femto-st SCIENCES & TECHNOLOGIES

L1 Sciences Fondamentales, S2
Langage SQL

Jean-François Couchot

Université de Franche-Comté, UFR-ST

UNIVERSITÉ DE FRANCHE-COMTÉ

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

cnrs

Plan

Plan

Langages de SGBD relationnelle : une histoire

De l'Alpha à . . . SQL

- Alpha, inventé par Edgar F. Codd (1970)
- . . . SEQUEL – Structured English QUEry Language . . .
- SQL – Structured Query Language (1974)

Quelques versions de SQL

- SQL–86 (SQL1) : norme proposée par l'ANSI [a]
  - Remarque : permet de manipuler des données mais pas de modifier un modèle relationnel
- SQL–92 (SQL2) : version utilisée dans la plupart des SGBD actuels
  - plus de types de données gérés et permet de modifier un modèle relationnel
- SQL—99 (SQL3) : version qui intègre un modèle objet-relationnel

a. American National Standard Institute

Structured Query Language

SQL : 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 comment l'obtenir
- Chaque SGBD : possède un interpréteur SQL qui exécute la requête (pour obtenir le résultat demandé)
- Traitement de la requête : optimisé pour accélérer les calculs
- Traitement sous-jacent : propre à chaque SGBD
- Résultats : indépendants du SGBD

Les 3 (ou 4) sous-langages de SQL

DML – Data Manipulation Language

Permet d'extraire et de mettre à jour les données contenues dans la base.
La partie *extraction de données* est parfois considérée à part entière sous la dénomination Data Query Language (DQL).

DDL – Data Definition Language

Permet de créer, de modifier ou de supprimer les relations de la base de données, de définir des liens entre ces relations et leurs clés primaires.

DCL – Data Control Language

Permet de gérer des protections d'accès aux relations ou tables en environnements multi-utilisateurs.

Plan du cours

Plan

Plan

Requêtes de sélection : SELECT

Détail des paramètres de SELECT

`SELECT` *liste d'attributs*
⇒ Noms des attributs à extraire 4
`FROM` *liste de relations*
⇒ Noms des relations utilisées
`WHERE` *critère de sélection*
⇒ Expression déterminant les t-uplets sélectionnés 1
`GROUP BY` *liste d'attributs de groupe*
⇒ Noms des attributs définissant un regroupement 2
`HAVING` *critère de sélection pour le groupe*
⇒ Expression déterminant les groupes sélectionnés 3
`ORDER BY` *liste d'attributs*
⇒ Noms des attributs sur lesquels se fait l'ordonnancement 5

- A minima : `SELECT ... FROM ...`

- Ordre d'évaluation : donné en rouge à titre indicatif

SELECT ... FROM ...

Syntaxe de SELECT

```
SELECT [DISTINCT] expr1 [AS Nom1] [, ...] | *
```

- Eléments entre crochets "[ ]" : optionnels
- Mot-clé `DISTINCT` : doublons supprimés dans le résultat ($\neq$ algèbre relationnelle)
- Expressions `expr` :
  - des noms d'attributs : NumeroEtudiant, NomEtudiant, ...
  - le résultat de fonctions : `SUM`, `AVG`, `COUNT` ... (étudié plus tard)
- Construction `expr AS Nom` : renommage `expr` pour qu'elle apparaisse sous le dénomination `Nom` dans le résultat.
- Etoile "*" : tous les attributs disponibles dans la source de données spécifiée dans la clause `FROM`

SELECT [DISTINCT] ... FROM ...

SELECT Prenom FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Avenia	Adrien	18-08-1998	aavenia	aavenia
32827	Signoret	Adrien	24-03-1998	asignoret	asignoret
32911	Gigant	Simon	26-02-1998	sgigant	sgigant
33818	Maillard	Cynthia	06-05-1998	cmaillard	cmaillard
34812	Smolinski	Marie	21-11-1998	ssmolinski	ssmolinski

Résultat :

Prenom
Adrien
Adrien
Simon
Cynthia
Marie

- SELECT : renvoie par défaut tous les doublons existants
- Résultat d'une requête : pas une relation

SELECT DISTINCT Prenom FROM ETUDIANT

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Avenia	Adrien	18-08-1998	aavenia	aavenia
32827	Signoret	Adrien	24-03-1998	asignoret	asignoret
32911	Gigant	Simon	26-02-1998	sgigant	sgigant
33818	Maillard	Cynthia	06-05-1998	cmaillard	cmaillard
34812	Smolinski	Marie	21-11-1998	ssmolinski	ssmolinski

Résultat :

Prenom
Adrien
Simon
Cynthia
Marie

SELECT expr AS Nom FROM ...

```
SELECT DISTINCT Prenom AS FirstName FROM ETUDIANT
```

ETUDIANT

Numero*	Nom	Prenom	DateNaissance	Login	MotDePasse
23794	Avenia	Adrien	18-08-1998	aavenia	aavenia
32827	Signoret	Adrien	24-03-1998	asignoret	asignoret
32911	Gigant	Simon	26-02-1998	sgigant	sgigant
33818	Maillard	Cynthia	06-05-1998	cmaillard	cmaillard
34812	Smolinski	Marie	21-11-1998	ssmolinski	ssmolinski

Résultat :

FirstName
Adrien
Simon
Cynthia
Marie

- Construction "*AS FirstName*" : colonne *Prenom* renommée en *FirstName*

SELECT ... FROM ...

Syntaxe de FROM ...

```
SELECT ...
FROM Rel1 [AS T1], Rel2 [AS T2]
```

- **FROM** : source de données de la requête en indiquant les tables ou les sous-requêtes qui seront utilisées
- Construction `Rel AS T` : renommage de la relation `Rel` en `T`

SELECT Nom, CodeModule FROM ETUDIANTS1, MODULES1

ETUDIANTS1

Numero*	Nom
23794	Avenia
32911	Gigant
34812	Smolinski

MODULES1

CodeModule*
BD_L1
PROG_L1

Résultat

Nom	CodeModule
Avenia	BD_L1
Avenia	PROG_L1
Gigant	BD_L1
Gigant	PROG_L1
Smolinski	BD_L1
Smolinski	PROG_L1

C'est un produit cartésien.

SELECT ... FROM ... WHERE...

Syntaxe de WHERE...

```
SELECT ...  FROM ...
WHERE CritereDeSelection
```

- `CritereDeSelection` : expression logique évaluée pour chaque *t*-uplet
  - comparaison entre 2 éléments : >, >=, <=, <, =, <>
  - comparaison avec `NULL` : `IS NULL`, `IS NOT NULL`
  - opérateurs d'appartenance : `IN`, `BETWEEN ... AND ...`, `LIKE`
  - connecteurs logiques : `AND`, `OR`, `NOT`
  - dans les chaînes de caractères, utilisation de caractères génériques : _ (remplace 1 caractère), % (remplace une chaîne de caractères)

SELECT NoEtudiant, CodeModule FROM INSCRIPTION WHERE Annee = 2016

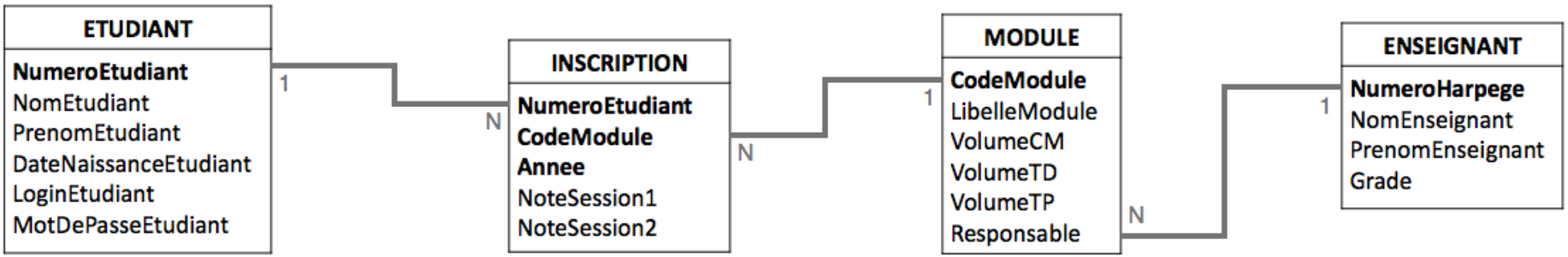
INSCRIPTION

NoEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2015	7	9
23794	BD_L1	2016	15	NULL
23794	PROG_L1	2015	16	NULL
32911	BD_L1	2016	7	12
32911	PROG_L1	2016	16	NULL
34812	BD_L1	2016	10	NULL

Résultat :

NoEtudiant	CodeModule
23794	BD_L1
32911	BD_L1
32911	PROG_L1
34812	BD_L1

SELECT ... FROM ... WHERE... : exemples 1



- Numéro des étudiants inscrits dans un module en 2015 ou en 2016

```
SELECT NumeroEtudiant FROM INSCRIPTION
WHERE Annee IN (2015, 2016)
```

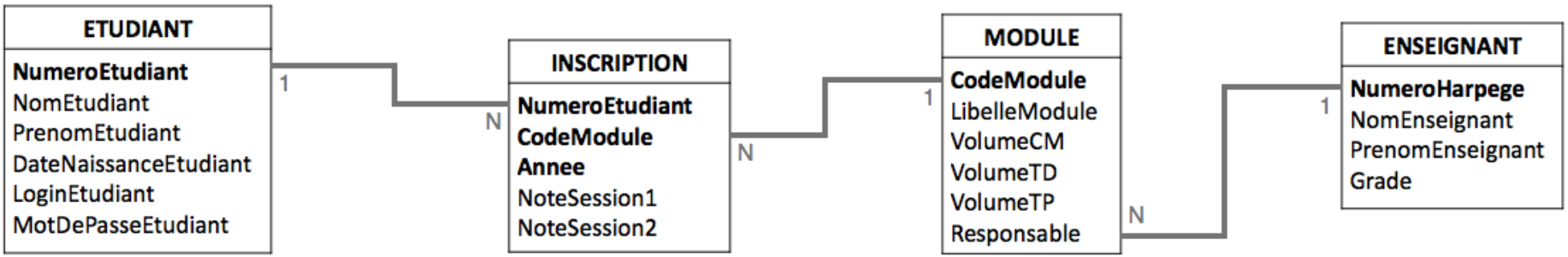
- Numéro des étudiants nés en 1998, ou après

```
SELECT NumeroEtudiant FROM ETUDIANT
WHERE DateNaissanceEtudiant >= '1-01-1998'
```

- Numéro des étudiants dont le prénom commence par A.

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

SELECT ... FROM ... WHERE... : exemples 2



- Numéro des étudiants ayant été en seconde session et code du module associé.

