# Le Langage SQL

Structured Query Language

est un langage informatique destiné à définir, modifier, exploiter des bases de données relationnelles.

C 'est un standard supporté par tous les logiciels SGBDR.

# Les trois volets de SQL

- DDL (Data Definition Language)

  Sert à définir et modifier la structure de la base.

- DML (Data Manipulation Language)

  Sert à manipuler (ajout, **interrogation**, modification) le contenu de la base.

- DCL (Data Control Language)

  Sert à contrôler les  accès à la base.

C 'est la partie **interrogation** du volet DML qui sera abordée dans ce cours.

# Requête sur une simple table

Une requête est une interrogation visant à extraire des informations d 'une base et pour commencer d 'une table unique. Le résultat de cette requête est elle même une table qui contient l 'information désirée.

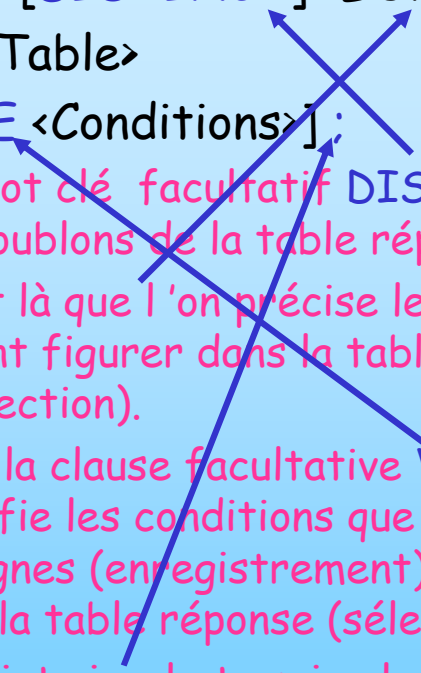

Cette extraction d 'information s 'opère selon deux opérations :

- ➔ Projection : on se limite à certaines colonnes de la table.
- ➔ La sélection : on ne retient que les lignes vérifiant une ou plusieurs conditions spécifiés.

Une même requête peut combiner ces deux opérations

# Structure de base d 'une requête SQL

SELECT [DISTINCT] <Liste_Attributs>
FROM <Table>
[WHERE <Conditions>] ;



- Le mot clé facultatif DISTINCT élimine les doublons de la table réponse.
- C 'est là que l 'on précise les attributs devant figurer dans la table réponse (projection).
- Dans la clause facultative WHERE, on spécifie les conditions que doivent vérifier les lignes (enregistrement) pour figurer dans la table réponse (sélection).
- Le point virgule termine la requête.

Exemples :

```
SELECT Ftitre
FROM film;

SELECT DISTINCT Cnat
FROM cineaste;

SELECT Mnom,  Mprenom
FROM membre;
```

# Expression d 'une condition

Une condition simple peut être :

- une comparaison :

  <Attribut> ***R*** <Valeur>

  où ***R*** est un des opérateurs :

  =, <, >, <=, >=, <>
- l 'appartenance à une liste :

  <Attribut> IN (<Valeur1>,… )
- l 'appartenance à un intervalle :

  <Attribut> BETWEEN <Valeurinf> AND <Valeursup>
- la ressemblance pour du texte

  <Attribut> LIKE <Motif>

  (avec Mysql les jokers sont _ et % au lieu de ? Et *)
- Absence ou non de valeur :

  <Attribut> IS NULL

  <Attribut> IS NOT NULL

Exemples :

```
SELECT *
FROM film
WHERE Fgenre = "Thriller";

SELECT Cnom
FROM cineaste
WHERE Cnaissance > 1950;

SELECT Mnom
FROM membre
WHERE Mnaissance BETWEEN 1970
    AND 1980;

SELECT Cnom, Cprenom
FROM cineaste
WHERE Cnom LIKE  "SC%";
```

# Expression de conditions

Les conditions simples peuvent être connectées par les opérateurs logiques :

⇨ La négation : NOT

NOT (<Condition>)

⇨ La conjonction AND

(<Condition1>) AND (<Condition2>)

⇨ La disjonction OR

(<Condition1>) OR (<Condition2>)

Ces opérateurs peuvent se combiner en respectant les règles de la logique...

