ORACLE SQL - LID -

Langage d'interrogation des données

Structure générale d'une requête SQL

L'ordre SELECT possède six clauses différentes, dont seules les deux premières sont obligatoires.

SELECT …

FROM …

WHERE …

GROUP BY …

HAVING …

ORDER BY …

Traduction francisée simplifiée : je sélectionne (select) les éléments à afficher à partir (from) des tables lorsque (where) des conditions sont vérifiées en groupant (group by) par tuples précisés dont chacun satisfait (having) une condition en les ordonnant (order by) par une condition de tri.

SELECT [DISTINCT] { [nomTable.]* | <expr> [,<expr>]... }

FROM nomTable [nomAlias] [,nomTable [nomAlias]] ...

[WHERE <conditionSelectionLigne>]

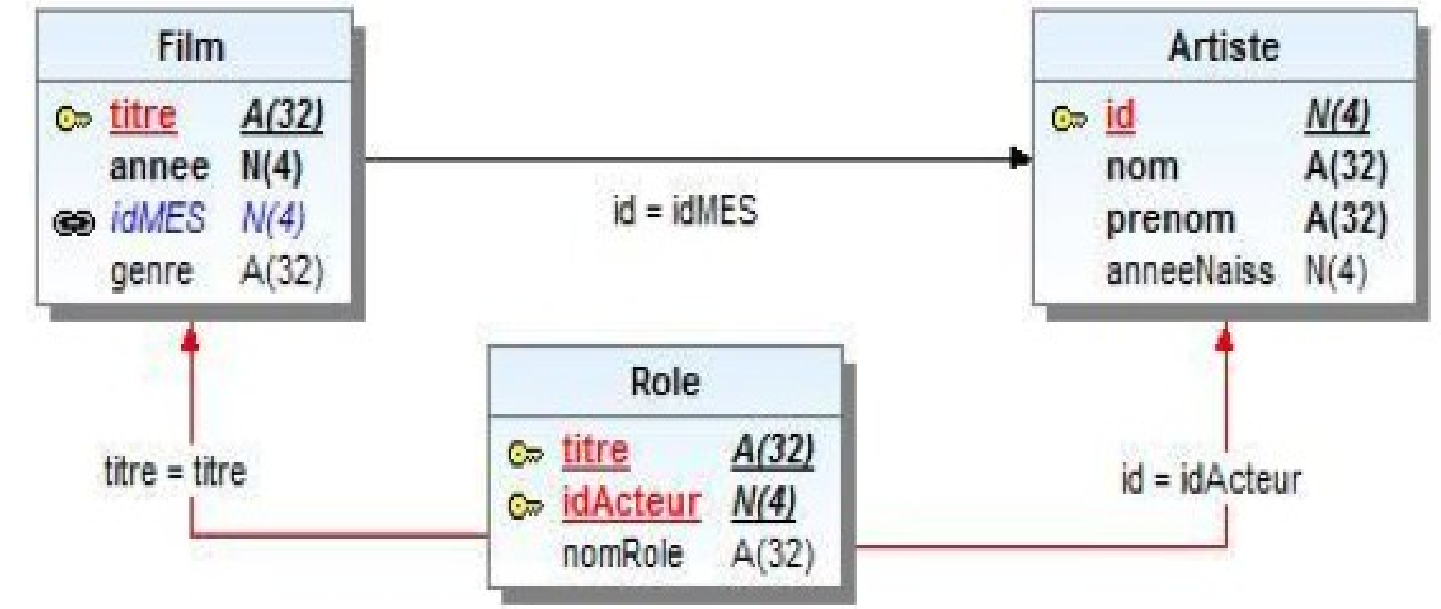
[GROUP BY <expr> [,<expr>]... [HAVING <conditionSelectionGroupe>]]

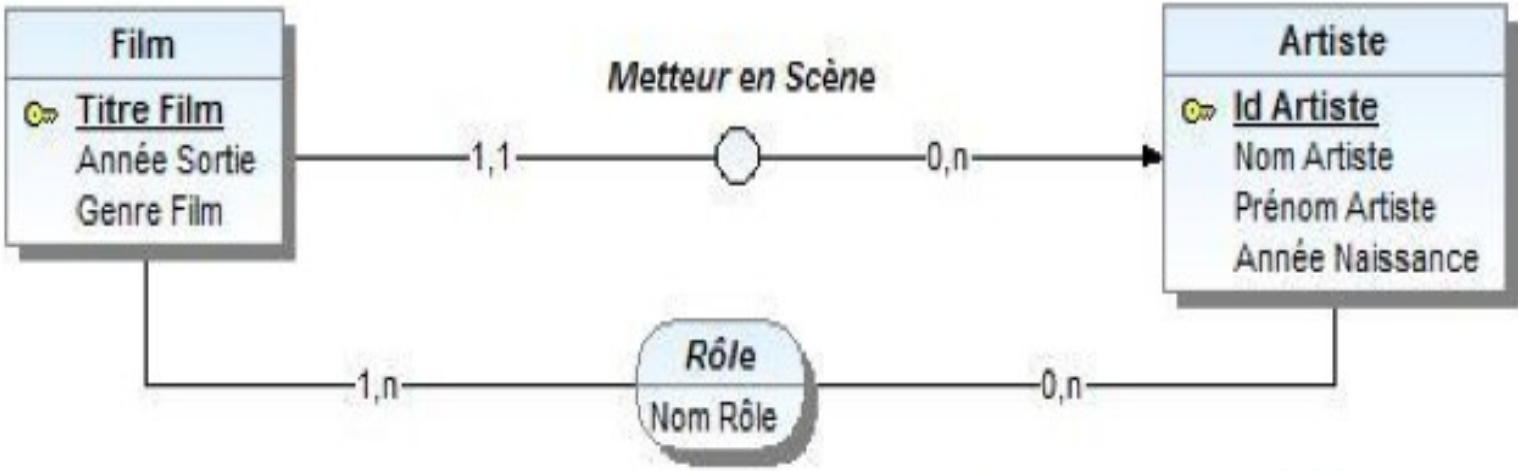
[{ UNION | INTERSECT | MINUS } <instructionSelect>]

[ORDER BY { <expr> | position | entête colonne } [ASC | DESC] [, { <expr> | position | entête colonne } [ASC | DESC] ...]]

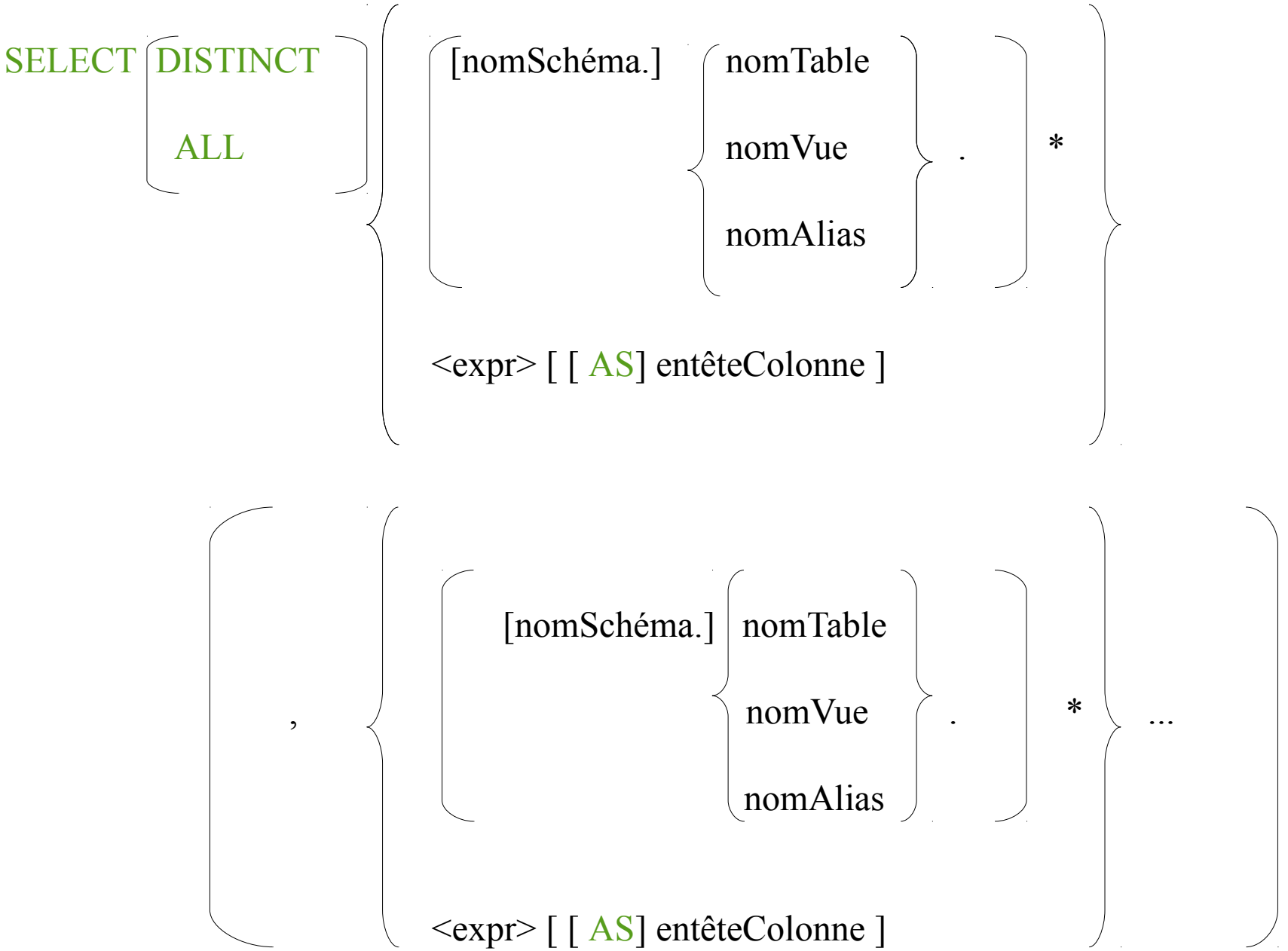
Une première vision simplifiée

Fil rouge du cours : films, artiste, metteur en scène, rôle,...





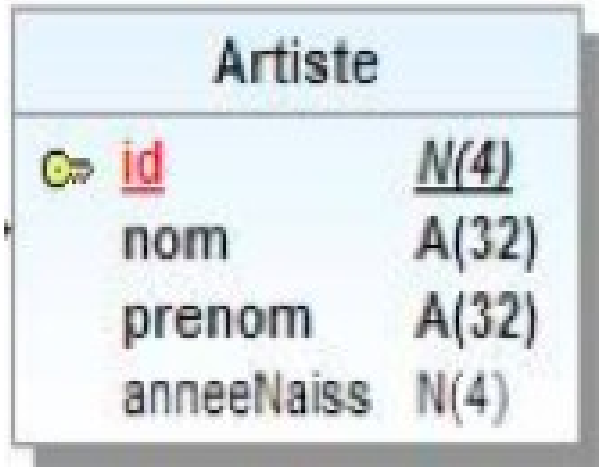
SELECT



Syntaxe du SELECT

(Q1) ? $\leftrightarrow \Pi_{\text{titre,annee,idMES,genre}}(\text{Film})$

Affiche tous les attributs de la table film pour chaque Film.



(Q2) ?

Affiche tous les genres de la table Film en éliminant les doublons.

(Q3) ?

Affiche pour chaque artiste son age en faisant 2017- son année de naissance

Cette façon de calculer n'est pas robuste car elle ne fonctionnera pas en 2018.

Nous verrons plus tard en introduisons les fonctions comment concevoir une requête robuste.

Exemples de base manipulant la clause SELECT

(Q1) SELECT * FROM Film $\leftrightarrow \Pi_{\text{titre,annee,idMES,genre}}(\text{Film})$

Affiche tous les attributs de la table film pour chaque Film.

(Q2) SELECT DISTINCT genre FROM Film

Affiche tous les genres de la table Film en éliminant les doublons.

(Q3) SELECT nom, prenom, 2017-anneeNaiss AS Age FROM Artiste

Affiche pour chaque artiste son age.

Cette façon de calculer n'est pas robuste car elle ne fonctionnera pas en 2014.