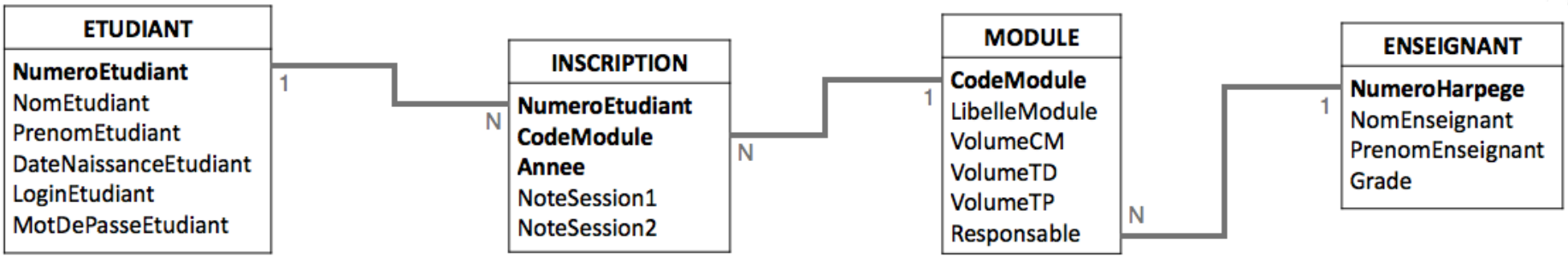
```
SELECT NumeroEtudiant, CodeModule FROM INSCRIPTION
WHERE NoteSession2 IS NOT NULL
```

- 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
```

Plan

Jointure : 2 syntaxes principales



Spécifiée dans `WHERE ...`

- Exemple : libellé des modules et nom de leur responsable

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

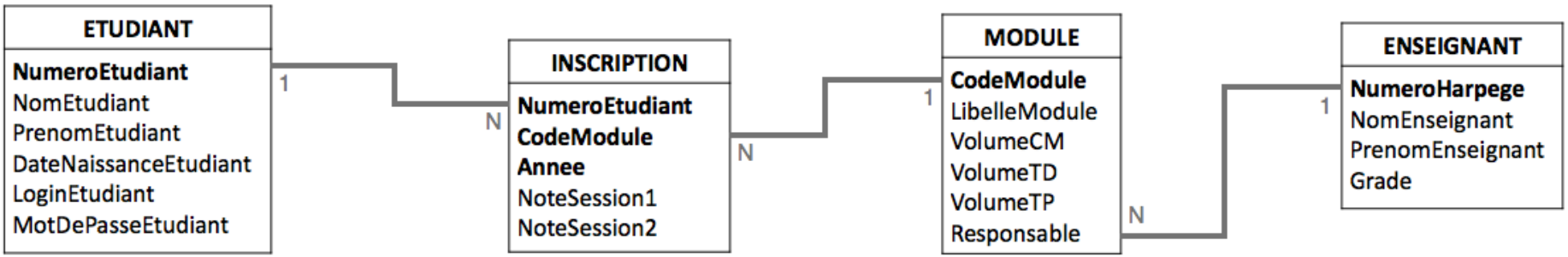
Spécifiée dans `FROM ...JOIN ...ON`

- Même exemple : libellé des modules et nom de leur responsable

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

Bonne pratique : préfixer les attributs par le nom de la relation

NATURAL JOIN pour des attributs homonymes



Deux relations T1 et T2 avec des attributs homonymes

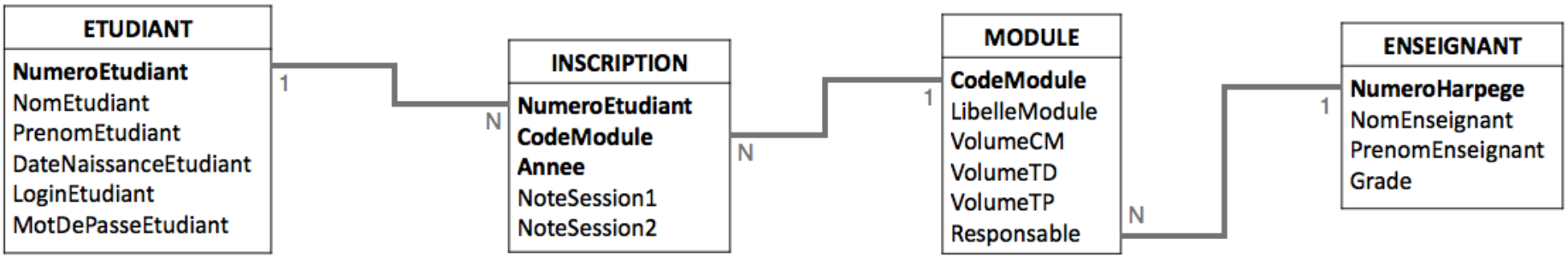
- Exemples : nom et numéro des étudiants inscrits en 2016

```
SELECT ETUDIANT.NomEtudiant, ETUDIANT.NumeroEtudiant
FROM ETUDIANT NATURAL JOIN INSCRIPTION
WHERE Annee = 2016
```

Auto-jointure

Définition pour une relation T

Jointure d'une relation T avec elle-même, mais renommées par `T AS T'` pour pouvoir comparer les attributs de T avec ceux de T'.



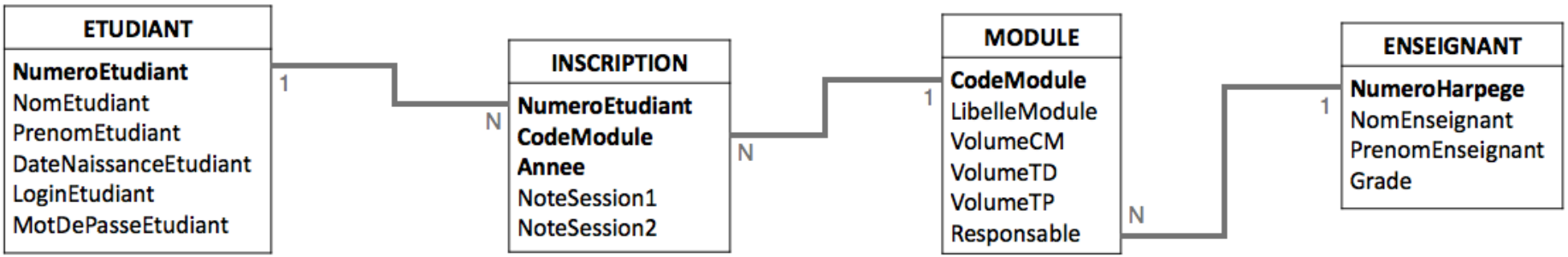
Numéro des étudiants inscrits au m̂ module, la m̂ année que 23974

```
SELECT INSCRIPTION.NumeroEtudiant FROM INSCRIPTION, INSCRIPTION AS I2
WHERE I2.NumeroEtudiant = 23974
AND INSCRIPTION.CodeModule = I2.CodeModule AND INSCRIPTION.Annee = I2.Annee
```

Nom, prénom des étudiants inscrits au m̂ module, la m̂ année que 23974

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

A vous de jouer



Donnez les requêtes permettant de calculer, sur notre exemple :

- Le nom et prénom des étudiants nés avant 1999.
- Le nom et le prénom des étudiants ayant validé des modules en session 1 en 2016.
- Le libellé des modules qui ont eu des responsables différents au cours du temps.

Lien entre SELECT et Alègbre relationnelle

Traduction d'un SELECT minimal en algèbre relationnelle

```
SELECT DISTINCT A1,A2,...,An
FROM R1,R2,...,Rm
WHERE <CONDITION(A1,...,An,An+1,...,An+k)>
```

est équivalent à l'expression en algèbre relationnelle

```
P1   = R1 pcart R2
...
Pmm1 = Pnm2 pcart Rm
SEL  = S(CONDITION(A1,...,An,An+1,...,An+k))Pnm1
RES  = [A1, ... ,An] SEL
```

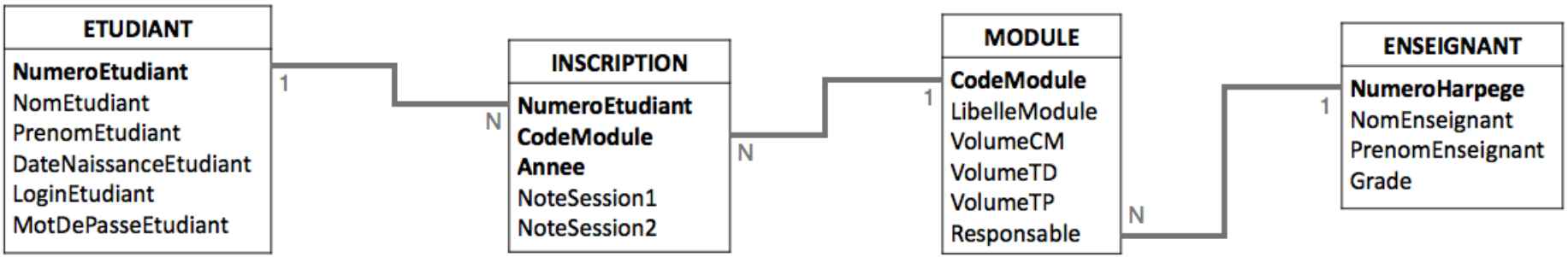
Plan

Mesures sur des valeurs d'un attribut

Fonctions de groupe

- Mesures sur des (regroupement de) valeurs d'un attribut d'une relation :
  - `MIN(Att)` : la plus petite valeur de l'attribut `Att`
  - `MAX(Att)` : la plus grande valeur de l'attribut `Att`
  - `SUM(Att)` : la somme des valeurs de l'attribut `Att`
  - `COUNT(Att)` : le nombre de lignes de l'attribut `Att`
  - `AVG(Att)` : la moyenne des valeurs de l'attribut `Att`
  - `VARIANCE(Att)` : la variance des valeurs de l'attribut `Att`
  - `STDDEV(Att)` : l'écart-type des valeurs de l'attribut `Att`
- Attention : fonctions applicables sur un attribut (une colonne), pas sur une ligne !

Mesures sur des valeurs d'un attribut : exemples



Date de naissance du plus vieil étudiant inscrit en BD_L1 en 2016

```
SELECT MIN(DateNaissanceEtudiant)
FROM ETUDIANT NATURAL JOIN INSCRIPTION
WHERE  INSCRIPTION.Annee = 2016 AND INSCRIPTION.CodeModule = 'BD_L1'
```

Nombre d'étudiants inscrits au module BD_L1 en 2016

```
SELECT COUNT(NumeroEtudiant)  FROM INSCRIPTION
WHERE Annee = 2016 AND CodeModule = 'BD_L1'
```

Moyenne des notes de première session au module BD_L1 en 2016