Exemples :

```
SELECT *
FROM film
WHERE
(Fgenre = "Comédie"  ) AND (Fnat = "USA");

SELECT Cnom
FROM cineaste
WHERE
(Cnaissance > 1980  ) OR (Csexe = "F");

SELECT Ftitre
FROM film
WHERE
NOT (Fnat = "USA" ) AND (Fremake IS NULL);
```

# Quelques règles de logique

- ✓ Le AND et le OR sont associatifs et commutatifs
- ✓ $(C_1$ OR $C_2)$ AND $C_3$ n 'équivaut pas à $C_1$ OR $(C_2$ AND $C_3)$ mais à $(C_1$ AND $C_3)$ OR $(C_2$ AND $C_3)$
- ✓ NOT$(C_1$ OR $C_2)$ équivaut à NOT$(C_1)$ AND NOT$(C_2)$
- ✓ NOT$(C_1$ AND $C_2)$ équivaut à NOT$(C_1)$ OR NOT$(C_2)$

# Compléments : présentation de la table réponse

Renommer un attribut ou une table :

Pour renommer un attribut ou la table réponse on utilise le mot clé AS

SELECT <Attribut> AS <Nouveau_Nom>
FROM <Table> AS <Nouveau_Nom>

Trier les lignes de la table réponse :

Pour trier les enregistrements dans la table réponse on ajoute à la fin de la requête une clause ORDER BY

SELECT… FROM… WHERE…
ORDER BY <Attribut1> [DESC],
<Attribut2> [DESC]…;

Remarque : un attribut peut être utilisé comme dans la clause ORDER BY même s 'il ne figure pas dans la clause SELECT.

## Exemples :

```
SELECT Ftitre AS Titre
FROM film AS Films_1995
WHERE Fannee = 1995;

SELECT *
FROM membre
WHERE Msexe = "F"
ORDER BY Mnaissance DESC,
    Mnom;
```

# Requête multitable

Des questions d 'apparence simple :

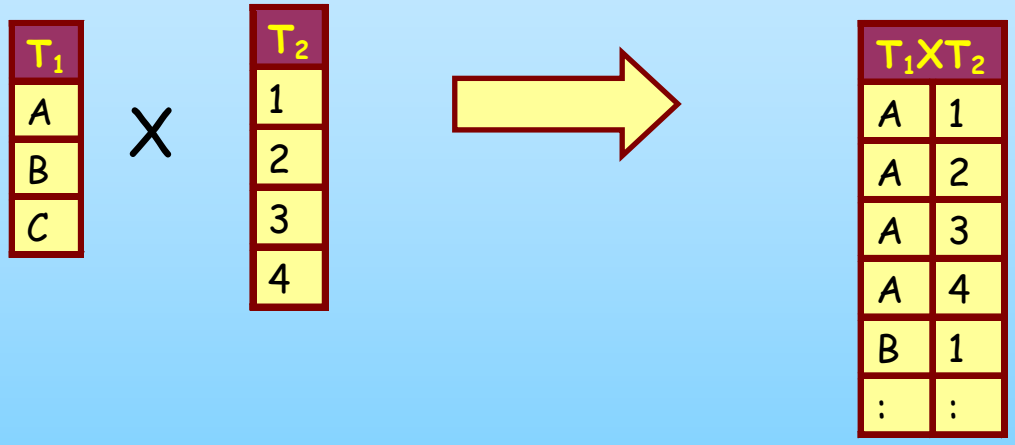
ex : Quel est le nom du réalisateur de tel film ?

ne peuvent en fait trouver leur réponse qu 'en explorant le contenu de plusieurs tables.

C 'est la technique des **jointures** qui permet d 'écrire en SQL de telles requêtes...

