Le langage SQL

Frédéric DADEAU

Institut FEMTO-ST – Département d'Informatique des Systèmes Complexes

Bureau 410 C

Email : frederic.dadeau@univ-fcomte.fr

Licence 1 – Semestre 2

Plan du cours

Data Query Language

Data Manipulation Language

Data Definition Language

Data Control Language

Structured Query Language

SQL est un langage de requêtes structuré, dont les instructions ressemblent à des phrases en anglais.

SQL, un langage déclaratif

- SQL décrit le résultat à obtenir, sans décrire la manière de l'obtenir
- Chaque SGBD possède un interpréteur SQL qui permet d'exécuter la requête pour obtenir le résultat demandé.
- Le traitement de la requête est souvent optimisé pour permettre d'accélérer les calculs.
- Suivant le SGBD utilisé, le traitement sous-jacent diffèrera, mais le résultat d'une requête sera toujours le même.

Requêtes de sélection simple

Les requêtes de sélection utilisent l'ordre SELECT.

Syntaxe d'une requête SELECT

Les requêtes SELECT sont composées d'un certain nombre de clauses :

SELECT *liste d'attributs*
FROM *liste de tables*
WHERE *critère de sélection*
GROUP BY *liste d'attributs de groupe*
HAVING *critère de sélection pour le groupe*
ORDER BY *liste d'attributs*

Seules les clauses SELECT et FROM sont obligatoires.

Requêtes de sélection simple

Les requêtes de sélection utilisent l'ordre SELECT.

Détail des paramètres des clauses

SELECT *liste d'attributs*

Requêtes de sélection simple

Les requêtes de sélection utilisent l'ordre SELECT.

Détail des paramètres des clauses

SELECT *liste d'attributs*
⇒ Noms des attributs à extraire
FROM *liste de tables*

Requêtes de sélection simple

Les requêtes de sélection utilisent l'ordre SELECT.

Détail des paramètres des clauses

SELECT *liste d'attributs*
⇒ Noms des attributs à extraire
FROM *liste de tables*
⇒ Noms des tables utilisées
WHERE *critère de sélection*

Requêtes de sélection simple

Les requêtes de sélection utilisent l'ordre SELECT.

Détail des paramètres des clauses

SELECT *liste d'attributs*
⇒ Noms des attributs à extraire
FROM *liste de tables*
⇒ Noms des tables utilisées
WHERE *critère de sélection*
⇒ Expression déterminant les t-uplets sélectionnés
GROUP BY *liste d'attributs de groupe*

Requêtes de sélection simple

Les requêtes de sélection utilisent l'ordre SELECT.

Détail des paramètres des clauses

SELECT *liste d'attributs*
⇒ Noms des attributs à extraire
FROM *liste de tables*
⇒ Noms des tables utilisées
WHERE *critère de sélection*
⇒ Expression déterminant les t-uplets sélectionnés
GROUP BY *liste d'attributs de groupe*
⇒ Noms des attributs définissant un regroupement
HAVING *critère de sélection pour le groupe*

Requêtes de sélection simple

Les requêtes de sélection utilisent l'ordre SELECT.

Détail des paramètres des clauses

SELECT *liste d'attributs*
⇒ Noms des attributs à extraire
FROM *liste de tables*
⇒ Noms des tables utilisées
WHERE *critère de sélection*
⇒ Expression déterminant les t-uplets sélectionnés
GROUP BY *liste d'attributs de groupe*
⇒ Noms des attributs définissant un regroupement
HAVING *critère de sélection pour le groupe*
⇒ Expression déterminant les groupes sélectionnés
ORDER BY *liste d'attributs*

Requêtes de sélection simple

Les requêtes de sélection utilisent l'ordre SELECT.

Détail des paramètres des clauses

SELECT *liste d'attributs*
⇒ Noms des attributs à extraire
FROM *liste de tables*
⇒ Noms des tables utilisées
WHERE *critère de sélection*
⇒ Expression déterminant les t-uplets sélectionnés
GROUP BY *liste d'attributs de groupe*
⇒ Noms des attributs définissant un regroupement
HAVING *critère de sélection pour le groupe*
⇒ Expression déterminant les groupes sélectionnés
ORDER BY *liste d'attributs*
⇒ Noms des attributs sur lesquels se fait l'ordonnancement

Requêtes de sélection simple

Ordre d'évaluation des clauses

1. clause WHERE
2. clause GROUP BY
3. Fonctions d'agrégation (vues ultérieurement)
4. clause HAVING
5. clause SELECT
6. clause ORDER

L'ordre d'évaluation est important, tâchez de vous en souvenir.

Requêtes de sélection simple

Définition

La clause SELECT permet de définir ce que retourne la requête. Il peut s'agir :

- d'attributs, issus des tables précisées dans la clause FROM
- d'expressions, qui réalisent des calculs utilisant les valeurs d'attributs

La clause SELECT est équivalente à l'opérateur de projection de l'algèbre relationnelle.

Requêtes de sélection simple

Syntaxe de la clause SELECT

La syntaxe générale est la suivante :

- SELECT [DISTINCT] $expr_1$ [AS Nom_1], $expr_2$ [AS Nom_2], . . .
- SELECT [DISTINCT] *

Les éléments entre crochets [ ] sont optionnels.

Requêtes de sélection simple

Syntaxe de la clause SELECT

La syntaxe générale est la suivante :

- SELECT [DISTINCT] $expr_1$ [AS Nom_1], $expr_2$ [AS Nom_2], . . .
- SELECT [DISTINCT] *

Les éléments entre crochets [ ] sont optionnels.

DISTINCT

Le mot-clé DISTINCT permet d'éliminer les doublons dans le résultat.
Attention, en algèbre relationnelle, la projection ne produit pas de doublons.

Requêtes de sélection simple

Syntaxe de la clause SELECT

La syntaxe générale est la suivante :

- SELECT [DISTINCT] $expr_1$ [AS Nom_1], $expr_2$ [AS Nom_2], . . .
- SELECT [DISTINCT] *

Les éléments entre crochets [ ] sont optionnels.

Les expressions

Les expressions $expr_i$ peuvent être :

- des noms d'attributs : NumeroEtudiant, NomEtudiant, ...
- le résultat de fonctions : SUM, AVERAGE, COUNT ... que nous verrons plus loin.

Requêtes de sélection simple

Syntaxe de la clause SELECT

La syntaxe générale est la suivante :

- SELECT [DISTINCT] $expr_1$ [AS Nom_1], $expr_2$ [AS Nom_2], . . .
- SELECT [DISTINCT] *

Les éléments entre crochets [ ] sont optionnels.

La construction AS

La construction *expr* AS *nom* permet de renommer l'expression *expr* pour qu'elle apparaisse sous le dénomination *nom* dans le résultat.

Requêtes de sélection simple

Syntaxe de la clause SELECT

La syntaxe générale est la suivante :

- SELECT [DISTINCT] $expr_1$ [AS Nom_1], $expr_2$ [AS Nom_2], . . .
- SELECT [DISTINCT] *

Les éléments entre crochets [ ] sont optionnels.

L'alias *

L'étoile * est un alias qui désigne tous les attributs disponibles dans la source de données spécifiée dans la clause FROM.
Préfixée du nom d'une relation $R.*$ cette construction désigne tous les attributs de la relation R.

Requêtes de sélection simple

Quelques exemples de requêtes utilisant la clause SELECT (et FROM).

La clause SELECT

SELECT Numero FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Requêtes de sélection simple

Quelques exemples de requêtes utilisant la clause SELECT (et FROM).

La clause SELECT

SELECT Numero FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Résultat :

Numero
23794
32827
32911
33818
34812

Requêtes de sélection simple

Quelques exemples de requêtes utilisant la clause SELECT (et FROM).

La clause SELECT

SELECT Prenom FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Requêtes de sélection simple

Quelques exemples de requêtes utilisant la clause SELECT (et FROM).

La clause SELECT

SELECT Prenom FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Résultat :

Prenom
Arnaud
Maxime
Maxime
Bastien
Vlad

Par défaut, la clause SELECT renvoie tous les doublons existants.
Le résultat d'une requête n'est donc pas une relation (ou une table).

Requêtes de sélection simple

Quelques exemples de requêtes utilisant la clause SELECT (et FROM).

La clause SELECT

SELECT DISTINCT Prenom FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Requêtes de sélection simple

Quelques exemples de requêtes utilisant la clause SELECT (et FROM).

La clause SELECT

SELECT DISTINCT Prenom FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Résultat :

Prenom
Arnaud
Maxime
Bastien
Vlad

L'utilisation du mot-clé DISTINCT permet d'éliminer les doublons du résultat.

Requêtes de sélection simple

Quelques exemples de requêtes utilisant la clause SELECT (et FROM).

La clause SELECT

SELECT DISTINCT Prenom AS FirstName FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Requêtes de sélection simple

Quelques exemples de requêtes utilisant la clause SELECT (et FROM).

La clause SELECT

SELECT DISTINCT Prenom AS FirstName FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Résultat :

FirstName
Arnaud
Maxime
Bastien
Vlad

L'utilisation de la construction *AS FirstName* permet de renommer la colonne *Prenom* en *FirstName*.

Requêtes de sélection simple

Syntaxe de la clause FROM

SELECT ...
FROM Table_1 [AS T1], Table_2 [AS T2]

Utilisation de la clause FROM

Cette clause permet de :

- spécifier la source de données de la requête en indiquant les relations ou les requêtes qui seront utilisées
- définir une relation qui est le produit cartésien des relations décrites

Note : la construction AS permet de renommer/copier la table qui la précède.

Requêtes de sélection simple

Exemple d'utilisation de la clause FROM

```
SELECT Nom, CodeModule
FROM ETUDIANTS1, MODULES1
```

ETUDIANTS1

Numero*	Nom
23794	Dornier
32911	Martin
34812	Sargu

MODULES1

CodeModule*
BD_L1
PROG_L1

Requêtes de sélection simple

Exemple d'utilisation de la clause FROM

```
SELECT Nom, CodeModule
FROM ETUDIANTS1, MODULES1
```

ETUDIANTS1

Numero*	Nom
23794	Dornier
32911	Martin
34812	Sargu

MODULES1

CodeModule*
BD_L1
PROG_L1

Résultat

Nom	CodeModule
Dornier	BD_L1
Dornier	PROG_L1
Martin	BD_L1
Martin	PROG_L1
Sargu	BD_L1
Sargu	PROG_L1

Requêtes de sélection simple

Exemple d'utilisation de la clause FROM

```
SELECT Nom
FROM ETUDIANTS1, MOD1
```

ETUDIANTS1

Numero*	Nom
23794	Dornier
32911	Martin
34812	Sargu

MODULES1

CodeModule*
BD_L1
PROG_L1

Requêtes de sélection simple

Exemple d'utilisation de la clause FROM

SELECT Nom
FROM ETUDIANTS1, MOD1

ETUDIANTS1

Numero*	Nom
23794	Dornier
32911	Martin
34812	Sargu

MODULES1

CodeModule*
BD_L1
PROG_L1

Résultat

Nom
Dornier
Dornier
Martin
Martin
Sargu
Sargu

La clause FROM est évaluée en premier, puis le SELECT $\Rightarrow$ d'où la présence de doublons.

Requêtes de sélection simple

Syntaxe de la clause WHERE

SELECT ...
FROM ...
WHERE *critère de sélection*

Utilisation de la clause WHERE

Cette clause permet de :

- spécifier les t-uplets à sélectionner dans une relation ou dans le produit cartésien de plusieurs relations.
- définir le critère de sélection : un prédicat qui est évalué pour chaque t-uplet

Requêtes de sélection simple

Exemple d'utilisation de la clause WHERE

```
SELECT NumeroEtudiant, CodeModule
FROM INSCRIPTION
WHERE Annee = 2010
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple d'utilisation de la clause WHERE

```
SELECT NumeroEtudiant, CodeModule
FROM INSCRIPTION
WHERE Annee = 2010
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple d'utilisation de la clause WHERE

```
SELECT NumeroEtudiant, CodeModule
FROM INSCRIPTION
WHERE Annee = 2010
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Résultat :

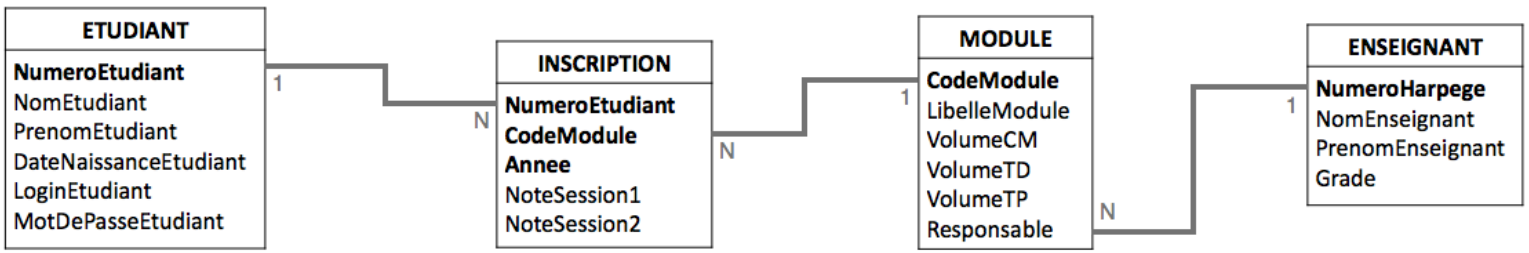
NoEtudiant	CodeModule
23794	BD_L1
32911	BD_L1
32911	PROG_L1
34812	BD_L1

Requêtes de sélection simple

Opérateurs utilisables dans les critères de sélection

- opérateurs de comparaison : $>$, $>=$, $<=$, $<$, $=$, $<>$
- connecteurs logiques : AND, OR, NOT
- prédicats servants à définir des ensembles : IN, BETWEEN ... AND ..., LIKE
- comparaison avec NULL : IS NULL, IS NOT NULL
- dans les chaînes de caractères, utilisation de caractères génériques :
  _ (remplace 1 caractère), % (remplace une chaîne de caractères)

Requêtes de sélection simple

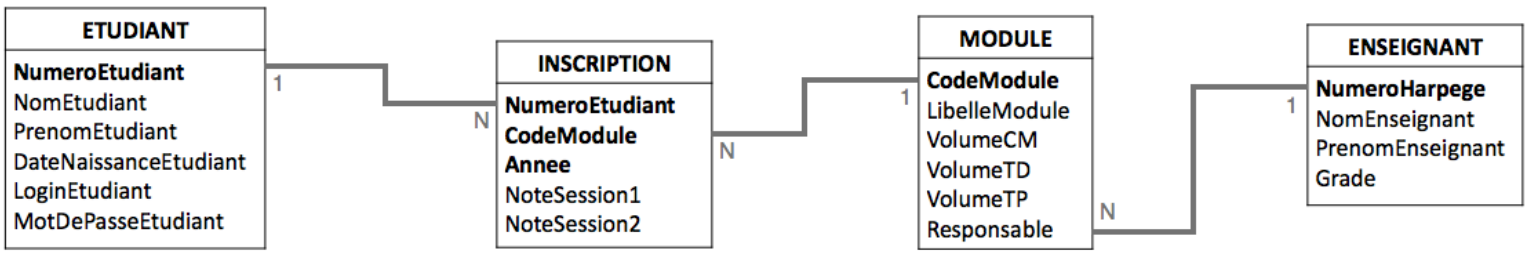


Exemples de requêtes

Numéro des étudiants inscrits dans un module en 2009 ou en 2010.

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE Annee = 2009 OR Annee = 2010
```

Requêtes de sélection simple

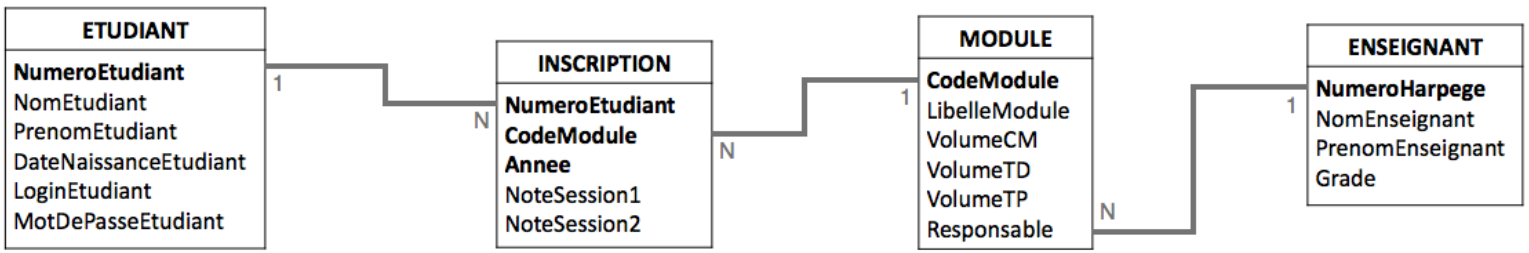


Exemples de requêtes

Numéro des étudiants inscrits dans un module en 2009 ou en 2010.

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE Annee IN (2009, 2010)
```

Requêtes de sélection simple

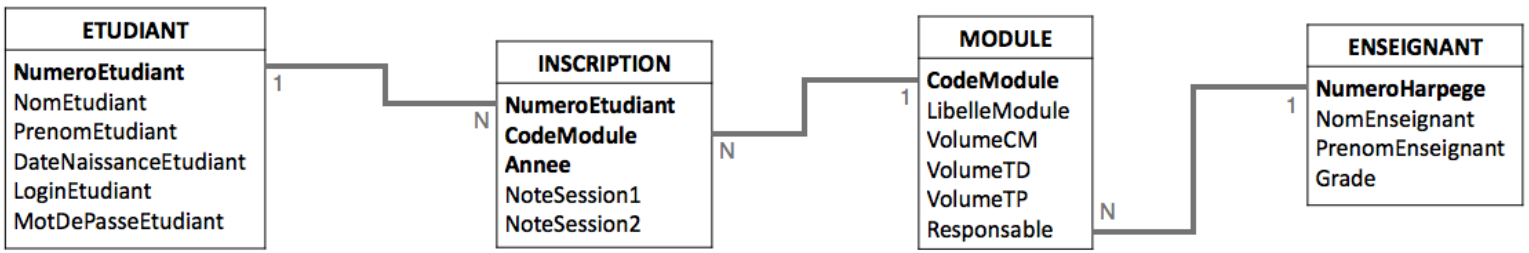


Exemples de requêtes

Numéro des étudiants nés en 1990, ou après.

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE DateNaissanceEtudiant >= '1-01-1990'
```

Requêtes de sélection simple

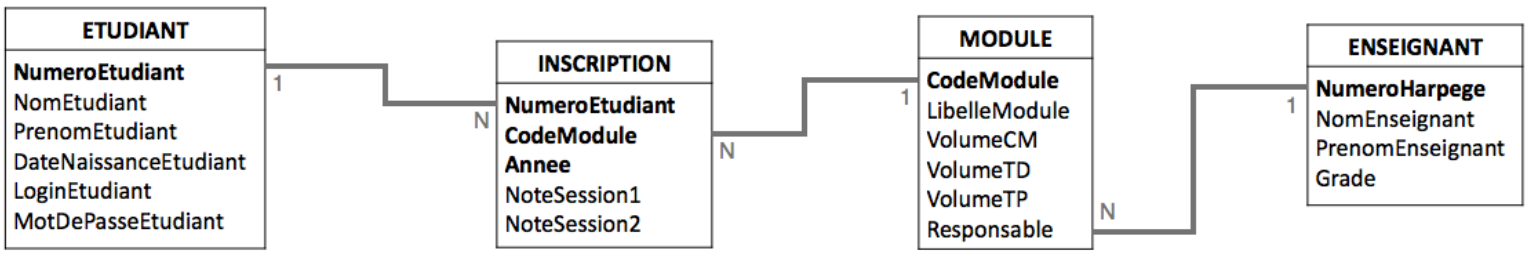


Exemples de requêtes

Numéro des étudiants dont le prénom commence par A.

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE PrenomEtudiant LIKE 'a%' OR PrenomEtudiant LIKE 'A%'
```

Requêtes de sélection simple

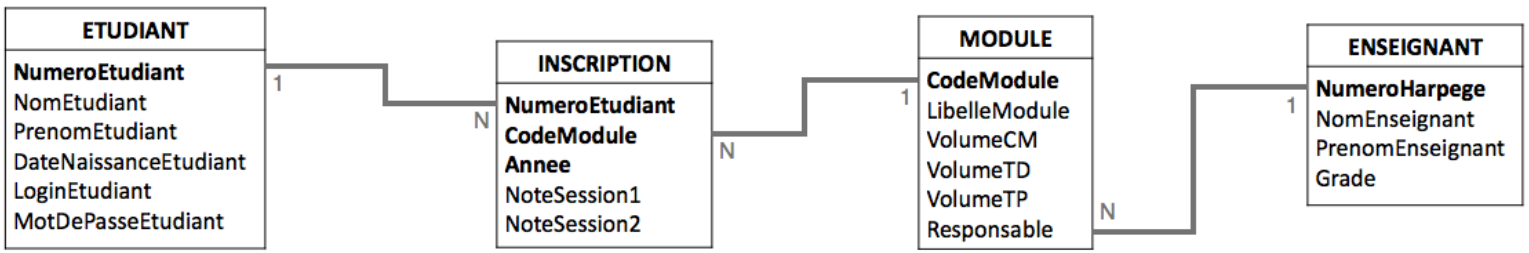


Exemples de requêtes

Numéro des étudiants ayant été en seconde session. On souhaite également avoir le code du module.

```
SELECT NumeroEtudiant, CodeModule
FROM INSCRIPTION
WHERE NoteSession2 IS NOT NULL
```

Requêtes de sélection simple

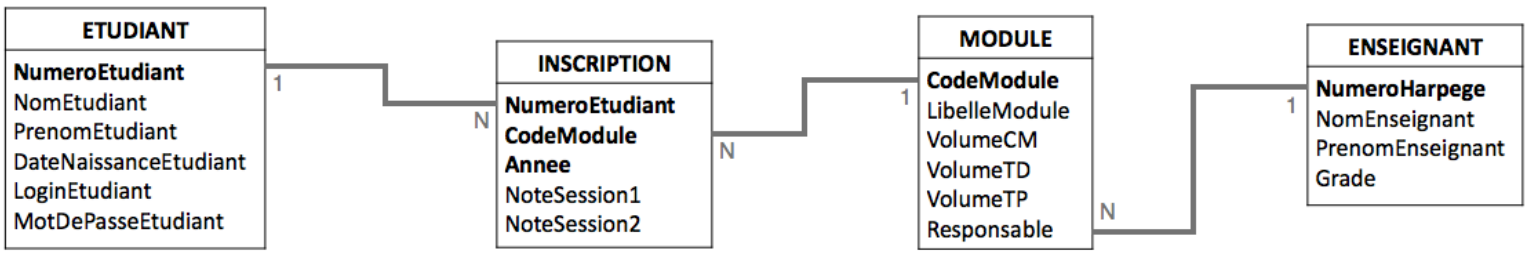


Exemples de requêtes

Numéro des étudiants ayant validé le module BD_L1 en session 1 avec une note entre 12 et 14.

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE CodeModule = 'BD_L1'
AND NoteSession1 >= 12 AND NoteSession1 <= 14
```

Requêtes de sélection simple



Exemples de requêtes

Numéro des étudiants ayant validé le module BD_L1 en session 1 avec une note entre 12 et 14.