```
SELECT AVG(NoteSession1) AS MoyenneSession1  FROM INSCRIPTION
WHERE Annee = 2016  AND CodeModule = 'B_DL1'
```

Particularité de la fonction COUNT

COUNT : 3 types d'appels

- `COUNT (*)` : nombre de t-uplets obtenus dans la requête (lignes)
- `COUNT (Att)` : nombre de valeurs de l'attribut `Att` différentes de `NULL`.
- `COUNT (DISTINCT Att)` : nombre de valeurs distinces de `Att` différentes de `NULL`

Exemple d'utilisation de la fonction COUNT, relation ENSEIGNANT

NumeroHarpege*	NomEnseignant	PrenomEnseignant	Grade
7358	Janey	Nicolas	MCF
7914	Dadeau	Frédéric	MCF
22223	Guyennet	Hervé	PR
32598	Paquette	Guillaume	NULL

- `SELECT COUNT(*) FROM ENSEIGNANT` $\rightsquigarrow 4$
- `SELECT COUNT(Grade) FROM ENSEIGNANT` $\rightsquigarrow 3$
- `SELECT COUNT(DISTINCT Grade) FROM ENSEIGNANT` $\rightsquigarrow 2$

GROUP BY : spécification du regroupement

Définition

- Dans une instruction `SELECT`, spécifie quels attributs doivent avoir des valeurs égales pour regrouper les t-uplets
- Syntaxe : `SELECT ... FROM ... WHERE ... GROUP BY ...`

GROUP BY : nombre de modules suivis par étudiant par année

```
SELECT COUNT(*) AS NbModSuivis FROM INSCRIPTION
GROUP BY NumeroEtudiant, Annee
```

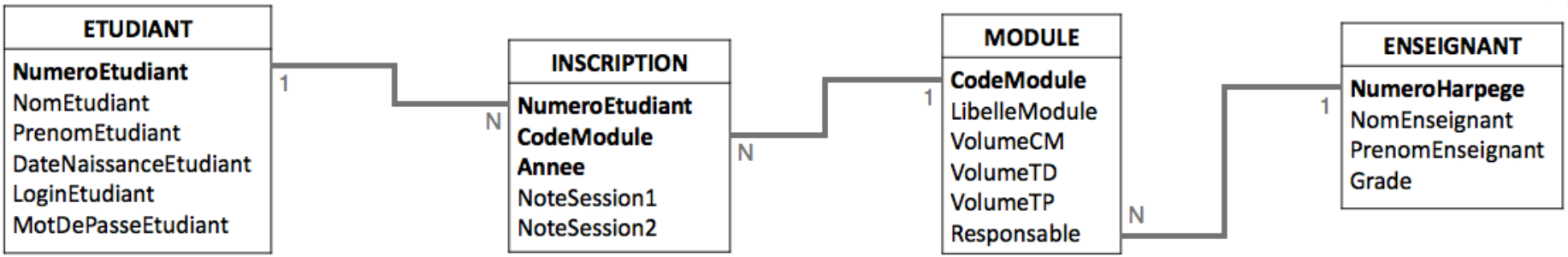
INSCRIPTION

NumeroEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2015	7	9
23794	BD_L1	2016	15	NULL
23794	PROG_L1	2015	16	NULL
32911	BD_L1	2016	7	12
32911	PROG_L1	2016	16	NULL
34812	BD_L1	2016	10	NULL

Résultat :

NbModSuivis
2
1
2
1

GROUP BY : exemples



Nombre d'étudiants par module et par année

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

INSCRIPTION

NumeroEtudiant*	CodeModule	Annee	NoteSession1	NoteSession2
23794	BD_L1	2015	7	9
23794	BD_L1	2016	15	NULL
23794	PROG_L1	2015	16	NULL
32911	BD_L1	2016	7	12
32911	PROG_L1	2016	16	NULL
34812	BD_L1	2016	10	NULL

Résultat :

Annee	CodeModule	Nb
2015	BD_L1	1
2016	BD_L1	3
2015	PROG_L1	1
2016	PROG_L1	1

Nom des enseignants et le nombre de modules dont ils sont responsables

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

HAVING : sélection selon une fct. de groupe

HAVING : définition

- Spécifie un critère `C` de sélection utilisant une fonction de groupe (interdit dans la clause `WHERE`)
- Réutilise les regroupements spécifiés par `GROUP BY`
- Syntaxe : `SELECT...FROM...WHERE...GROUP BY...HAVING C`

Nom des enseignants qui sont responsables d'au moins 2 modules

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

ORDER BY : ordonner les résultats

ORDER BY : définition

- Résultats ordonnés suivant `ASC` (croissant, par déf.) `DESC` (décroissant).
- Syntaxe :
  `SELECT...ORDER BY expr1 [ASC ou DESC], expr2 [ASC ou DESC]`

Nom, Prénom des étudiants, par ordre alphab. sur les noms et prénoms

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

ETUDIANT

NumeroEtudiant*	NomEtudiant	PrenomEtudiant	DateNaissanceEtudiant	LoginEtudiant	MotDePasseEtudiant
23794	Avenia	Adrien	18-08-1998	aavenia	aavenia
32827	Signoret	Adrien	24-03-1998	asignoret	asignoret
32911	Gigant	Simon	26-02-1998	sgigant	sgigant
33818	Maillard	Cynthia	06-05-1998	cmaillard	cmaillard
34812	Smolinski	Marie	21-11-1998	ssmolinski	ssmolinski

Résultat :

NomEtudiant	PrenomEtudiant
Avenia	Adrien
Gigant	Simon
Maillard	Cynthia
Signoret	Adrien
Smolinski	Marie

Plan

Sous-requête

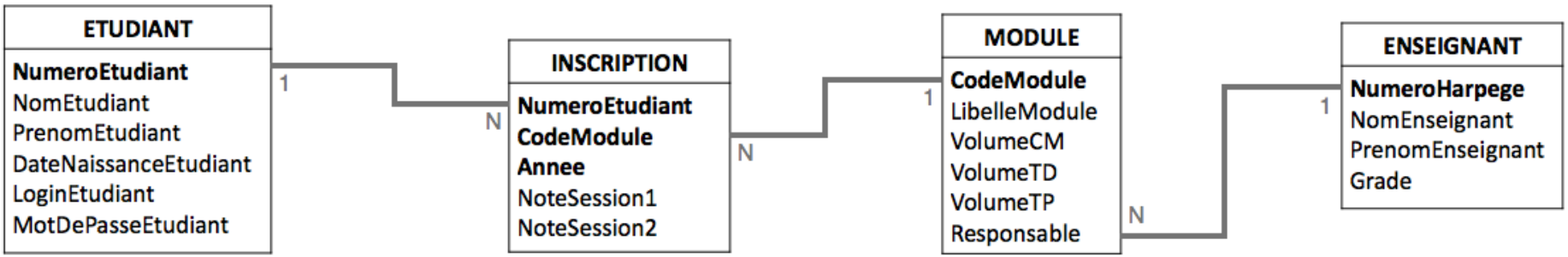
Intuition : imbrication de requêtes

Les résultats d'une sous-requête : utilisables dans une autre requête via un opérateur relationnel (IN, =,...)

Code générique

```
SELECT ... FROM ... WHERE
Attr <OPERATEUR>  ( SELECT  ... FROM ... WHERE ... )
```

Sous-requête : exemple



Modules qui ont le même responsable que le module BD_L1

- Avec une auto-jointure :

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

- Avec une sous-requête :

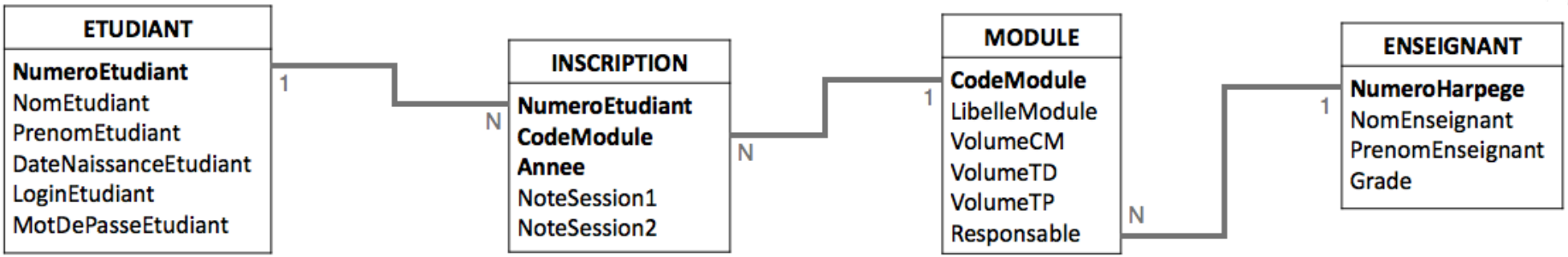
```
SELECT CodeModule FROM Module
WHERE Responsable = (SELECT Responsable FROM Module
                     WHERE CodeModule = 'BD_L1')
```

Sous-requêtes : 2 types

Indépendance vs. synchronisation

- Sous-requête indépendante : aucune référence aux attributs manipulés dans la requête principale
  - $\rightsquigarrow$ évaluée avant la requête principale
- Sous-requête synchronisée : des références aux attributs de la requête principale
  - $\rightsquigarrow$ évaluée après la requête principale

Sous-requête indépendante : exemple



Modules qui ont le même responsable que le module BD_L1

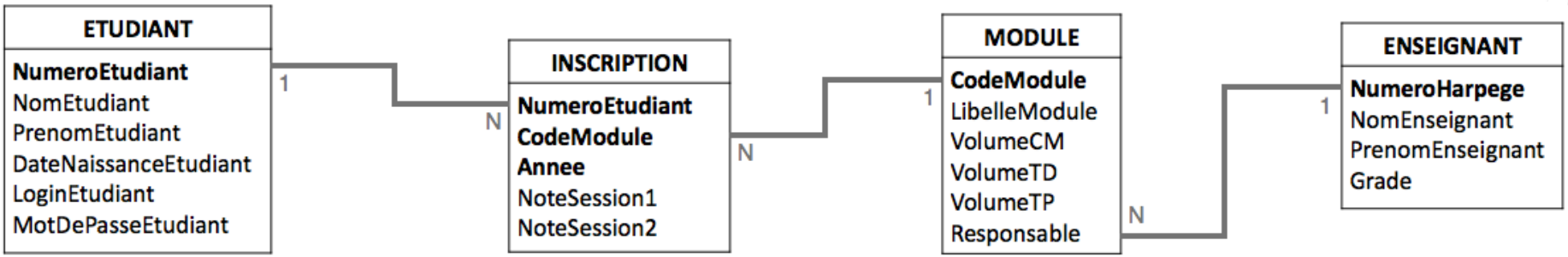
Module

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

1. Requêtes imbriquées :

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

Sous-requête indépendante : exemple



Modules qui ont le même responsable que le module 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

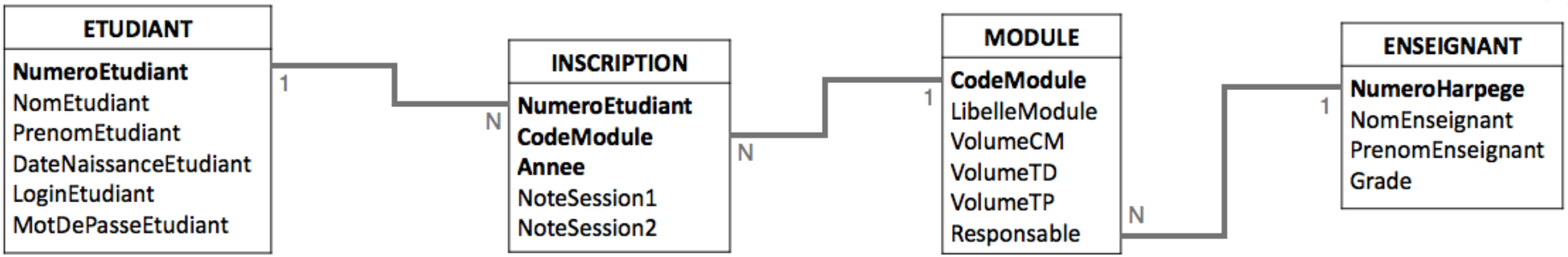
1. Requêtes imbriquées :

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

2. Sous-requête :