# Produit cartésien de deux tables

⇨ Le produit cartésien de deux tables $T_1$ , $T_2$ retourne une table qui combine chacune des lignes de la première table avec chacune des lignes de la deuxième :




⇨ Cette opération se réalise en SQL en écrivant :

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

Mais elle est à proscrire absolument parce que…

# Produit cartésien de deux tables (suite et fin)

- La table réponse n 'a guère de sens : on mélange tout…
- La table réponse peut être **énorme** :
  - ✓ Si $T_1$ a $l_1$ lignes et $c_1$ colonnes
  - ✓ Si $T_2$ a $l_2$ lignes et $c_2$ colonnes
  - ✓ $T_1$x $T_2$ a $c_1$+$c_2$ colonnes et **$l_1$x$l_2$ lignes**.

  De quoi mettre à genoux le serveur de données… ☠

Moralité : ne jamais citer deux tables dans une clause FROM sans imposer une condition de **jointure**…

# Jointure 1 : Le principe

**film**

| FilmID | Ftitre | FrealisateurID | Fgenre | Fnat | Fduree | Fannee | Fremake |
|---|---|---|---|---|---|---|---|
| F001 | Million Dollar Baby | C001 | Drame | USA | 132 | 2004 | NULL |
| : | : | | | | | | : |

La jointure s 'opère en spécifiant l 'égalité entre l 'occurrence d 'un attribut de la 1° table avec l 'occurrence d 'un attribut de la 2° table.

**cineaste**

| CineasteID | Cnom | Cprenom | Cnat | Cnaissance | Csexe |
|---|---|---|---|---|---|
| C001 | EASTWOOD | Clint | USA | 1930 | H |
| C040 | SWANK | Hilary | USA | 1974 | F |
| : | | | | | : |

Seul les lignes pour lesquelles ces 2 attributs sont égaux seront incluses dans la table réponse.

# Jointure 2 : La traduction en SQL

⇨ Syntaxe :

```
SELECT <liste_attributs>
FROM <T1> JOIN <T2>
    ON (<T1.Attribut> =
          <T2.Attribut>)
[WHERE
 <autres_conditions>] ;
```

Exemple : quels sont les noms des réalisateurs de comédie ?

```
SELECT  Cnom
FROM film JOIN cineaste
      ON (FrealisateurID =
           CineasteID)
WHERE
 (Fgenre = "Comédie" ) ;
```

# Jointure 3 : le préfixage des attributs

Le préfixage des attributs consiste à les faire précéder du nom de la table dont ils sont issus. On écrit :

<Table>.<Attribut>

Il peut contribuer à la clarté des requêtes.

Il est **obligatoire**, si deux attributs issus de deux tables différentes portent le même nom.

Il sera aussi obligatoire dans les auto-jointures qui vont suivre...

Exemple, voici la requête précédente préfixée :

```
SELECT  cineaste.Cnom
FROM film JOIN cineaste
ON (film.FrealisateurID = cineaste.CineasteID)
WHERE
 (film.Fgenre = "Comédie" ) ;
```

# Jointure 4 : l 'auto-jointure

Il peut arriver que l 'obtention d 'une information exige une double lecture d 'une même table, par exemple : trouver les titres de films qui ont des remake. SQL permet alors d 'effectuer une jointure d 'une table avec elle même en utilisant un « **alias** ».

```
SELECT <Liste_attributs>*
FROM <T> JOIN <T> AS <T1>
ON (<T.Attribut> = <T1.Attribut>)
WHERE <Conditions> ;
```

* Les attributs doivent être préfixés pour indiquer s 'ils proviennent de la 1° lecture <T> ou de la deuxième <T1>.