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE CodeModule = 'BD_L1'
AND NoteSession1 BETWEEN 12 AND 14
```

Requêtes de sélection simple

La jointure, on l'a vu précédemment, est la composition d'un produit cartésien et d'une sélection selon le critère de jointure.

Jointure en SQL

SQL propose deux manières de réaliser une jointure :

- en utilisant la clause WHERE
- en utilisant l'ordre JOIN dans la clause FROM

Requêtes de sélection simple

Jointure en SQL avec la clause WHERE

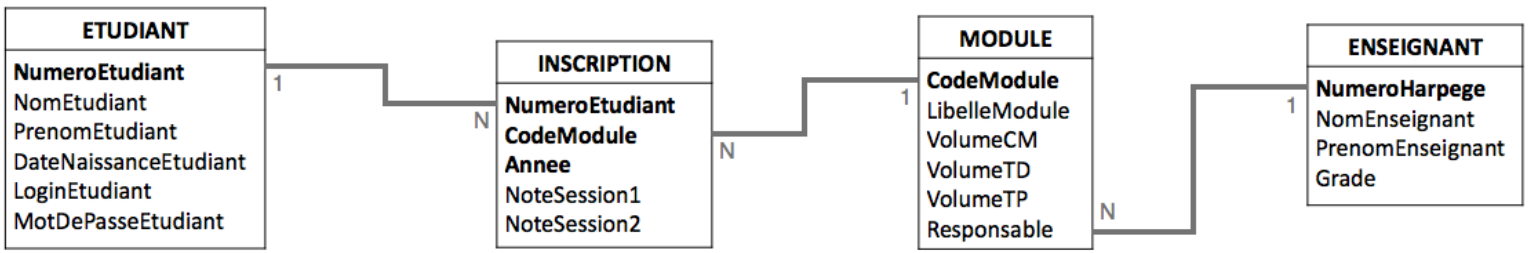
- Cette solution a été proposée dans SQL1
- Elle permet de réaliser des jointures internes, mais pas externes [a].

a. Certaines versions récentes de SQL permettent d'utiliser cette construction pour réaliser une jointure externe

Syntaxe de $R_1[C]R_2$

SELECT ...
FROM R_1, R_2
WHERE C

Requêtes de sélection simple



Jointure avec WHERE

Libellé des modules et nom de leur responsable.

```
SELECT LibelleModule, NomEnseignant
FROM MODULE, ENSEIGNANT
WHERE MODULE.Responsable = ENSEIGNANT.NoHarpege
```

Requêtes de sélection simple

Cette solution a été introduite dans SQL2. Elle permet de réaliser aussi bien des jointures internes qu'externes.

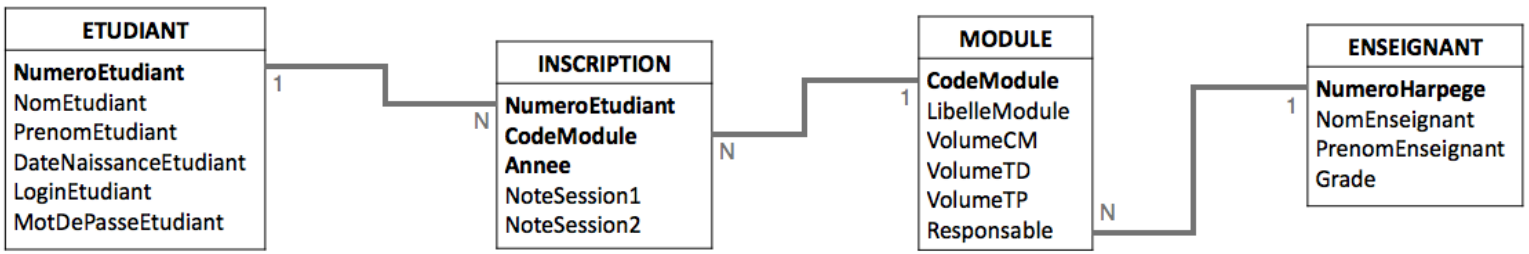
Syntaxe d'une jointure avec l'ordre JOIN

SELECT ...
FROM R_1 Opérateur Jointure R_2 ON C

Opérateur de jointure

- Jointure interne : INNER JOIN
- Jointure externe : OUTER JOIN, avec plusieurs déclinaisons
  - LEFT OUTER JOIN - jointure gauche
  - RIGHT OUTER JOIN - jointure droite
  - FULL OUTER JOIN - jointure gauche et droite
- Jointure naturelle : NATURAL JOIN

Requêtes de sélection simple

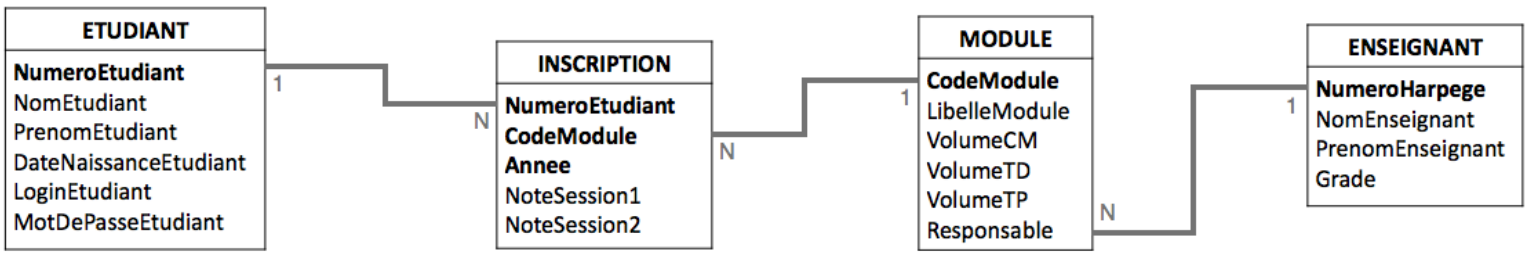


Jointure avec JOIN

Libellé des modules et nom de leur responsable.

```
SELECT LibelleModule, NomEnseignant
FROM MODULE INNER JOIN ENSEIGNANT
ON MODULE.Responsable = ENSEIGNANT.NoHarpege
```

Requêtes de sélection simple

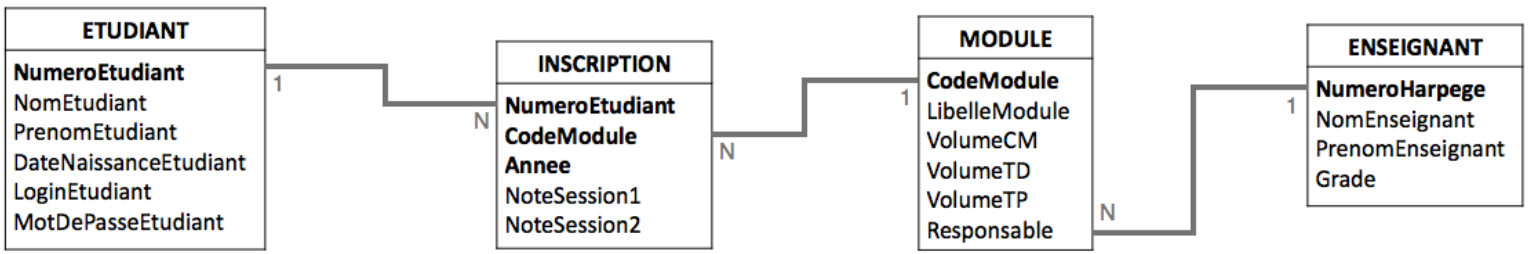


Jointure avec JOIN

Noms des étudiants qui ont déjà été inscrits à au moins un module en 2010

```
SELECT NomEtudiant
FROM ETUDIANT INNER JOIN INSCRIPTION
ON ETUDIANT.NumeroEtudiant = INSCRIPTION.NumeroEtudiant
WHERE Annee = 2010
```

Requêtes de sélection simple

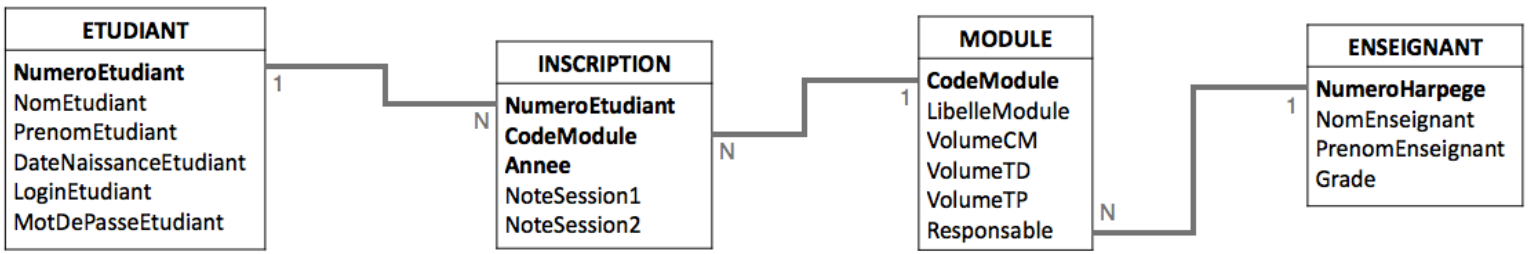


Jointure avec JOIN

Numéro des étudiants qui ont déjà été inscrits à au moins un module en 2010

```
SELECT NumeroEtudiant
FROM ETUDIANT INNER JOIN INSCRIPTION
ON ETUDIANT.NumeroEtudiant = INSCRIPTION.NumeroEtudiant
WHERE Annee = 2010
```

Requêtes de sélection simple

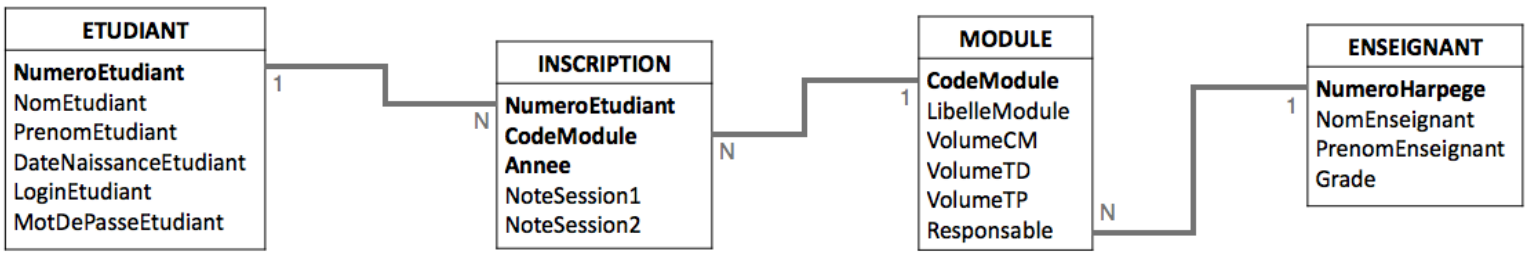


Jointure avec JOIN

Numéro des étudiants qui ont déjà été inscrits à au moins un module en 2010

SELECT NumeroEtudiant ← **Ambiguité !**
FROM ETUDIANT INNER JOIN INSCRIPTION
ON ETUDIANT.NumeroEtudiant = INSCRIPTION.NumeroEtudiant
WHERE Annee = 2010

Requêtes de sélection simple



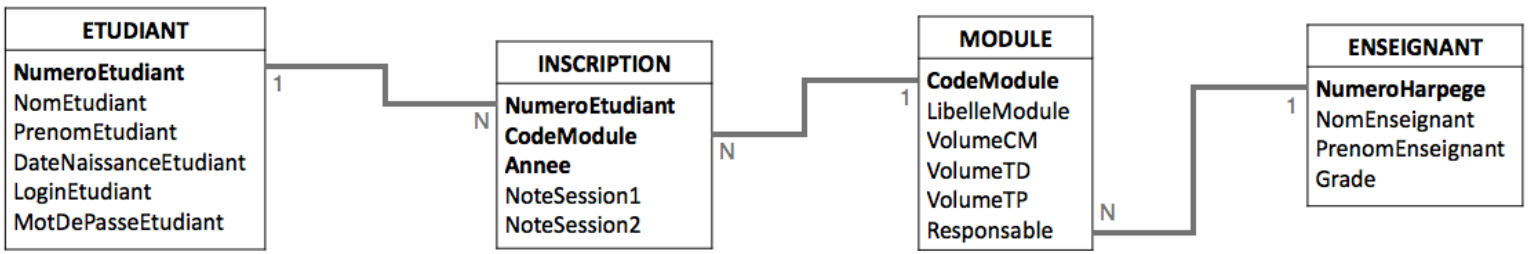
Jointure avec JOIN

Numéro des étudiants qui ont déjà été inscrits à au moins un module en 2010

```
SELECT ETUDIANT.NumeroEtudiant
FROM ETUDIANT INNER JOIN INSCRIPTION
ON ETUDIANT.NumeroEtudiant = INSCRIPTION.NumeroEtudiant
WHERE Annee = 2010
```

En cas d'ambiguité, si deux champs existent dans la source de donnée avec le même nom, on préfixe ces champs par le nom de leur table d'origine.

Requêtes de sélection simple



Jointure avec JOIN

Numéro des étudiants qui ont déjà été inscrits à au moins un module en 2010

```
SELECT ETUDIANT.NumeroEtudiant
FROM ETUDIANT NATURAL JOIN INSCRIPTION
WHERE Annee = 2010
```

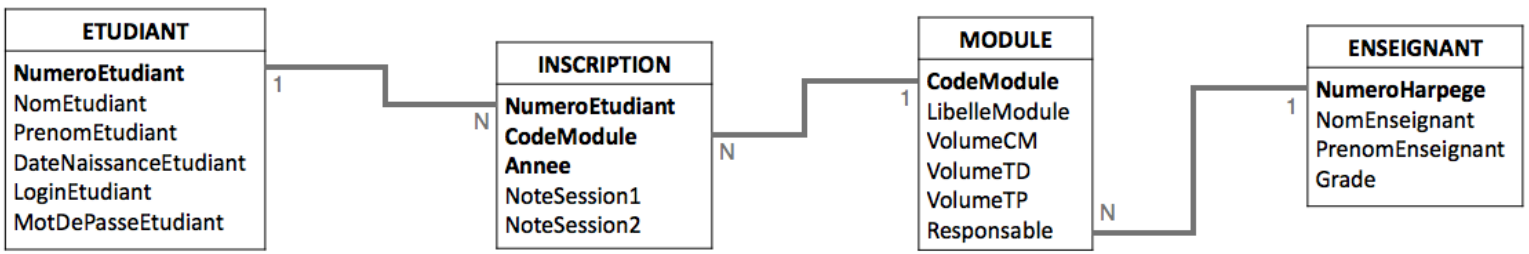
Requêtes de sélection simple

Un mot sur NATURAL JOIN

SQL choisit automatiquement les attributs possédant le même nom dans les deux tables comme attributs de jointure entre les tables.

Pour n'utiliser que certains attributs, on précisera lesquels avec la construction USING(att_1).

Requêtes de sélection simple

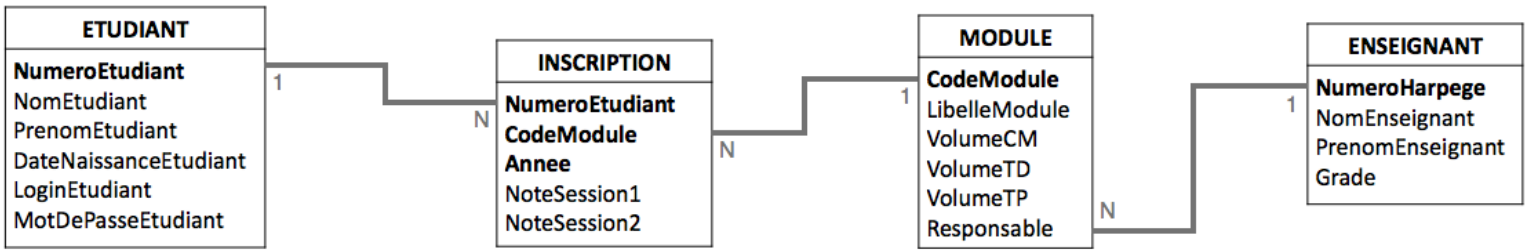


Jointure externe avec OUTER JOIN

Nom des enseignants, et libellé des modules dont ils sont responsables. Tous les enseignants doivent apparaître dans le résultat.

```
SELECT NomEnseignant, LibelleModule
FROM ENSEIGNANT LEFT OUTER JOIN MODULE
ON ENSEIGNANT.NoHarpege = MODULE.Responsable
```

Requêtes de sélection simple

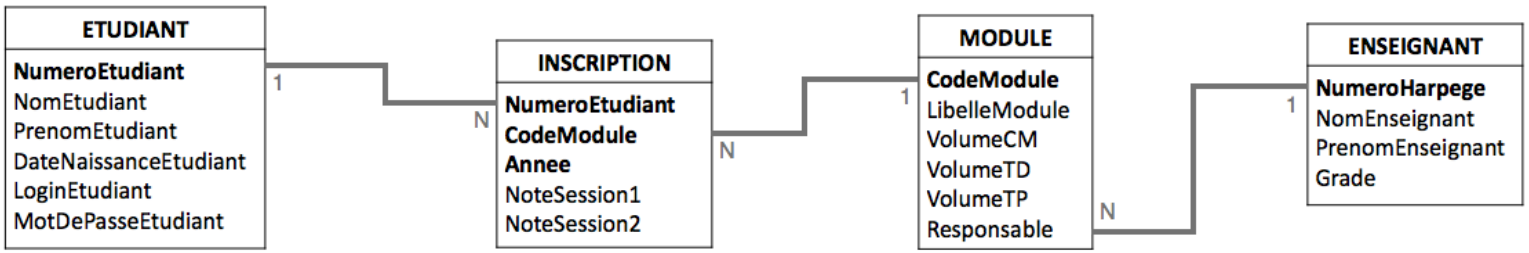


Jointure externe avec OUTER JOIN

Nom des enseignants, et libellé des modules dont ils sont responsables. Tous les enseignants doivent apparaître dans le résultat.

```
SELECT NomEnseignant, LibelleModule
FROM MODULE RIGHT OUTER JOIN ENSEIGNANT
ON MODULE.Responsable = ENSEIGNANT.NoHarpege
```

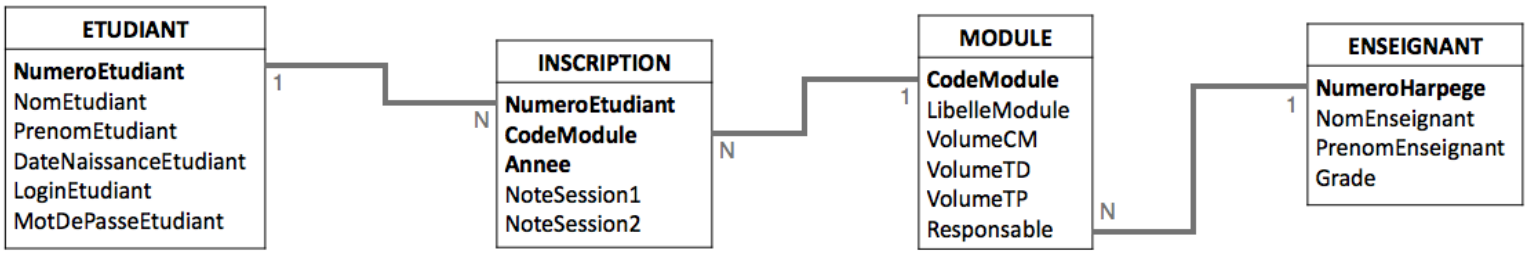
La jointure en SQL



Auto-jointure

Les noms et prénoms des étudiants qui ont côtoyé l'étudiant 23974.
Autrement dit : les étudiants qui ont été inscrits au même module la même année.

La jointure en SQL

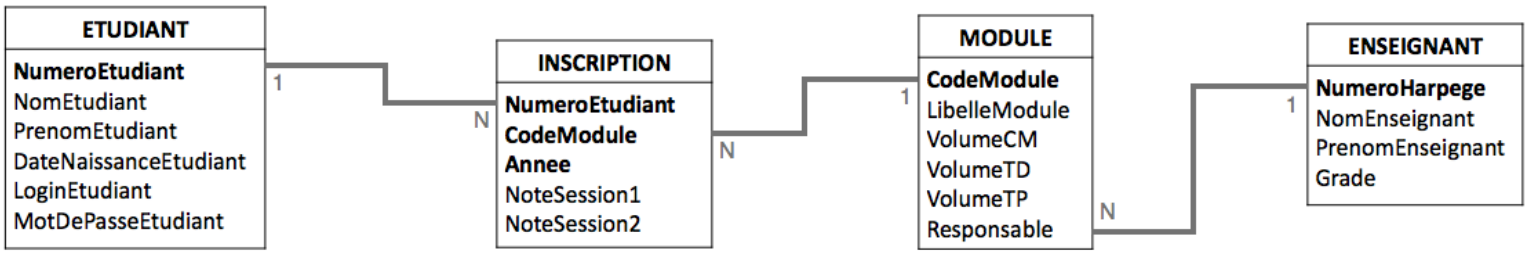


Auto-jointure

Les noms et prénoms des étudiants qui ont côtoyé l'étudiant 23974.
Autrement dit : les étudiants qui ont été inscrits au même module la même année.

Principe : réaliser une jointure entre une table et elle-même (auto-jointure). Ici, on utilisera la table inscription dont on réalisera une copie, en utilisant la construction AS dans la clause FROM. Le critère de jointure portera à la fois sur les attributs *CodeModule* et *Annee*.

La jointure en SQL

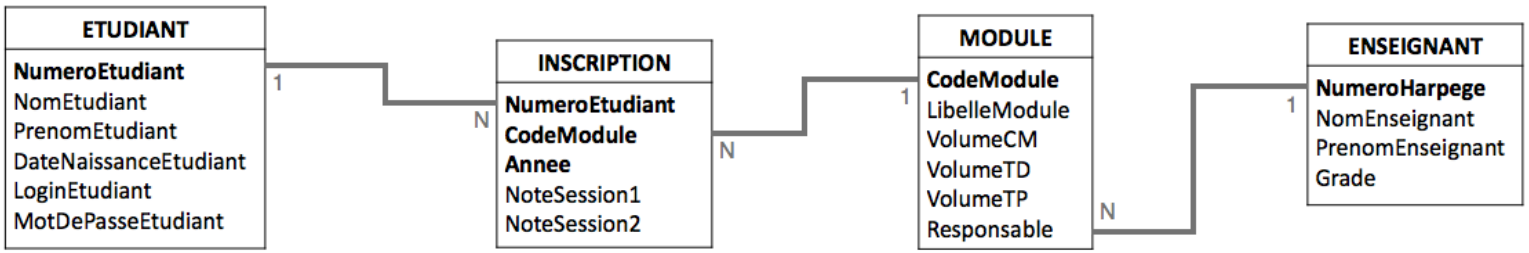


Auto-jointure

Les noms et prénoms des étudiants qui ont côtoyé l'étudiant 23974.
Autrement dit : les étudiants qui ont été inscrits au même module la même année.