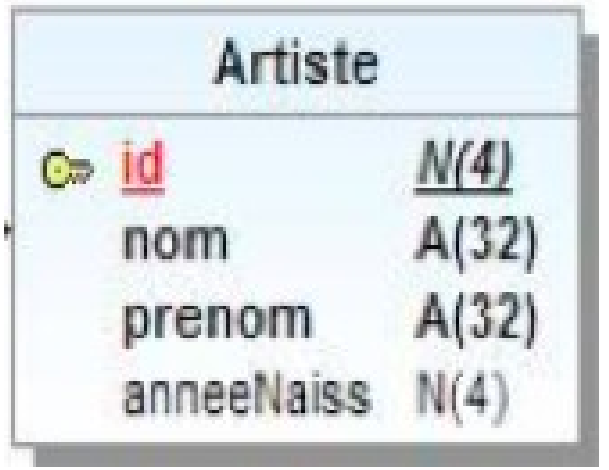
Nous verrons plus tard en introduisons les fonctions comment concevoir une requête robuste.

Exemples de base manipulant la clause SELECT

On souhaite maintenant un affichage colonne « Age Artiste » au lieu de « Age »

Que faut il ajouter à cette requête (Q3) ?

SELECT nom, prenom, 2017-anneeNaiss AS Age FROM Artiste



Exemples de base manipulant la clause SELECT

(Q4) SELECT nom, prenom, 2017-anneeNaiss "Age Artiste" FROM Artiste

Notons les guillemets autour de Age Artiste du fait qu'elle contienne un espace.

(Q5) La requête suivante va provoquer une erreur car on utilise « Age » dans la clause where

SELECT nom, prenom, 2017-anneeNaiss Age FROM Artiste

WHERE Age > 20

Exemples de base manipulant la clause SELECT

Remarque : Le nom complet d'une colonne d'une table est le nom de la table suivi d'un point et du nom de la colonne (cf. Q6)

(Q6) SELECT Artiste.nom, Artiste.prenom FROM Artiste

Le nom de la table peut être omis quand il n'y a pas d'ambiguïté. Il doit être précisé s'il y a une ambiguïté, ce qui peut arriver quand on fait une sélection sur plusieurs tables à la fois et que celles-ci contiennent des colonnes qui ont le même nom.

La clause SELECT

Cette clause permet d'indiquer quelles colonnes, ou quelles expressions doivent être retournées par l'interrogation.

SELECT [DISTINCT] *

SELECT [DISTINCT] exp1 [[AS] nom1], exp2 [[AS] nom2], ...

exp1, exp2, ... sont des expressions : nom d'une colonne, opération sur des colonnes, etc (cf. Q3)

nom1, nom2, ... sont des noms facultatifs de 30 caractères maximum, donnés aux expressions.

Si un nom contient des séparateurs (espace, caractère spécial), ou s'il est identique à un mot réservé SQL (exemple : DATE), il doit être mis entre guillemets (cf. Q4)

La clause SELECT

SELECT [DISTINCT] *

SELECT [DISTINCT] exp1 [[AS] nom1], exp2 [[AS] nom2], ...

Le mot clé AS (optionnel) constitue le titre de la colonne dans l'affichage du résultat (cf. Q3)

Les nom1,... ne peuvent pas être utilisés dans les autres clauses (where par exemple) (cf. Q5)

* signifie que toutes les colonnes de la table sont sélectionnées (cf. Q1)

Le mot clé DISTINCT permet d'éliminer les doublons (cf. Q2)

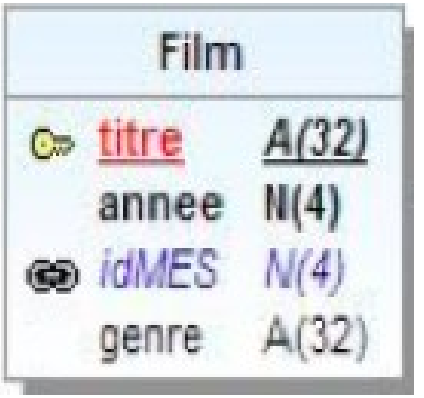
Des fonctions dans le SELECT : chaines de caractères

|| → concaténation de chaines

LENGTH(chaîne) → prend comme valeur la longueur de la chaîne.

SUBSTR(chaîne, pos [,long]) → extrait de la chaîne chaîne une sous-chaîne de longueur « long » commençant en position « pos » de la chaîne

UPPER(chaîne) → convertit les minuscules en majuscules (resp LOWER(chaîne))



Exemple :

?

Donne les trois premières lettre de chaque film concaténées à son année de sortie.

?

Donne les titres des films en majuscule.

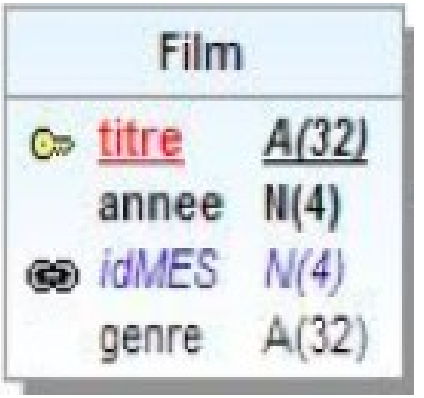
Des fonctions dans le SELECT : chaines de caractères

|| → concaténation de chaines

LENGTH(chaîne) → prend comme valeur la longueur de la chaîne.

SUBSTR(chaîne, pos [,long]) → extrait de la chaîne chaîne une sous-chaîne de longueur « long » commençant en position « pos » de la chaîne

UPPER(chaîne) → convertit les minuscules en majuscules (resp LOWER(chaîne))



Exemple :

SELECT SUBSTR(titre,1,3) || annee AS "Titre concat année" From Film

Donne les trois premières lettre de chaque film concaténées à son année de sortie.

SELECT UPPER(titre) AS "Titre en majuscule" From Film

Donne les titres des films en majuscule.

Des fonctions dans le SELECT : manipulation de date

TO_CHAR (date, format) → date est un attribut déclarée « DATE » et format est une chaine entre cotes simples indiquant le format sous lequel sera affichée la date . C'est une combinaison de codes :

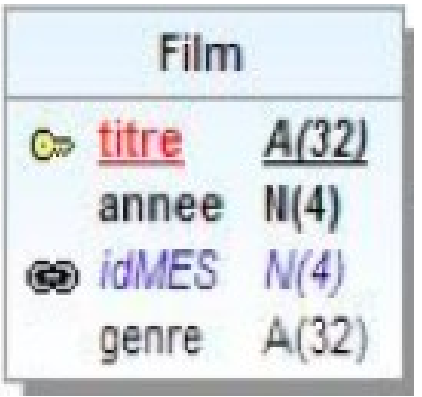
YYYY → année MM → numéro du mois DD → numéro du jour dans le mois

HH24 → heure sur 24 heures MI → minutes SS → secondes

Tout caractère spécial inséré dans le format sera reproduit dans la chaîne de caractère résultat.

Exemple : supposons que la table film contienne un attribut dateDebutTournage de type « DATE ». La requête suivant affiche le titre et date de tournage de chaque film :

?



Des fonctions dans le SELECT : manipulation de date

TO_CHAR (date, format) → date est un attribut déclarée « DATE » et format est une chaine entre cotes simples indiquant le format sous lequel sera affichée la date . C'est une combinaison de codes :

YYYY → année MM → numéro du mois DD → numéro du jour dans le mois

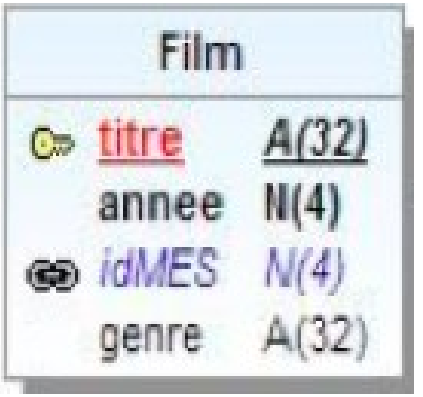
HH24 → heure sur 24 heures MI → minutes SS → secondes

Tout caractère spécial inséré dans le format sera reproduit dans la chaîne de caractère résultat.

Exemple :

Supposons que la table film contienne un attribut dateDebutTournage de type « DATE ».

La requête suivant affiche le titre et date de tournage de chaque film au format 31/12/2017 14:00:00



Des fonctions dans le SELECT : arithmétique

ABS(n) → valeur absolue de n

MOD(n1, n2) → n1 modulo n2

POWER(n, e) → n à la puissance e

SIGN(n) → -1 si n<0, 0 si n=0, 1 si n>0

SQRT(n) → racine carrée de n

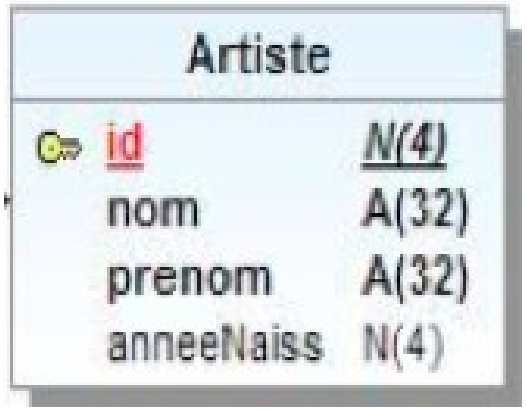
GREATEST(n1, n2,...) → maximum de n1, n2,...

LEAST(n1, n2,...) → minimum de n1, n2,...

TO_CHAR(n, format) → convertit n en chaîne de caractères

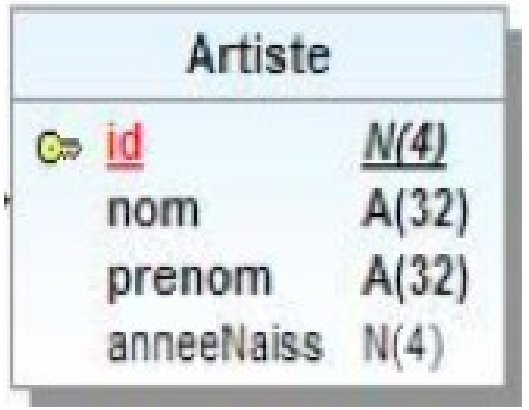
TO_NUMBER(chaîne) → convertit la chaîne de caractères en numérique

Exemple : afficher le carré de l'année de naissance en nommant la colonne : Date de naissance au carrée



Des fonctions dans le SELECT : arithmétique

POWER(n, e) → n à la puissance e



SELECT POWER(anneeNaiss,2) AS "Date de naissance au carrée" FROM Artiste

Des fonctions dans le SELECT : manipulation de date

SYSDATE → renvoie la date et heure du système. Il faut alors la combiner avec TO_CHAR (date, format) pour formater l'affichage.

Exemple : supposons que la table film contienne un attribut dateDebutTournage de type « DATE ».

La requête suivante affiche la date et heure du système formatée en DD/MM/YYYY HH24:MI:SS

```
SELECT TO_CHAR(SYSDATE, 'DD/MM/YYYY HH24:MI:SS')
            AS "date et heure à cet instant précis"
            FROM DUAL
```

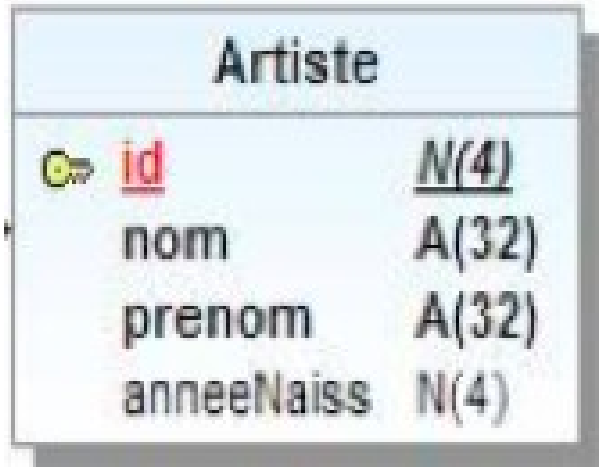
La table DUAL est une table spéciale d'une seule colonne utilisée généralement comme "pseudo-colonne" pour les requêtes du genre SYSDATE ou USER. Elle contient une seule colonne de type VARCHAR(1) appelée DUMMY

Des fonctions dans le SELECT : manipulation de date (suite)

On peut combiner des fonctions.

Par exemple on souhaite afficher l'age de chaque artiste et on souhaite faire ça avec une requête robuste : c'est-à-dire qu'on aura pas à la changer avec le temps.

Il faut combiner TO_NUMBER , TO_CHAR et SYSDATE



Des fonctions dans le SELECT : manipulation de date (suite)

On peut combiner des fonctions.