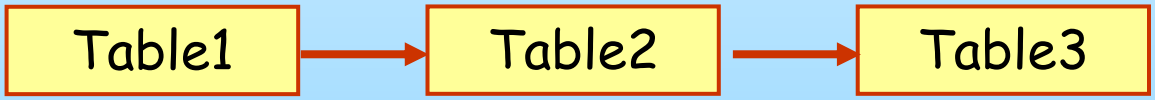
Exemple :

```
SELECT f1.Ftitre, f1.Fannee
FROM film JOIN film AS f1
        ON film.Fremake = f1.FilmID
WHERE film.Fremake IS NOT NULL ;
```

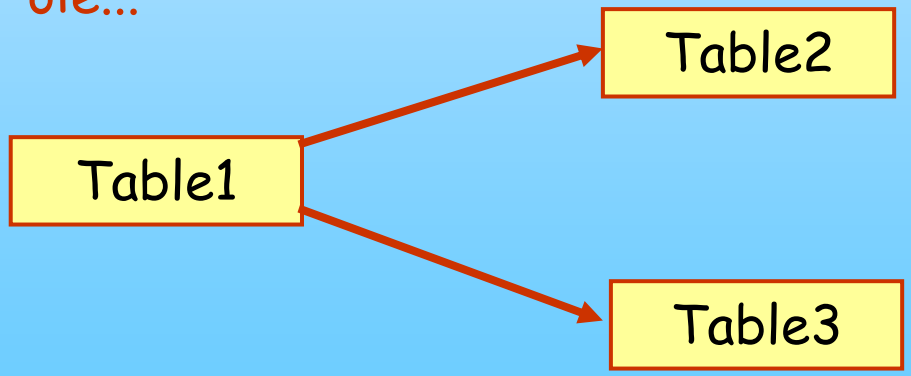
# Jointure 5 : 3 tables (voir + si affinité)

En fait, bien souvent, c 'est dans trois tables (sinon plus) qu 'il faut quérir les informations. La jointure de trois tables peut s 'opérer, selon le cas, suivant deux schémas :

⇨ A la queue leu leu...

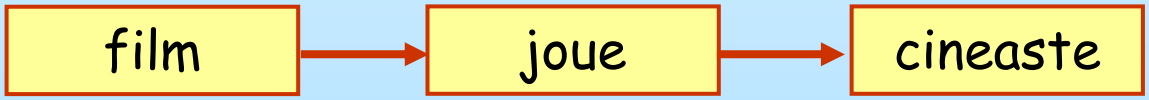



⇨ En patte d 'oie...



# Jointure 6 : 3 tables à la queue leu leu

Quels sont les noms et prénoms des acteurs de «Titanic » ? Question simple, mais dont la réponse exige les visites successives des tables :



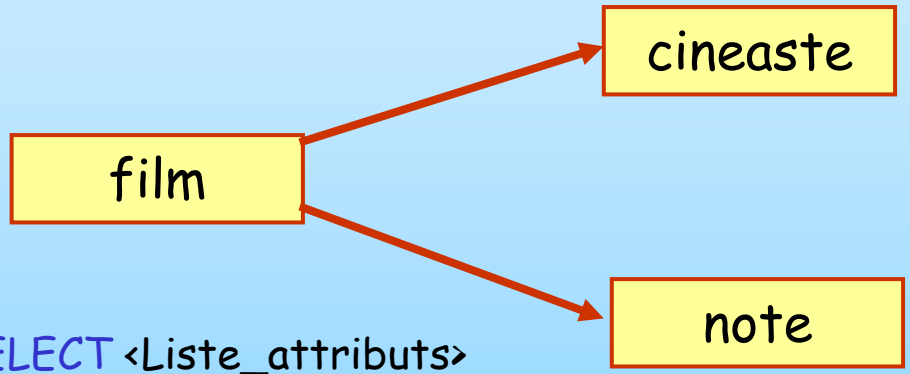

```
SELECT <Liste_attributs>
FROM <T1> JOIN <T2> JOIN <T3>
    ON (<T1.Attribut> = <T2.Attribut>)
    ON (<T2.Attribut> = <T3.Attribut>)
WHERE <Conditions> ;
```