```
SELECT NomEtudiant, PrenomEtudiant
FROM ETUDIANT, INSCRIPTION AS I1, INSCRIPTION AS I2
WHERE ETUDIANT.NumeroEtudiant = I1.NumeroEtudiant
AND I1.CodeModule = I2.CodeModule
AND I1.Annee = I2.Annee
AND I2.NumeroEtudiant = 23974
```

A vous de jouer



Donnez les requêtes permettant de calculer, sur notre exemple :

- Le nom et prénom des étudiants nés avant 1991.
- Le nom et le prénom des étudiants ayant validé leurs modules en session 1 en 2010.
- Les nom et prénom des enseignants ayant le grade “MCF”, avec le libellé des modules dont ils sont responsables (même s’ils n’ont pas de module).
- Le libellé des modules où est inscrit l’étudiant numéro 23794 en 2010.
- Le libellé des modules communs aux étudiants 23974 et 33818.

Requêtes de sélection simple

Fonctions de groupe

Egalement appelées *fonctions d'agrégation*, elles permettent d'obtenir des informations sur un ensemble de t-uplets dans une relation.

Ces fonctions permettent de travailler sur les colonnes, et non sur les lignes

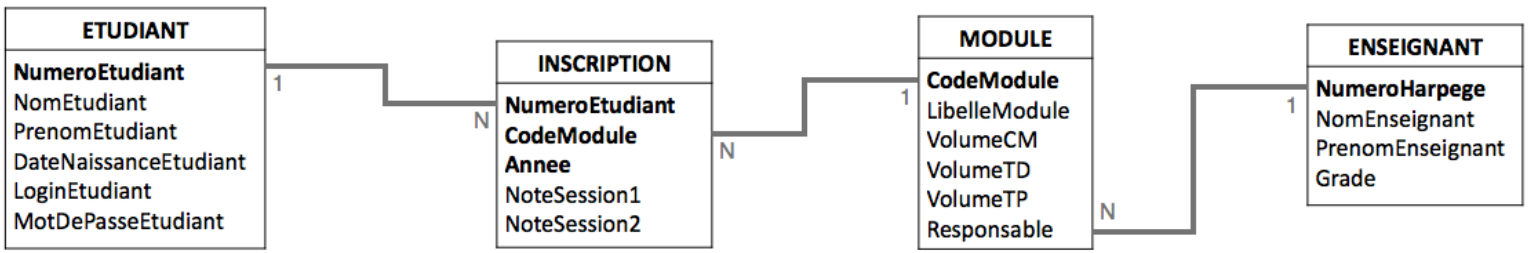
Requêtes de sélection simple

Liste des fonctions de groupe

- AVG : moyenne des valeurs
- SUM : somme des valeurs
- MIN : plus petite valeur
- MAX : plus grande valeur
- VARIANCE : variance des valeurs
- STDDEV : écart-type des valeurs
- COUNT : nombre de lignes

Ces fonctions prennent en argument le nom d'une colonne, sur laquelle on cherche à appliquer le calcul souhaité.

Requêtes de sélection simple

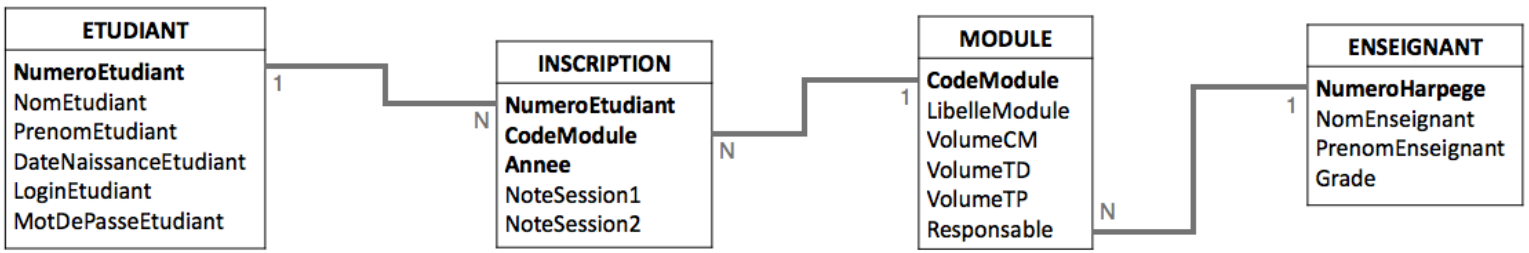


Utilisation des fonctions de groupe

Date de naissance du plus vieil étudiant inscrit en BD_L1 en 2010.

```
SELECT MIN(DateNaissanceEtudiant)
FROM ETUDIANT, INSCRIPTION
WHERE ETUDIANT.NumeroEtudiant = INSCRIPTION.NumeroEtudiant
AND INSCRIPTION.CodeModule = 'BD_L1'
AND INSCRIPTION.Annee = 2010
```

Requêtes de sélection simple

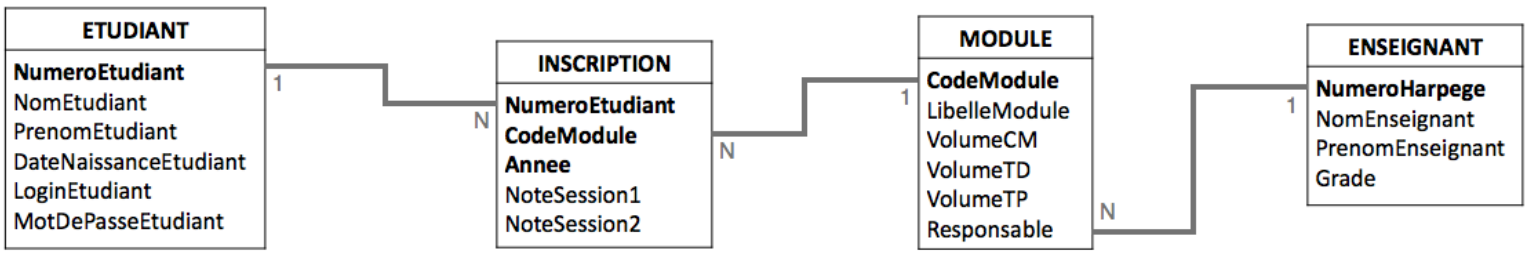


Utilisation des fonctions de groupe

Date de naissance du plus jeune étudiant inscrit en BD_L1 en 2010.

```
SELECT MAX(DateNaissanceEtudiant)
FROM ETUDIANT, INSCRIPTION
WHERE ETUDIANT.NumeroEtudiant = INSCRIPTION.NumeroEtudiant
AND INSCRIPTION.CodeModule = 'BD_L1'
AND INSCRIPTION.Annee = 2010
```

Requêtes de sélection simple

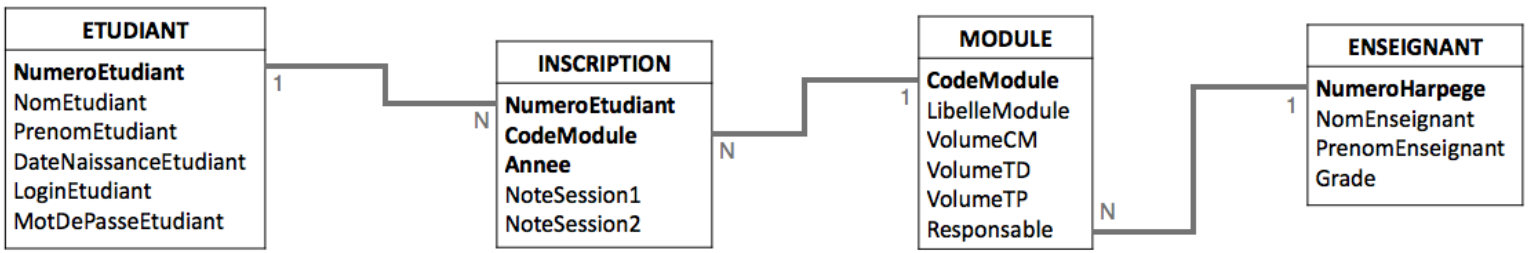


Utilisation des fonctions de groupe

Moyenne des notes de première session au module BD_L1 en 2010.

```
SELECT AVG(NoteSession1) AS MoyenneSession1
FROM INSCRIPTION
WHERE Annee = 2010
AND CodeModule = 'BD_L1'
```

Requêtes de sélection simple



Utilisation des fonctions de groupe

Nombre d'étudiants inscrits au module BD_L1 en 2010.

```
SELECT COUNT(NumeroEtudiant)
FROM INSCRIPTION
WHERE Annee = 2010
AND CodeModule = 'BD_L1'
```

Requêtes de sélection simple

Particularité de la fonction COUNT

On distingue 3 types d'appels de cette fonction :

- COUNT (*) : compte le nombre de t-uplets obtenus dans la requête (lignes).
- COUNT (*att*) : compte le nombre de valeurs de l'attribut *att* différentes de NULL.
- COUNT (DISTINCT *att*) : compte le nombre de valeurs différentes de *att* [a].

a. N'existe pas dans Access

Requêtes de sélection simple

Utilisation de la fonction COUNT

ENSEIGNANT

NoHarpege*	Nom	Prenom	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF
22223	Guyennet	Hervé	PR
32598	Paquette	Guillaume	NULL

Requêtes de sélection simple

Utilisation de la fonction COUNT

ENSEIGNANT

NoHarpege*	Nom	Prenom	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF
22223	Guyennet	Hervé	PR
32598	Paquette	Guillaume	NULL

SELECT COUNT(*) FROM ENSEIGNANT $\Rightarrow$ 4

Requêtes de sélection simple

Utilisation de la fonction COUNT

ENSEIGNANT

NoHarpege*	Nom	Prenom	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF
22223	Guyennet	Hervé	PR
32598	Paquette	Guillaume	NULL

SELECT COUNT(Grade) FROM ENSEIGNANT $\Rightarrow$ 3

Requêtes de sélection simple

Utilisation de la fonction COUNT

ENSEIGNANT

NoHarpege*	Nom	Prenom	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF
22223	Guyennet	Hervé	PR
32598	Paquette	Guillaume	NULL

SELECT COUNT(DISTINCT Grade) FROM ENSEIGNANT $\Rightarrow$ 2

Requêtes de sélection simple

Syntaxe de la clause GROUP BY

SELECT ...
FROM ...
WHERE ...
GROUP BY $expr_1$, $expr_2$, ...

Définition

La clause GROUP BY permet de subdiviser la relation en groupes :

- une seule ligne représente l'ensemble des t-uplets de la relation regroupée,
- tous les t-uplets regroupés ont la même valeur pour les expressions du regroupement.

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant, Annee

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant, Annee

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant, Annee

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant, Annee

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Exemple de clause GROUP BY

GROUP BY NoEtudiant, Annee

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Requêtes de sélection simple

Utilisation de la clause GROUP BY

La clause GROUP BY est principalement utilisée en complément de fonctions d'agrégation.

SELECT F($expr_1$)
FROM ...
WHERE ...
GROUP BY $expr_2$, $expr_3$

A comprendre comme : *"je veux évaluer F($expr_1$) pour chaque groupe défini par une seule valeur de ($expr_2$,$expr_3$)"*

Utilisation de GROUP BY et COUNT

On cherche à obtenir le nombre d'inscriptions par module pour une année donnée.

- on fait un regroupement sur les attributs "Annee" et "CodeModule"
- on compte les lignes de chaque groupe

Requêtes de sélection simple

Utilisation de GROUP BY et COUNT

```
SELECT COUNT(*) AS NbInscr
FROM INSCRIPTION
GROUP BY Annee, CodeModule
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Résultat :

NbInscr

Requêtes de sélection simple

Utilisation de GROUP BY et COUNT

```
SELECT COUNT(*) AS NbInscr
FROM INSCRIPTION
GROUP BY Annee, CodeModule
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Résultat :

NbInscr
1

Requêtes de sélection simple

Utilisation de GROUP BY et COUNT

```
SELECT COUNT(*) AS NbInscr
FROM INSCRIPTION
GROUP BY Annee, CodeModule
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Résultat :

NbInscr
1
3

Requêtes de sélection simple

Utilisation de GROUP BY et COUNT

```
SELECT COUNT(*) AS NbInscr
FROM INSCRIPTION
GROUP BY Annee, CodeModule
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Résultat :

NbInscr
1
3
1

Requêtes de sélection simple

Utilisation de GROUP BY et COUNT

```
SELECT COUNT(*) AS NbInscr
FROM INSCRIPTION
GROUP BY Annee, CodeModule
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Résultat :

NbInscr
1
3
1
1

Requêtes de sélection simple

Utilisation de GROUP BY et COUNT

```
SELECT Annee, CodeModule, COUNT(*) AS Nb
FROM INSCRIPTION
GROUP BY Annee, CodeModule
```

INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2009	7	9
23794	BD_L1	2010	15	NULL
23794	PROG_L1	2009	16	NULL
32911	BD_L1	2010	7	12
32911	PROG_L1	2010	16	NULL
34812	BD_L1	2010	10	NULL

Résultat :

Annee	CodeModule	Nb
2009	BD_L1	1
2010	BD_L1	3
2009	PROG_L1	1
2010	PROG_L1	1

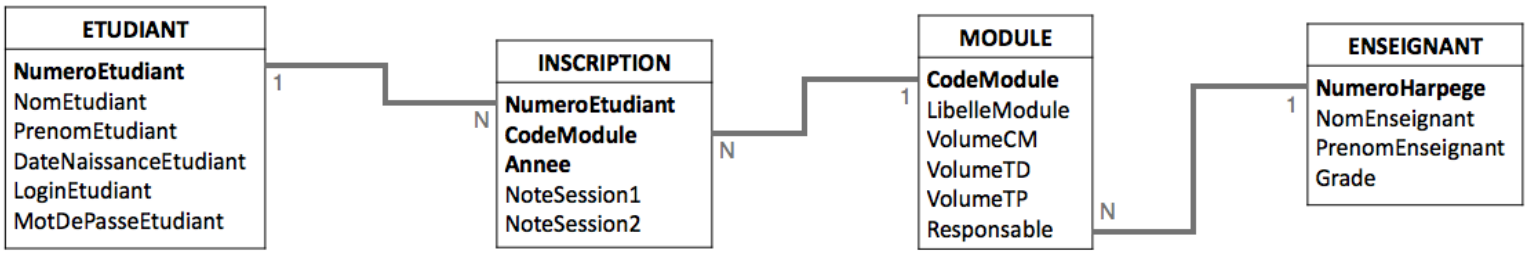
Requêtes de sélection simple

Restrictions de la clause GROUP BY

Les seules opérations réalisables sur une “table regroupée” sont :

- les opérations qui prennent en compte un ensemble de lignes (fonctions de groupe)
- les opérations de projection portant sur des attributs impliqués dans le regroupement

Requêtes de sélection simple

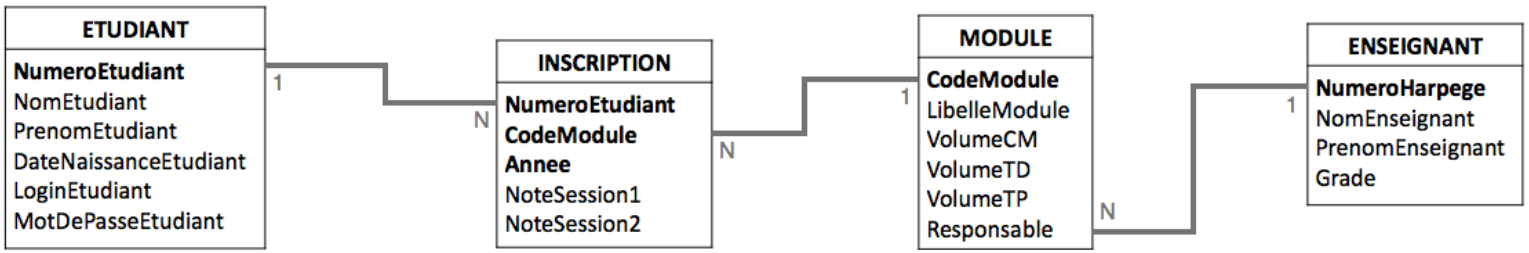


Utilisation de GROUP BY et COUNT

On veut le nom des enseignants et le nombre de modules dont ils sont responsables.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE, ENSEIGNANT
WHERE MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NomEnseignant
```

Requêtes de sélection simple



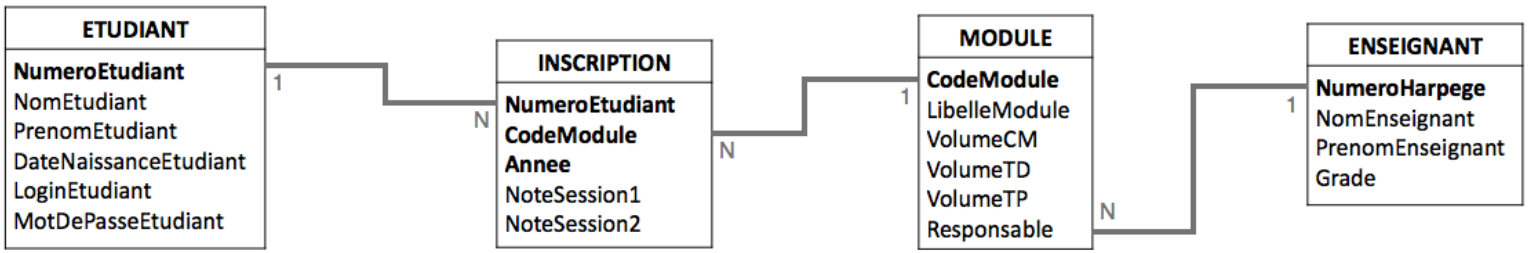
Utilisation de GROUP BY et COUNT

On veut le nom des enseignants et le nombre de modules dont ils sont responsables.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE, ENSEIGNANT
WHERE MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NomEnseignant
```

Que faire si deux enseignants ont le même nom ? Le résultat sera erroné !

Requêtes de sélection simple

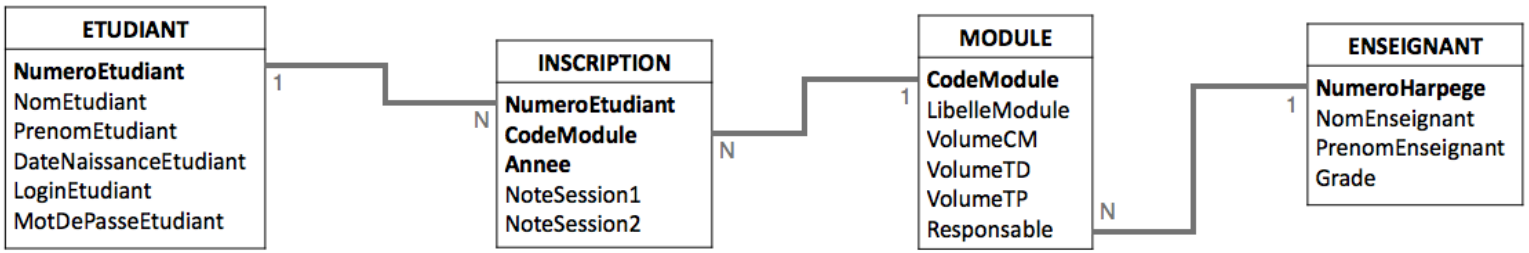


Utilisation de GROUP BY et COUNT

On veut le nom des enseignants et le nombre de modules dont ils sont responsables.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE, ENSEIGNANT
WHERE MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NumeroHarpege
```

Requêtes de sélection simple



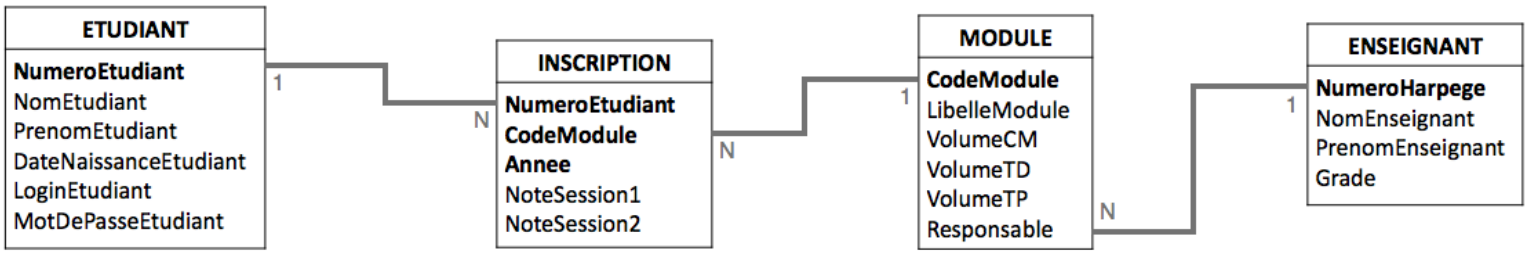
Utilisation de GROUP BY et COUNT

On veut le nom des enseignants et le nombre de modules dont ils sont responsables.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE, ENSEIGNANT
WHERE MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NumeroHarpege
```

“Les opérations de projection portant sur des attributs impliqués dans le regroupement”
Erreur de syntaxe, il faut rajouter les attributs du SELECT dans le GROUP BY.