```
SELECT Responsable FROM Module WHERE CodeModule = 'BD_L1'
```
$\rightsquigarrow$ 7914

Sous-requête indépendante : exemple



Modules qui ont le même responsable que le module 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

Résultat :

CodeModule
BD_L1
TEST_M2
MCOO_L3

1. Requêtes imbriquées :

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

2. Sous-requête :

```
SELECT Responsable FROM Module WHERE CodeModule = 'BD_L1'
```
$\rightsquigarrow$ 7914

3. Requête principale :

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

Sous-requête indépdendante : type du résultat

Sous-requête indépdendante retournant un seul t-uplet

- Emploi des opérateurs de comparaison entre 2 t-uplets : $=, <>, <, <= \ldots$
- Exemple : étudiants qui ne sont pas "majors" de BD_L1 en 2017.

```
SELECT NumeroEtudiant FROM INSCRIPTION
WHERE CodeModule = 'BD_L1' AND Annee = 2017
AND NoteSession1 < (SELECT MAX(NoteSession1) FROM INSCRIPTION
                    WHERE CodeModule = 'BD_L1' AND Annee = 2017)
```

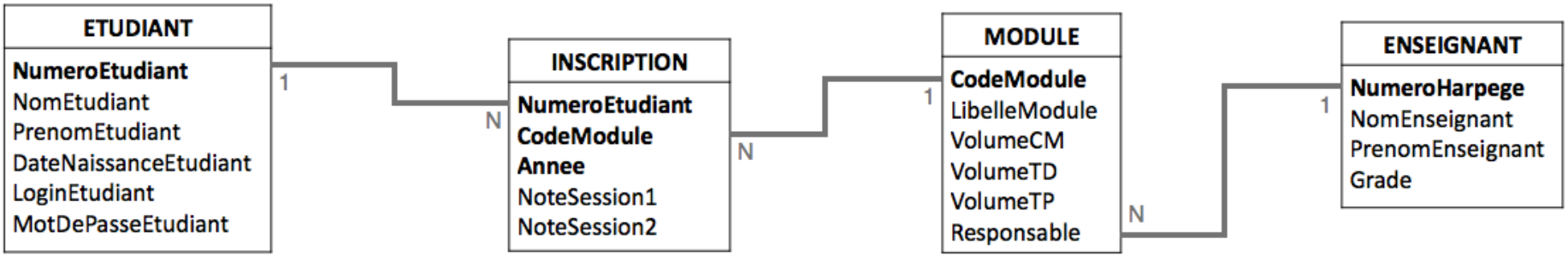
Sous-requête indép. retournant un ensemble de t-uplets

- Dans la syntaxe générique

```
SELECT ... FROM ... WHERE
Attr <OPERATEUR>  ( SELECT  ... FROM ... WHERE ... )
```

Emploi des opérateurs ensemblistes : `IN`, `NOT IN`, `ANY`, `ALL` ...

Sous-requête indép. ensembliste : IN, NOT IN



numéros des étudiants qui ont côtoyé l'étudiant 23794 en 2017

```
SELECT NumeroEtudiant FROM INSCRIPTION  WHERE Annee = 2017
AND CodeModule IN  (SELECT CodeModule FROM INSCRIPTION
                    WHERE Annee = 2017 AND NumeroEtudiant = 23794)
```

Numéro des étudiants qui n'ont jamais eu l'enseignant 7914

```
SELECT NumeroEtudiant FROM ETUDIANT WHERE
NumeroEtudiant NOT IN (SELECT NumeroEtudiant FROM INSCRIPTION
                       NATURAL JOIN MODULE WHERE Responsable = 7914)
```

Sous-requête indép. ensembliste : ANY, ALL

Definition

- Dans la syntaxe générique

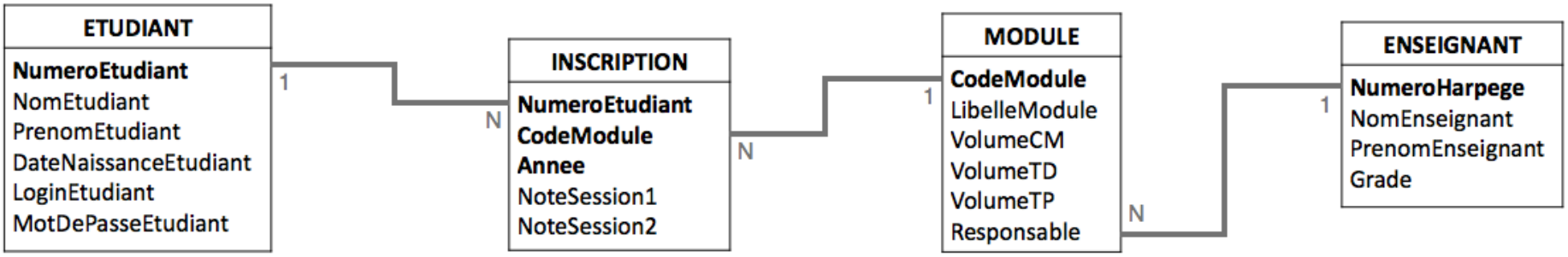
  ```
  SELECT ... FROM ... WHERE
  Attr <OPERATEUR>  (SELECT  ... FROM ... WHERE ...)
  ```

  <OPERATEUR> est composé

  - d'un opérateur op de comparaison entre 2 t-uplets : $=, <>, <, <= \ldots$
  - suivi de ANY ou ALL

- op ANY (SELECT ... FROM ... WHERE ...) : vrai si la condition utilisant op est vraie pour au moins un des résultats de la sous-requête.
  - Rem. : comparaison fausse si la sous-requête renvoie $\emptyset$
- op ALL (SELECT ... FROM ... WHERE ...) : vrai si la condition utilisant op est vraie pour tous les résultats de la sous-requête.
  - Rem. : comparaison vraie si la sous-requête renvoie $\emptyset$

Sous-req. indép. ensembliste : exemples ANY, ALL



Numéros des étudiants inscrits dans un des modules de l'enseignant 7914

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

Enseignant qui totalise le plus grand nombre de responsabilités

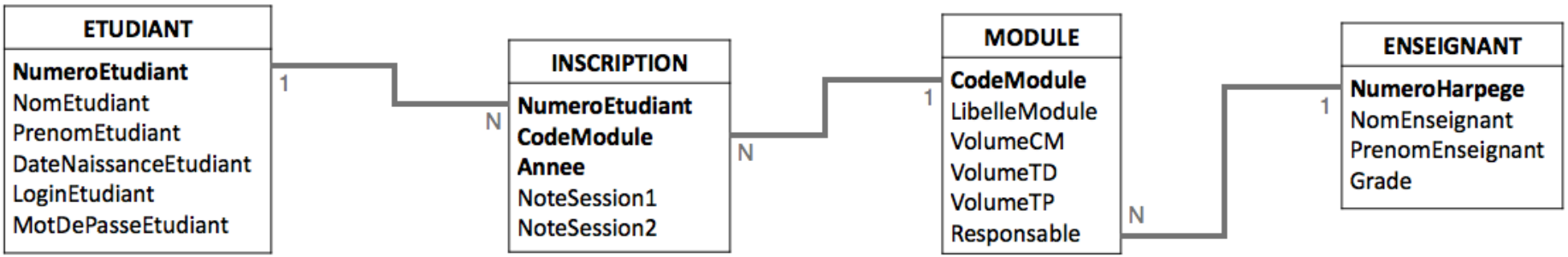
```
SELECT NumeroHarpege FROM ENSEIGNANT INNER JOIN MODULE
ON ENSEIGNANT.NumeroHarpege = MODULE.Responsable
GROUP BY NumeroHarpege
HAVING count(CodeModule) >= ALL (SELECT count(CodeModule)
                                 FROM MODULE
                                 GROUP BY Responsable)
```

Sous-requêtes synchronisées : définition

Définition

- 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 synchronisées : exemple



Numéro des étudiants dont la note en session 1 dans un des modules est supérieure à la moyenne de celui-ci

- Requête globale :

```
SELECT NumeroEtudiant FROM INSCRIPTION AS INSCR
WHERE NoteSession1 > (SELECT AVG(NoteSession1) FROM INSCRIPTION
                      WHERE CodeModule = INSCR.CodeModule)
```

- Sous-requête :

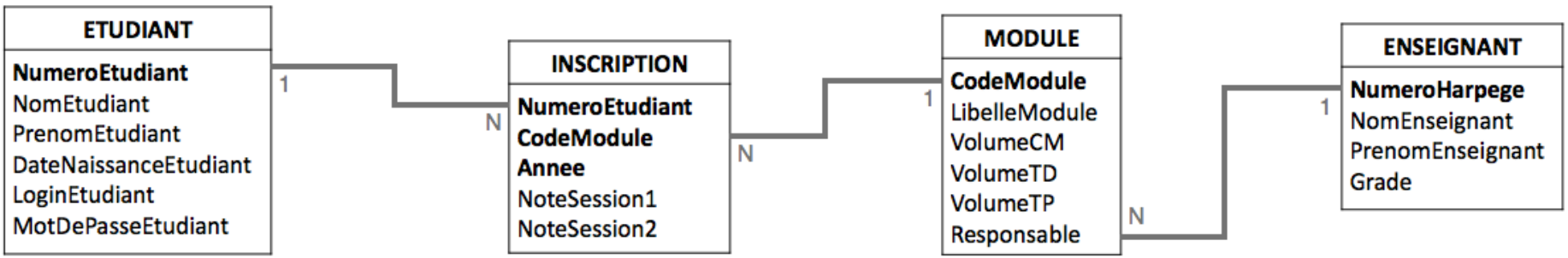
```
SELECT AVG(NoteSession1) FROM INSCRIPTION
WHERE CodeModule = INSCR.CodeModule
```

doit être exécutée à chaque exécution de la requête principale à cause de `CodeModule = INSCR.CodeModule`

Sous-requêtes synchronisées avec EXISTS

Définition

- Syntaxe : `SELECT ... WHERE [NOT] EXISTS (Sous-requête synchronisée)`
- `EXISTS` : vraie si la sous-requête synchronisée renvoie au moins un résultat



Numéros des étudiants qui n'ont été inscrits à aucun module en 2017