Exemple :

```
SELECT Cnom, Cprenom
FROM film JOIN joue JOIN cineaste
    ON (FilmID = JFilmID)
    ON (JCineasteID = CineasteID)
WHERE Ftitre =  "Titanic" ;
```

# Jointure 7 : 3 tables en patte d'oie

Quel est le nom du réalisateur de «Titanic » et les notes obtenues ?
Question simple, mais dont la réponse exige les visites des tables :




```
SELECT <Liste_attributs>
FROM <T1> JOIN <T2>
      ON (<T1.Attribut> = <T2.Attribut>)
            JOIN <T3>
       ON (<T1.Attribut> = <T3.Attribut>)
WHERE <Conditions> ;
```

Exemple :

```
SELECT Ftitre, Cnom, Nnote
FROM film JOIN cineaste
    ON (FrealisateurID = CineasteID)
            JOIN note
    ON (FilmID = NFilmID)
WHERE Ftitre =  "Titanic" ;
```

# ✈Jointure 8 : jointure "externe"

Jusqu'à présent, seul les lignes où une correspondance était trouvée entre les attributs des tables jointes figurait dans la table réponse. Avec les jointures "externes", cette exigence s'évanouit :

```
SELECT <Liste_attributs>
FROM <T1> LEFT [OUTER] JOIN <T2>
      ON (<T1.Attribut> = <T2.Attribut)
WHERE <Conditions> ;
```

Dans la table réponse figureront toutes les lignes issues de la table de **gauche** T1, y compris celles qui n'ont pas de correspondance dans la table T2. Pour ces dernières l'attribut manquant de la table T2 sera mis à NULL.

Une construction symétrique se fait avec RIGHT au lieu de LEFT.

Exemple :

```
SELECT DISTINCT Cnom, CineasteID,
                FrealisateurID
FROM cineaste LEFT JOIN film
      ON CineasteID = FrealisateurID ;
```

Listera tous les cinéastes y compris ceux n'ayant jamais réalisé de films. Pour ces derniers, l'attribut FrealisateurID sera mis à NULL.

En ajoutant la clause :

```
WHERE FrealisateurID IS NULL
```

elle listera donc les "purs" interprètes.

# ✈Jointure 9 : jointure "non équi"

Jusqu'à présent, la condition de jointure a été une égalité (on parle alors d'équi-jointure). Mais il est possible d'utiliser d'autres opérateurs de comparaison que l'égalité :

```
SELECT <Liste_attributs>
FROM <T1> JOIN <T2>
      ON (<T1.Attribut> R <T2.Attribut>)
WHERE <Conditions> ;
```

Où R désigne un des opérateurs de comparaison:

<, <=, >, >=, <>

Exemple : quels sont les films tournés après la date de naissance de "DICAPRIO" (Leonardo pour les intimes) ?

```
SELECT Ftitre, Fannee, Cnaissance
FROM cineaste JOIN film
    ON (Cnaissance < Fannee)
WHERE Cnom =  "DICAPRIO" ;
```

# Le calcul

SQL permet d'effectuer nombre de calculs.
On distingue :

- ✓ le calcul **horizontal** qui vise, sur chaque ligne, à calculer et afficher des valeurs à partir des valeurs des attributs de la ligne.

- ✓ Le calcul **vertical** qui permet l'obtention de statistiques sur les valeurs d'un attribut

# Calcul horizontal 1 : expressions

Ces expressions sont constituée à partir d'attributs, d'opérateurs et de fonctions. Selon la nature des attributs en jeu, il faut distinguer :

➔ Calcul arithmétique : il utilise les opérateurs classiques de l'arithmétique :

**+, -, *, /, DIV, MOD, ^**

plus, si besoin est, de nombreuses fonctions...

Exemples : YEAR(CURDATE())- Cnaissance, Fduree DIV 60, Fduree MOD 60

➔ Calcul textuel : il utilise des fonctions de traitement des chaînes de caractères :

CONCAT(), UPPER(), LOWER(), LENGTH()...