Requêtes de sélection simple

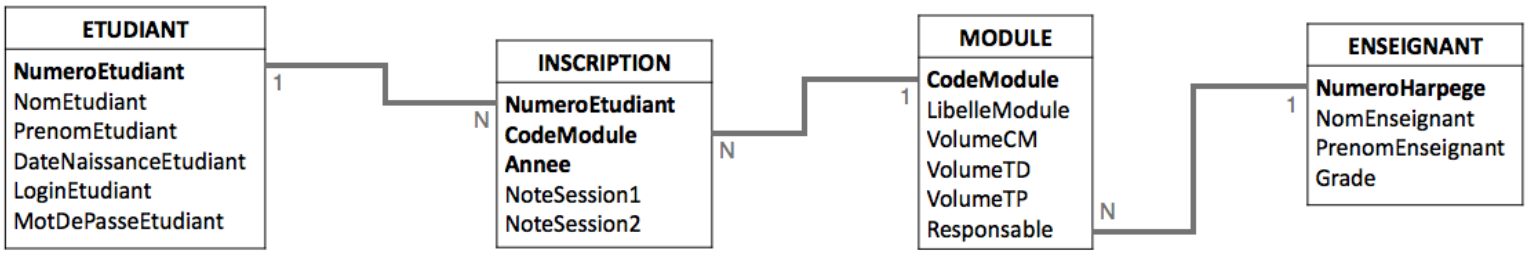


Utilisation de GROUP BY et COUNT

On veut le nom des enseignants et le nombre de modules dont ils sont responsables.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE, ENSEIGNANT
WHERE MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NumeroHarpege, NomEnseignant
```

Requêtes de sélection simple



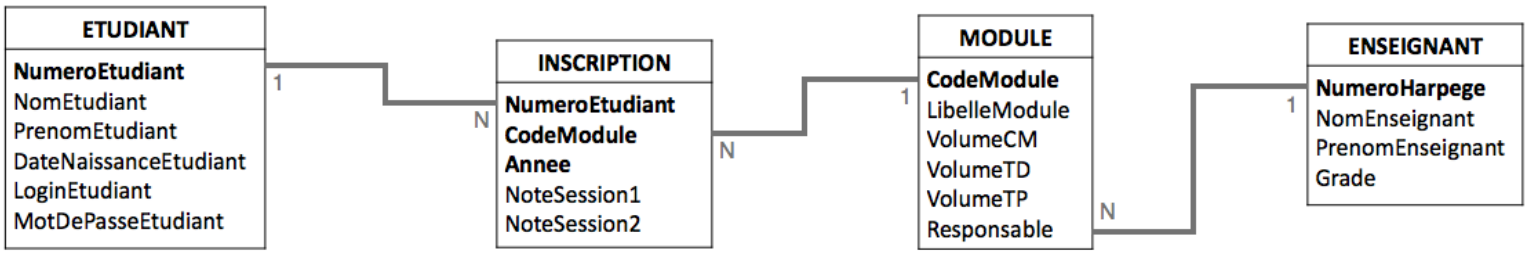
Utilisation de GROUP BY et COUNT

On veut le nom des enseignants et le nombre de modules dont ils sont responsables.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE, ENSEIGNANT
WHERE MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NumeroHarpege, NomEnseignant
```

Pas mal, mais si un enseignant n'est responsable d'aucun module, il n'apparaît pas.

Requêtes de sélection simple



Utilisation de GROUP BY et COUNT

On veut le nom des enseignants et le nombre de modules dont ils sont responsables.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE RIGHT OUTER JOIN ENSEIGNANT
ON MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NumeroHarpege, NomEnseignant
```

On utilise donc une jointure externe pour avoir tous les enregistrements de ENSEIGNANT.

Requêtes de sélection simple

Syntaxe de la clause HAVING

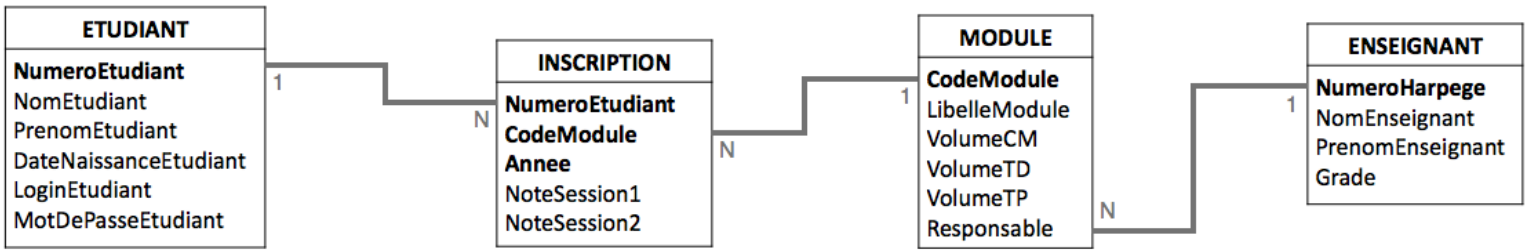
SELECT ...
FROM ...
WHERE ...
GROUP BY ...
HAVING *C*

Définition

La clause HAVING :

- est similaire à la clause WHERE
- permet de définir un critère de sélection utilisant une fonction d'agrégation (interdit dans la clause WHERE)
- réutilise les regroupements dans le GROUP BY

Requêtes de sélection simple

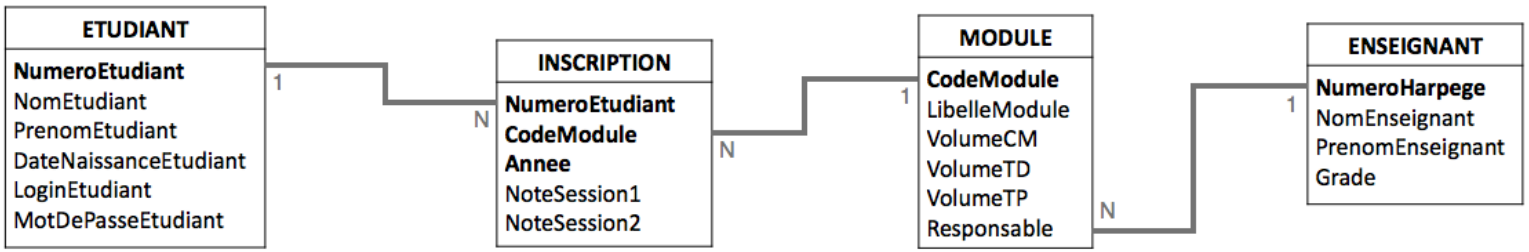


Utilisation de HAVING

On veut le nom des enseignants qui sont responsables d'au moins 2 modules.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE RIGHT OUTER JOIN ENSEIGNANT
ON MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NumeroHarpege, NomEnseignant
HAVING COUNT(CodeModule) >= 2
```

Requêtes de sélection simple



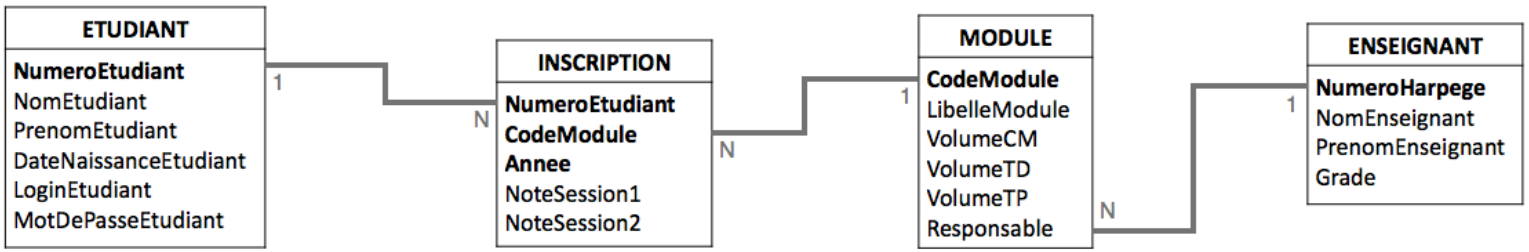
Utilisation de HAVING

On veut le nom des enseignants qui sont responsables d'au moins 2 modules.

```
SELECT NomEnseignant, COUNT(CodeModule)
FROM MODULE RIGHT OUTER JOIN ENSEIGNANT
ON MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NumeroHarpege, NomEnseignant
HAVING COUNT(CodeModule) >= 2
```

Il n'est pas obligatoire de renvoyer le résultat du COUNT.

Requêtes de sélection simple



Utilisation de HAVING

On veut le nom des enseignants qui sont responsables d'au moins 2 modules.

```
SELECT NomEnseignant
FROM MODULE RIGHT OUTER JOIN ENSEIGNANT
ON MODULE.Responsable = ENSEIGNANT.NumeroHarpege
GROUP BY NumeroHarpege, NomEnseignant
HAVING COUNT(CodeModule) >= 2
```

Requêtes de sélection simple

Syntaxe de la clause ORDER BY

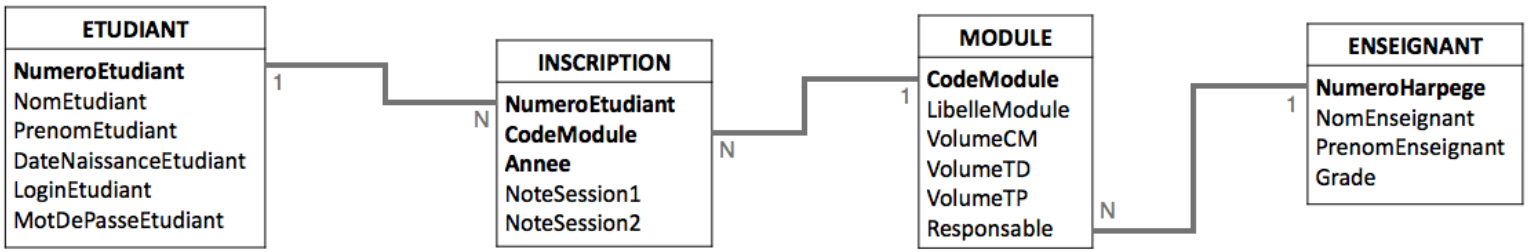
SELECT ...
FROM ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY $expr_1$ [ASC ou DESC], $expr_2$ [ASC ou DESC]

Définition

La clause ORDER BY permet d'ordonner le résultat suivant un ou plusieurs critères.

- ASC correspond à un tri croissant (ascendant)
- DESC correspond à un tri décroissant (descendant)
- par défaut, ASC s'applique

Requêtes de sélection simple



Utilisation de la clause ORDER BY

```
SELECT NomEtudiant, PrenomEtudiant
FROM ETUDIANT
ORDER BY PrenomEtudiant DESC, NomEtudiant ASC
```

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Dornier	Arnaud	27-09-1990	adornier	adornier
32827	Bros	Maxime	3-10-1992	mbros	mbros
32911	Martin	Maxime	2-12-1992	mmartin	mmartin
33818	Schatt	Bastien	23-05-1992	bschatt	bschatt
34812	Sargu	Vlad	15-12-1990	vsargu	vsargu

Résultat :

Nom	Prenom
Sargu	Vlad
Bros	Maxime
Martin	Maxime
Schatt	Bastien
Dornier	Arnaud

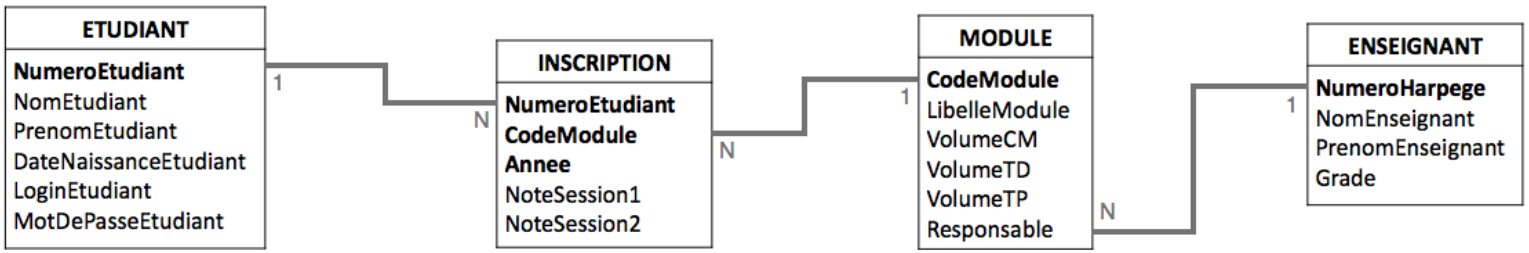
Sous-requêtes

Définition

Les sous-requêtes permettent l'expression d'un prédicat (dans la clause WHERE ou HAVING) en utilisant le résultat d'un SELECT.

SELECT ...
FROM ...
WHERE *attr* < *operateur* > (SELECT *attr*
FROM ...
WHERE ...)

Sous-requêtes



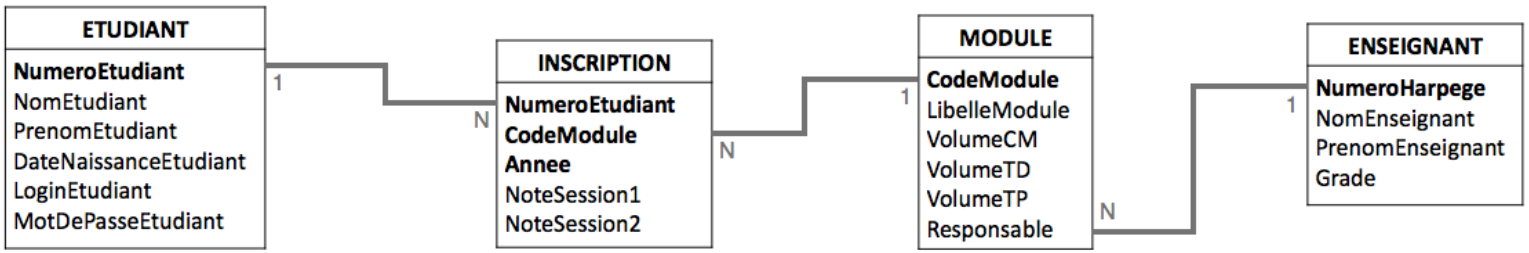
Une requête pas si simple ...

On souhaite connaître les modules qui ont le même responsable que le module BD_L1.

Deux solutions :

- (auto-)jointure
- sous-requête

Sous-requêtes



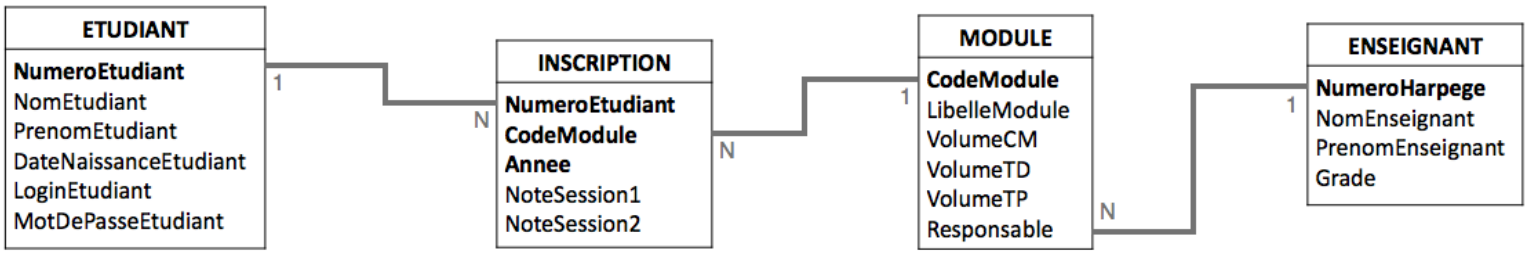
Une requête pas si simple ...

On souhaite connaître les modules qui ont le même responsable que le module BD_L1.

Solution 1 : avec une auto-jointure

```
SELECT M1.CodeModule
FROM Module AS M1, Module AS M2
WHERE M1.Responsable = M2.Responsable
AND M2.CodeModule = 'BD_L1'
```

Sous-requêtes



Une requête pas si simple ...

On souhaite connaître les modules qui ont le même responsable que le module BD_L1.

Solution 2 : avec une sous-requête

```
SELECT CodeModule
FROM Module
WHERE Responsable =   ( SELECT Responsable
                        FROM Module
                        WHERE CodeModule = 'BD_L1')
```

Sous-requêtes

Deux types de sous-requêtes

Il existe en réalité deux types de sous-requêtes, en fonction de leur ordre d'évaluation :

- les sous-requêtes indépendantes

  $\Rightarrow$ la sous-requête est évaluée avant la requête principale

- les sous-requêtes synchronisées

  $\Rightarrow$ la sous-requête est évaluée après la requête principale

Sous-requêtes

Définition

Une sous-requête indépendante :

- est une sous requête qui ne fait aucune référence aux attributs manipulés dans la requête principale
- sera évaluée avant la requête principale

Sous-requêtes

Sous-requête indépendante

```
SELECT CodeModule
FROM Module
WHERE Responsable =   ( SELECT Responsable
                        FROM Module
                        WHERE CodeModule = 'BD_L1')
```

Module

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test logiciel	...	7914
SR_L3	Systèmes et réseaux	...	22223
MCOO_L3	Modélisation objet	...	7914

Sous-requêtes

Sous-requête indépendante

```
SELECT CodeModule
FROM Module
WHERE Responsable =   ( SELECT Responsable
                        FROM Module
                        WHERE CodeModule = 'BD_L1')
```

Module

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test logiciel	...	7914
SR_L3	Systèmes et réseaux	...	22223
MCOO_L3	Modélisation objet	...	7914

Sous-requêtes

Sous-requête indépendante

```
SELECT CodeModule
FROM Module
WHERE Responsable =   ( SELECT Responsable
                        FROM Module
                        WHERE CodeModule = 'BD_L1')
```

Module

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test logiciel	...	7914
SR_L3	Systèmes et réseaux	...	22223
MCOO_L3	Modélisation objet	...	7914

Sous-requêtes

Sous-requête indépendante

SELECT CodeModule
FROM Module
WHERE Responsable = 7914

Module

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test logiciel	...	7914
SR_L3	Systèmes et réseaux	...	22223
MCOO_L3	Modélisation objet	...	7914

Sous-requêtes

Sous-requête indépendante

```
SELECT CodeModule
FROM Module
WHERE Responsable =    7914
```

Module

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test logiciel	...	7914
SR_L3	Systèmes et réseaux	...	22223
MCOO_L3	Modélisation objet	...	7914

Résultat :

CodeModule
BD_L1
TEST_M2
MCOO_L3

Sous-requêtes

Résultat d'une sous-requête

Une sous-requête peut renvoyer deux types de résultat :

- soit un singleton = un seul t-uplet
- soit un ensemble de t-uplets

Opérateurs associés

En fonction du nombre d'enregistrements (t-uplets) renvoyés, différents opérateurs s'emploient devant la sous-requête :

- les opérateurs de comparaison "classiques" si la sous-requête renvoie un seul enregistrement
- les opérateurs ensemblistes si la sous-requête renvoie un ensemble de t-uplets

Sous-requêtes

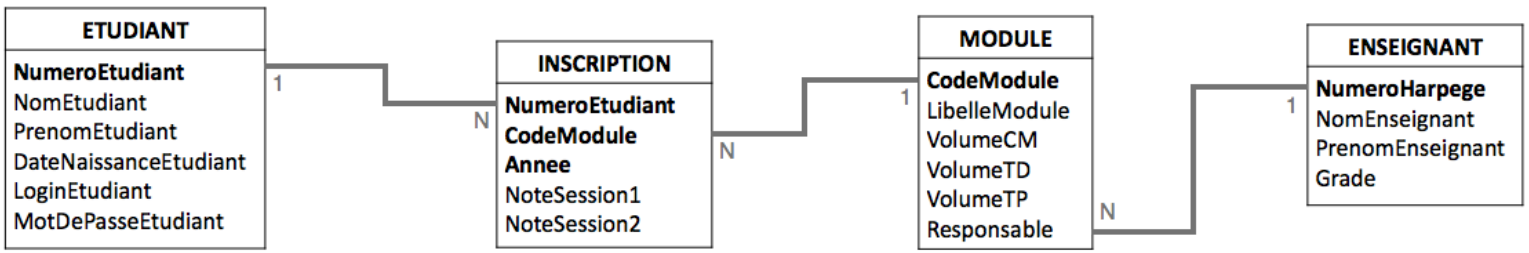
Si la sous-requête ne renvoie qu'un seul enregistrement

Dans ce cas, il est autorisé d'employer des opérateurs de comparaison tels que :

$$= \quad <> \quad < \quad <= \quad > \quad >=$$

Attention, il faut être certain que la sous-requête ne renvoie pas plus d'un enregistrement pour employer ces opérateurs.

Sous-requêtes

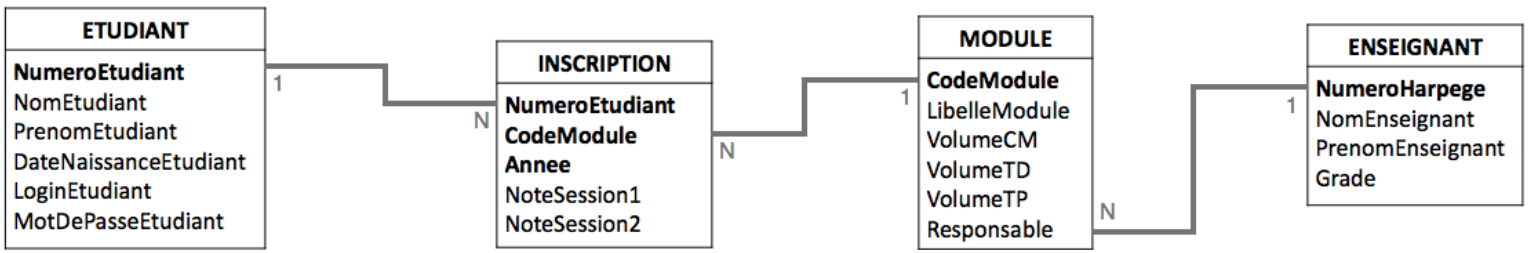


Sous-requêtes renvoyant un seul résultat

Les étudiants qui ne peuvent pas prétendre au titre de “major” du module BD_L1 en 2010.

La sous-requête renverra la note la plus haute du module BD_L1 (en session 1).
On sélectionnera les étudiants qui ont une note inférieure à celle-ci.