```
SELECT NumeroEtudiant FROM ETUDIANT
WHERE NOT EXISTS (SELECT * FROM INSCRIPTION
                  WHERE INSCRIPTION.Annee = 2017
                  AND INSCRIPTION.NumeroEtudiant = ETUDIANT.NumeroEtudiant)
```

Plan

Opérateurs ensemblistes : définition, syntaxe

Définition

SQL propose l'utilisation des 3 opérateurs ensemblistes classiques :

- union : `UNION`
- intersection : `INTERSECT`
- différence : `MINUS`

Ces opérateurs permettent de composer le résultat de deux requêtes `SELECT`

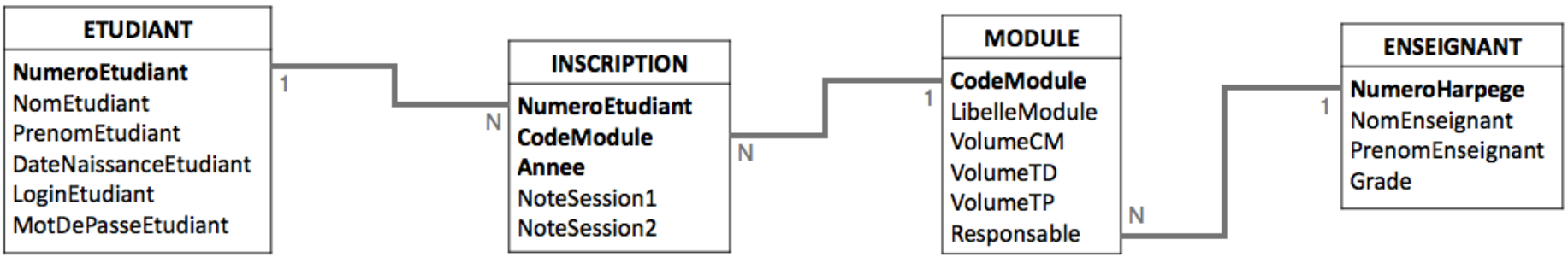
Syntaxe

- Syntaxe générale :

```
(SELECT ... FROM ...)
<Opérateur Ensembliste>
(SELECT ... FROM ...)
```

- Remarque : comme en algèbre relationnelle, les deux requêtes doivent retourner la même en-tête

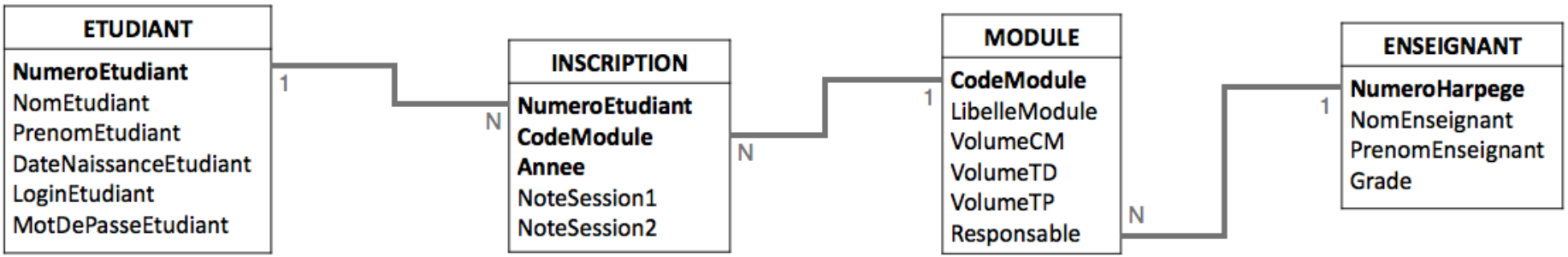
Opérateurs ensemblistes : exemple avec UNION



Noms, prénoms des enseignants MCF ou resp. d'un module sans TD

```
SELECT NomEnseignant, PrenomEnseignant FROM ENSEIGNANT WHERE Grade='MCF'
UNION
SELECT NomEnseignant, PrenomEnseignant FROM ENSEIGNANT INNER JOIN MODULE
ON NumeroHarpage = Responsable WHERE VolumeTD = 0
```

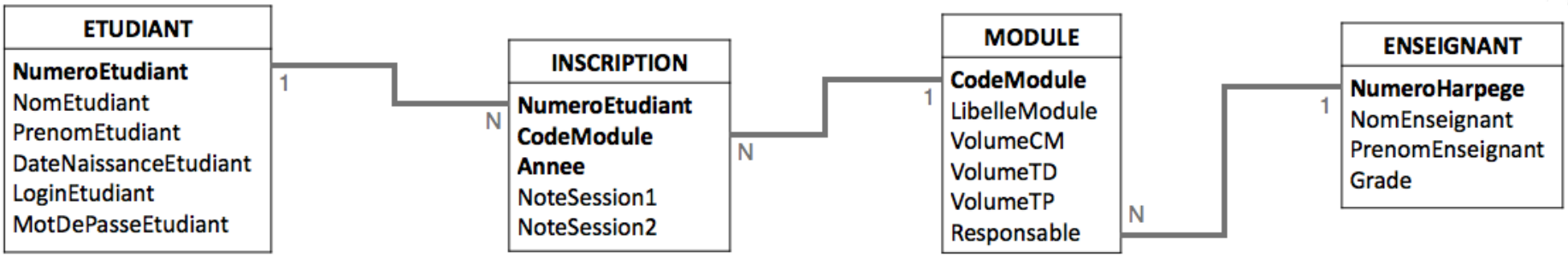
Opérateurs ensemblistes : exemple avec MINUS



Noms, prénoms des enseignants MCF mais qui ne sont pas responsables d'un module sans TD

```
SELECT NomEnseignant, PrenomEnseignant FROM ENSEIGNANT WHERE Grade='MCF'
MINUS
SELECT NomEnseignant, PrenomEnseignant FROM ENSEIGNANT INNER JOIN MODULE
ON NumeroHarpage = Responsable WHERE VolumeTD = 0
```

Opérateurs ensemblistes : expl. avec INTERSECT



Noms, prénoms des enseignants MCF et resp. d'un module sans TD

```
SELECT NomEnseignant, PrenomEnseignant FROM ENSEIGNANT WHERE Grade='MCF'
INTERSECT
SELECT NomEnseignant, PrenomEnseignant FROM ENSEIGNANT INNER JOIN MODULE
ON NumeroHarpage = Responsable WHERE VolumeTD = 0
```

Remarque

- Opérateurs ensemblistes : pas toujours implantés dans les SGBD
  - `INTERSECT` absent par exemple en MySQL

Intersection avec une sous-requête

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

Plan

La division : rappels et définition en logique

Division de $R_1 \subset A_1 \times \cdots \times A_m \times B_1 \times \cdots \times B_n$ par $R_2 \subset B_1 \times \cdots \times B_n$

- La plus grande relation Q dont le produit $Q \times R_2$ est inclus dans R_1.

$$\begin{array}{c|c} R_1 & R_2 \\ \hline \ldots & Q \\ R & \end{array} \quad \text{avec } R_1 = Q \times R_2 \ \texttt{union} \ R$$

Exemple : Numéros des étudiants inscrits dans tous les modules

INSCRIPTION =

NoEtudiant*	CodeModule*
23794	PROG_L1
32911	BD_L1
32911	PROG_L1
32911	TEST_L3
34812	PROG_L1
34812	BD_L1
34812	TEST_L3

MOD1

CodeModule*
BD_L1
PROG_L1
TEST_L3

INSCRIPTION DIV MOD1

NoEtudiant*
32911
34812

Traduction ensembliste

$$Q = \{(a_1, \ldots, a_m) | (a_1, \ldots, a_m) \in [A_1 \ldots, A_m]R_1 \text{ et } \forall (b_1, \ldots, b_n) \in R_2 \, (a_1, \ldots, a_m, b_1, \ldots, b_n) \in R_1\}$$

Division : 2 sous-req. NOT EXISTS imbriquées

Division de $R_1 \subset A_1 \times \cdots \times A_m \times B_1 \times \cdots \times B_n$ par $R_2 \subset B_1 \times \cdots \times B_n$

$$Q = \{(a_1, \ldots, a_m) | (a_1, \ldots, a_m) \in [A_1 \ldots, A_m]R_1 \text{ et } \forall (b_1, \ldots, b_n).(b_1, \ldots, b_n) \in R_2 \Rightarrow (a_1, \ldots, a_m, b_1, \ldots, b_n) \in R_1\}$$

$$\Leftrightarrow$$

$$Q = \{(a_1, \ldots, a_m) | (a_1, \ldots, a_m) \in [A_1 \ldots, A_m]R_1 \text{ et } \neg (\exists (b_1, \ldots, b_n).(b_1, \ldots, b_n) \in R_2 \text{ et } (a_1, \ldots, a_m, b_1, \ldots, b_n) \notin R_1)\}$$

$$\Leftrightarrow$$

$$Q = \{(a_1, \ldots, a_m) | (a_1, \ldots, a_m) \in [A_1 \ldots, A_m]R_1 \text{ et } \nexists (b_1, \ldots, b_n).(b_1, \ldots, b_n) \in R_2 \text{ et } (a_1, \ldots, a_m, b_1, \ldots, b_n) \notin R_1\}$$

Traduction SQL $\rightsquigarrow$ formule à particulariser