Exemples : CONCAT(Mprenom, ".", Mnom, "@uhb.fr"), UPPER(Ftitre)

# Le calcul horizontal 2 : la mise en place

Ces expressions calculées peuvent se loger dans la clause SELECT pour faire apparaître de nouveaux attributs dans la réponse :

SELECT …, <Expression> AS <Nom>,…

ou dans la clause WHERE pour exprimer des conditions :

WHERE (<Expression> **R** <Valeur>) AND…

Exemples :

```
SELECT Cnom,
            YEAR(CURDATE())-Cnaissance AS Age
FROM cineaste ;

SELECT CONCAT(Mprenom, ".", Mnom,"@uhb.fr")
    AS Mail
FROM membre ;


SELECT Ftitre
FROM film
WHERE LENGTH(Ftitre) < 15 ;
```

# Le calcul vertical 1 : les fonctions

Il vise à donner des statistiques sur les occurrences d'un (ou plusieurs ) attribut en utilisant les fonctions :

- ✓ COUNT( <Attribut> ) : pour le nombre d'occurrences d'un attribut.
- ✓ COUNT(DISTINCT <Attribut> ) : pour le nombre d'occurrences distinctes d'un attribut.
- ✓ COUNT(*) : pour le nombre de lignes d'une table.

et pour les attributs numériques :

- ✓ SUM( <Attribut> ), pour la somme.
- ✓ MAX(<Attribut> ), MIN(<Attribut> ), pour le minimum et le maximum.
- ✓ AVG( <Attribut> ), pour la moyenne.
- ✓ STD( <Attribut> ), pour l'écart-type.
- ✓ VARIANCE( <Attribut> ), pour la variance.

# Le calcul vertical 2 : la mise en place

La (ou les) fonction de calcul se loge dans la clause SELECT

SELECT ...<Fonction(Attribut)> AS* <Nom>...
FROM... [WHERE*...];

* Le AS est recommandé pour la lisibilité de la réponse.
* Si la requête inclut une clause WHERE le calcul ne prend en compte que les lignes qu'elle sélectionne.

☠ **Ces fonctions retournent un résultat unique et la table réponse ne contient donc qu'une seule ligne. Elles sont donc incompatibles avec l'affichage d'un attribut.**

```
SELECT COUNT(DISTINCT Cnat)
                AS "Nb nationalités"
FROM cineaste ;

SELECT AVG(Fduree) AS "Durée moyenne ",
          STD(Fduree) AS "Ecart-type"
FROM film
WHERE Fgenre = "Drame";
```

~~SELECT COUNT(Fgenre), Fgenre~~
~~FROM film ;~~

# Regroupement et calcul

Est il possible de faire opérer ces fonctions statistiques sur des des groupes de ligne, pour calculer par exemple le nombre de films par genre ? Oui en utilisant une clause GROUP BY.

SELECT <Attribut_groupement>* ,
<Fonction(Attribut)>
FROM <Table>
GROUP BY <Attribut_groupement>* ;
*<Attribut_groupement> doit figurer impérativement dans les deux clauses.

Une clause HAVING peut être ajoutée pour opérer une sélection sur les groupes :
HAVING <Condition_sur_groupe>
Cette condition peut porter sur :

- ✓ la valeur retournée par la fonction
- ✓ la valeur de l'attribut de regroupement.

```
SELECT Fgenre, COUNT(FilmID)
FROM film
GROUP BY Fgenre ;

SELECT Fgenre, COUNT(FilmID)
FROM film
GROUP BY Fgenre
HAVING COUNT(FilmID)>2;

SELECT Fgenre, COUNT(FilmID)
FROM film
GROUP BY Fgenre
HAVING Fgenre <> "SF" ;
```

# Les Requêtes imbriquées ou sous-requêtes

Une requête interroge des tables pour produire une… table. L'idée donc de chaîner deux (ou plus) requêtes est donc assez naturelle. Une requête dite **principale** va alors utiliser dans sa clause **WHERE** les résultats d'une autre requête dite **sous-requête**.