Sous-requêtes



Sous-requêtes renvoyant un seul résultat

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE CodeModule = 'BD_L1'
AND Annee = 2010
AND NoteSession1 <    ( SELECT MAX(NoteSession1)
                        FROM INSCRIPTION
                        WHERE CodeModule = 'BD_L1'
                        AND Annee = 2010)
```

Sous-requêtes

Si la sous-requête renvoie potentiellement plusieurs enregistrements

Si la sous-requête renvoie entre 0 et N ($N > 1$) t-uplets, on peut utiliser deux types d'opérateurs :

- l'opérateur IN
- les opérateurs ANY et ALL suivi des opérateurs de comparaison classiques :

$$= \quad <> \quad < \quad <= \quad > \quad >=$$

Ces opérateurs fonctionnent dans tous les cas, même si la sous-requête ne renvoie qu'un seul enregistrement.

Sous-requêtes

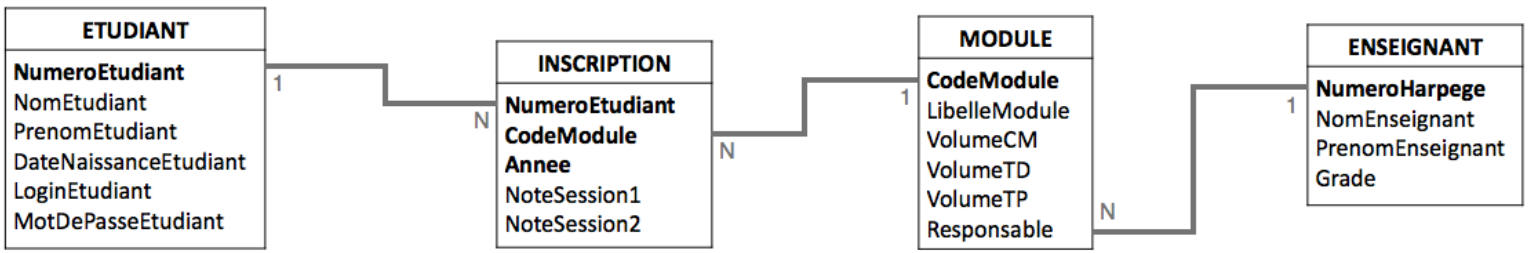
L'opérateur IN

C'est l'opérateur ensembliste d'appartenance classique, il permet de vérifier que la valeur de l'attribut spécifié dans la requête principale appartient à l'ensemble des valeurs renvoyées par la sous-requête.

Négation

Il est possible de composer avec l'opérateur NOT (sous la forme : NOT IN) pour vérifier que la valeur de l'attribut spécifié dans la requête principale n'appartient pas à l'ensemble des valeurs renvoyées par la sous-requête.

Sous-requêtes

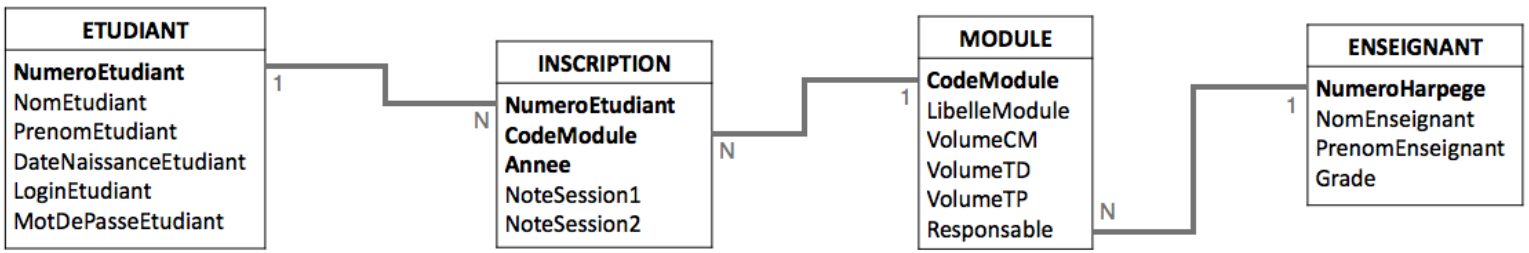


Sous-requêtes renvoyant plusieurs résultats

On veut les numéros des étudiants qui ont côtoyé l'étudiant 23794 en 2010.

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE Annee = 2010
AND CodeModule IN    ( SELECT CodeModule
                       FROM INSCRIPTION
                       WHERE Annee = 2010
                       AND NumeroEtudiant = 23794)
```

Sous-requêtes



Sous-requêtes renvoyant plusieurs résultats

Les numéro des étudiants qui n'ont jamais été en contact avec l'enseignant 7914.

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NumeroEtudiant NOT IN    ( SELECT NumeroEtudiant
                                 FROM INSCRIPTION, MODULE
                                 WHERE INSCRIPTION.CodeModule = MODULE.CodeModule
                                 AND Responsable = 7914 )
```

Sous-requêtes

Les opérateurs ANY et ALL sont associés aux opérateurs de comparaison classique.

L'opérateur ANY

Attribut *op* ANY (*Sous-Requête*)

est évalué à vrai si la condition utilisant l'opérateur *op* est vraie pour au moins un des résultats de la sous-requête.

Cas particulier : la comparaison est fausse si la sous-requête ne renvoie aucun enregistrement.

Sous-requêtes

Les opérateurs ANY et ALL sont associés aux opérateurs de comparaison classique.

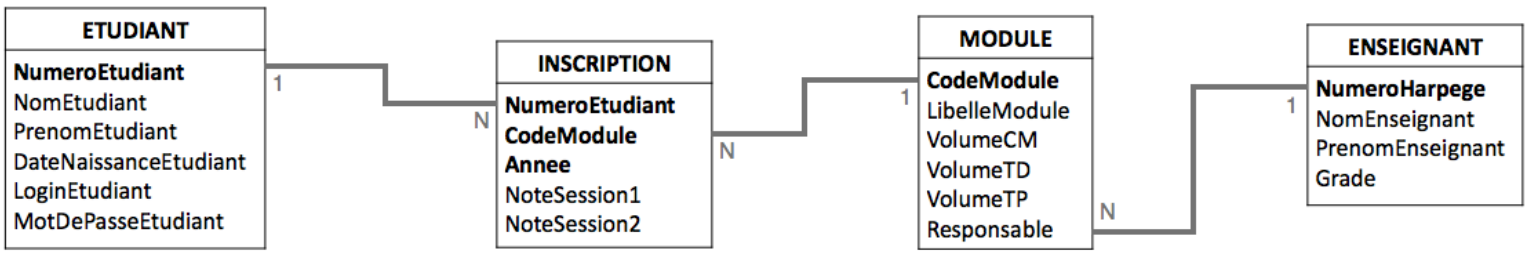
L'opérateur ALL

Attribut *op* ALL (*Sous-Requête*)

est évalué à vrai si la condition utilisant l'opérateur *op* est vraie pour tous les résultats de la sous-requête.

Cas particulier : la comparaison est vraie si la sous-requête ne renvoie aucun enregistrement.

Sous-requêtes

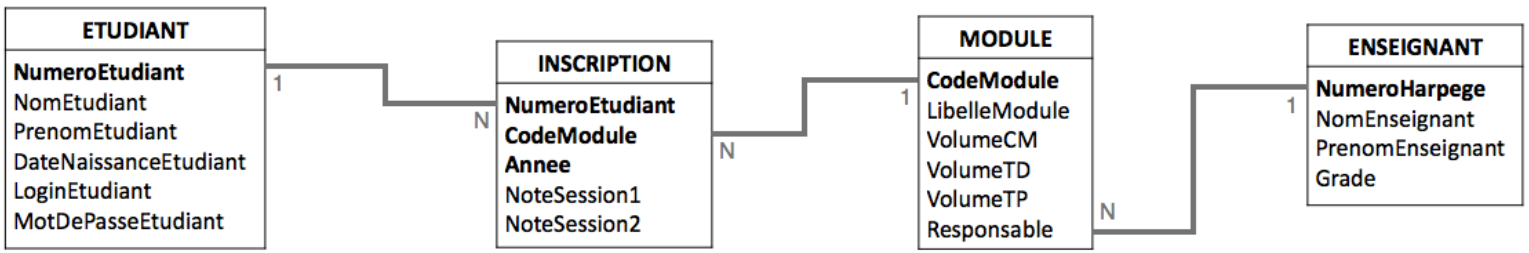


Sous-requêtes renvoyant plusieurs résultats

Les numéros des étudiants inscrits à au moins un des modules de l'enseignant 7914.

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE CodeModule = ANY    ( SELECT CodeModule
                            FROM MODULE
                            WHERE Responsable = 7914)
```

Sous-requêtes



Sous-requêtes renvoyant plusieurs résultats

L'enseignant qui totalise le plus grand nombre de responsabilités.

```
SELECT NumeroHarpege
FROM ENSEIGNANT LEFT OUTER JOIN MODULE
ON ENSEIGNANT.NumeroHarpege = MODULE.Responsable
GROUP BY NumeroHarpege
HAVING count(CodeModule) >= ALL    ( SELECT count(CodeModule)
                                     FROM MODULE
                                     GROUP BY Responsable )
```

Sous-requêtes

Définition

Une sous-requête synchronisée :

- est une sous requête qui fait référence aux attributs manipulés dans la requête principale
- nécessite la réévaluation de la sous-requête pour chaque t-uplet résultat de la requête principale

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

```
SELECT DISTINCT Responsable
FROM MODULE AS M1
WHERE M1.Responsable =    ( SELECT DISTINCT Responsable
                            FROM MODULE AS M2
                            WHERE M1.Responsable = M2.Responsable
                            AND M1.CodeModule <> M2.CodeModule )
```

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

1ère évaluation de la requête principale :

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

Evaluation de la sous-requête en remplaçant les entités issues de la requête principale :

```
SELECT DISTINCT Responsable
FROM MODULE AS M2
WHERE 7914 = M2.Responsable
AND 'BD_L1' <> M2.CodeModule
```

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

Evaluation de la sous-requête en remplaçant les entités issues de la requête principale :

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Le numéro de responsable 7914 apparaîtra dans le résultat.

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

2ème évaluation de la requête principale :

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

Evaluation de la sous-requête en remplaçant les entités issues de la requête principale :

```
SELECT DISTINCT Responsable
FROM MODULE AS M2
WHERE 7358 = M2.Responsable
AND 'PROG_L1' <> M2.CodeModule
```

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

Evaluation de la sous-requête en remplaçant les entités issues de la requête principale :

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Aucun enregistrement ne correspond.

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

3ème évaluation de la requête principale :

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

Evaluation de la sous-requête en remplaçant les entités issues de la requête principale :

```
SELECT DISTINCT Responsable
FROM MODULE AS M2
WHERE 7914 = M2.Responsable
AND 'TEST_M2' <> M2.CodeModule
```

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

Evaluation de la sous-requête en remplaçant les entités issues de la requête principale :

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Le numéro de responsable 7914 apparaîtra dans le résultat (déjà trouvé précédemment).

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

4ème évaluation de la requête principale :

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

Evaluation de la sous-requête en remplaçant les entités issues de la requête principale :

```
SELECT DISTINCT Responsable
FROM MODULE AS M2
WHERE 22223 = M2.Responsable
AND 'SR_L3' <> M2.CodeModule
```

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

Evaluation de la sous-requête en remplaçant les entités issues de la requête principale :

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test de logiciels	...	7914
SR_L3	Système et réseaux	...	22223

Aucun enregistrement ne correspond.

Sous-requêtes

Sous-requête synchronisée

Les enseignants qui sont responsables d'au moins deux modules.

```
SELECT DISTINCT Responsable
FROM MODULE AS M1
WHERE M1.Responsable =
  ( SELECT DISTINCT Responsable
    FROM MODULE AS M2
    WHERE M1.Responsable = M2.Responsable
    AND M1.CodeModule <> M2.CodeModule )
```

Responsable
7914

Sous-requêtes

Opérateur EXISTS

L'opérateur EXISTS peut être utilisé dans une clause WHERE de la manière suivante :

WHERE [NOT] EXISTS (*Sous-requête synchronisée*)

Cette expression s'évalue à vrai si la sous-requête renvoie au moins un résultat.
La sous-requête doit être synchronisée avec la requête principale.

Sous-requêtes

Utilisation de l'opérateur EXISTS

Les numéros des étudiants qui n'ont été inscrits à aucun module en 2010.

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS    (SELECT *
                     FROM INSCRIPTION
                     WHERE INSCRIPTION.Annee = 2010
                     AND INSCRIPTION.NumeroEtudiant = ETUDIANT.NumeroEtudiant)
```

Remarque : la sous-requête ne nécessite pas de projeter d'attributs spécifiques ; seul le nombre de résultats renvoyés importe.

Opérateurs ensemblistes

Utilisation des opérateurs ensemblistes

SQL propose l'utilisation des 3 opérateurs ensemblistes classiques :

- union (UNION),
- intersection (INTER),
- différence (MINUS).

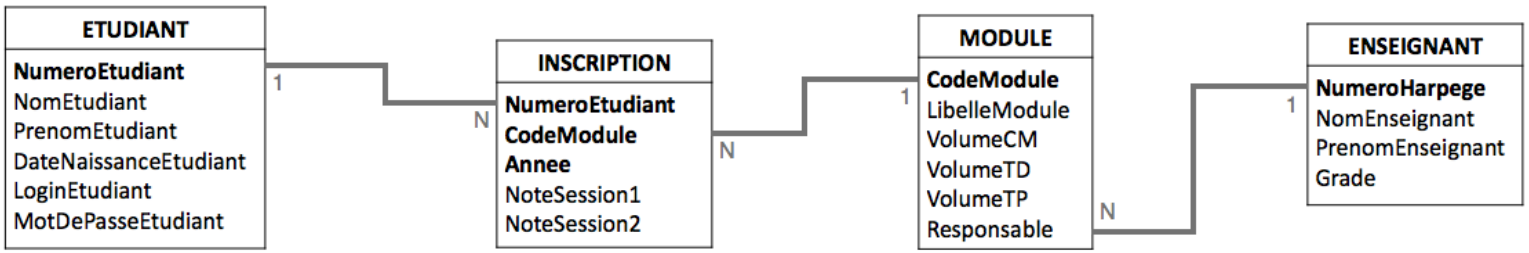
Ces opérateurs permettent de composer le résultat de deux requêtes SELECT

Syntaxe

(SELECT ... FROM ...)
Opérateur Ensembliste
(SELECT ... FROM ...)

Comme pour l'algèbre relationnelle, les deux requêtes doivent retourner la même entête.

Opérateurs ensemblistes



Exemple d'union

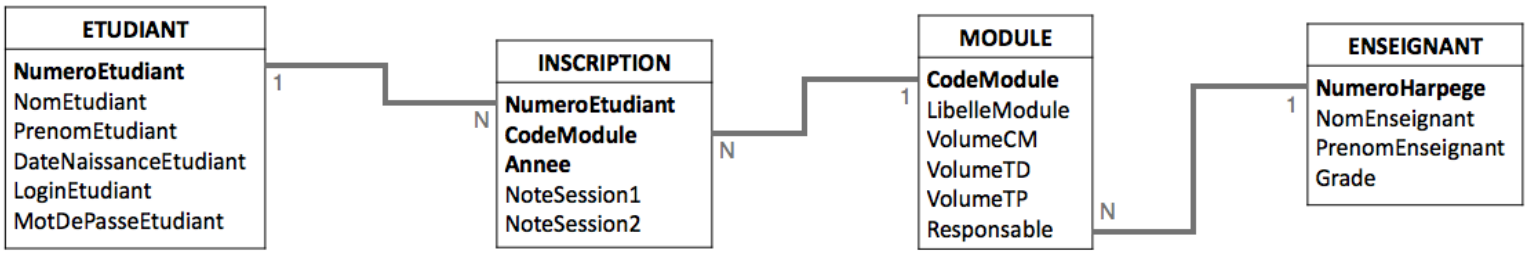
On veut les noms et prénoms des enseignants qui ont le grade MCF ou qui sont responsables d'un module qui n'a pas de TD.

```
SELECT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT
WHERE Grade = 'MCF'

UNION

SELECT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT, MODULE
WHERE ENSEIGNANT.NumeroHarpage = MODULE.Responsable
AND VolumeTD = 0
```

Opérateurs ensemblistes

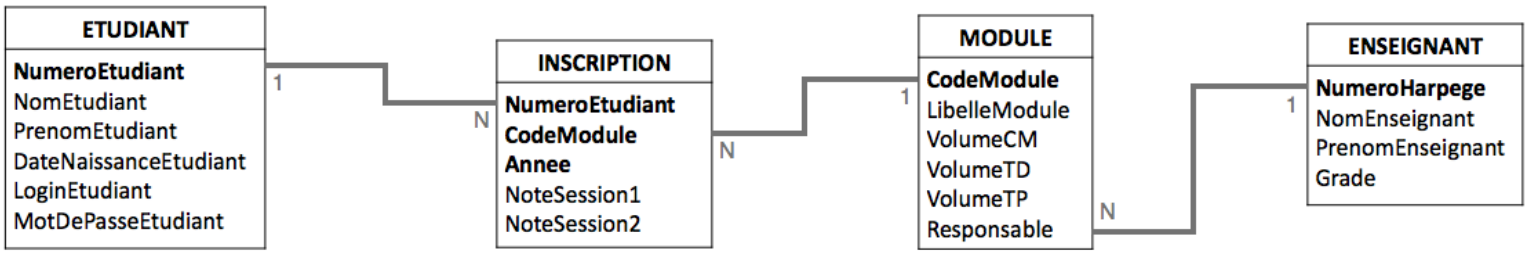


Exemple d'intersection

On veut les noms et prénoms des enseignants qui ont le grade MCF et qui sont responsables d'un module qui n'a pas de TD.

```
SELECT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT
WHERE Grade = 'MCF'
INTERSECT
SELECT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT, MODULE
WHERE ENSEIGNANT.NumeroHarpage = MODULE.Responsable
AND VolumeTD = 0
```

Opérateurs ensemblistes



Exemple de différence

On veut les noms et prénoms des enseignants qui ont le grade MCF mais qui ne sont pas responsables d'un module qui n'a pas de TD.

```
SELECT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT
WHERE Grade = 'MCF'

MINUS

SELECT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT, MODULE
WHERE ENSEIGNANT.NumeroHarpage = MODULE.Responsable
AND VolumeTD = 0
```

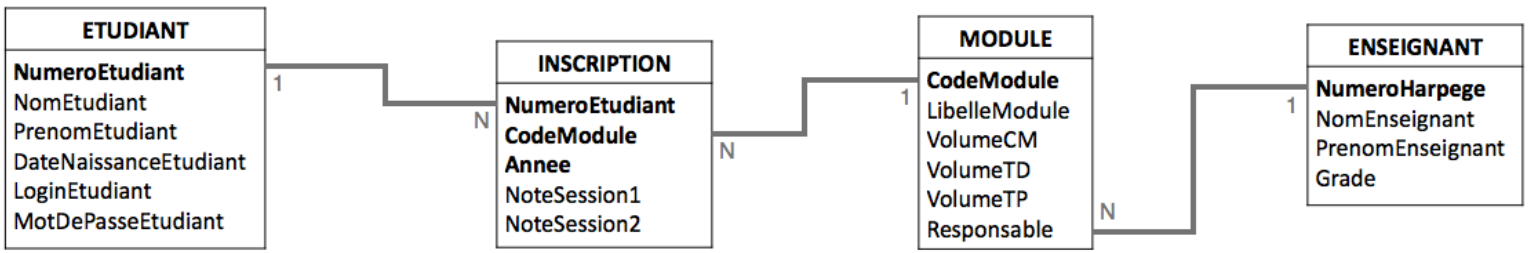
Opérateurs ensemblistes

Remarque

Tous les SGBD n'intègrent pas forcément (tous) ces opérateurs.
Par exemple, ACCESS n'autorise pas l'intersection, ni la différence.

En revanche, il est possible de réexprimer ces opérateurs ensemblistes, éventuellement à l'aide de sous-requêtes.

Opérateurs ensemblistes

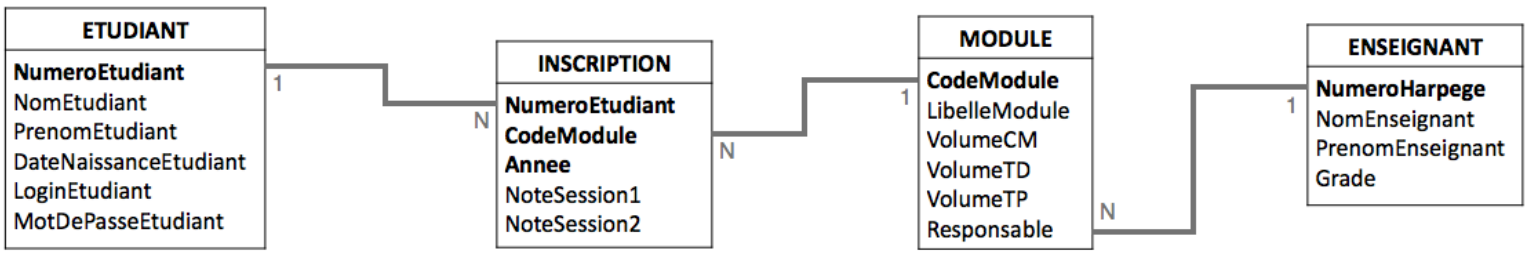


Exemple d'union

On veut les noms et prénoms des enseignants qui ont le grade MCF ou qui sont responsables d'un module qui n'a pas de TD.

```
SELECT DISTINCT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT
WHERE Grade = 'MCF' OR NumeroHarpege IN    ( SELECT Responsable
                                             FROM MODULE
                                             WHERE VolumeTD = 0)
```

Opérateurs ensemblistes

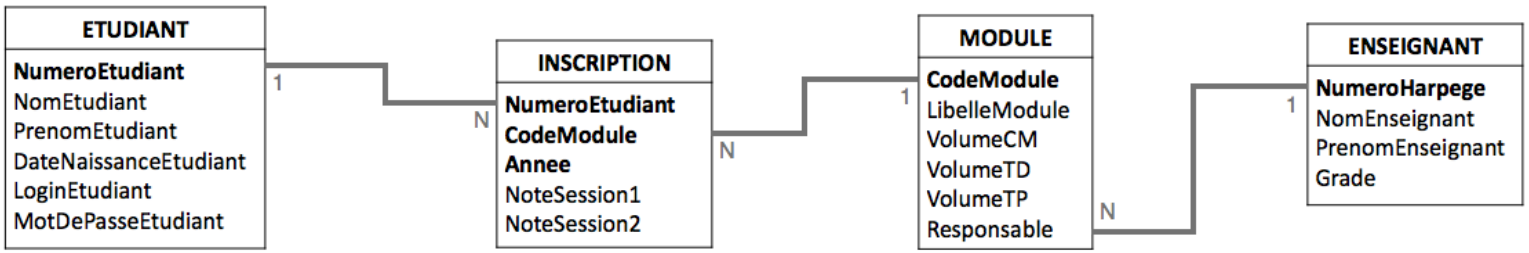


Exemple d'intersection

On veut les noms et prénoms des enseignants qui ont le grade MCF et qui sont responsables d'un module qui n'a pas de TD.