```
SELECT A1,...,Am FROM R1          WHERE
   NOT EXISTS (SELECT B1,...,Bn FROM R2          WHERE
      NOT EXISTS (SELECT A1,...,Am,B1,...,Bn FROM R1 AS R1P
                  WHERE R1.A1 = R1P.A1 AND ... R1.Am = R1P.Am AND
                        R2.B1 = R1P.B1 AND ... R2.Bn = R1P.Bn))
```

Division : 2 sous-req. NOT EXISTS imbriquées

Division de $R_1 \subset A_1 \times \cdots \times A_m \times B_1 \times \cdots \times B_n$ par $R_2 \subset B_1 \times \cdots \times B_n$

$$Q = \{(a_1, \ldots, a_m) | (a_1, \ldots, a_m) \in [A_1 \ldots, A_m]R_1 \text{ et } \forall (b_1, \ldots, b_n).(b_1, \ldots, b_n) \in R_2 \Rightarrow (a_1, \ldots, a_m, b_1, \ldots, b_n) \in R_1\}$$

$$\Leftrightarrow$$

$$Q = \{(a_1, \ldots, a_m) | (a_1, \ldots, a_m) \in [A_1 \ldots, A_m]R_1 \text{ et } \neg (\exists (b_1, \ldots, b_n).(b_1, \ldots, b_n) \in R_2 \text{ et } (a_1, \ldots, a_m, b_1, \ldots, b_n) \notin R_1)\}$$

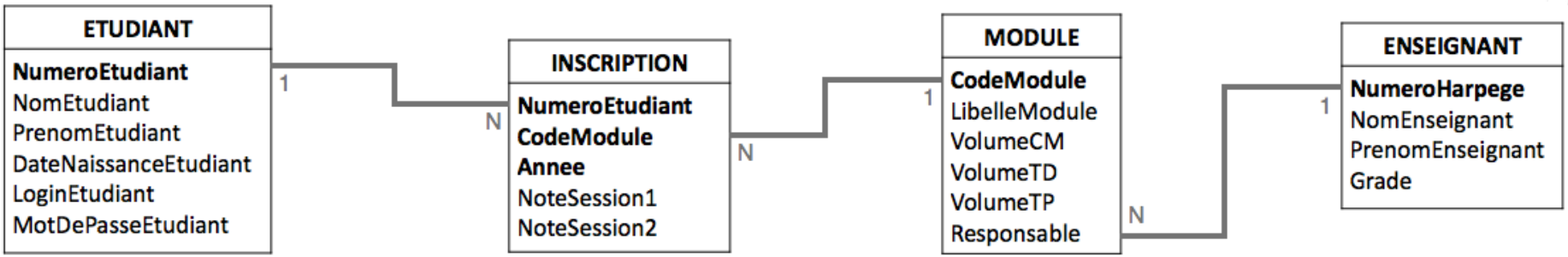
$$\Leftrightarrow$$

$$Q = \{(a_1, \ldots, a_m) | (a_1, \ldots, a_m) \in [A_1 \ldots, A_m]R_1 \text{ et } \nexists (b_1, \ldots, b_n).(b_1, \ldots, b_n) \in R_2 \text{ et } (a_1, \ldots, a_m, b_1, \ldots, b_n) \notin R_1\}$$

Traduction SQL $\rightsquigarrow$ formule à particulariser

```
SELECT A1,...,Am FROM R1 AS T1  WHERE
   NOT EXISTS (SELECT B1,...,Bn FROM R2 AS T2  WHERE
      NOT EXISTS (SELECT A1,...,Am,B1,...,Bn FROM R1 AS T1P
                  WHERE T1.A1 = T1P.A1 AND ... T1.Am = T1P.Am AND
                        T2.B1 = T1P.B1 AND ... T2.Bn = T1P.Bn))
```

Division : exemple



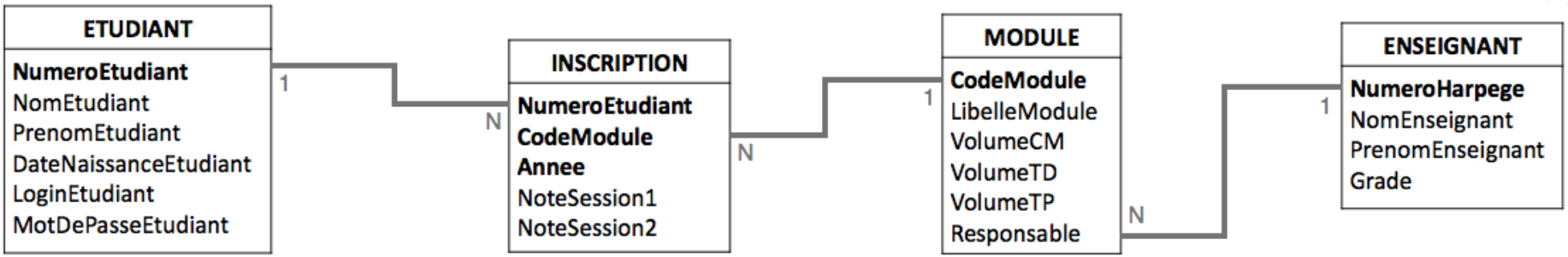
Formule à particulariser

```
SELECT A1,...,Am FROM R1 AS T1  WHERE
   NOT EXISTS (SELECT B1,...,Bn FROM R2 AS T2  WHERE
      NOT EXISTS (SELECT A1,...,Am,B1,...,Bn FROM R1 AS T1P
                  WHERE T1.A1 = T1P.A1 AND ... T1.Am = T1P.Am AND
                        T2.B1 = T1P.B1 AND ... T2.Bn = T1P.Bn))
```

Numéros des étudiants inscrits dans tous les modules

- R1 dans INSCRIPTION : A1 = NumeroEtudiant, B1 = CodeModule
- R2 dans MODULE : B1 = CodeModule

Division : exemple



Formule à particulariser

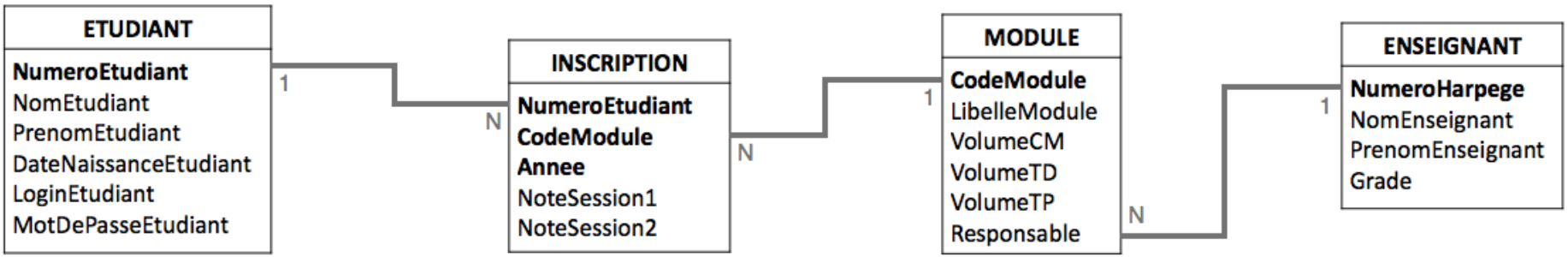
```
SELECT A1,...,Am FROM R1 AS T1  WHERE
   NOT EXISTS (SELECT B1,...,Bn FROM R2 AS T2  WHERE
      NOT EXISTS (SELECT A1,...,Am,B1,...,Bn FROM R1 AS T1P
                  WHERE T1.A1 = T1P.A1 AND ... T1.Am = T1P.Am AND
                        T2.B1 = T1P.B1 AND ... T2.Bn = T1P.Bn))
```

Numéros des étudiants inscrits dans tous les modules

- R1 dans INSCRIPTION : A1 = NumeroEtudiant, B1 = CodeModule
- R2 dans MODULE : B1 = CodeModule

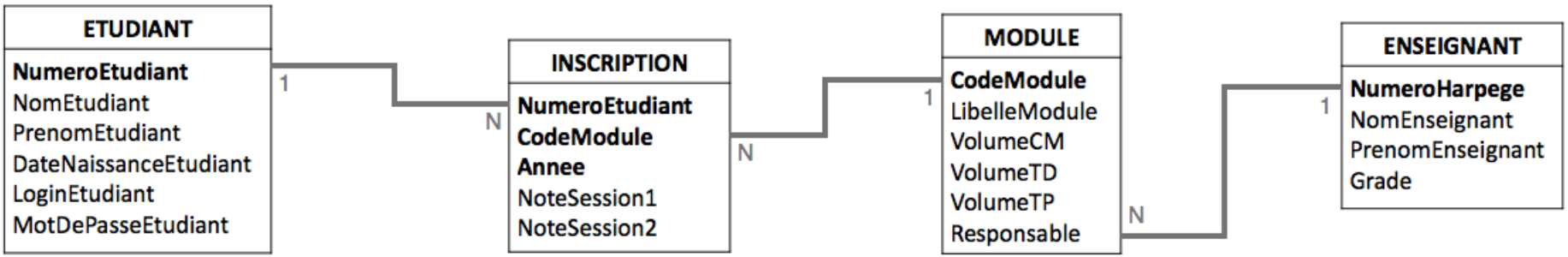
```
SELECT NumeroEtudiant FROM INSCRIPTION AS T1 WHERE
   NOT EXISTS (SELECT CodeModule FROM MODULE AS T2 WHERE
      NOT EXISTS (SELECT NumeroEtudiant, CodeModule FROM INSCRIPTION AS T1P
                  WHERE T1.NumeroEtudiant = T1P.NumeroEtudiant AND
                        T2.CodeModule = T1P.CodeModule))
```

Division : à vous de jouer



Numéros des étudiants inscrits toutes les années entre 2012 et 2017

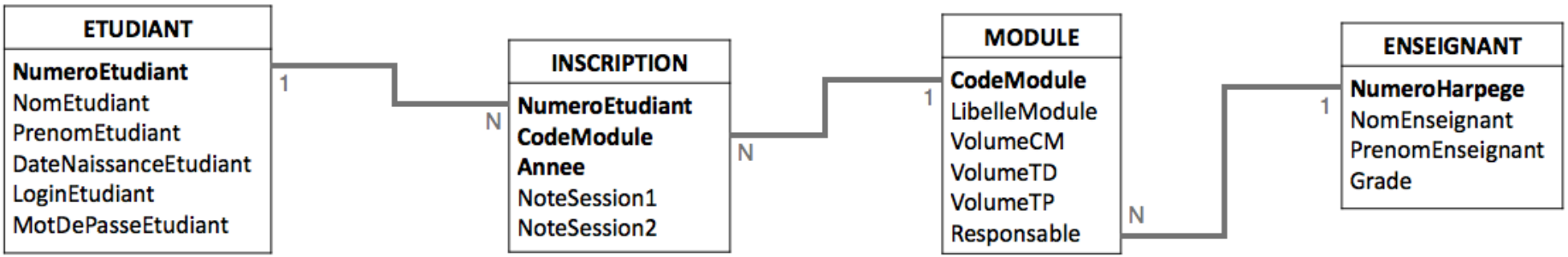
Division : à vous de jouer



Numéros des étudiants inscrits toutes les années entre 2012 et 2017

- R1 dans INSCRIPTION : A1 = NumeroEtudiant, B1 = Annee
- R2 dans INSCRIPTION : B1 = Annee (entre 2012 et 2017)

Division : à vous de jouer

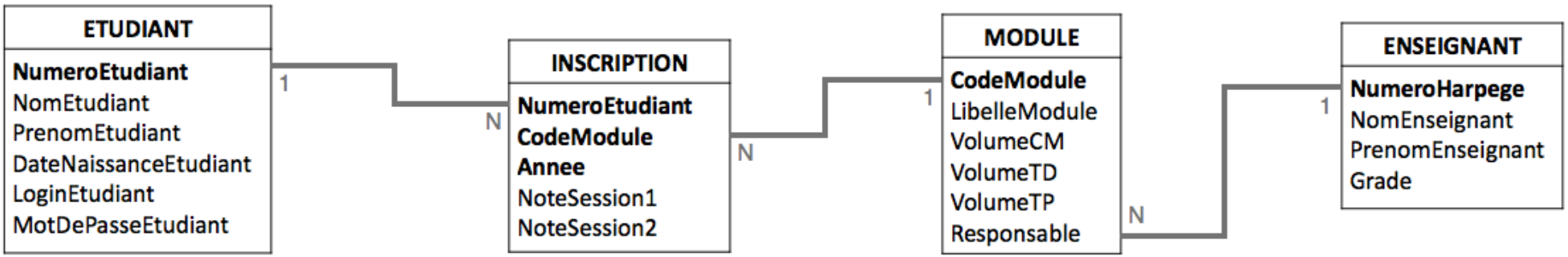


Numéros des étudiants inscrits toutes les années entre 2012 et 2017

- R1 dans INSCRIPTION : A1 = NumeroEtudiant, B1 = Annee
- R2 dans INSCRIPTION : B1 = Annee (entre 2012 et 2017)