Par exemple on souhaite afficher l'age de chaque artiste et on souhaite faire ça avec une requête robuste : c'est-à-dire qu'on aura pas à la changer avec le temps.

```
SELECT  nom, prenom,
        TO_NUMBER(TO_CHAR(SYSDATE,'YYYY'))- anneeNaiss
        As " Age artiste"  FROM Artiste
```

Pseudos colonnes et SELECT

user : le nom de l'utilisateur courant.

rownum : le numéro des lignes sélectionnées par une clause where.

Select rownum, nom, prenom FROM Artiste WHERE …

FROM

Une seconde vision détaillé : FROM … WHERE

FROM { [nomSchéma.] { nomTable | NomVue } [nomAlias] | (< sous requête SELECT >) nomAlias }

[, { [nomSchéma.] { nomTable | NomVue } [nomAlias] | (< sous requête SELECT >) nomAlias } ...]

[WHERE <conditionSelectionLigne>]

La clause : FROM Table1 [synonyme1] , Table2 [synonyme2] , ...

Définie la liste des tables participant à l'interrogation.

Il est possible de lancer des interrogations utilisant plusieurs tables à la fois.

synonyme1, synonyme2,... sont des synonymes attribués facultativement aux tables pour le temps de la sélection → utilisée pour lever certaines ambiguïtés, quand la même table est utilisée de plusieurs façons différentes dans une même interrogation.

Quand on a donné un synonyme à une table dans une requête, elle n'est plus reconnue sous son nom d'origine dans cette requête.

Exemple : les requêtes suivantes sont équivalentes.

$$\Pi_{Art.nom,Art.prenom}(\rho_{Art}(Artiste))$$

SELECT Art.nom, Art.prenom FROM Artiste Art

SELECT nom, prenom FROM Artiste

La clause : FROM Table1 [synonyme1] , Table2 [synonyme2] , ...

Quand on précise plusieurs tables dans la clause FROM, on obtient le produit cartésien des tables.
A partir d'ORACLE 9 l'instruction FROM Table 1 CROSS JOIN Table 2 spécifie plus explicitement ce produit.

Exemple : $\Pi_{\text{Table1.attribut1,Table2.attribut2}}(\text{Table1} \times \text{Table2})$

ORACLE <9 : SELECT Table1.Attribut1, Table2.Attribut2 FROM Table1,Table2

ORACLE >8 : SELECT Table1.Attribut1, Table2.Attribut2
FROM Table1 CROSS JOIN Table2

Exemple : $\Pi_{\text{T1.attribut1,T2.attribut2}}(\rho_{T1}(\text{Table1}) \times \rho_{T2}(\text{Table2}))$

ORACLE <9 : SELECT T1.Attribut1, T2.Attribut2 FROM Table1 T1,Table2 T2

ORACLE >8 : SELECT T1.Attribut1, T2.Attribut2
FROM Table1 T1 CROSS JOIN Table2 T2

WHERE

La clause : WHERE <expression booléenne>

Permet de spécifier quelles sont les lignes à sélectionner dans une table ou dans le produit cartésien de plusieurs tables.

Elle est suivie d'une expression booléenne qui sera évaluée pour chaque ligne.

Les lignes pour lesquelles l'expression est vraie seront sélectionnées.

Cette clause s'utilise aussi dans les commandes UPDATE et DELETE.

WHERE exp1 = exp2 WHERE exp1 != exp2 WHERE exp1 < exp2

WHERE exp1 > exp2 WHERE exp1 <= exp2 WHERE exp1 >= exp2

Pour les types date, la relation d'ordre est l'ordre chronologique

Pour les types caractères, la relation d'ordre est l'ordre lexicographique.

La clause : WHERE <expression booléenne>

WHERE exp1 BETWEEN exp2 AND exp3

WHERE exp1 LIKE exp2

WHERE exp1 NOT LIKE exp2

« _ » remplace un caractère exactement
« % » remplace une chaîne de caractères de longueur quelconque, y compris de longueur Nulle

Where nom LIKE '_AR%' → le nom doit commencer par un caractère quelconque puis « AR » et enfin une chaine éventuellement vide.

WHERE exp1 IN (exp2, exp3,...)

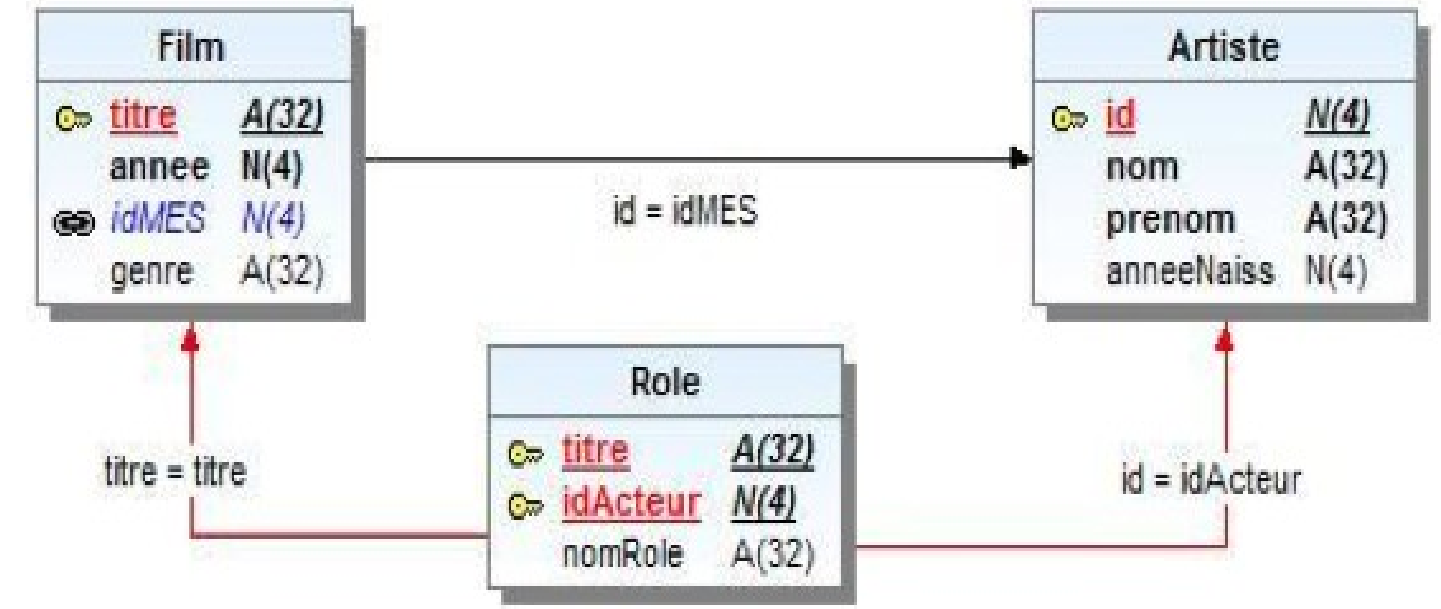
WHERE exp1 NOT IN (exp2, exp3,...)

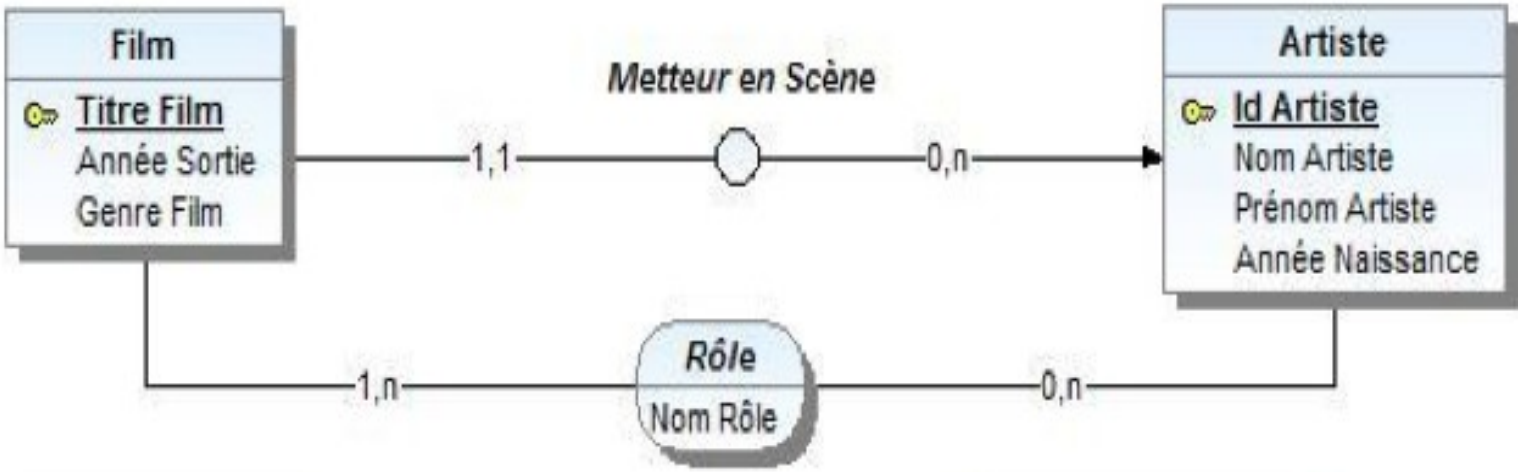
WHERE exp IS NULL

WHERE exp IS NOT NULL

Les opérateurs logiques AND, OR, NOT peuvent être utilisés pour combiner plusieurs expressions.

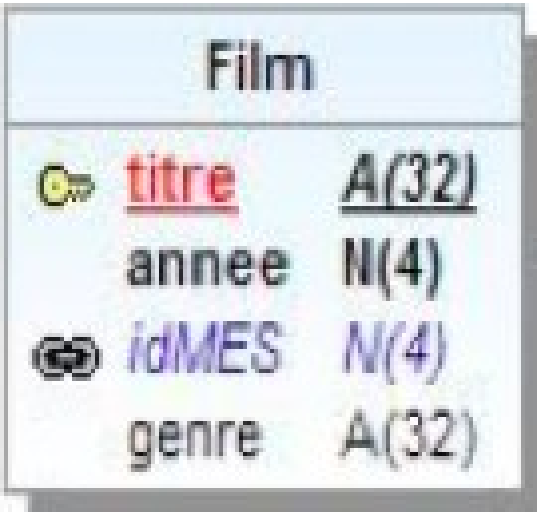
Fil rouge du cours : films, artiste, metteur en scène, rôle,...





La clause : WHERE <expression booléenne>

$\Pi_{titre}(\sigma_{genre= 'drame'}(Film))$



$\Pi_{titre,annee}(\sigma_{genre= 'policier' \wedge annee \geq 1990 \wedge annee \leq 2000}(Film))$

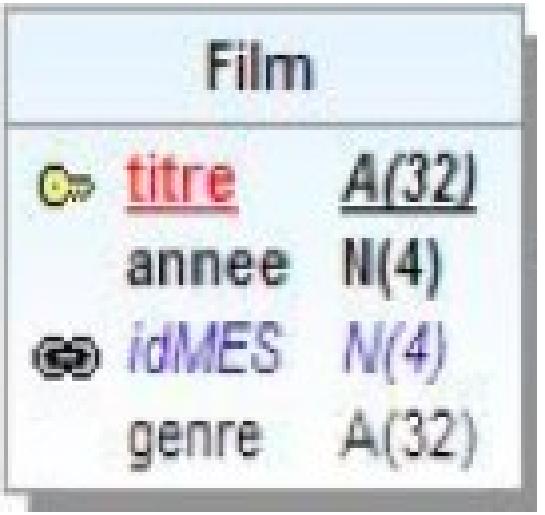
?

?

La clause : WHERE <expression booléenne>

$\Pi_{titre}(\sigma_{genre= 'drame'}(Film))$

SELECT titre FROM Film WHERE genre='drame'



$\Pi_{titre,annee}(\sigma_{genre= 'policier' \wedge annee \geq 1990 \wedge annee \leq 2000}(Film))$

```
SELECT titre,annee  FROM Film
      WHERE genre='policier' AND annee>=1990 AND annee <=2000
```

```
SELECT titre,annee  FROM Film
      WHERE genre='policier' AND annee BETWEEN 1990 AND 2000
```

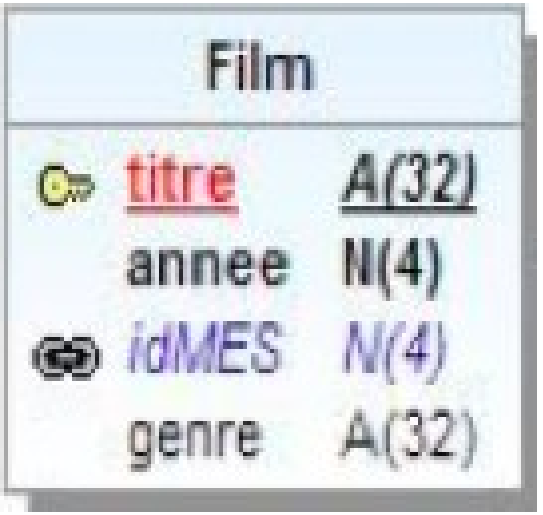
La clause : WHERE <expression booléenne>

$\Pi_{\text{titre}} (\sigma_{\text{genre} \neq \text{'drame'} \wedge \text{genre} \neq \text{'policier'}} (\text{Film}))$

?