```
SELECT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT
WHERE Grade = 'MCF' AND NumeroHarpege IN    ( SELECT Responsable
                                              FROM MODULE
                                              WHERE VolumeTD = 0)
```

Opérateurs ensemblistes



Exemple de différence

On veut les noms et prénoms des enseignants qui ont le grade MCF mais qui ne sont pas responsables d'un module qui n'a pas de TD.

```
SELECT NomEnseignant, PrenomEnseignant
FROM ENSEIGNANT
WHERE Grade = 'MCF' AND NumeroHarpege NOT IN    ( SELECT Responsable
                                                  FROM MODULE
                                                  WHERE VolumeTD = 0)
```

Division

Rappel : le principe de la division

Description avec l'opérateur relationnel :

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

Toute valeur t de l'attribut At_1 appartenant à t est telle que : pour toute valeur y de At_2 dans Y, le couple (t, y) appartient à la relation X.

Division

Rappel : le principe de la division

X

At_1	At_2
a	x
a	y
a	z
b	x
b	y
b	t
c	x

Y

At_2
x
y

Division

Rappel : le principe de la division

X

At_1	At_2
a	x
a	y
a	z
b	x
b	y
b	t
c	x

Y

At_2
x
y

T

At_1
a
b

Division

Comment exprimer la division en SQL ?

Il y a deux solutions possibles :

- avec la fonction COUNT

  $\Rightarrow$ Solution assez intuitive, mais qui ne s'applique pas toujours.

- avec la clause (NOT) EXISTS

  $\Rightarrow$ Solution plus complexe, mais qui s'applique dans tous les cas.

Division

Principe de la division avec COUNT

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

t appartient à T si le nombre de valeurs distinctes de At_2 apparaissant avec t dans X est égal au nombre de valeurs de At_2 dans Y.

Division avec COUNT

```
SELECT At1
FROM X
GROUP BY At1
HAVING COUNT(DISTINCT At2) =    (SELECT COUNT(*)
                                 FROM Y)
```

Division

Principe de la division avec COUNT

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

t appartient à T si le nombre de valeurs distinctes de At_2 apparaissant avec t dans X est égal au nombre de valeurs de At_2 dans Y.

Division avec COUNT

On souhaite connaître les étudiants qui ont été inscrits à tous les modules en 2010.

R1 = S (Annee = 2010) INSCRIPTION
R2 = [NumeroEtudiant, CodeModule] R1
R3 = [CodeModule] MODULE
R4 = R2 / R3

Division

Principe de la division avec COUNT

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

t appartient à T si le nombre de valeurs distinctes de At_2 apparaissant avec t dans X est égal au nombre de valeurs de At_2 dans Y.

Division avec COUNT

On souhaite connaître les étudiants qui ont été inscrits à tous les modules en 2010.

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE Annee = 2010
GROUP BY NumeroEtudiant
HAVING COUNT(DISTINCT CodeModule) =    (SELECT COUNT(*)
                                        FROM MODULE)
```

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE Annee = 2010
GROUP BY NumeroEtudiant
HAVING COUNT(DISTINCT CodeModule) = (SELECT COUNT(*) FROM MODULE)
```

INSCRIPTION

NumeroEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	LibelleModule	...
BD_L1	Bases de données	...
PROG_L1	Programmation	...
TEST_M2	Test de logiciels	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM INSCRIPTION
WHERE Annee = 2010
GROUP BY NumeroEtudiant
HAVING COUNT(DISTINCT CodeModule) = 3
```

INSCRIPTION

NumeroEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	LibelleModule	...
BD_L1	Bases de données	...
PROG_L1	Programmation	...
TEST_M2	Test de logiciels	...

Division

Condition d'utilisation de la solution avec COUNT

Les valeurs de l'attribut At_2 apparaissant dans la relation X ne peuvent pas prendre des valeurs différentes de celles qui apparaissent dans la relation Y.

Un exemple de division qui utilise COUNT

On souhaite connaître les étudiants qui ont été inscrits à tous les modules en 2010.

R1 = S (Annee = 2010) INSCRIPTION
R2 = [NumeroEtudiant, CodeModule] R1
R3 = [CodeModule] MODULE
R4 = R2 / R3

Les valeurs de *CodeModule* dans R2 ne peuvent pas prendre de valeurs différentes des *CodeModule* qui sont dans R3. De plus, on ne pourra pas avoir deux valeurs identiques de *CodeModule* dans R2 $\Rightarrow$ le compte ne sera donc pas faussé.

Division

Condition d'utilisation de la solution avec COUNT

Les valeurs de l'attribut At_2 apparaissant dans la relation X ne peuvent pas prendre des valeurs différentes de celles qui apparaissent dans la relation Y.

Un exemple de division qui ne peut pas utiliser COUNT

X

At_1	At_2
a	x
a	y
a	z
b	x
b	y
b	t
c	x

Y

At_2
x
y

La solution avec COUNT ne fonctionne pas car, dans X, At_2 peut prendre d'autres valeurs que celles issues de Y.

Division

Principe de la division avec EXISTS

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

Quelque soit la valeur At_1 de T, et quelque soit At_2 de Y, (At_1, At_2) appartient à X.

Division

Principe de la division avec EXISTS

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

Quelque soit la valeur At_1 de T, et quelque soit At_2 de Y, (At_1, At_2) appartient à X.

$$\forall At_1 \in T, \forall At_2 \in Y \Rightarrow (At_1, At_2) \in X$$

Division

Principe de la division avec EXISTS

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

Quelque soit la valeur At_1 de T, et quelque soit At_2 de Y, (At_1, At_2) appartient à X.

$$\forall At_1 \in T, \forall At_2 \in Y \quad \Rightarrow \quad (At_1, At_2) \in X$$

$$\Leftrightarrow$$

$$\forall At_1 \in T, \nexists At_2 \in Y . (At_1, At_2) \notin X$$

Division

Principe de la division avec EXISTS

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

Quelque soit la valeur At_1 de T, et quelque soit At_2 de Y, (At_1, At_2) appartient à X.

$$\forall At_1 \in T, \forall At_2 \in Y \quad \Rightarrow \quad (At_1, At_2) \in X$$

$$\Leftrightarrow$$

$$\forall At_1 \in T, \nexists At_2 \in Y . (At_1, At_2) \notin X$$

$$\Leftrightarrow$$

$$\forall At_1 \in T, \nexists At_2 \in Y . \nexists (At_1, At_2) \in X$$

Division

La formule magique

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

$$\forall At_1 \in T, \nexists At_2 \in Y.\ \nexists (At_1, At_2) \in X$$

Division

La formule magique

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

```
SELECT At1
FROM X AS X1
WHERE NOT EXISTS   (SELECT At2
                    FROM Y
                    WHERE NOT EXISTS   (SELECT *
                                        FROM X AS X2
                                        WHERE X2.At1 = X1.At1
                                        AND X2.At2 = Y.At2))
```

Division

La formule magique

$$X(At_1, At_2) / Y(At_2) \rightarrow T(At_1)$$

```
SELECT At1
FROM X AS X1
WHERE NOT EXISTS   (SELECT At2
                    FROM Y
                    WHERE NOT EXISTS   (SELECT *
                                        FROM X AS X2
                                        WHERE X2.At1 = X1.At1
                                        AND X2.At2 = Y.At2))
```

La dernière sous-requète doit utiliser une source de données contenant les attributs At_1 et At_2 utilisés pour réaliser la synchronisation.

Division

Exemple de division avec la clause EXISTS

On souhaite connaître les étudiants qui ont été inscrits à tous les modules en 2010.

$\Leftrightarrow$ on veut donc connaître les étudiants pour lesquels il n'existe pas de module pour auquel ces étudiants n'étaient pas inscrits en 2010.

Division

Exemple de division avec la clause EXISTS

On souhaite connaître les étudiants qui ont été inscrits à tous les modules en 2010.

```
SELECT NumeroEtudiant
FROM INSCRIPTION AS I1
WHERE NOT EXISTS
            (SELECT CodeModule
             FROM MODULE
             WHERE NOT EXISTS
                        (SELECT *
                         FROM INSCRIPTION AS I2
                         WHERE I1.NumeroEtudiant = I2.NumeroEtudiant
                         AND I2.CodeModule = MODULE.CodeModule
                         AND I2.Annee = 2010))
```

Division

Exemple de division avec la clause EXISTS

On souhaite connaître les étudiants qui ont été inscrits à tous les modules en 2010.

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
            (SELECT CodeModule
             FROM MODULE
             WHERE NOT EXISTS
                        (SELECT *
                         FROM INSCRIPTION AS I
                         WHERE ETUDIANT.NumeroEtudiant = I.NumeroEtudiant
                         AND I.CodeModule = MODULE.CodeModule
                         AND I.Annee = 2010))
```

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT CodeModule
                 FROM MODULE
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I
                                 WHERE ETUDIANT.NumeroEtudiant = I.NumeroEtudiant
                                 AND I.CodeModule = MODULE.CodeModule
                                 AND I.Annee = 2010))
```

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT CodeModule
                 FROM MODULE
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I
                                 WHERE 23794 = I.NumeroEtudiant
                                 AND I.CodeModule = MODULE.CodeModule
                                 AND I.Annee = 2010))
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
            (SELECT CodeModule
             FROM MODULE
             WHERE NOT EXISTS
                        (SELECT *
                         FROM INSCRIPTION AS I
                         WHERE 23794 = I.NumeroEtudiant
                         AND I.CodeModule = 'BD_L1'
                         AND I.Annee = 2010))
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT CodeModule
                 FROM MODULE
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I
                                 WHERE 23794 = I.NumeroEtudiant
                                 AND I.CodeModule = 'PROG_L1'
                                 AND I.Annee = 2010))
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT CodeModule
                 FROM MODULE
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I
                                 WHERE 23794 = I.NumeroEtudiant
                                 AND I.CodeModule = 'TEST_M2'
                                 AND I.Annee = 2010))
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                ('TEST_M2')
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE FALSE
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT CodeModule
                 FROM MODULE
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I
                                 WHERE 32911 = I.NumeroEtudiant
                                 AND I.CodeModule = MODULE.CodeModule
                                 AND I.Annee = 2010))
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT CodeModule
                 FROM MODULE
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I
                                 WHERE 32911 = I.NumeroEtudiant
                                 AND I.CodeModule = 'BD_L1'
                                 AND I.Annee = 2010))
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT CodeModule
                 FROM MODULE
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I
                                 WHERE 32911 = I.NumeroEtudiant
                                 AND I.CodeModule = 'PROG_L1'
                                 AND I.Annee = 2010))
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT CodeModule
                 FROM MODULE
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I
                                 WHERE 32911 = I.NumeroEtudiant
                                 AND I.CodeModule = 'TEST_M2'
                                 AND I.Annee = 2010))
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                ( )
```

RESULTAT

NřEtudiant*

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE TRUE
```

RESULTAT

NřEtudiant*
32911

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                ('PROG_L1, TEST_M2')
```

RESULTAT

NřEtudiant*
32911

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE FALSE
```

RESULTAT

NřEtudiant*
32911

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                ( )
```

RESULTAT

NřEtudiant*
32911

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

Division

Evaluation de la division

```
SELECT NumeroEtudiant
FROM ETUDIANT
WHERE TRUE
```

RESULTAT

NřEtudiant*
32911
34812

ETUDIANT

NřEtudiant*	...
23794	...
32911	...
33818	...
34812	...

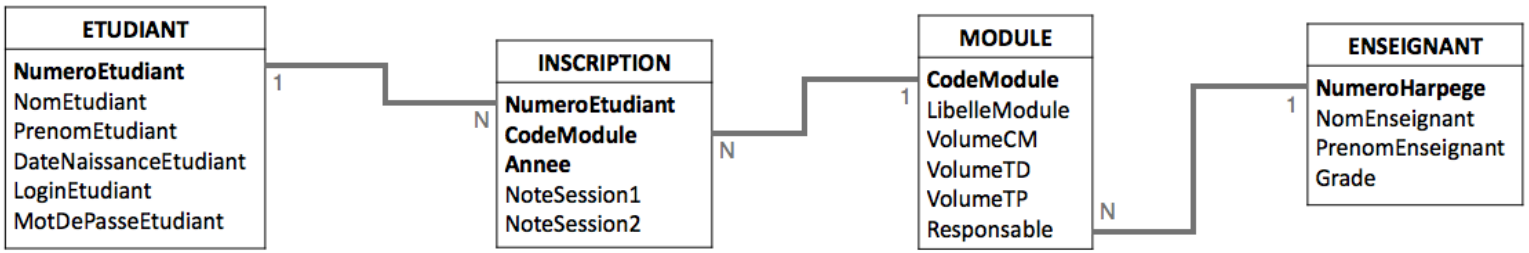
I = INSCRIPTION

NřEtudiant*	CodeModule*	Annee	...
23794	BD_L1	2010	...
23794	PROG_L1	2010	...
32911	BD_L1	2010	...
32911	PROG_L1	2010	...
32911	TEST_M2	2010	...
33818	BD_L1	2010	...
34812	PROG_L1	2010	...
34812	BD_L1	2010	...
34812	TEST_M2	2010	...

MODULE

CodeModule*	...
BD_L1	...
PROG_L1	...
TEST_M2	...

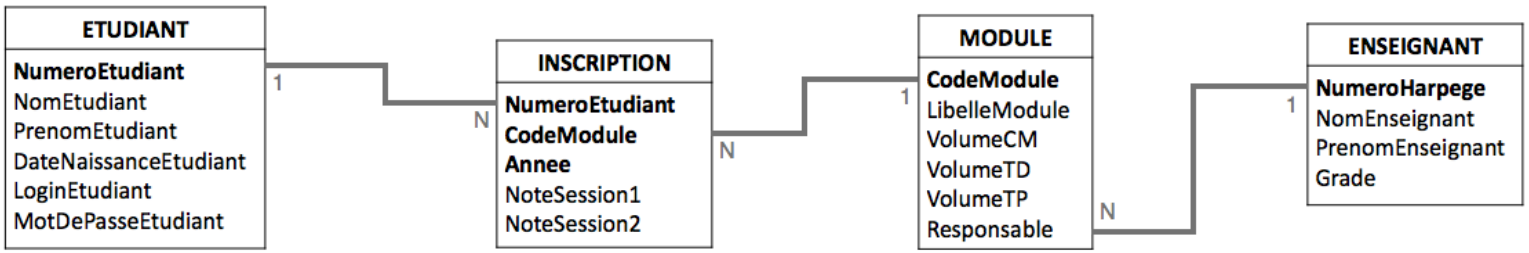
Division



Donner les 2 solutions (COUNT et EXISTS) pour les requètes suivantes :

- Libellés des modules où tous les étudiants étaient inscrits en 2010
- Noms et prénom des étudiants qui ont été inscrits toutes les années entre 2005 et 2010

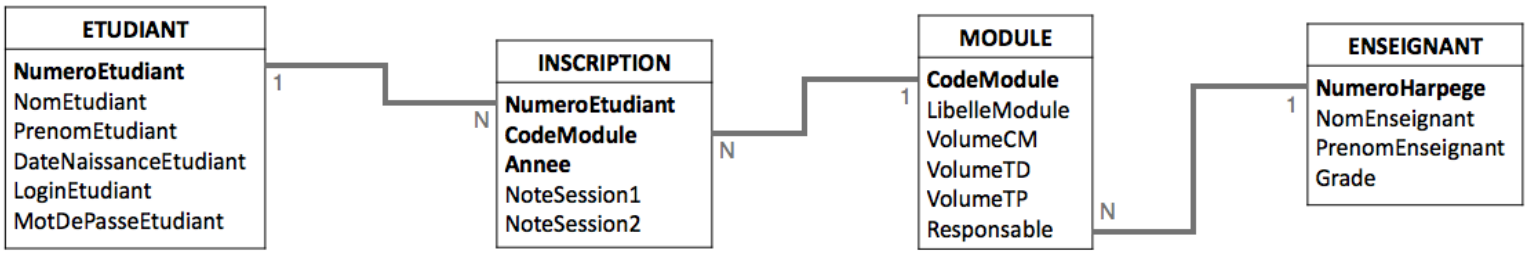
Division



Libellés des modules où tous les étudiants étaient inscrits en 2010

```
SELECT LibelleModule
FROM MODULE, INSCRIPTION
WHERE MODULE.CodeModule = INSCRIPTION.CodeModule
AND Annee = 2010
GROUP BY CodeModule, LibelleModule
HAVING COUNT (DISTINCT NumeroEtudiant) =    (SELECT COUNT(*)
                                             FROM ETUDIANT)
```

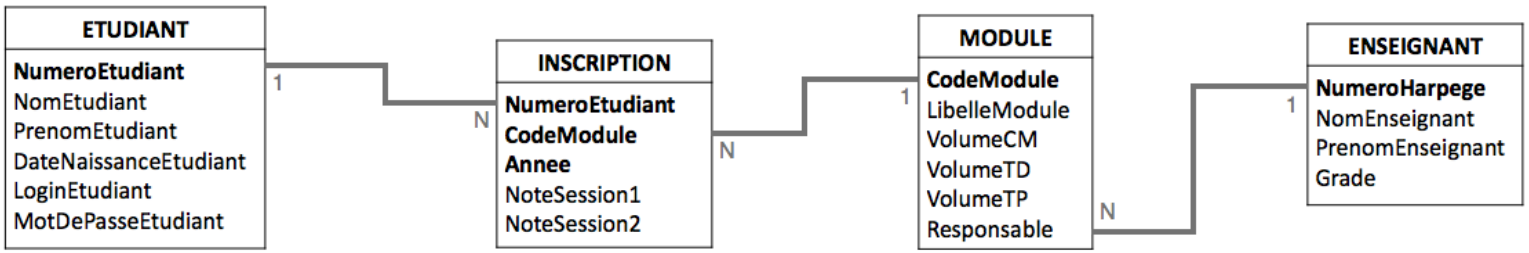
Division



Libellés des modules où tous les étudiants étaient inscrits en 2010

```
SELECT LibelleModule
FROM MODULE
WHERE NOT EXISTS
                (SELECT NumeroEtudiant
                 FROM ETUDIANT
                 WHERE NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION
                                 WHERE INSCRIPTION.CodeModule = MODULE.CodeModule
                                 AND INSCRIPTION.NumeroEtudiant = ETUDIANT.NumeroEtudiant
                                 AND Annee = 2010))
```

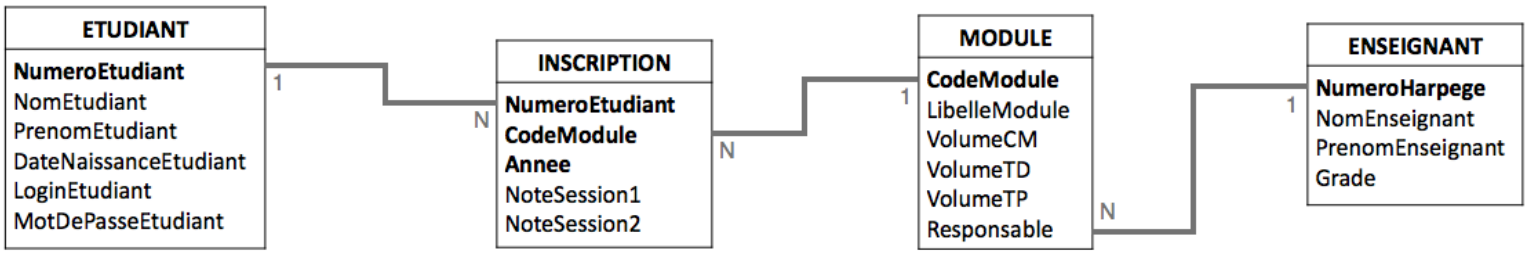
Division



Noms et prénom des étudiants qui ont été inscrits toutes les années entre 2005 et 2010

```
SELECT NomEtudiant, PrenomEtudiant
FROM ETUDIANT, INSCRIPTION
WHERE ETUDIANT.NumeroEtudiant = INSCRIPTION.NumeroEtudiant
GROUP BY ETUDIANT.NumeroEtudiant, NomEtudiant, PrenomEtudiant
HAVING COUNT(DISTINCT) Annee =  (SELECT COUNT(DISTINCT Annee)
                                 FROM INSCRIPTION
                                 WHERE Annee BETWEEN 2005 AND 2010)
```

Division



Noms et prénom des étudiants qui ont été inscrits toutes les années entre 2005 et 2010

```
SELECT NomEtudiant, PrenomEtudiant
FROM ETUDIANT
WHERE NOT EXISTS
                (SELECT Annee
                 FROM INSCRIPTION AS I1
                 WHERE Annee BETWEEN 2005 AND 2010
                 AND NOT EXISTS
                                (SELECT *
                                 FROM INSCRIPTION AS I2
                                 WHERE I2.Annee = I1.Annee
                                 AND I2.NumeroEtudiant = ETUDIANT.NumeroEtudiant))
```

Fonctions et expressions

Deux types de calcul

On distingue deux types de calculs :

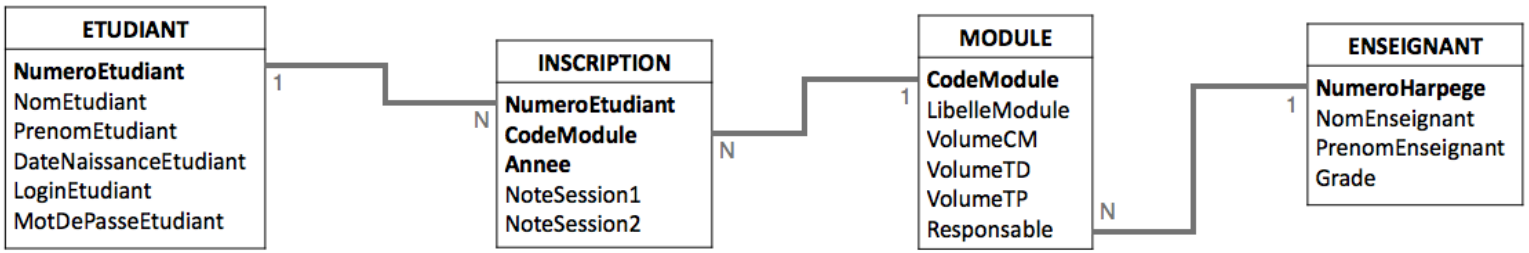
- Les fonctions d'agrégation

  $\Rightarrow$ permettent de réaliser des calculs "verticaux" sur un attribut, utilisant un ensemble d'enregistrements

- Les **expressions et les fonctions**

  $\Rightarrow$ permettent de réaliser des calculs "horizontaux", sur un seul enregistrement, utilisant plusieurs attributs

Fonctions et expressions



Moyenne du temps passé dans un module par semaine

Sachant qu'il y a 14 semaines dans un semestre, donnez le temps moyen passé dans chaque module par semaine.