```
SELECT A1,...,Am FROM R1 AS T1  WHERE
   NOT EXISTS (SELECT B1,...,Bn FROM R2 AS T2  WHERE
      NOT EXISTS (SELECT A1,...,Am,B1,...,Bn FROM R1 AS T1P
                  WHERE T1.A1 = T1P.A1 AND ... T1.Am = T1P.Am AND
                        T2.B1 = T1P.B1 AND ... T2.Bn = T1P.Bn))
```

Division : à vous de jouer

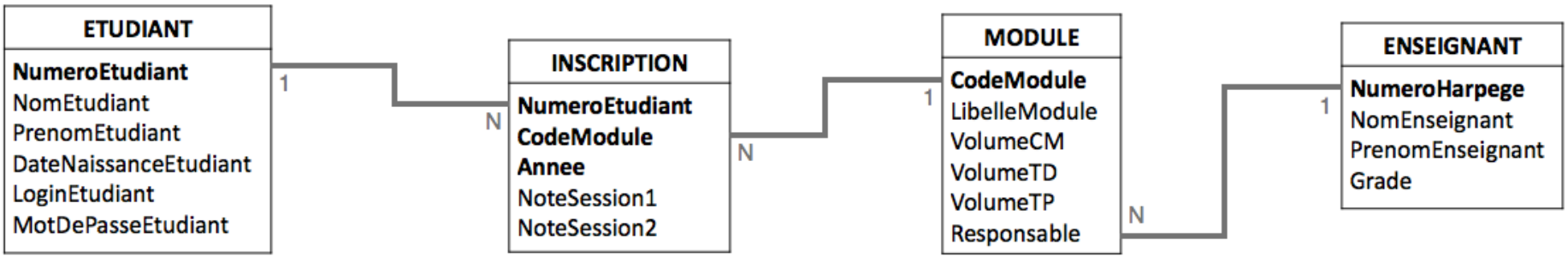


Numéros des étudiants inscrits toutes les années entre 2012 et 2017

- R1 dans INSCRIPTION : A1 = NumeroEtudiant, B1 = Annee
- R2 dans INSCRIPTION : B1 = Annee (entre 2012 et 2017)

```
SELECT NumeroEtudiant FROM INSCRIPTION AS T1  WHERE
   NOT EXISTS (SELECT B1,...,Bn FROM R2 AS T2  WHERE
      NOT EXISTS (SELECT NumeroEtudiant,B1,...,Bn FROM INSCRIPTION AS T1P
                  WHERE T1.NumeroEtudiant = T1P.NumeroEtudiant AND
                        T2.B1 = T1P.B1 AND ... T2.Bn = T1P.Bn))
```

Division : à vous de jouer



Numéros des étudiants inscrits toutes les années entre 2012 et 2017

- R1 dans INSCRIPTION : A1 = NumeroEtudiant, B1 = Annee
- R2 dans INSCRIPTION : B1 = Annee (entre 2012 et 2017)

```
SELECT NumeroEtudiant FROM INSCRIPTION AS T1  WHERE
   NOT EXISTS (SELECT Annee FROM INSCRIPTION AS T2  WHERE
      T2.Annee BETWEEN 2012 AND 2017 AND
      NOT EXISTS (SELECT NumeroEtudiant,Annee FROM INSCRIPTION AS T1P
                  WHERE T1.NumeroEtudiant = T1P.NumeroEtudiant AND
                        T2.Annee = T1P.Annee))
```

Plan

Fonctions et expressions

Rappel sur les fonctions de groupe

- Mesures sur des (regroupement de) valeurs d'un attribut d'une relation
- Applicables sur un attribut (verticalement), pas sur une ligne

Expressions et fonctions

- Mesures sur des valeurs d'un seul enregistrement, utilisant plusieurs attributs
- Applicables sur un enregistrement (horitzontalement), sur une ligne

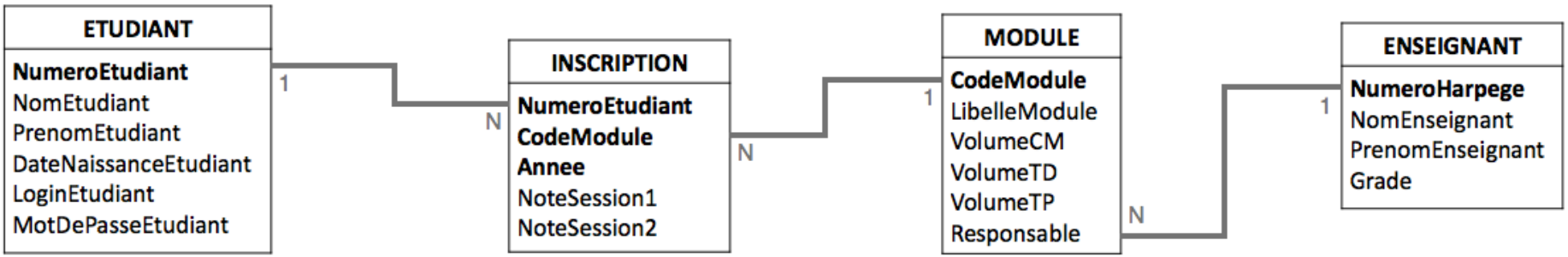
Fonctions et expressions

Où utiliser ces fonctions et expressions ?

A utilisées dans des instructions :

- `SELECT`
- `WHERE`
- `GROUP BY`
- `ORDER BY`

Fonctions et expressions : exemple avec SELECT



Moyenne du temps passé dans un module sur 14 semaines

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

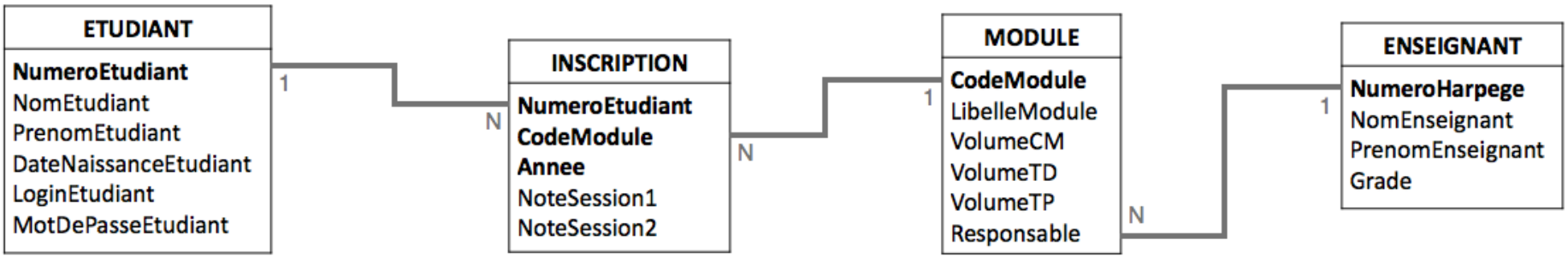
MODULE

CodeModule	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 : exemple avec WHERE



Modules dont le volume horaire supérieur à 40 heures par semestre

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

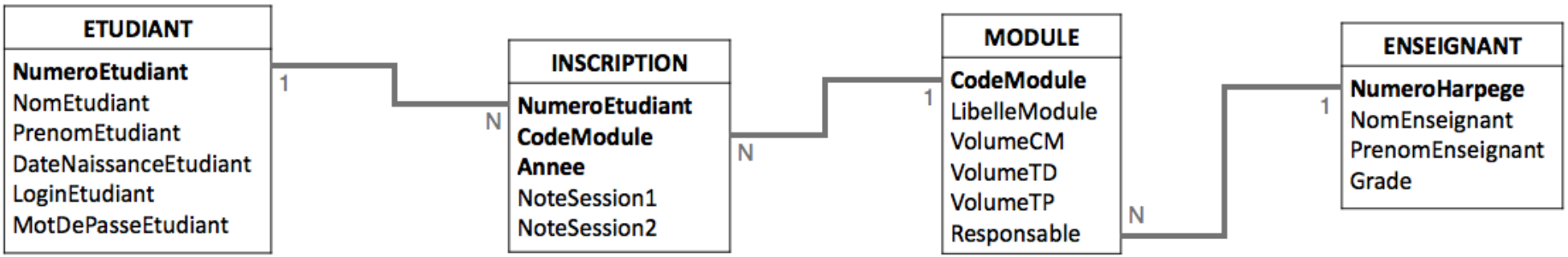
MODULE

CodeModule	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 : expl. avec ORDER BY



Tri des modules par ordre décroissant de volume horaire global

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

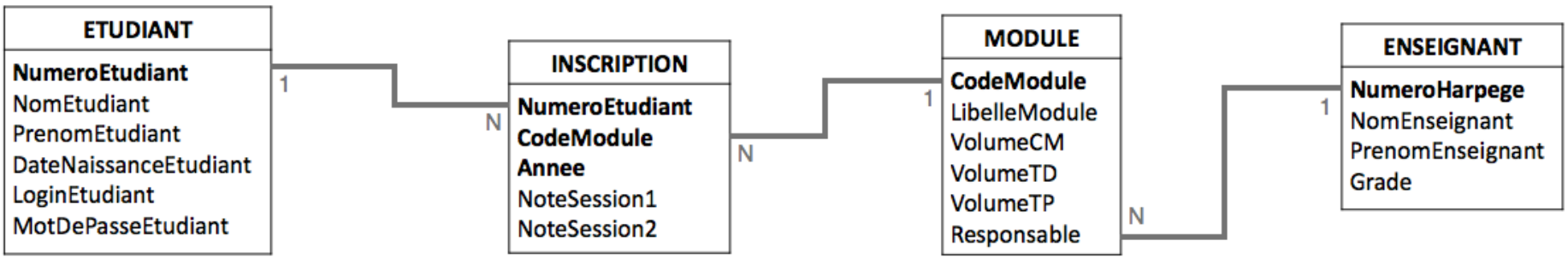
MODULE

CodeModule	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 : expl. avec GROUP BY



Moyenne par module en fonction de l'année de naissance