Le schéma général :

```
SELECT ...
FROM ...
WHERE <expression> (SELECT
                    FROM ...
                    )
;
```

La sous-requête

doit être précisé selon la nature du résultat retourné par la sous-requête.

# Sous requête 1

## La sous-requête retourne une valeur unique d'un attribut unique

```
SELECT…
FROM..
WHERE <attribut> R (SELECT
                    FROM …
                    )
                    ;
```

R est un des opérateurs :

=, <, >, <=, >=, <>

Spécifie comme condition de sélection pour <attribut> la vérification de la comparaison R avec la valeur unique retournée par la sous-requête.

Exemple :

Liste des films d'une durée supérieure à la moyenne.

```
SELECT Ftitre
FROM film
WHERE Fduree > (SELECT
                AVG(Fduree)
                FROM film)
                ;
```

- ✓ La sous-requête retourne la durée moyenne des films.
- ✓ La requête principale sélectionne les films d'une durée supérieure.

# Sous requête 2

## La sous-requête retourne une liste de valeurs d'un attribut unique

```
SELECT…
FROM…
WHERE <attribut>  [NOT] IN
                  (SELECT
                  FROM…
                  )
                  ;
```

Spécifie comme condition de sélection pour <attribut>
l'appartenance IN
ou non NOT IN
à la liste retournée par la sous-requête.

Exemple :

Liste des cinéastes dont la nationalité n'est pas une nationalité de film.

```
SELECT Cnom
FROM cineaste
WHERE Cnat NOT IN (SELECT Fnat
                   FROM film)
                   ;
```

- ✓ La sous-requête retourne la liste des pays producteurs de films.
- ✓ La requête principale sélectionne les cinéastes dont la nationalité n'est pas dans cette liste.

# Sous requête 3

## La sous-requête retourne une liste de valeurs d'un attribut unique (bis)

```
SELECT...
FROM...
WHERE <attribut> R ALL|SOME
                    (SELECT
                    FROM...
                    ) ;
```

R est un des opérateurs :

=, <, >, <=, >=, <>

Avec ALL, la comparaison R doit être vérifiée pour tous les éléments de la liste

Avec SOME (on peut aussi utiliser ANY), la comparaison doit être vraie pour un élément au moins de la liste.

Exemple :

Liste des films qui durent plus longtemps que tous les films américains.

```
SELECT Ftitre
FROM film WHERE Fduree > ALL
            (SELECT Fduree FROM film
            WHERE Fnat ="USA") ;
```

- ✓ La sous-requête retourne la liste des durées des films américains.
- ✓ La requête principale sélectionne les films dont la durée est strictement supérieure à toutes les valeurs de cette liste.

# Sous requête 4

## La sous-requête retourne un n-uplet unique d'attributs

```
SELECT…
FROM…
WHERE (<attribut1>, <attribut2>…)  R
                    (SELECT
                    FROM…
                    ) ;
```

Entre n-uplets, les seuls opérateurs de comparaisons possibles sont l'égalité et la différence
R est donc un des opérateurs :

=, <>

Exemple :

Liste des films de même genre et même nationalité que Titanic.

```
SELECT Ftitre
FROM  film
WHERE (Fgenre, Fnat) =
            (SELECT Fgenre, Fnat
            FROM film
            WHERE Ftitre ="Titanic")
            ;
```

- ✓ la sous requête retourne le 2-uplet (Drame,USA)
- ✓ La requête principale sélectionne donc les films tels que :
  (Fgenre, Fnat) = (Drame,USA)

# ✈ Sous requête 5

## La sous-requête retourne une liste de n-uplets d'attributs (une table avec plusieurs lignes et colonnes)

```
SELECT...
FROM...
WHERE (<attribut1>, <attribut2>...)
                    [NOT] IN
                    (SELECT
                    FROM...
                    ) ;
SELECT...
FROM...
WHERE (<attribut1>, <attribut2>...)
                    =|<> SOME|ALL
                    (SELECT
                    FROM...
                    ) ;
```

Exemple :

Liste des membres du club ayant le même âge et le même sexe qu'un cinéaste.

```
SELECT Mnom,
FROM membre
WHERE (Mnaissance, Msexe) IN
            (SELECT  Cnaissance, Csexe
            FROM  cineaste) ;
```

- ✓ la sous requête retourne une liste de 2-uplets du genre (1973, H), (1940, F)...
- ✓ La requête principale sélectionne donc les membres pour lesquels : (Mnaissance, Msexe) est dans la liste.