```
SELECT LibelleModule, (VolumeCM + VolumeTD + VolumeTP) / 14 AS TempsMoyen
FROM MODULE
```

MODULE

Code*	LibelleModule	CM	TD	TP	...
BD_L1	Bases de données	18	21	21	...
PROG_L1	Programmation	15	21	24	...
ANG_L1	Anglais	0	18	0	..

RESULTAT

LibelleModule	TempsMoyen
Bases de données	4.28
Programmation	4.28
Anglais	1.28

Fonctions et expressions

Où utiliser ces fonctions et expressions ?

Les fonctions et les expressions peuvent être utilisées dans certaines clauses :

- SELECT
- WHERE
- GROUP BY
- ORDER

Fonctions et expressions

Quelques exemples

Liste des modules ayant un volume horaire supérieur à 40 heures par semestre.

```
SELECT CodeModule, LibelleModule
FROM MODULE
WHERE VolumeCM + VolumeTD + VolumeTP > 40
```

MODULE

Code*	LibelleModule	CM	TD	TP	...
BD_L1	Bases de données	18	21	21	...
PROG_L1	Programmation	15	21	24	...
ANG_L1	Anglais	0	18	0	..

RESULTAT

Code	LibelleModule
BD_L1	Bases de données
PROG_L1	Programmation

Fonctions et expressions

Quelques exemples

Tri des modules par ordre décroissant de volume horaire par semestre.

```
SELECT CodeModule, LibelleModule
FROM MODULE
ORDER BY VolumeCM + VolumeTD + VolumeTP
```

MODULE

Code*	LibelleModule	CM	TD	TP	...
BD_L1	Bases de données	18	21	21	...
PROG_L1	Programmation	15	21	24	...
ANG_L1	Anglais	0	18	0	..

RESULTAT

Code	LibelleModule
BD_L1	Bases de données
PROG_L1	Programmation
ANG_L1	Anglais

Fonctions et expressions

On compte 4 grands types de fonctions, qui dépendent beaucoup du SGBD.

Fonctions arithmétiques

- fonctions mathématiques : valeur absolue, puissance, plusgrand, etc.
- fonctions financières : calcul d'amortissement, etc.

Fonctions et expressions

On compte 4 grands types de fonctions, qui dépendent beaucoup du SGBD.

Fonctions sur les chaînes

- calcul de la longueur d'une chaîne
- transformation d'une chaîne en majuscules ou en minuscules
- extraction d'une sous-chaîne
- etc.

Fonctions et expressions

On compte 4 grands types de fonctions, qui dépendent beaucoup du SGBD.

Les expressions/fonctions sur les dates

- addition/soustraction d'une date/d'un nombre
- fonctions retournant la date du système, le jour de la semaine, etc.

Fonctions et expressions

On compte 4 grands types de fonctions, qui dépendent beaucoup du SGBD.

Les fonctions de conversion

Ces fonctions permettent de passer d'un type à l'autre

- pour effectuer des calculs
- pour permettre des affichages

Plan du cours

Data Manipulation Language

Data Manipulation Language

Jusqu'à présent, les requêtes effectuées permettaient d'extraire des informations de la base de données (SELECT).

DML représente le sous-ensemble de SQL qui permet :

- d'ajouter des enregistrements,
- de supprimer des enregistrements,
- de mettre à jour (i.e. modifier) des enregistrements.

Ajout de données

Syntaxe générale

L'ajout de données, ou insertion, se décrit en SQL de la manière suivante :

INSERT INTO *table* [(Att_1, Att_2, ...)]
VALUES ($Val1_1$, $Val2_1$, ...), ($Val1_2$, $Val2_2$, ...), ...

Liste d'attributs

- La liste des attributs mentionnée après le nom de la table est facultative (c'est ce qu'indiquent les crochets [ ]).
- Elle permet de restreindre les attributs qui seront renseignés dans le nouvel enregistrement.
- Tous les attributs non renseignés prendront soit leur valeur par défaut (si elle a été définie), soit NULL.
- Si cette liste est absente, par défaut, tous les attributs de la table sont considérés dans l'ordre de la création de la table.

Ajout de données

Syntaxe générale

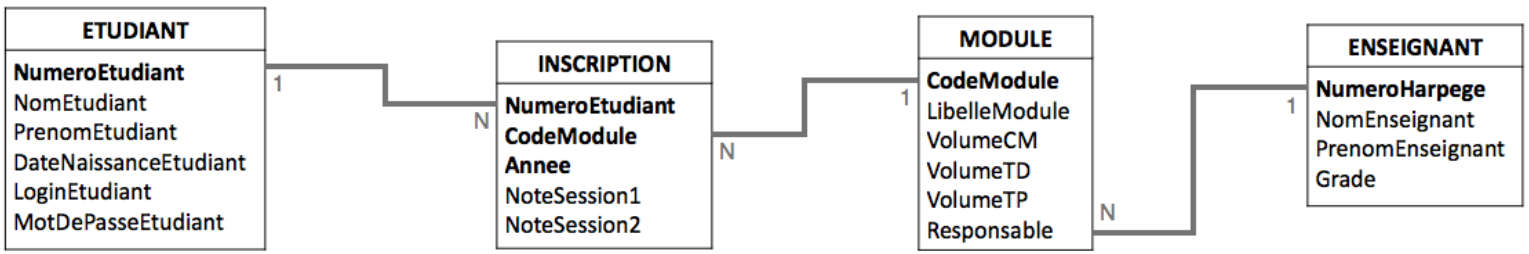
L'ajout de données, ou insertion, se décrit en SQL de la manière suivante :

INSERT INTO *table* [(Att_1, Att_2, ...)]
VALUES ($Val1_1$, $Val2_1$, ...), ($Val1_2$, $Val2_2$, ...), ...

Liste de valeurs

- La liste de valeurs est obligatoire et doit être délimitée par des parenthèses.
- Les valeurs sont déclarées dans le même ordre que les attributs auxquels ils se rapportent (par exemple, $Val1_N$ correspond à Att_1).
- Chaque valeur doit appartenir au domaine de l'attribut auquel il se rapporte.
- Pour insérer plusieurs enregistrements en une seule requête, on sépare les listes de valeurs par des virgules.

Ajout de données



Exemple d'insertion

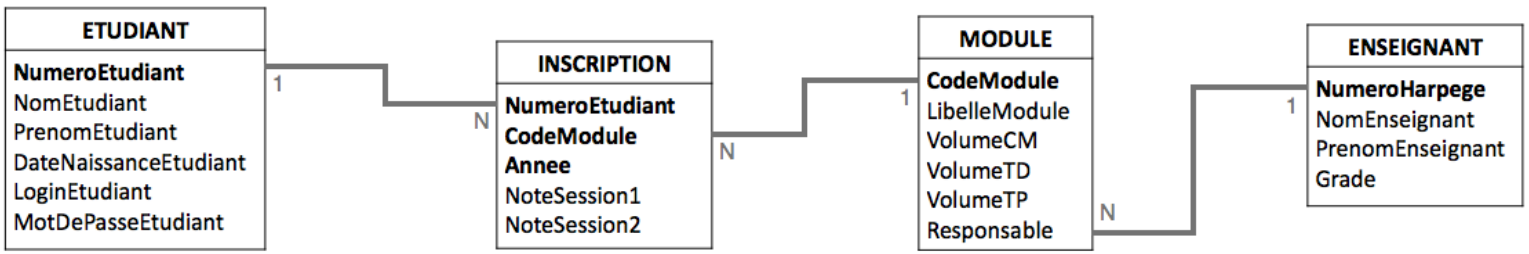
Ajout dans la table ENSEIGNANT :

ENSEIGNANT

NoHarpege*	Nom	Prenom	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF

INSERT INTO ENSEIGNANT (NumeroHarpege, NomEnseignant, PrenomEnseignant)
VALUES (32598, 'Paquette', 'Guillaume')

Ajout de données



Exemple d'insertion

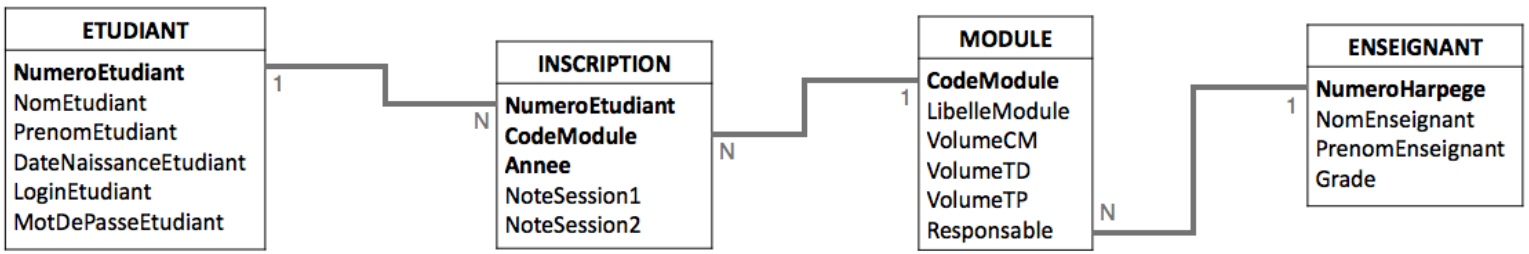
Ajout dans la table ENSEIGNANT :

ENSEIGNANT

NoHarpege*	Nom	Prenom	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF
32598	Paquette	Guillaume	

INSERT INTO ENSEIGNANT (NumeroHarpege, NomEnseignant, PrenomEnseignant)
VALUES (32598, 'Paquette', 'Guillaume')

Ajout de données



Exemple d'insertion

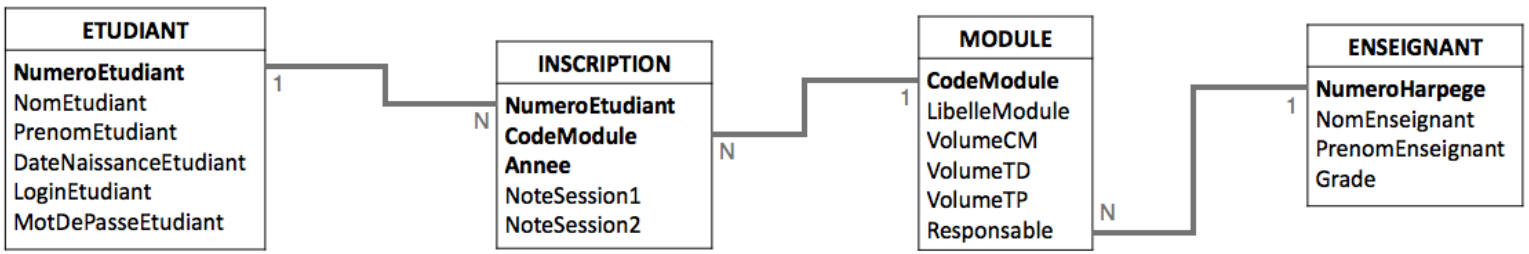
Ajout dans la table ENSEIGNANT :

ENSEIGNANT

NoHarpege*	Nom	Prenom	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF
32598	Paquette	Guillaume	

INSERT INTO ENSEIGNANT
VALUES (1797, 'Damy', 'Sylvie', 'MCF')

Ajout de données



Exemple d'insertion

Ajout dans la table ENSEIGNANT :

ENSEIGNANT

NoHarpege*	Nom	Prenom	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF
32598	Paquette	Guillaume	
1797	Damy	Sylvie	MCF

```
INSERT INTO ENSEIGNANT
VALUES (1797, 'Damy', 'Sylvie', 'MCF')
```

Ajout de données

Autre syntaxe

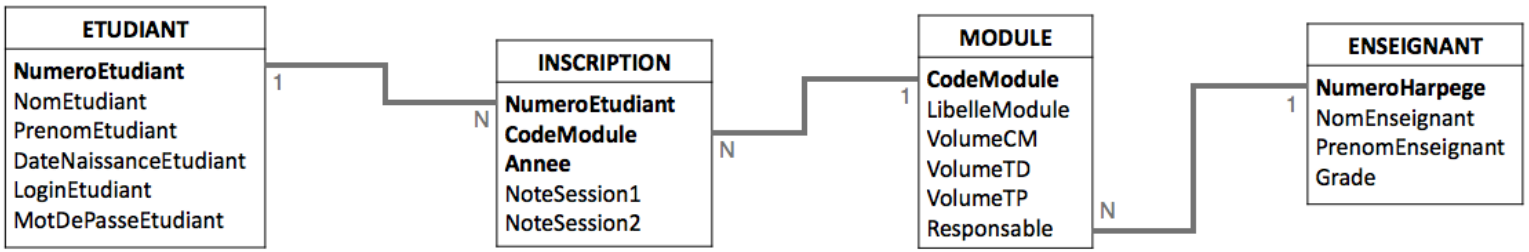
Il est possible d'insérer plusieurs enregistrements qui sont le résultat d'une requête SELECT.

INSERT INTO *table* [(Att_1, Att_2, ...)]
(SELECT Att_1, Att_2 ...)

Quelques remarques

- La requête SELECT devra bien évidemment renvoyer des attributs "compatibles" et dans le même ordre que les attributs de la table dans laquelle on insère un t-uplet.
- La requête SELECT ne pourra pas contenir de clause ORDER BY.

Ajout de données



Exemple d'insertion

Inscription automatique de tous les étudiants dans les modules qu'ils n'ont pas validé en 2010 l'année suivante :

```
INSERT INTO INSCRIPTION (NumeroEtudiant, CodeModule, Annee)
SELECT NumeroEtudiant, CodeModule, Annee + 1
FROM MODULE
WHERE Annee = 2010
AND NoteSession2 IS NOT NULL
AND NoteSession2 < 10
```

Modification de données

Syntaxe générale

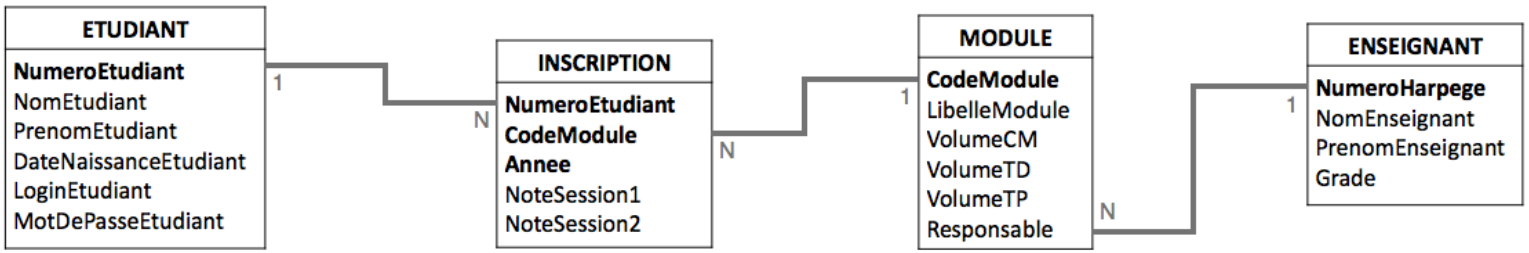
La modification de données (ou mise à jour) se décrit en SQL de la manière suivante :

UPDATE *table*
SET $Att_1 = Val_1$, $Att_2 = Val_2$, ... WHERE *condition*

Sémantique

- Elle met à jour les t-uplet de la table mentionnée qui satisfont la condition de la clause WHERE
- Elle ne concerne que les attributs listés dans la clause SET

Modification de données



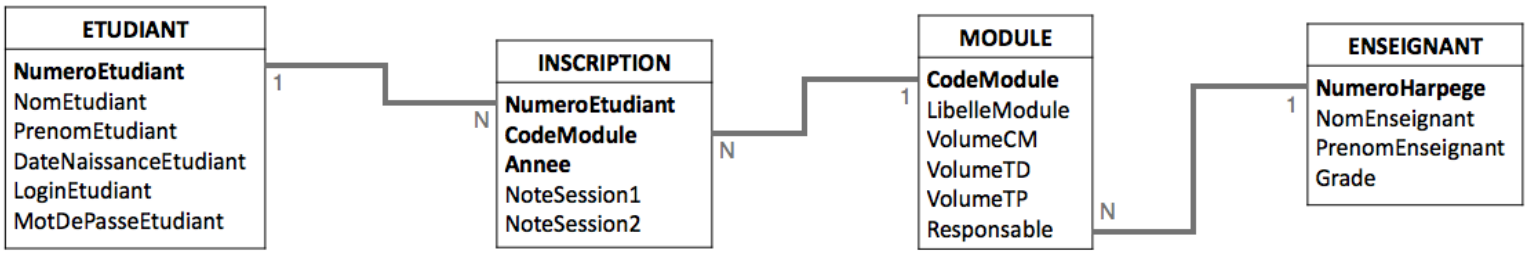
Exemple de mise à jour

L'étudiant 23794 met à jour son mot de passe pour le remplacer par 'azerty'.

```
UPDATE ETUDIANT
SET MotDePasseEtudiant = 'azerty'
WHERE NumeroEtudiant = 23794
```

$\Rightarrow$ un seul t-uplet sera modifé.

Modification de données



Exemple de mise à jour

L'enseignant 7914 en a assez des étudiants qui viennent pas en cours, il se met en disponibilité et donne la responsabilité de ses cours à l'enseignant 32598.

```
UPDATE MODULE
SET Responsable = 32598
WHERE Responsable = 7914
```

$\Rightarrow$ potentiellement, plusieurs t-uplets seront modifés.

Suppression de données

Syntaxe générale

La suppression de données se décrit en SQL de la manière suivante :

DELETE FROM *table*
WHERE *condition*

Sémantique

- Supprime les t-uplet de la table mentionnée qui satisfont la condition de la clause WHERE

Une remarque sur les requêtes de type DML

Attention aux règles de cohérence de la base de données

Les requêtes INSERT, UPDATE, DELETE qui ne respectent pas l'intégrité référentielles, par exemple :

- INSERT d'une valeur déjà existante pour une clé primaire,
- UPDATE d'un attribut clé primaire et porte un lien avec une politique de mise à jour RESTRICTED et référencé dans la table liée,
- DELETE d'un enregistrement dont la clé primaire porte un lien avec une politique de suppression RESTRICTED et référencée dans la table liée

vont toujours produire des erreurs renvoyées par le SGBD !

Plan du cours

Data Definition Language

Data Definition Language

Nous avons vu qu'à maintenant comment manipuler (au sens large du terme) les données : sélectionner des enregistrements (SELECT), ajouter des enregistrements (INSERT), modifier (UPDATE) ou supprimer (DELETE).

Nous allons désormais voir comment construire le modèle de la base de données au travers des 3 entités :

- les tables
- les index
- les vues

que nous allons pouvoir créer (CREATE), modifier (ALTER), ou supprimer (DROP).

Création d'une table

Création d'une table

La création d'une table consiste à décrire :

- les attributs de la table
- les contraintes d'intégrité concernant la table

Création d'une table

Création d'une table

La syntaxe "basique" pour la création d'une table est la suivante :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1*, *NomAttribut2 Type2*, ...)

Les types dépendent en général du SGBD utilisé, mais certains sont immuables : INTEGER, CHAR(longueur), DATE, TIME, ENUM(v_1, v_2, ...)

Exemple de création de table

Ajoutons à notre exemple fil rouge une table INTERVENANT listant les enseignants intervenants dans un module.

CREATE TABLE INTERVENANT
(CodeModule CHAR(5),
Enseignant INTEGER)

Création d'une table

Création et remplissage d'une table

La syntaxe pour la création et le remplissage d'une table est la suivante :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1*, *NomAttribut2 Type2*, ...) AS SELECT ...

Cette instruction va créer une table et l'initialiser avec les résultats de la requête SELECT.

Quelques remarques sur cette syntaxe

- Par défaut, les noms des colonnes de la nouvelle table (s'ils ne sont pas précisés) sont les noms des colonnes du SELECT.
- Si des expressions apparaîssent dans le SELECT, les colonnes doivent être renommées.
- Le SELECT peut contenir des fonctions de groupe, mais pas de clause ORDER BY.

Création d'une table

Exemple de création et de remplissage d'une table

Ajoutons à notre exemple fil rouge une table INTERVENANT listant les enseignants intervenants dans un module. On peut d'ores et déjà l'initialiser en considérant que tous les responsables de modules interviennent dans leurs modules.

```
CREATE TABLE INTERVENANT
(CodeModule CHAR(5),
Enseignant INTEGER)
AS SELECT CodeModule, Responsable FROM MODULE
```

Création d'une table

Exemple de création et de remplissage d'une table

Ajoutons à notre exemple fil rouge une table INTERVENANT listant les enseignants intervenants dans un module. On peut d'ores et déjà l'initialiser en considérant que tous les responsables de modules interviennent dans leurs modules.