?

?



Les noms d'artistes qui ont un « a » en deuxième position

?

La clause : WHERE <expression booléenne>

$\Pi_{\text{titre}} (\sigma_{\text{genre} \neq \text{'drame'} \wedge \text{genre} \neq \text{'policier'}} (\text{Film}))$

SELECT titre FROM Film WHERE genre<>'drame' AND genre!='policier'

SELECT titre FROM Film WHERE NOT (genre='drame' OR genre='policier')

SELECT titre FROM Film WHERE genre NOT IN ('drame','policier')

SELECT * FROM Artiste WHERE nom LIKE '_a%'

Les noms d'artistes qui ont un « a » en deuxième position

La clause : WHERE <expression booléenne>

La valeur NULL signifie « absence de valeur » donc on ne peut lui appliquer aucune opération ou comparaison usuelle :

- Toute opération ou fonction appliquée sur NULL donne comme résultat NULL

- Toute comparaison avec NULL donne un résultat ni vrai ni faux : « UNKNOWN »

Pour tester la nullité on utilise IS NULL et IS NOT NULL

Exemple : SELECT * FROM Artiste WHERE anneeNaiss IS NULL

Liste des acteurs dont on ne connait pas la date de naissance.

La clause : WHERE <expression booléenne>

Remarque : la fonction NVL(expression,message) remplace par « message » l'expression si celle-ci est NULL. Une expression est de type chaine de caractères ou nombre.

SELECT NVL(TO_CHAR(anneeNaiss),'année non renseigné') FROM Artiste

Affiche les dates de naissance des artistes et pour ceux dont on ne connait pas la date on affiche le message « année non renseignée ».

Notons la conversion via TO_CHAR de la date vers une chaine de caractère pour qu'elle soit conforme à la syntaxe de NVL.

La clause WHERE peut aussi être utilisée pour faire des jointures et des sous-interrogations : une des valeurs utilisées dans un WHERE provient d'une requête SELECT (emboîtée).

NATURAL JOIN

La jointure naturelle en ORACLE SQL

La jointure naturelle Table1 ⋈ Table 2 s'exprime en SQL dans la clause FROM :

FROM Table1 NATURAL JOIN TABLE2

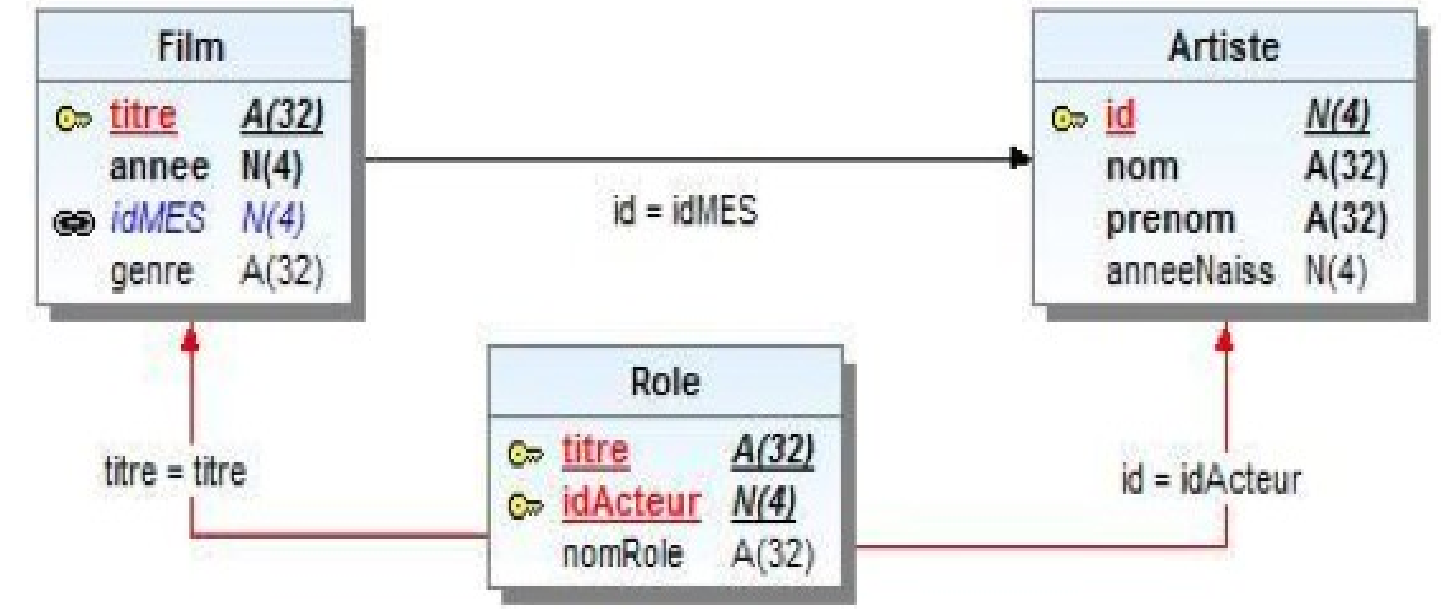
Pour obtenir une jointure naturelle sur une partie seulement des colonnes qui ont le même nom, il faut utiliser la clause :

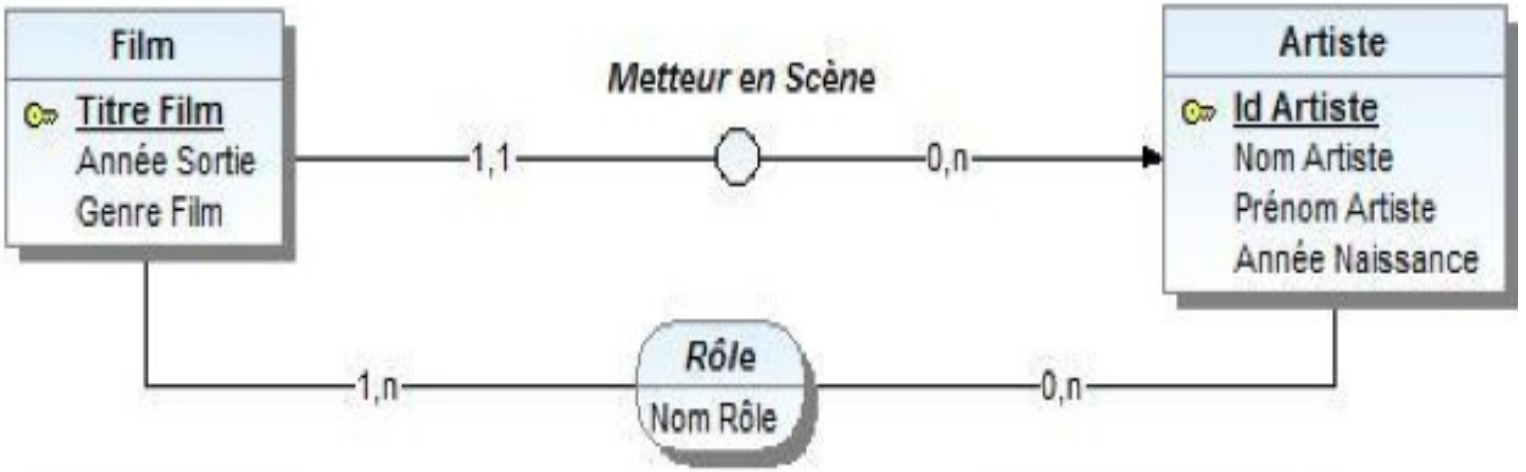
Table 1 JOIN Table 2 USING (Col1,...,Coln)

{Col1,...,Coln} est un sous ensemble d'attributs communs des tables : Table1 et Table2.

Si cet ensemble représente tous les attributs communs alors cette syntaxe est équivalente à celle vue précédemment via Table1 NATURAL JOIN Table2.

Fil rouge du cours : films, artiste, metteur en scène, rôle,...





La jointure naturelle en ORACLE SQL

Attention : dans une jointure naturelle via « NATURAL JOIN » il est interdit de préfixer une colonne commune (utilisée pour la jointure) par un nom de table. La requête suivante génère une erreur :

```
SELECT F.titre, nomRole FROM Film F NATURAL JOIN Role
```

La colonne « titre » est une colonne commune aux deux tables Film et Rôle, elle sera donc utilisée dans le NATURAL JOIN et ne doit donc pas être préfixée sous ORACLE.

JOIN

La théta-jointure en ORACLE SQL

La théta-jointure Table1 $\bowtie_{condition}$ Table 2 s'exprime en SQL dans la clause FROM :

FROM Table1 JOIN TABLE2 ON Condition

Exemple : titre de chaque film avec le nom du metteur en scène.

On note que les colonnes idMES de la table film et id de la table d'artiste ont des noms différents. On ne peut donc pas faire de jointure naturelle. On passe par une théta-jointure

$$\Pi_{titre,\ nom} (Film \bowtie_{id=idMes} Artiste)$$

?

La théta-jointure en ORACLE SQL

La théta-jointure Table1 $\bowtie_{condition}$ Table 2 s'exprime en SQL dans la clause FROM :

FROM Table1 JOIN TABLE2 ON Condition

Exemple : titre de chaque film avec le nom du metteur en scène.

On note que les colonnes idMES de la table film et id de la table d'artiste ont des noms différents. On ne peut donc pas faire de jointure naturelle. On passe par une théta-jointure

$$\Pi_{titre,\ nom} (Film \bowtie_{id=idMes} Artiste)$$

SELECT titre, nom FROM Film JOIN Artiste ON id=idMes

La théta-jointure en ORACLE SQL

Reprenons l'exemple : SELECT titre, nom FROM Film JOIN Artiste ON id=idMes

Tout comme le NATURAL JOIN, l'instruction JOIN n'est disponible qu'à partir de la version 9.

Pour les versions antérieures on passe par la clause WHERE en spécifiant l'égalité des attributs de jointure.

SELECT titre, nom FROM Film, Artiste WHERE id=idMES

La théta-jointure en ORACLE SQL

Jointure d'une table sur elle même :

On a des fois besoins de joindre une table sur elle même, dans ce cas un renommage est obligatoire pour pouvoir préfixer sans ambiguité chaque attribut. Dans l'exemple ci-dessous on joint la table « Tab » sur elle même :

ORACLE >8 : SELECT Table1.Attribut1 , Table2.Attribut2,...

FROM Tab Table1 JOIN Tab Table2 ON …

ORACLE <9 : SELECT Table1.Attribut , Table2.Attribut,...

FROM Tab Table1, Tab Table2 WHERE ...

NATURAL OUTER JOIN

La jointure naturelle externe Table1 ⟗ Table2

Parfois, perdre les tuples défaillants de la jointure n'est pas convenable...

La jointure externe est une opération de jointure qui permet de retrouver aussi les tuple défaillants, en mettant des valeurs NULL (ou ⊥) pour les attributs qui ne sont pas définis dans les résultats.

(1) Jointure naturelle externe Table1 avec Table2

R_1 =

A	B	C
1	2	5
3	4	6
2	5	12
6	7	8

R_2 =

B	C	D	E
2	5	6	3
4	6	8	10
5	7	10	3
4	6	7	8
3	4	6	8

R1⋈R2 =

A	B	C	D	E
1	2	5	6	3
3	4	6	8	10

$R_1 ⟗ R_2$ =

A	B	C	D	E
1	2	5	6	3
3	4	6	8	10
3	4	6	7	8
2	5	12	⊥	⊥
6	7	8	⊥	⊥
⊥	5	7	10	3
⊥	3	4	6	8

FROM Table1 NATURAL FULL OUTER JOIN Table2

FROM R1 NATURAL FULL OUTER JOIN R2

La jointure naturelle externe Table1 ⟕ Table2

(2) Jointure naturelle externe gauche Table1 avec Table2

R_1 =

A	B	C
1	2	5
3	4	6
2	5	12
6	7	8

R_2 =

B	C	D	E
2	5	6	3
4	6	8	10
5	7	10	3
4	6	7	8
3	4	6	8

R1⋈R2 =

A	B	C	D	E
1	2	5	6	3
3	4	6	8	10
3	4	6	7	8
2	5	12	⊥	⊥
6	7	8	⊥	⊥

$R_1 \overset{\circ}{\bowtie}_L R_2$ =

A	B	C	D	E
1	2	5	6	3
3	4	6	8	10
3	4	6	7	8
2	5	12	⊥	⊥
6	7	8	⊥	⊥

FROM Table1 NATURAL LEFT OUTER JOIN Table2

FROM R1 NATURAL LEFT OUTER JOIN R2

Pour une jointure à droite on remplace LEFT par RIGHT.