# ✈ Sous requête 6
## Sous-requête corrélée (synchronisée)

Dans les schémas précédents, la sous-requête était exécutée, puis la requête principale. Dans les requêtes dites corrélées, la sous-requête dépend pour son exécution de la valeur d'un attribut fournie par la requête principale. L'exécution se fait alors ligne à ligne.

```
SELECT <Liste_attributs>
FROM  <T1>
WHERE <T1>.<attribut> R
      (SELECT <T2>.<attribut>
      FROM <T2>
      WHERE <T2>.<attribut> R
            <T1> .<attribut>) ;
```

Exemple :

Quels sont les films d'un genre différent de celui dont ils sont le remake ?

```
SELECT F1.Ftitre FROM film AS F1
WHERE F1.Fgenre <>
      (SELECT F2.Fgenre FROM film AS F2
      WHERE F2.FilmID = F1.Fremake) ;
```

# Sous requête 7
# A quoi servent les sous-requêtes ?

- Pour certaines questions elles sont indispensables. Par exemple, la question :

  **Quels sont les films d'une durée supérieure à la moyenne ?**

  ne saurait se traduire en SQL sans recourir à cette méthode.
- Pour d'autres questions elles sont une alternative à la jointure. Par exemple :

  **Quels sont les cinéastes de même âge que Michel Bouquet ?**

  Peut se traduire :

```
SELECT c1.Cnom, c1.Cprenom
FROM cineaste JOIN cineaste AS c1
  ON cineaste.cnaissance = c1.cnaissance
WHERE (cineaste.Cnom = "BOUQUET") ;
```

```
SELECT Cnom, Cprenom
FROM cineaste
WHERE Cnaissance = (SELECT Cnaissance
                    FROM  cineaste
                    WHERE Cnom = "BOUQUET") ;
```

La méthode des sous-requêtes qui fragmente la question en deux facilite souvent la mise au point.

# Opérateurs ensemblistes 1
## Le principe

- La table réponse retournée lors de l'exécution d'une requête est fondamentalement un ensemble d'occurrences. D'où l'idée d'utiliser les opérateurs ensemblistes :
  - Réunion : ∪
  - Intersection : ∩
  - Différence : -

  afin de combiner les résultats de deux requêtes dans une table unique.

- Cependant, ce procédé n'est envisageable que si les tables engendrées par les deux requêtes :
  - possèdent le même nombre d'attribut,
  - ont leurs attributs correspondants de même type.

  et s'il n'existe pas de solution simple avec OR ou AND.

# Opérateurs ensemblistes 2
## La mise en œuvre

<requête1>
UNION
<requete2>

Liste de tous les pays présents dans la base :

```
(SELECT Fnat FROM film)
UNION
(SELECT Cnat FROM cineaste) ;
```

Mysql ignore INTERSECT et MINUS, mais on peut y pallier par des sous-requêtes :

<requête1>
INTERSECT
<requete2>

Liste des pays de films et de cinéastes :

```
SELECT DISTINCT Fnat FROM film
WHERE Fnat IN (SELECT Cnat FROM cineaste) ;
```

<requête1>
MINUS
<requete2>

Liste des pays de films qui ne sont pas des pays de cinéastes :

```
SELECT DISTINCT Fnat FROM film
WHERE Fnat NOT IN (SELECT Cnat FROM cineaste) ;
```

# Opérateurs ensemblistes 3

## Illustration