```
CREATE TABLE INTERVENANT
AS SELECT CodeModule, Responsable AS Enseignant FROM MODULE
```

Création d'une table

Exemple de création et de remplissage d'une table

Ajoutons à notre exemple fil rouge une table INTERVENANT listant les enseignants intervenants dans un module. On peut d'ores et déjà l'initialiser en considérant que tous les responsables de modules interviennent dans leurs modules.

CREATE TABLE INTERVENANT
AS SELECT CodeModule, Responsable AS Enseignant FROM MODULE

MODULE

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test logiciel	...	7914
SR_L3	Systèmes et réseaux	...	22223
MCOO_L3	Modélisation objet	...	7914

Création d'une table

Exemple de création et de remplissage d'une table

Ajoutons à notre exemple fil rouge une table INTERVENANT listant les enseignants intervenants dans un module. On peut d'ores et déjà l'initialiser en considérant que tous les responsables de modules interviennent dans leurs modules.

CREATE TABLE INTERVENANT
AS SELECT CodeModule, Responsable AS Enseignant FROM MODULE

MODULE

CodeModule*	LibelleModule	...	Responsable
BD_L1	Bases de données	...	7914
PROG_L1	Programmation	...	7358
TEST_M2	Test logiciel	...	7914
SR_L3	Systèmes et réseaux	...	22223
MCOO_L3	Modélisation objet	...	7914

INTERVENANT

CodeModule	Enseignant
BD_L1	7914
PROG_L1	7358
TEST_M2	7914
SR_L3	22223
MCOO_L3	7914

Création d'une table

Les contraintes d'intégrité

Les contraintes d'intégrité sont :

- des règles prises en charge par le SGBD pour préserver la cohérence des données lors de la saisie, la modification et la suppression des données.
- décrites dans deux grands types de clauses : sur les attributs et sur les tables.

Création d'une table

Les contraintes d'intégrité sur les attributs

Les contraintes d'intégrité sur les attributs sont décrites à la suite de leur type :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1 [Contrainte*$_1$*] [Contrainte*$_2$*]*, ...)

La contrainte DEFAULT

La contrainte

DEFAULT *val*

spécifie que l'attribut auquel il se rapporte vaut *val* par défaut.

Création d'une table

Les contraintes d'intégrité sur les attributs

Les contraintes d'intégrité sur les attributs sont décrites à la suite de leur type :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1 [Contrainte*$_1$*] [Contrainte*$_2$*]*, ...)

La contrainte NOT NULL

La contrainte

NOT NULL

spécifie que l'attribut auquel il se rapporte ne peut pas prendre la valeur NULL.

Création d'une table

Les contraintes d'intégrité sur les attributs

Les contraintes d'intégrité sur les attributs sont décrites à la suite de leur type :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1 [Contrainte*$_1$*] [Contrainte*$_2$*]*, ...)

La contrainte UNIQUE

La contrainte

UNIQUE

spécifie que l'attribut auquel il se rapporte ne peut pas prendre deux fois la même valeur (i.e. c'est une clé candidate).

Création d'une table

Les contraintes d'intégrité sur les attributs

Les contraintes d'intégrité sur les attributs sont décrites à la suite de leur type :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1 [Contrainte*$_1$*] [Contrainte*$_2$*]*, ...)

La contrainte CHECK

La contrainte

CHECK (*condition*)

spécifie que la valeur de l'attribut doit satisfaire la condition pour être valide. Cette condition n'est vérifiée que si l'attribut n'est pas NULL (ça ne signifie pas que l'attribut porte forcément la contrainte NOT NULL.

Création d'une table

Exemple de création de table

Décrivons la table ETUDIANT de notre exemple fil rouge.

```
CREATE TABLE ETUDIANT
(NumeroEtudiant INTEGER NOT NULL UNIQUE,
NomEtudiant CHAR(30) NOT NULL,
PrenomEtudiant CHAR(50) NOT NULL,
DateNaissanceEtudiant DATE CHECK (DateNaissanceEtudiant < CURRENT_DATE),
LoginEtudiant CHAR(8) NOT NULL UNIQUE,
MotDePasseEtudiant CHAR(8) NOT NULL DEFAULT '1234'
```

Création d'une table

Les contraintes d'intégrité sur les tables

Les contraintes d'intégrité sur les tables sont décrites après les attributs :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1* ..., ...)
[CONSTRAINT $NomContrainte_1$] $Contrainte_1$, ...

La clause CONSTRAINT

La clause

CONSTRAINT *id*

permet de nommer la contrainte avec un identifiant *id* qui sera utilisé par le SGBD pour indiquer l'erreur si cette contrainte est, à un moment donné, violée. Cette clause est optionnelle.

Création d'une table

Les contraintes d'intégrité sur les tables

Les contraintes d'intégrité sur les tables sont décrites après les attributs :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1* ..., ...)
[CONSTRAINT $NomContrainte_1$] $Contrainte_1$, ...

La contrainte PRIMARY KEY

La contrainte

$$\text{PRIMARY KEY } (att_1, att_2, \ldots)$$

permet de spécifier que les attributs mentionnés entre parenthèses définissent la clé primaire de la table. Il n'y a qu'une seule contrainte PRIMARY KEY par table. Automatiquement, tous les attributs portent la propriété NOT NULL et chaque valeur de clé doit être unique (équivalent à UNIQUE si la clé n'est composée que d'un seul attribut).

Création d'une table

Les contraintes d'intégrité sur les tables

Les contraintes d'intégrité sur les tables sont décrites après les attributs :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1* ..., ...)
[CONSTRAINT $NomContrainte_1$] $Contrainte_1$, ...

La contrainte FOREIGN KEY

La clause

$$\text{FOREIGN KEY } (att_1, att_2, ...) \text{ REFERENCE } table\_liee(att'_1, att'_2, ...)$$

permet de spécifier que les attributs $att_1, att_2, ...$ sont clé étrangère et sont liés aux attributs $att'_1, att'_2, ...$ dans la table *table_liee*.

Création d'une table

Les contraintes d'intégrité sur les tables

Les contraintes d'intégrité sur les tables sont décrites après les attributs :

CREATE TABLE *nom_table*
(*NomAttribut1 Type1* ..., ...)
[CONSTRAINT $NomContrainte_1$] $Contrainte_1$, ...

La contrainte FOREIGN KEY

FOREIGN KEY (...) REFERENCE ...(...) [ON UPDATE P_1] [ON DELETE P_2]

La clause ON UPDATE (resp. ON DELETE) suivie d'un type de politique de modification (resp. suppression) permet de définir la politique à appliquer en cas de modification (resp. suprresion) de la clé primaire du lien.

On trouve pour P_1 et P_2 les valeurs classiques vues dans le chapitre 2 : RESTRICT, CASCADE, SET NULL, SET DEFAULT or NO ACTION.

Création d'une table

Exemple de création de table

Nous pouvons décrire complètement la table INTERVENANT que l'on rajoute à notre exemple fil rouge.

```
CREATE TABLE INTERVENANT
(CodeModule CHAR(5), Enseignant INTEGER)
CONSTRAINT Pk_intervenant PRIMARY KEY (CodeModule, Enseignant)
CONSTRAINT Fk_module FOREIGN KEY (CodeModule)
REFERENCE MODULE(CodeModule)
ON DELETE CASCADE   ON UPDATE CASCADE
CONSTRAINT Fk_enseignant FOREIGN KEY (Enseignant)
REFERENCE ENSEIGNANT(NumeroHarpege)
ON DELETE CASCADE   ON UPDATE CASCADE
```

Modification d'une table

Modification d'une table

La modification d'une table peut être réalisée par :

- l'ajout d'un attribut
- la modification d'un attribut existant
- la suppression d'un attribut

Modification d'une table

Syntaxe générale

La syntaxe générale permettant l'ajout d'un attribut est la suivante :

ALTER TABLE *table*
ADD (Att_1 $Type_1$ [*Contraintes*], Att_2 $Type_2$ [*Contraintes*], ...)

Quelques remarques

- L'ajout des attributs de la clause ADD se fera en fin de table.
- Il est possible d'appliquer des contraintes d'intégrité sur les attributs (cf. CREATE TABLE)

Modification d'une table

Syntaxe générale

La syntaxe générale permettant l'ajout d'un attribut est la suivante :

ALTER TABLE *table*
ADD (Att_1 $Type_1$ [*Contraintes*], Att_2 $Type_2$ [*Contraintes*], ...)

Exemple d'ajout de colonne

On ajoute à la table INTERVENANT un attribut comptant le nombre d'heures effectuées dans un module par un enseignant :

ALTER TABLE INTERVENANT
ADD (NbHeures INTEGER NON NULL CHECK (NbHeures > 0))

Modification d'une table

Syntaxe générale

La syntaxe générale permettant la modification d'un attribut est la suivante :

ALTER TABLE *table*
MODIFY Att_1 $Type_1$ [*Contraintes*], MODIFY Att_2 $Type_2$ [*Contraintes*], ...

Quelques remarques

- Cette commande permet de modifer le type de donnée des attributs existants.
- Il est possible d'appliquer des contraintes d'intégrité sur les attributs

Modification d'une table

Syntaxe générale

La syntaxe générale permettant la modification d'un attribut est la suivante :

ALTER TABLE *table*
MODIFY Att_1 $Type_1$ [*Contraintes*], MODIFY Att_2 $Type_2$ [*Contraintes*], ...

Exemple de modification d'attribut

On modifie la table ETUDIANT pour pouvoir prendre en compte des noms d'étudiants plus longs (100 caractères vs. 50 actuellement).

ALTER TABLE ETUDIANT
MODIFY NomEtudiant char(100) NOT NULL

Modification d'une table

Syntaxe générale

La syntaxe générale permettant la suppression d'un attribut est la suivante :

ALTER TABLE *table*
DROP Att_1, DROP Att_2, ...

Quelques remarques

- Cette commande permet de supprimer des attributs existants.
- Attention à la cohérence de la base en particulier si les attributs supprimés sont clés étrangère dans un lien ...

Modification d'une table

Syntaxe générale

La syntaxe générale permettant la suppression d'un attribut est la suivante :

ALTER TABLE *table*
DROP Att_1, DROP Att_2, ...

Exemple de suppression d'un attribut

On modifie la table ETUDIANT pour supprimer la date de naissance des étudiants :

ALTER TABLE ETUDIANT
DROP DateNaissanceEtudiant

Modification d'une table

Une remarque sur la syntaxe ALTER TABLE

Il est possible de réaliser plusieurs modifications différentes sur une même table en une seule invocation de ALTER TABLE, en enchaînant les ADD/MODIFY/DROP, séparées par des virgules.

ALTER TABLE *table*
ClauseModification1, *ClauseModification2*, ...

Modification multiples

On modifie la table ETUDIANT pour réaliser en une seule fois les modifications précédentes :

ALTER TABLE ETUDIANT
MODIFY NomEtudiant char(100) NOT NULL, DROP DateNaissanceEtudiant

Suppression d'une table

Syntaxe générale

La syntaxe générale permettant la suppression d'une table est la suivante :

DROP TABLE *nom_table*

Remarque

Si la table détruite par la commande DROP TABLE a une clé primaire utilisée pour la définition d'un lien vers une autre table, alors l'ordre DROP TABLE échoue.

Suppression d'une table

Syntaxe générale

La syntaxe générale permettant la suppression d'une table est la suivante :

DROP TABLE *nom_table*

Exemple de suppression de table

On souhaite supprimer la table INTERVENANT qui a été précédemment créée.

DROP TABLE INTERVENANT

Les index

Qu'est-ce qu'un index ?

Considérons la requète :

```
SELECT *
FROM ETUDIANT
WHERE NomEtudiant = "Dornier"
```

Pour trouver le (ou les enregistrements) concerné(s) il est nécessaire de parcourir tous les enregistrements de la table ETUDIANT.
⇒ Un tel traitement risque de conduire à des temps d'accès très longs dès que l'on aura beaucoup d'informations dans la table.

Les index

Qu'est-ce qu'un index ?

Les index représentent une solution à ce problème :

- Un index permet d'accéder rapidement et directement à certains t-uplets dans la table.
- L'index se compose d'un ou plusieurs attributs de la table.
- Les requêtes SQL (de type SELECT) sont transparentes au fait qu'il existe des index ou non.

  $\Rightarrow$ les index sont surtout utilisés par l'interpréteur SQL et plus précisément l'optimiseur de requêtes.

Les index

Qu'est-ce qu'un index ?

Concrètement,

- Un index peut être créé juste après la création de la table ou lorsque celle-ci contient déjà des enregistrements.
- Les index sont mis à jour automatiquement à chaque mise à jour de la table (ajout/modification/suppression d'enregistrements).
- Dans beaucoup de SGBD, la déclaration d'une clé primaire implique la création d'un index sur cette même clé.

Les index

Quand faut-il créer des index ?

Il peut être intéressant de créer un index si :

- on effectue souvent des sélections sur un ensemble d'attributs,
- on effectue beaucoup de jointures entre deux tables par rapport à un ensemble d'attributs.

Quand ne faut-il pas créer d'index ?

Il n'est pas nécessaire de créer un index si :

- la table contient peu d'enregistrements ; dans ce cas, la recherche dans toute la table sera aussi rapide sans index,
- l'attribut ou l'ensemble d'attributs prennent peu de valeurs différentes.

Les index

Syntaxe générale

La syntaxe générale de création d'un index est la suivante :

CREATE [UNIQUE] INDEX *NomIndex* ON *table*(Att_1, Att_2, ...)

Quelques remarques

- Cette commande permet de créer un index nommé *NomIndex* sur les attributs Att_1, Att_2, ... de *table*.
- Le mot-clé optionnel UNIQUE signifie qu'il y a unicité des attributs indexés.

Les index

Syntaxe générale

La syntaxe générale de création d'un index est la suivante :

CREATE [UNIQUE] INDEX *NomIndex* ON *table*(Att_1, Att_2, ...)

Exemple de création d'index

Accélérons les recherches dans la table ETUDIANT en créant un index sur les noms d'étudiants :

CREATE INDEX IdxNom ON ETUDIANT(NomEtudiant)

Les index

Syntaxe de la création d'un index lors de la création de la table

La syntaxe pour inclure la création d'un index lors de la création de la table est la suivante :

CREATE TABLE *NomTable*
(*Att*$_1$ *Type*$_1$ ..., ...)
INDEX *NomIndex* [UNIQUE] (*Att*$_1$, ...)

La création d'un index se déclare après avoir déclaré les attributs. Il n'y a pas d'ordre précis entre les clauses INDEX, PRIMARY/FOREIGN KEY, etc. qui peuvent apparaître à cet endroit.

Les index

Syntaxe générale

La syntaxe générale de suppression d'un index est la suivante :

DROP INDEX *NomIndex*

Exemple de destruction d'index

Supprimons l'index "IdxNom" créé précédemment sur la table ETUDIANT.

```
DROP INDEX IdxNom
```

Les vues

Qu'est-ce qu'une vue ?

- Une vue est une table dont les données ne sont pas stockées physiquement, mais qui se réfère à d'autres tables réelles.
- Seule la définition de la vue est stockée dans la base, mais pas son contenu.

Une vue peut être assimilée à une *table virtuelle*.

Les vues

Que permet cette notion ?

Une vue permet :

- de dissocier la façon dont les utilisateurs voient les données et les tables réelles, en séparant les aspects externes (vues) et les aspects internes (tables du modèle),
- de favoriser l'indépendance des données et des programmes,
- de restreindre l'accès à certaines parties des tables,
- de protéger l'accès aux tables en fonction des utilisateurs.

Les vues

Syntaxe générale

La création d'une vue se réalise par la syntaxe suivante, à l'aide d'une requête SELECT qui sélectionne dans la base les informations pour la vue :

CREATE VIEW *NomVue*
AS SELECT ...

Création d'une vue

Création d'une vue donnant les inscriptions des étudiants de 2010.

```
CREATE VIEW INSCRIPTION2010 AS
SELECT *
FROM INSCRIPTION
WHERE Annee = 2010
```

Les vues

Vue = table virtuelle

- Il est donc possible de faire des requêtes SELECT à partir d'une vue (qui sera référencée dans la clause FROM).
- On peut imaginer vouloir réaliser des opérations de mise à jour de données (INSERT, UPDATE, DELETE) à partir de la vue.

  $\Rightarrow$ C'est possible, mais sous certaines conditions bien précises.

Les vues

Conditions à respecter pour effectuer des mises à jour à partir d'une vue

- Le SELECT définissant la vue ne fait référence qu'à une seule table.
- Ce SELECT ne comporte pas de DISTINCT.
- Les colonnes résultat du SELECT correspondent à des noms d'attributs et pas à des résultats d'expressions (même renommés).
- La clause WHERE ne doit pas contenir de sous-requête synchronisée.
- Il n'y a ni clause GROUP BY, ni HAVING.

Si la vue ne respecte pas ces conditions, elle est en *lecture seule*.

Les vues

Syntaxe générale

La syntaxe générale de suppression d'une vue est la suivante :

DROP VIEW *NomVue*

Exemple de destruction d'index

Supprimons la vue créée précédemment sur les inscriptions de 2010.

DROP VIEW INSCRIPTIONS2010

Plan du cours

Data Query Language

Data Manipulation Language

Data Definition Language

Data Control Language

- Utilisateurs
- Donner des droits
- Retirer des droits

Data Control Language

Data Control Language

Nous avons vu qu'à maintenant comment manipuler (au sens large du terme) les données, et la structure de la base de données.

Nous allons désormais voir une partie du langage DCL qui permet de contrôler, pour les environnements multi-utilisateurs, les accès à des éléments d'une base de données, à travers l'assignations de *droits* à des *utilisateurs*.

Utilisateurs

Contrôle des utilisateurs

Dans certains SGBD multi-utilisateurs, il est possible de définir des utilisateurs. On peut ainsi :

- leur donner ou leur interdire l'accès à certaines parties du SGBD (par exemple : accès à une base en particulier).
- leur autoriser ou leur interdire certaines opérations sur une base de données.

En général, ce genre de système possède un super-utilisateur, généralement appelé *root*, qui possède tous les droits, notamment celui de créer d'autres utilisateurs et de leur affecter des droits.

Exemple de nécessité de contrôler les utilisateurs

En Licence 2 Informatique, les étudiants travaillent avec une base de données MySQL dans le module "Langage du Web". Pour éviter toute fausse manipulation, chacun a un accès personnalité (un utilisateur existe pour chaque étudiant), et restreint (ils n'ont pas le droit de modifier la structure de la base).

Utilisateurs

Syntaxe générale

La syntaxe générale pour la création d'un utilisateur est la suivante :

CREATE USER $user_1$ [IDENTIFIED BY [PASSWORD]] '$password_1$', $user_2$ ...

Création d'un utilisateur

Création d'un utilisateur *etudiant* identifié par le mot de passe '1234'.

CREATE USER *etudiant* IDENTIFIED BY '1234'

Utilisateurs

Renommage d'un utilisateur

La syntaxe pour renommer un utilisateur est la suivante :

RENAME USER $ancienNom_1$ TO $nouveauNom_1$, $ancienNom_2$ TO $nouveauNom_2$, ...

Changer le mot de passe d'un utilisateur

La syntaxe pour changer le mot de passe d'un utilisateur est la suivante :

SET PASSWORD [FOR USER *user*] = PASSWORD('*nouveauPassword*')

Sans la clause FOR USER le changement de mot de passe concerne l'utilisateur courant.

Donner des droits

Syntaxe générale

La syntaxe générale pour donner des droits à des utilisateurs est la suivante :

GRANT *TypePrivilege*$_1$, *TypePrivilege*$_2$, ...
ON *Entite*
TO *user*$_1$ [IDENTIFIED BY [PASSWORD] '*password*$_1$'], *user*$_2$ ...

Donner des droits

Syntaxe générale

La syntaxe générale pour donner des droits à des utilisateurs est la suivante :

GRANT *TypePrivilege*$_1$, *TypePrivilege*$_2$, ...
ON *Entite*
TO *user*$_1$ [IDENTIFIED BY [PASSWORD] '*password*$_1$'], *user*$_2$ ...

Les types de privilèges

Ils distinguent les actions pour lesquels les droits sont définis sur l'entité considérée :

- CREATE, ALTER, DROP pour une table.
- SELECT, INSERT, UPDATE, DELETE pour une table.
- INDEX pour un index (autorise CREATE INDEX et DROP INDEX).
- GRANK, REVOKE pour un utilisateur.
- ALL [PRIVILEGES] : tous les droits possibles pour l'entité considérée.

Donner des droits

Syntaxe générale

La syntaxe générale pour donner des droits à des utilisateurs est la suivante :

GRANT *TypePrivilege*$_1$, *TypePrivilege*$_2$, ...
ON *Entite*
TO *user*$_1$ [IDENTIFIED BY [PASSWORD] '*password*$_1$'], *user*$_2$ ...

Les entités

L'entité mentionnée représente sur quoi s'appliquent les privilèges :

- $*$: les droits s'appliquent pour toutes les bases.
- $*.*$: les droits s'appliquent pour toutes les tables de toutes les bases.
- *NomBase*.$*$: les droits s'appliquent pour toutes les tables de la base spécifiée.
- *NomBase*.*NomTable* : les droits s'appliquent pour la table spécifiée de la base spécifiée.

Donner des droits

Exemple de don de droits

On souhaite donner aux étudiants de Licence 2, partageant l'utilisateur *etudiant*, des droits sécurisés sur la base de données *projet*.

GRANT SELECT, INSERT, UPDATE, DELETE
ON *projet*.∗
TO *etudiant*

Retirer des droits

Syntaxe générale

La syntaxe générale pour retirer des droits à des utilisateurs est la suivante :

REVOKE $TypePrivilege_1$, $TypePrivilege_2$, ...
ON *Entite*
TO $user_1$ [IDENTIFIED BY [PASSWORD] '$password_1$'], $user_2$ ...

Exemple de retrait de droits

On souhaite retirer aux étudiants de Licence 2, partageant l'utilisateur *etudiant*, la possibilité de démanteler la base de données *projet*.

REVOKE CREATE, ALTER, DROP
ON *projet*.*
TO *etudiant*

Retirer des droits

Syntaxe générale

La syntaxe générale pour retirer des droits à des utilisateurs est la suivante :

REVOKE *TypePrivilege*$_1$, *TypePrivilege*$_2$, ...
ON *Entite*
TO *user*$_1$ [IDENTIFIED BY [PASSWORD] '*password*$_1$'], *user*$_2$ ...

Exemple de retrait de droits

On souhaite aussi empêcher la modification du contenu de la table *log* (car elle est remplie automatiquement).

REVOKE INSERT, UPDATE, DELETE
ON *projet*.*log*
TO *etudiant*

Structured Query Language

SQL est un langage de requêtes structuré, dont la syntaxe est standardisée, mais pour laquelle certaines variantes peuvent exister en fonction du SGBD considéré.

Bilan sur SQL

SQL permet d'exprimer :

- des requêtes d'interrogation de la base de données (SELECT)
- des requêtes de mise à jour de données (INSERT, UPDATE, DELETE)
- des requêtes de mise à jour de la base (CREATE, ALTER, DROP)
- des requêtes de contrôle des accès aux données (GRANT, REVOKE)

Nous n'avons pas fait un tour d'horizon complet de SQL, la suite sera pour la 3ème année de Licence Informatique (module "Administration de Bases de Données"). Néanmoins, nous en avons vu assez pour faire déjà pas mal de choses, cette année et l'année prochaine, dans le module "Langages du Web".