Pour la théta-jointure externe on ajoute « ON » suivi des conditions de jointure.

La jointure naturelle externe Table ⟗ Table2

Exprimer le titre et l'année de tous les films avec les rôles. Si pour un film aucun rôle n'existe encore dans la table on affiche uniquement le titre et l'année de ce film.

$$\Pi_{\text{titre, année, nomRole}} ((\text{Film} \overset{\circ}{\bowtie}_{L} \text{Role})$$

La jointure est externe gauche car on souhaite garder tous les films et pour ceux dont on ne connait aucun rôle on complète la ligne par NULL.

C'est donc la table Film qui est affichée entièrement d'où la jointure GAUCHE !

ORACLE >8 : SELECT titre, année, nomRole

FROM (Film NATURAL LEFT OUTER JOIN Role)

ORACLE <9 ?

La jointure naturelle externe Table ⟕ Table2

Exprimer le titre et l'année de tous les films avec les rôles. Si pour un film aucun rôle n'existe encore dans la table on affiche uniquement le titre et l'année de ce film.

$$\Pi_{\text{titre, année, nomRole}} ((\text{Film} \overset{\diamond}{\bowtie}_{L} \text{Role})$$

La jointure est externe gauche car on souhaite garder tous les films et pour ceux dont on ne connait aucun rôle on complète la ligne par NULL.

C'est donc la table Film qui est affichée entièrement d'où la jointure GAUCHE !

ORACLE >8 : ?

ORACLE <9 ?

La jointure naturelle externe Table ⟕ Table2

Exprimer le titre et l'année de tous les films avec les rôles. Si pour un film aucun rôle n'existe encore dans la table on affiche uniquement le titre et l'année de ce film.

$$\Pi_{\text{titre, année, nomRole}} ((\text{Film} \overset{\circ}{\bowtie}_{L} \text{Role})$$

La jointure est externe gauche car on souhaite garder tous les films et pour ceux dont on ne connait aucun rôle on complète la ligne par NULL.

C'est donc la table Film qui est affichée entièrement d'où la jointure GAUCHE !

ORACLE <9 : SELECT titre, année, nomRole FROM Film, Role
WHERE Film.titre =Role.titre (+)

Notons le (+) à droite de l'équation pour spécifier que c'est la table « Role » qui sera complétée par des NULL → jointure externe GAUCHE.

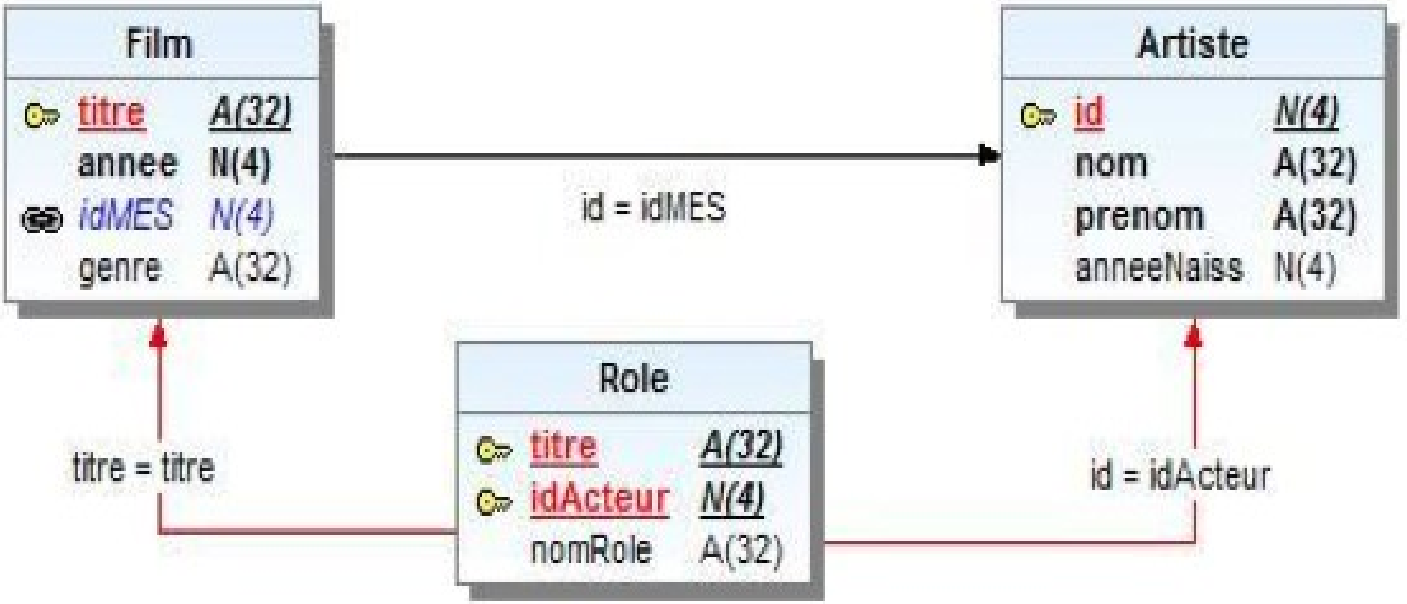
Combinaison de jointures

Exprimer pour chaque film : le titre, l'année, les rôles et les noms des acteurs.

D’abord en AR puis en Oracle >8 puis Oracle < 9

L'information est distribuée sur 3 tables : Film, Artiste et Rôle.

?



Combinaison de jointures

Exprimer pour chaque film : le titre, l'année, les rôles et les noms des acteurs.

L'information est distribuée sur 3 tables : Film, Artiste et Rôle.

Il faut donc deux jointures : une naturelle entre Film et Rôle et une autre en théta entre le résultat obtenu de la jointure naturelle et la table Artiste.

$$\Pi_{\text{titre, année, nom, nomRole}} ((\text{Film} \bowtie \text{Role}) \bowtie_{\text{id=idActeur}} \text{Artiste})$$

ORACLE >8 : ?

Combinaison de jointures

Exprimer pour chaque film : le titre, l'année, les rôles et les noms des acteurs.

L'information est distribuée sur 3 tables : Film, Artiste et Rôle.

Il faut donc deux jointures : une naturelle entre Film et Rôle et une autre en théta entre le résultat obtenu de la jointure naturelle et la table Artiste.

$$\Pi_{\text{titre, année, nom, nomRole}} ((\text{Film} \bowtie \text{Role}) \bowtie_{\text{id=idActeur}} \text{Artiste})$$

ORACLE >8 : SELECT titre, année, nom, nomRole

FROM (Film NATURAL JOIN Role) JOIN Artiste ON id=idActeur

Combinaison de jointures

Exprimer pour chaque film : le titre, l'année, les rôles et les noms des acteurs.

L'information est distribuée sur 3 tables : Film, Artiste et Rôle.

Il faut donc deux jointures : une naturelle entre Film et Rôle et une autre en théta entre le résultat obtenu de la jointure naturelle et la table Artiste.

$$\Pi_{\text{titre, année, nom, nomRole}} ((\text{Film} \bowtie \text{Role}) \bowtie_{\text{id=idActeur}} \text{Artiste})$$

ORACLE <9 : ?

Combinaison de jointures

Exprimer pour chaque film : le titre, l'année, les rôles et les noms des acteurs.

L'information est distribuée sur 3 tables : Film, Artiste et Rôle.

Il faut donc deux jointures : une naturelle entre Film et Rôle et une autre en théta entre le résultat obtenu de la jointure naturelle et la table Artiste.

$$\Pi_{\text{titre, année, nom, nomRole}} ((\text{Film} \bowtie \text{Role}) \bowtie_{\text{id=idActeur}} \text{Artiste})$$

ORACLE <9 : SELECT titre, année, nom, nomRole

FROM Film, Role, Artiste

WHERE Film.titre=Role.titre AND id=idActeur.

UNION

INTERSECT

MINUS

L'union Table1 ∪ Table2

L'opérateur UNION permet de fusionner deux sélections de tables pour obtenir un ensemble de lignes égal à la réunion des lignes des deux sélections. Les lignes communes n'apparaîtront qu'une fois.

Si on veut conserver les doublons, on peut utiliser la variante UNION ALL.

Exemple : afficher toutes les dates présentes dans la base de données.

Π_{annee} (Film) ∪ $\Pi_{\text{anneeNaiss}}$ (Artiste)

?

UNION

?

L'union Table1 ∪ Table2

L'opérateur UNION permet de fusionner deux sélections de tables pour obtenir un ensemble de lignes égal à la réunion des lignes des deux sélections. Les lignes communes n'apparaîtront qu'une fois.

Si on veut conserver les doublons, on peut utiliser la variante UNION ALL.

Exemple : afficher toutes les dates présentes dans la base de données.

Π_{annee} (Film) ∪ $\Pi_{\text{anneeNaiss}}$ (Artiste)

SELECT annee FROM Film

UNION

SELECT anneeNaiss FROM Artiste

L'intersection Table1 ∩ Table2

L'opérateur INTERSECT permet d'obtenir l'ensemble des lignes communes à deux interrogations.

Exemple : afficher les noms de rôles qui sont également des titres de films

$$\Pi_{\text{titre}}(\text{Film}) \cap \Pi_{\text{nomRole}}(\text{Role})$$

?

INTERSECT

?

L'intersection Table1 ∩ Table2

L'opérateur INTERSECT permet d'obtenir l'ensemble des lignes communes à deux interrogations.

Exemple : afficher les noms de rôles qui sont également des titres de films

$$\Pi_{\text{titre}}(\text{Film}) \cap \Pi_{\text{nomRole}}(\text{Role})$$

SELECT titre FROM Film

INTERSECT

SELECT nomRole FROM Role

La soustraction Table1 \ Table2

L'opérateur MINUS permet d'ôter d'une sélection les lignes obtenues dans une deuxième sélection.

Exemple : donnez le nom des rôles qui ne sont pas des titres de films.

Π_{nomRole} (Role) \ Π_{titre} (Film)

?

MINUS

?

Important : pour « union », « union all », « intersect » et « minus » il faut qu'il y est cohérence entre le nombre d'attributs dans chaque select ainsi qu'une cohérence entre leur groupe type (data type group) respectif, par exemple numérique ou caractère.

La soustraction Table1 \ Table2

L'opérateur MINUS permet d'ôter d'une sélection les lignes obtenues dans une deuxième sélection.

Exemple : donnez le nom des rôles qui ne sont pas des titres de films.

Π_{nomRole} (Role) \ Π_{titre} (Film)

SELECT nomRole FROM Role

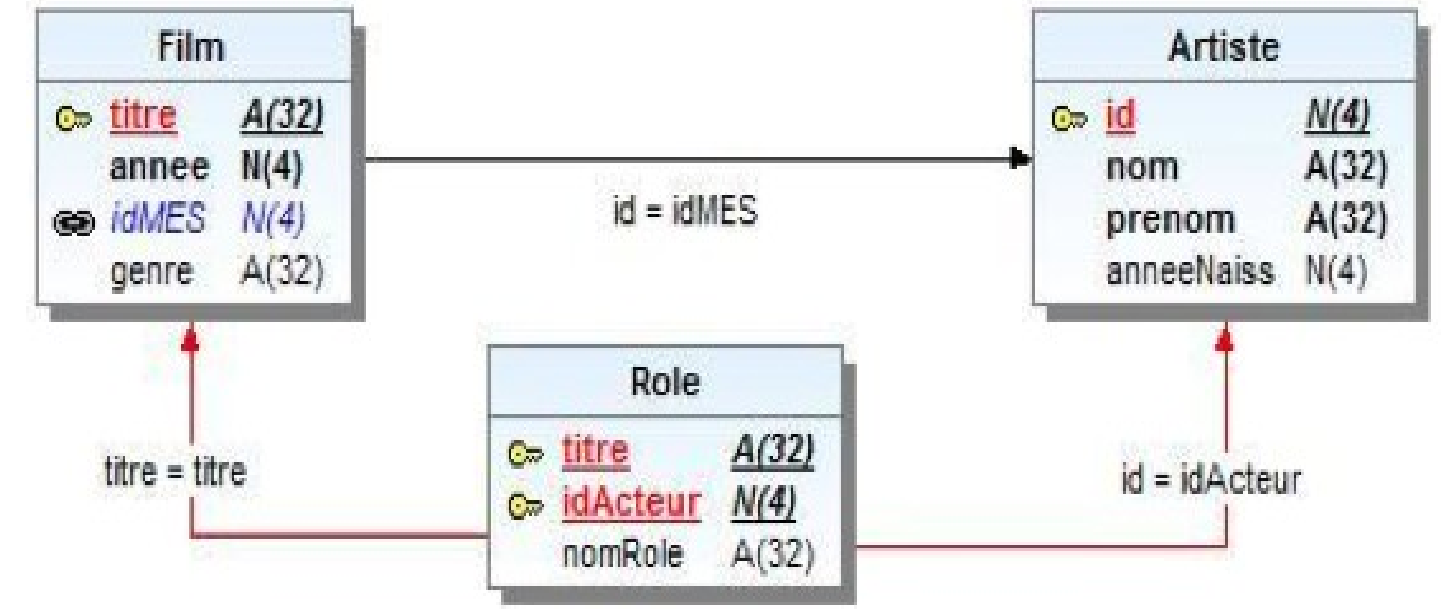
MINUS

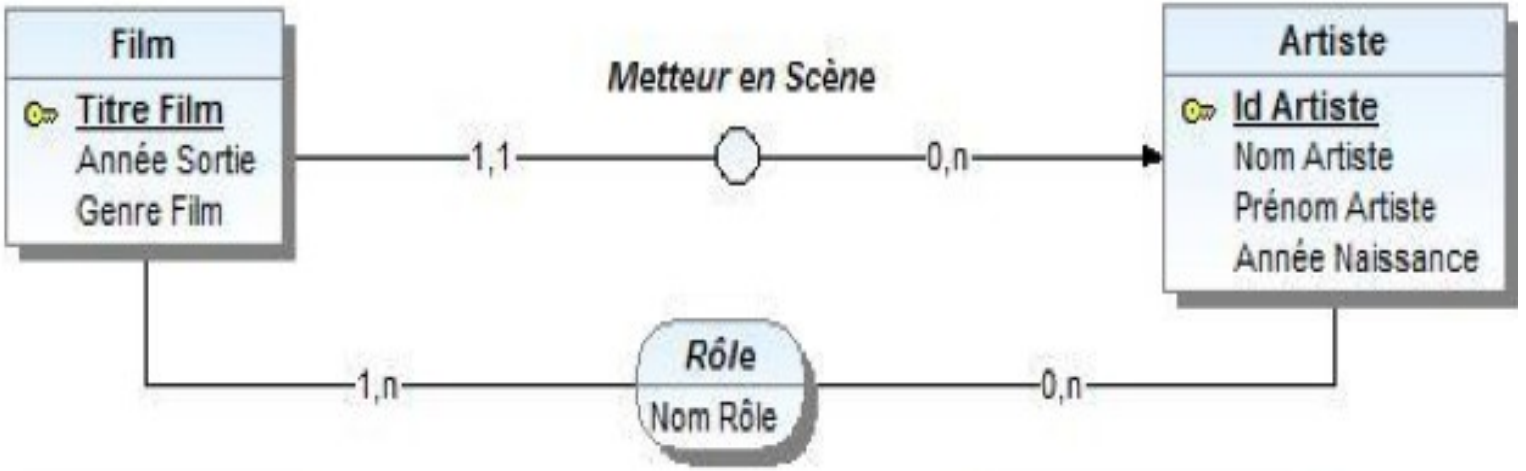
SELECT titre FROM Film

Important : pour « union », « union all », « intersect » et « minus » il faut qu'il y est cohérence entre le nombre d'attributs dans chaque select ainsi qu'une cohérence entre leur groupe type (data type group) respectif, par exemple numérique ou caractère.

REQUETES IMBRIQUEES

Fil rouge du cours : films, artiste, metteur en scène, rôle,...





Les requêtes imbriquées : sous-interrogations

Une caractéristique puissante de SQL est la possibilité qu'une expression à droite d'un opérateur de comparaison employé dans une clause WHERE comporte un SELECT emboîté.

Exemple : la requête précédente (donnez le nom des rôles qui ne sont pas des titres de films) peut s'exprimer de deux manières

(1) Codage à plat :

```
SELECT nomRole FROM Role MINUS SELECT titre FROM Film
```

(2) Codage par imbrication :

```
SELECT nomRole FROM Role
WHERE  nomRole NOT IN
            ( SELECT  titre
              FROM Film
             )
```

Sous-interrogations à une ligne et une colonne : WHERE exp op (SELECT ...)

le SELECT imbriqué équivaut à une valeur : op est un des opérateurs =, !=, < >, <=, >=

exp est toute expression légale.

Exemple : liste des artiste ayant la même année de naissance que l'acteur dont l'identifiant est égal à 100.

```
SELECT nom FROM Artiste WHERE annee =
        ( SELECT annee FROM Artiste WHERE id=100
         )
```

Une sous-interrogation à une seule ligne doit ramener une seule ligne ; dans le cas où plusieurs lignes, ou pas de ligne du tout seraient ramenées, un message d'erreur sera affiché et l'interrogation sera abandonnée.

Un SELECT peut comporter plusieurs sous-interrogations, soit imbriquées, soit au même niveau dans différents prédicats combinés par des AND ou des OR.

Sous-interrogations ramenant plusieurs lignes

WHERE exp op ANY (SELECT ...)

WHERE exp op ALL (SELECT ...)

WHERE exp IN (SELECT ...)

WHERE exp NOT IN (SELECT ...)

Une sous-interrogation peut ramener plusieurs lignes à condition que l'opérateur de comparaison admette à sa droite un ensemble de valeurs.

Les opérateurs autorisés sont :

(1) L'opérateur IN

(2) Les opérateurs obtenus en ajoutant ANY ou ALL à la suite des opérateurs de comparaison classique =, !=, <, >, <=, >=.

- ANY : la comparaison sera vraie si elle est vraie pour au moins un élément de l'ensemble (elle est donc fausse si l'ensemble est vide).

- ALL : la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble (elle est vraie si l'ensemble est vide).

L'opérateur IN est équivalent à = ANY, et l'opérateur NOT IN est équivalent à != ALL.

Sous-interrogations ramenant plusieurs lignes

Exemple 1 : donnez les années de sorties des films tournés par l'acteur dont l'identifiant est égal à 100.

```
SELECT annee FROM Film
    WHERE titre IN ( SELECT titre FROM Role
                     WHERE idacteur=100)
```

Exemple 2 : quels sont les films qui sont sortis après tous les films de drame qui figurent dans la base.

```
SELECT titre FROM Film
 WHERE annee  > ALL  ( SELECT annee FROM Film
                       WHERE genre='drame')
```

Sous-interrogations ramenant plusieurs lignes

Il est possible de synchroniser une sous-interrogation avec l'interrogation principale : SQL sait traiter une sous-interrogation faisant référence à une colonne de la table de l'interrogation principale.

Le traitement dans ce cas est plus complexe car il faut évaluer la sous-interrogation pour chaque ligne de l'interrogation principale alors que dans la requête précédente on avait juste à évaluer la sous-requête et finaliser par l'évaluation de la requête principale.

Exemple : liste des film dont le metteur en scène n'est pas un acteur du film.

```
SELECT titre FROM Film
    WHERE idMES NOT IN  (SELECT idArtiste FROM Role
                                WHERE  Role.titre = Film.titre )
```

Pour chaque film on évalue d'abord la sous-requête qui nous renvoie la liste des acteurs de ce film puis on exige dans la requête principale que le metteur en scène ne soit pas un élément de cette liste.

Sous-interrogations ramenant plusieurs colonnes

Il est possible de comparer le résultat d'un SELECT ramenant plusieurs colonnes à une liste de colonnes.

La liste de colonnes figurera entre parenthèses à gauche de l'opérateur de comparaison.

Soit op l'un des opérateurs « = » ou « != » :

(A) Avec une seule ligne sélectionnée

WHERE (exp, exp,...) op (SELECT ...)

(B) Avec plusieurs lignes sélectionnées

WHERE (exp, exp,...) op ANY (SELECT ...)
WHERE (exp, exp,...) op ALL (SELECT ...)
WHERE (exp, exp,...) IN (SELECT ...)
WHERE (exp, exp,...) NOT IN (SELECT ...)

Les expressions figurant dans la liste entre parenthèses seront comparées à celles qui sont ramenées par le SELECT.

Sous-interrogations ramenant plusieurs colonnes

Exemple : metteurs en scène qui ne jouent pas dans au moins de leur propre film

```
SELECT DISCTINCT F1.idMES FROM Film F1

 WHERE (F1.titre,F1.idMES) IN

     (SELECT F2.titre, F2.idMES FROM Film F2)

       MINUS

     (SELECT Role.titre, Role.idActeur FROM Role)
```

On sélectionne le couple (titre,idMES) à condition que celui-ci soit un élément de l'ensemble des couple obtenu par le MINUS.

La soustraction garde un couple (tite,idMES) si le metteur en scène de ce film n'est pas un acteur de ce film.

On utilise DISTINCT car un metteur en scène peut répondre positivement à la requête pour plusieurs Film et sera donc affiché plusieurs fois.

La clause : EXISTS / NOT EXISTS

La clause EXISTS est suivie d'une sous-interrogation entre parenthèses, et prend la valeur vrai s'il existe au moins une ligne satisfaisant les conditions de la sous-interrogation.

Souvent on peut utiliser IN à la place de la clause EXISTS.

Exemple 1 : donnez les années de sorties des films tournés par l'acteur dont l'identifiant est égal à 100.

On avait déjà répondu à cette requête via « IN » :

```
SELECT annee FROM Film
 WHERE titre IN ( SELECT titre FROM Role
                  WHERE idActeur=100)
```

On peut utiliser EXISTS :

```
SELECT annee FROM Film
WHERE EXISTS
                  ( SELECT * FROM Role
                    WHERE Film.titre=Role.titre AND idActeur=100)
```

Des sous requêtes dans FROM et SELECT

Il est possible d'insérer des sous requêtes dans les expressions qui suivent le SELECT ou le FROM.

SELECT (REQ1), (REQ2),... FROM (REQ3) Synonyme3, (REQ4) Synonyme4, …

Synonyme3 et Synonyme4 sont les synonymes des tables renvoyées par REQ3 et REQ4.

Exemple : Trouver les acteurs qui ont été partenaires de l'acteur dont l'identifiant est égale à 100.

ORACLE <9

```
SELECT A1.idActeur From ( SELECT titre, idActeur FROM Role
                          WHERE idActeur !=100) A1,

                        ( SELECT titre  FROM Role
                          WHERE idActeur=100) A2

                  WHERE A1.titre=A2.titre
```

Des sous requêtes dans FROM et SELECT

Il est possible d'insérer des sous requêtes dans les expressions qui suivent le SELECT ou le FROM.

SELECT (REQ1), (REQ2),... FROM (REQ3) Synonyme3, (REQ4) Synonyme4, …

Synonyme3 et Synonyme4 sont les synonymes des tables renvoyées par REQ3 et REQ4.

Exemple : Trouver les acteurs qui ont été partenaires de l'acteur dont l'identifiant est égale à 100.

ORACLE >=9

```
SELECT A1.idActeur From ( SELECT titre, idActeur FROM Role
                            WHERE idActeur !=100) A1
                          JOIN
                          ( SELECT titre  FROM Role
                            WHERE idActeur=100) A2

                          ON A1.titre=A2.titre
```

AGREGATIONS ET GROUPES

GROUP BY … HAVING

Agrégation

Des fonctions d'agrégation très utilisées notamment dans le domaine des statistiques

COUNT(*) : compte le nombre de ligne même s'il y a des valeurs NULL
COUNT(Col1) : compte le nombre de ligne pour lesquelles Col1 n'a pas la valeur NULL
COUNT(DISTINCT Col1) : similaire à COUNT(Col1) mais éjecte les doublons.

SUM, MAX, MIN, AVG, STD (écart type).

Le moyen de partitionner une table en groupes selon certains critères (GROUP BY)

Le moyen d'exprimer des conditions sur ces groupes (HAVING)

Il existe un groupe par défaut : la table toute entière ! ce qui permet d'appliquer les fonctions d'agrégation directement sur la table.

Syntaxe simplifiée d'un SELECT agrégé

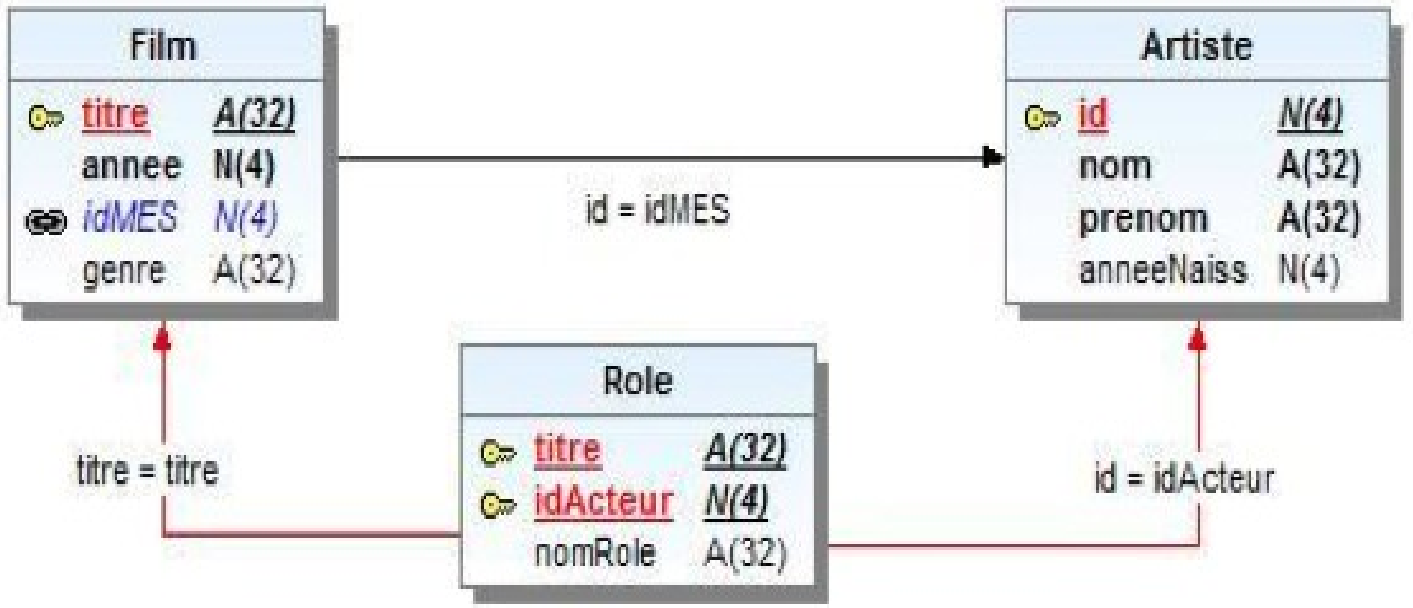
```
SELECT < expr > [,< expr > ]...
FROM nomTable[ nomAlias ][,nomTable[ nomAlias ]]...
[WHERE < conditionSélectionLigne >]
[GROUP BY < expr >[,< expr >] ...
    [HAVING < conditionSélectionGroupe >]] ;
```

Agrégation

?
Calcule le nombre de ligne de la table Artiste.

?
Calcul le nombre d'année de naissance distinctes de la table Artiste.

?
Calcul le nombre de films réalisée par Client Eastwood.

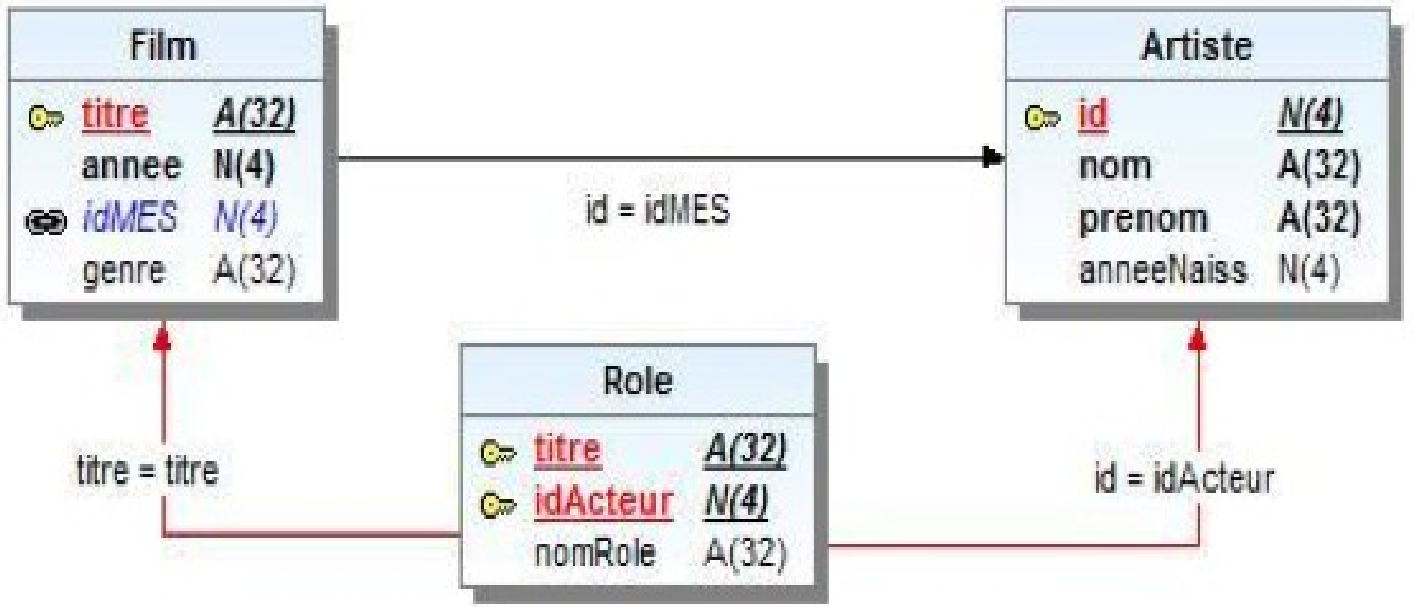


Agrégation

SELECT COUNT(*) FROM Artiste
Calcule le nombre de ligne de la table Artiste.

SELECT COUNT(DISTINCT anneeNaiss) FROM Artiste.
Calcul le nombre d'année de naissance distinctes de la table Artiste.

SELECT COUNT(*)
FROM Film JOIN Artiste
ON (idMES=id AND prenom='Clint' AND nom='Eastwood')
Calcul le nombre de films réalisée par Client Eastwood.



Agrégation

Attention : à un niveau de profondeur (relativement aux sous- interrogations), d'un SELECT, les fonctions de groupe et les colonnes doivent être toutes du même niveau de regroupement.

Exemple : si on veut le nom de l'artiste le plus jeune (par rapport à l'année de naissance) la requête suivante provoquera une erreur :

```
SELECT nom, anneeNaiss FROM Artiste
WHERE anneeNaiss = MAX(anneeNaiss)
```

Il faut une sous-interrogation car MAX(anneeNaiss) n'est pas au même niveau de regroupement que anneeNais :

```
SELECT nom, anneeNaiss FROM Artiste
WHERE anneeNaiss= (SELECT MAX(anneeNaiss) FROM Artiste)
```

Agrégation

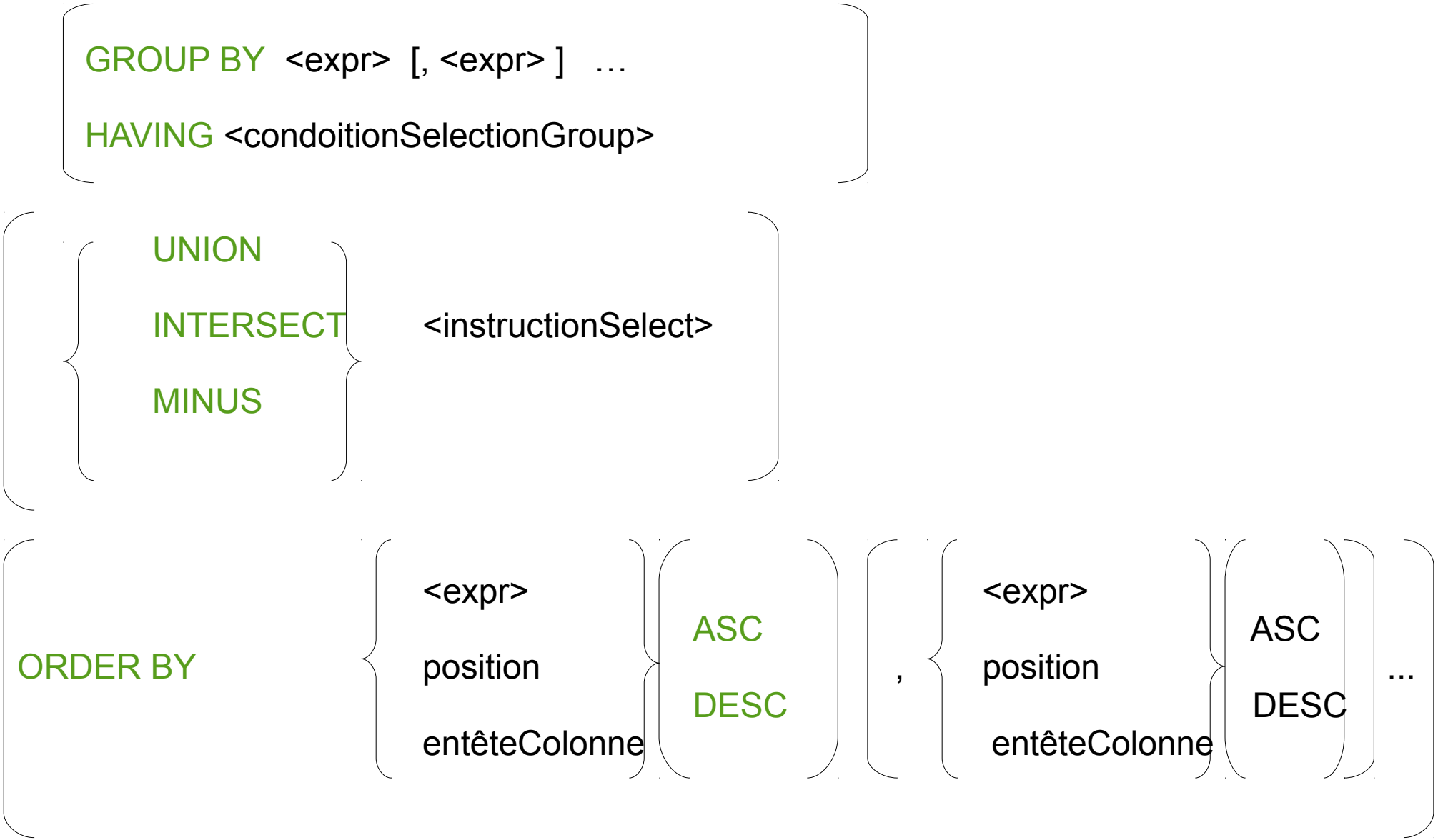
De même cette version n'est pas bonne ! On aura le bon max mais pas le bon nom !

SELECT nom, MAX(anneeNaiss) FROM Artiste

Il faut une sous-interrogation. La requête ci-dessous est correcte :

SELECT nom, anneeNaiss FROM Artiste
WHERE anneeNaiss= (SELECT MAX(anneeNaiss) FROM Artiste)

GROUP BY … ORDER BY ...



GROUP BY

Il est possible de subdiviser la table en groupes, chaque groupe étant l'ensemble des lignes ayant une valeur commune.

GROUP BY exp1, exp2,... → groupe en une seule ligne toutes les lignes pour lesquelles exp1, exp2,... ont la même valeur.

Cette clause se place juste après la clause WHERE, ou après la clause FROM si la clause WHERE n'existe pas.

Des lignes peuvent être éliminées avant que le groupe ne soit formé grâce à la clause WHERE.

L'exécution s'exécute en deux étape : création des groupes puis application de l'opération d'agrégation sur chaque groupe → une ligne du résultat représente un groupe.

GROUP BY

Une expression d'un SELECT avec clause GROUP BY ne peut être que :

- soit une fonction de groupe,

- soit une expression figurant dans le GROUP BY.

```
SELECT genre, COUNT(titre)
FROM Film
GROUP BY idMES
```

Erreur : « genre » est une colonne de Film mais du fait du GROUP BY on a le droit qu'à des fonctions de groupes ou à l'attribut « idMES ».

Correction :

```
SELECT idMES,COUNT(titre)...
FROM Film
GROUP BY idMES
```

affiche pour chaque metteur en scène le nombre des films dont il est le metteur en scène.

HAVING

Permet d'exprimer des conditions de groupe : garder que les groupes satisfaisant une certaine condition, alors que la clause WHERE exprime des conditions de sélection de ligne avant la formation des groupes.

Exemple : genre de film pour lesquels ont connait au moins deux films. Pour chaque genre retenu, afficher également la plus récente des années de sortie.

```
SELECT genre, MAX(annee) "Le plus récent"

FROM Film

GROUP BY genre

HAVING COUNT(*)>=2
```

Expression de la division par groupage

Dans le cas où il est certain que la table dividende (celle qui est divisée) ne contient dans la colonne qui sert pour la division que des valeurs qui existent dans la table diviseur, on peut exprimer la division en utilisant les regroupements et en comptant les lignes regroupées.

Exemple : les acteurs qui jouent dans tous les films

$$\text{Role} / \Pi_{\text{titre}} (\text{Film})$$

```
SELECT idActeur FROM Rôle
GROUP BY idActeur
HAVING COUNT(*)=
                    (SELECT COUNT(*) FROM Film)
```

Attention au COUNT(*) qui ne prend pas en compte l'élimination des doublons. Dans ce cas passer par des DISTINCT en précisant la colonne que l'on compte.

Ici nous sommes sure qu'il n'y a pas de doublons dans aucun des deux COUNT, d'où l'utilisation de COUNT(*).

TRI : ORDER BY

Tri par ORDER BY

La clause ORDER BY précise l'ordre dans lequel la liste des lignes sélectionnées sera donnée.

ORDER BY exp1 [DESC], exp2 [DESC], …

L'option facultative DESC donne un tri par ordre décroissant. Par défaut, l'ordre est croissant (ASC).

Le tri se fait d'abord selon la première expression, puis les lignes ayant la même valeur pour la première expression sont triées selon la deuxième, etc.

Les valeurs NULL sont toujours en tête quel que soit l'ordre du tri.

Pour préciser lors d'un tri sur quelle expression va porter le tri, il est possible de donner le rang relatif (par rapport au SELECT) de la colonne plutôt que son nom.

SELECT titre,annee FROM Film ORDER BY annee DESC

↔

SELECT titre,annee FROM Film ORDER BY 2 DESC

Tri par ORDER BY

Possibilité de trier sur une fonction d'agrégation.

Exemple : ajoutons à une des requêtes déjà étudiée un tri sur des fonction d'agrégation.

Afficher les genres de film pour lesquels ont connait au moins deux films. Pour chaque genre retenu, afficher également la plus récente des années de sortie avec un tri décroissant.

```
SELECT genre, MAX(annee) "Le plus récent"
FROM Film
GROUP BY genre
HAVING COUNT(*)>=2
ORDER BY MAX(annee) DESC
```

Les vues

Les vues

Une vue est une vision partielle ou particulière des données d'une ou plusieurs tables de la base.

La définition d'une vue est donnée par un SELECT qui indique les données de la base qui seront vues.

Les utilisateurs pourront consulter la base, ou modifier la base (avec certaines restrictions) à travers la vue, c'est-à-dire manipuler les données renvoyées par la vue comme si c'était des données d'une table réelle.

Seule la définition de la vue est enregistrée dans la base, et pas les données de la vue. On peut parler de table virtuelle.

Les vues

La commande CREATE VIEW permet de créer une vue en spécifiant le SELECT constituant la définition de la vue :

CREATE VIEW nom_vue (col1, col2...) AS SELECT …

Exemple : on souhaite créer une vue contenant les jeune artiste nés après 1980 à partir de la table Artiste.

```
CREATE VIEW Artiste_jeune (id,nom,prenom,anneeNaiss)
 AS SELECT id, nom,prenom,anneeNaiss From Artiste WHERE anneeNaiss > 1980
```

La spécification des noms des colonnes de la vue est facultative : par défaut, les colonnes de la vue ont pour nom les noms des colonnes résultats du SELECT. Si certaines colonnes résultats du SELECT sont des expressions sans nom, il faut alors obligatoirement spécifier les noms de colonnes de la vue.

Le SELECT peut contenir toutes les clauses d'un SELECT, sauf la clause ORDER BY. Il est également possible de créer une vue de vue.

Les vues

Une vue est supprimée via DROP VUE nom_vue.

Une vue peut être référencée dans un SELECT de la même façon qu'une table.

Il est possible d'effectuer des DELETE, INSERT et des UPDATE à travers des vues :

Pour effectuer un DELETE, le select qui définit la vue ne doit pas comporter de GROUP BY, de DISTINCT, de fonction de groupe.

Pour un UPDATE, en plus des conditions précédentes, les colonnes modifiées doivent être des colonnes réelles de la table sous-jacente ;

Pour un INSERT, en plus des conditions précédentes, toute colonne NOT NULL de la table sous-jacente doit être présente dans la vue.

Utilité des vues

Dissocier la façon dont les utilisateurs voient les données, du découpage en tables :

On sépare l'aspect externe (ce que voit un utilisateur particulier de la base) de l'aspect conceptuel (comment a été conçu l'ensemble de la base).

Ceci favorise l'indépendance entre les programmes et les données.

Si la structure des données est modifiée, les programmes ne seront pas à modifier si l'on a pris la précaution d'utiliser des vues (ou si on peut se ramener à travailler sur des vues).

Par exemple, si une table est découpée en plusieurs tables après l'introduction de nouvelles données, on peut introduire une vue, jointure des nouvelles tables, et la nommer du nom de l'ancienne table pour éviter de réécrire les programmes qui utilisaient l'ancienne table.

Utilité des vues

Une vue peut aussi être utilisée pour restreindre les droits d'accès à certaines colonnes et à certaines lignes d'une table

Un utilisateur peut ne pas avoir accès à une table mais avoir les autorisations pour utiliser une vue qui ne contient que certaines colonnes de la table; on peut de plus ajouter des restrictions d'utilisation sur cette vue

Une vue peut également simplifier la consultation de la base en enregistrant des SELECT complexes.

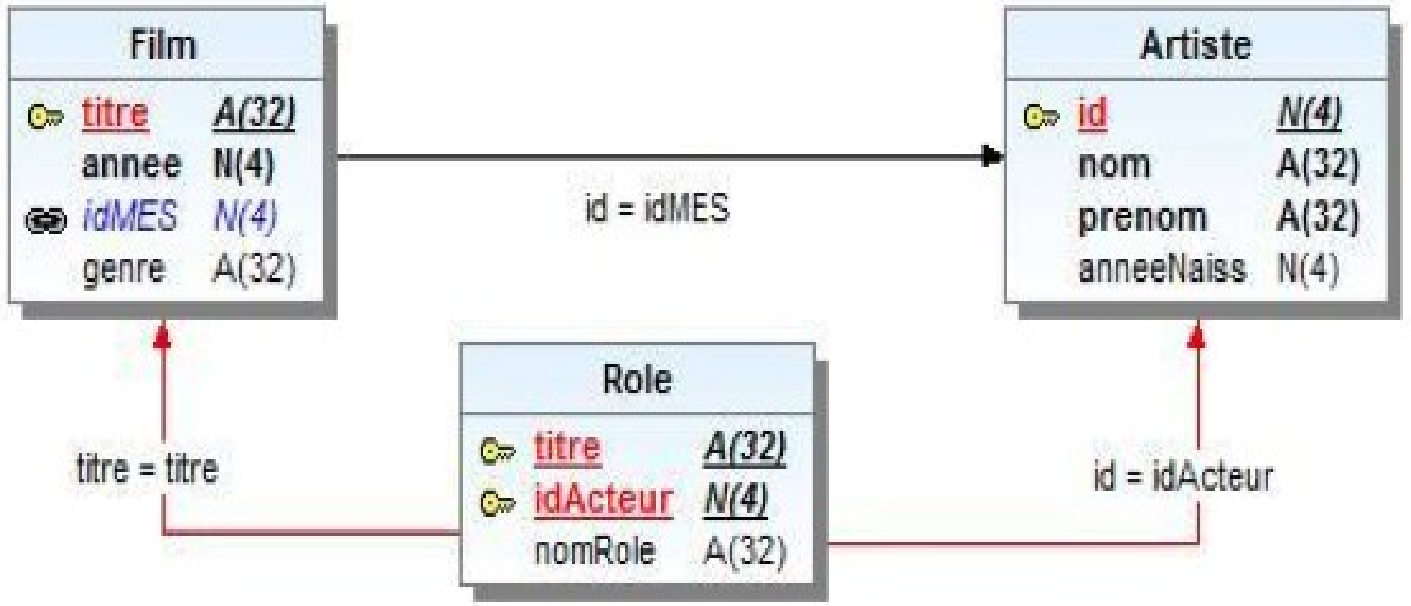
L'écriture des requêtes complexe est également simplifiée. Il suffit de créer plusieurs vues et de les utiliser ensuite pour définir nos requêtes complexes sous une forme lisible et simple.

Exemple de requêtes faisant appelle à une vue

Nom et prénom des metteur en scène pour lesquels on ne connait pas plus de deux films, avec le nombre de films, triés par ordre alphabétique des noms et prénoms des metteurs en scène.

```
DROP VIEW VnbFilms ;
CREATE VIEW VnbFilms (idMES,nbFilms) AS
        SELECT idMES, COUNT(Film.titre) FROM Film GROUP BY idMES
        HAVING COUNT(*)<=2 ;

SELECT nom AS "nom MES" , prenom AS "prénom MES", nbFilms
          FROM VnbFilms JOIN Artiste ON idMES=id
          ORDER BY nom,prenom
```



De l'algèbre relationnel vers ORACLE SQL

Algèbre	SQL
$\Pi_{A1,...,An}$ Table	SELECT A1, A2, ..., An FROM Table
δ(Table)	SELECT DISTINCT * FROM R
$\rho_{T(A1,...,An)}$ Table1	SELECT A1, ...,An FROM Table1 T
τ_{L} (Table)	SELECT * FROM Table ORDER BY L
$\sigma_{condition}$ (Table)	SELECT * FROM Table WHERE Condition
Table1 × Table2	SELECT * From Table 1 CROSS JOIN Table2
Table 1 ⋈ Table 2	SELECT * From Table 1 NATURAL JOIN Table2
Table 1 ⋈condition Table 2	SELECT * From Table 1 JOIN Table2 ON Condition
Table1 ⟗ Table 2	SELECT * FROM Table1 NATURAL FULL OUTER JOIN Table2
Table1 ∪ Table2	Table1 UNION Table2
Table1 ∩ Table2	Table1 INTERSECT Table2
Table1 \ Table2	Table1 MINUS Table2
Table1 / Table2	groupage+comptage

CASE WHEN (forme 1)

Depuis la version 9i. Elle correspond à la structure switch du langage C.

Elle remplace avantageusement la fonction decode d'Oracle (qui n'est pas dans la norme SQL2).

Il existe deux syntaxes pour CASE : une qui donne une valeur suivant des conditions quelconques et une qui donne une valeur suivant la valeur d'une expression.

CASE

WHEN condition1 THEN expression1

[WHEN condition2 THEN expression2] ...

[ELSE expression_défaut]

END

La condition qui suit un WHEN peut être n'importe quelle expression booléenne.

Si aucune condition n'est remplie et s'il n'y a pas de ELSE, NULL est retourné.

CASE WHEN (forme 2)

```
CASE
WHEN valeur1 THEN expression1
[WHEN valeur2 THEN expression2] ...
[ELSE expression_défaut]
END
```

Attention, ne fonctionne pas pour le cas où une des valeurs n'est pas renseignée (when null then). Pour ce cas, il faut utiliser l'autre syntaxe de case : WHEN colonne IS NULL THEN.

Si l'expression n'est égale à aucune valeur et s'il n'y a pas de ELSE, NULL est retourné.

Les différentes expressions renvoyées doivent être de même type.

```
SELECT titre,
   CASE genre
   WHEN 'dessin animé' THEN 'Enfant'
   WHEN 'animation'       THEN 'Familiale'
   ELSE   'Adulte '
   END
FROM film
```

Renvoie le titre et un libellé classant le public selon le genre du film

Forme 2

```
SELECT AVG
(CASE
   WHEN salaire > 2000 THEN salaire
   ELSE 1100
   END
 )   "Salaire moyen" FROM Employé
```

Renvoie la moyenne des salaires en prenant tous les salaires >2000 et 1100 pour les salaire <=2000

Forme 1