```
SELECT CodeModule, YEAR(DateNaissanceEtudiant) AS ANNEE,
       AVG(NoteSession1) AS Moyenne
FROM INSCRIPTION NATURAL JOIN ETUDIANT
GROUP BY CodeModule, YEAR(DateNaissanceEtudiant)
```

Types Fonctions et expressions

Classement des différentes fonctions

- Fonctions numériques
  - valeur absolue `ABS()`, puissance `POW()`,...
- Fonctions sur les chaînes de caractères
  - longueur d'une chaîne `LENGTH()`, ...
  - transformation en majuscules `UPPER()`, ...
  - extraction d'une sous-chaîne `SUBSTRING()`,...
- Fonctions sur les dates
  - addition/soustraction de dates `DATEDIFF()`, des jours à une date `DATE_ADD(expr,days)`...
  - fonctions retournant la date du système `CURDATE()`, le jour de la semaine `DAYNAME(date)`,...

Plan

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.

Plan

Ajout de données avec VALUES : définition

Syntaxe

```
INSERT INTO Table [(Att1, Att2,...)] VALUES (x1, x2,...), (y1, y2,...),...
```

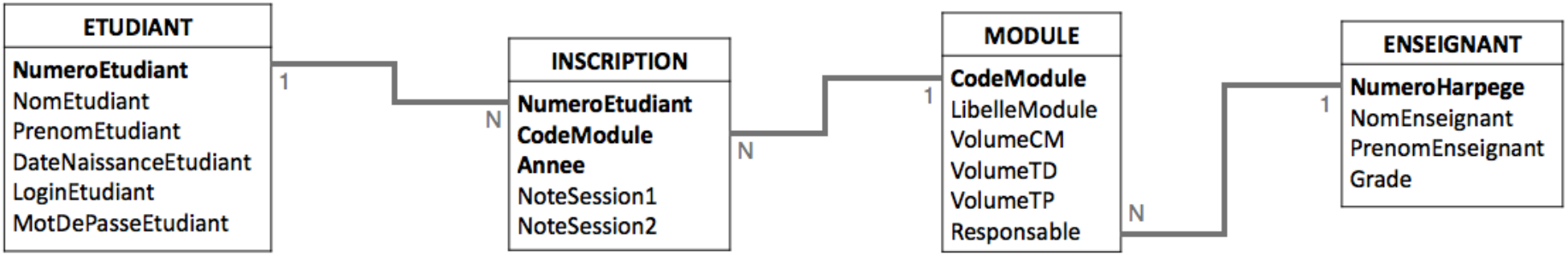
Liste facultative d'attributs (Att1, Att2, ...)

- Restreindre les attributs à renseigner dans le nouvel enregistrement
- Attributs non renseignés : valeur par défaut (si définie), NULL sinon
- Si absente : renseigner tous les attributs de manière ordonnée

Liste obligatoire de valeurs (x1, x2,...), (y1, y2, ...), ...

- Ordre des valeurs : même que celui des attributs (`xn` $\leftrightarrow$ `Attn` par ex.).
- Chaque valeur `xn` : doit appartenir au domaine de `Attn`
- Insertion multiples : t-uplets séparés des virgules

Ajout de données avec VALUES : exemple



Exemple d'insertions d'enseignants

ENSEIGNANT

NumeroHarpege	NomEnseignant	PrenomEnseignant	Grade
7358	Féléa	Violeta	MCF
7914	Dadeau	Frédéric	MCF

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

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

ENSEIGNANT

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

Ajout de données issues de SELECT : définition

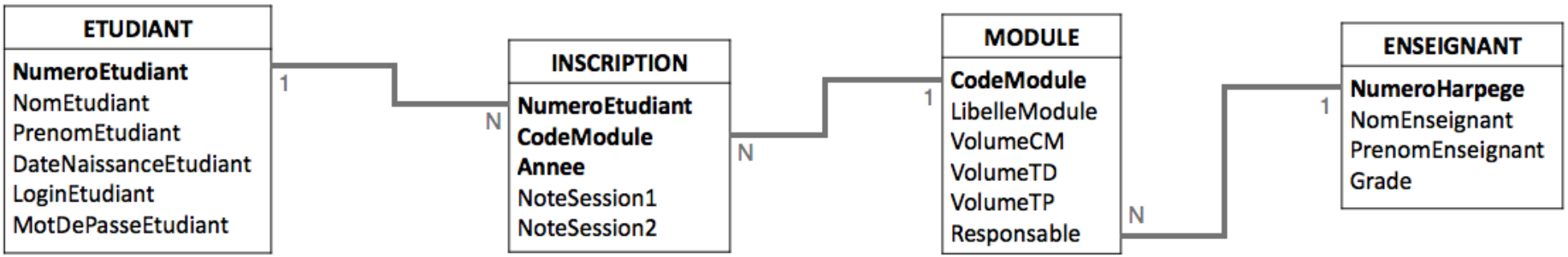
Syntaxe

```
INSERT INTO Table [(Att1, Att2, ...)]  SELECT Att1, Att2, ...
```

Remarques

- t-uplets retournés par SELECT : “compatibles” et dans le bon ordre
- Instruction SELECT : sans instruction ORDER BY.

Ajout de données issues de SELECT : exemple



Réinscription en 2018 des étudiants dans les modules non validés de 2017

```
INSERT INTO INSCRIPTION (NumeroEtudiant, CodeModule, Annee)
   SELECT NumeroEtudiant, CodeModule, Annee + 1 FROM INSCRIPTION
   WHERE Annee = 2017 AND
   NoteSession2 IS NOT NULL AND NoteSession2 < 10
```

Plan

Modification de données avec UPDATE : définition

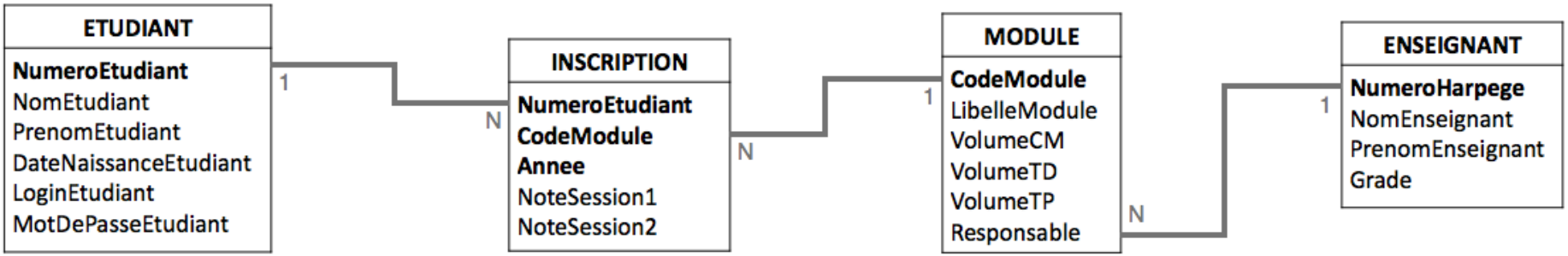
Syntaxe

```
UPDATE Table  SET Att1 = Val1, Att2 = Val2, ... WHERE Condition
```

Remarque

- Mise à jour des t-uplets de `Table` satisfaisant la condition du WHERE
- Valeurs modifiées : celles des attributs listés `Att1`, `Att2`

Modification de données avec UPDATE : exemple



'azerty' : nouveau mot passe de l'étudiant 23794

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

- Remarque : un seul *t*-uplet modifié

Responsabilités assumées par 7914 : assumées par 32598

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

- Remarque : potentiellement, plusieurs *t*-uplets modifiés

Plan

Suppression avec DELETE : définition

Syntaxe

```
DELETE FROM Table WHERE Condition
```

Remarque

- Tous les t-uplets de `Table` qui satisfont la condition du `WHERE` : supprimés

INSERT, UPDATE, DELETE : intégrité

Requêtes INSERT, UPDATE, DELETE violant l'intégrité référentielle

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

vont toujours produire des erreurs renvoyées par le SGBD !

Plan

Data Definition Language

Bilan sur ce qui est (ou devrait être) su

- Concernant les enregistrements dans une base de données, on sait :
  - en sélectionner (`SELECT`)
  - en ajouter (`INSERT`)
  - en modifier (`UPDATE`)
  - en supprimer (`DELETE`)
- Considère donc comme acquis le modèle de la base de données. . .

Objectif : construire le modèle de la base de données

- les tables
- les index

à créer (`CREATE`), modifier (`ALTER`), ou supprimer (`DROP`).

Plan

Création d'une table : CREATE TABLE

Description générale

- des attributs de la table (leur nom, leur type)
- des contraintes d'intégrité de cette table

Syntaxe basique

```
CREATE TABLE NOM_TABLE
    (NomAttribut1 Type1, NomAttribut2 Type2, ... )
```

- Type1, Type2 : types dépendant du SGBD (MySQL $\neq$ ORACLE...)
- Types communs : `INTEGER`, `CHAR(length)`, `DATE`, `ENUM(v1, v2, ...)`

Exemple : création de la table INTERVENANT listant les enseignants intervenant dans un module

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

Créat° & rempliss. : CREATE TABLE...AS SELECT

Syntaxe de création et remplissage initial d'une table

```
CREATE TABLE NOM_TABLE
    (NomAttribut1 Type1, NomAttribut2 Type2, ... )
    AS SELECT ...
```

Remarques sur cette syntaxe

- Noms des attributs de la table créée : ceux des attributs du SELECT
- $\rightsquigarrow$ SELECT avec des expressions : renommage des colonnes avec AS

Créat° d'INTERVENANT et initialisat° avec les resp. de modules

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

Contraintes d'intégrité sur les attributs

Syntaxe

```
CREATE TABLE NOMTABLE
(Att1 Type1 [Contrainte1 ... ContrainteN], ...)
```

Différents types de contraintes sur les attributs

- `Att Type DEFAULT val` : `Att` vaut `val` si non renseigné dans `INSERT`
- `Att Type NOT NULL` : `Att` ne peut pas prendre la valeur `NULL`
- `Att ... UNIQUE` : deux occurr. d'une même valeur interdit (clé candidate)
- `Att ... CHECK (condition)` : condition à vérifier pour la valeur de `Att`

Exemple de création de la table ETUDIANT

```
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')
```

Contraintes d'intégrité sur les tables

Syntaxe : après la définition des attributs

```
CREATE TABLE NOMTABLE
(Att1 Type1 [CA11 ... ],...,
Attk Typek [CAk1 ...],
[CONSTRAINT nom_contrainte_table_1] CONTRAINTE_TABLE_1,...,
[CONSTRAINT nom_contrainte_table_1] CONTRAINTE_TABLE_1)
```

Remarques

- `CONSTRAINT nom_contrainte_table` : nommage optionnel verbeux
- `CONTRAINTE_TABLE` : peuvent concerner les clefs
  - `PRIMARY KEY (Att1, Att2,...)` : définit° de la clé primaire
    - Une seule contrainte `PRIMARY KEY` par table
    - `NOT NULL` implicite pour chaque attribut de la clef
  - `FOREIGN KEY (Atti) REFERENCES NOMTABLE(Atti)`
    `[ON UPDATE REF_01] [ON DELETE REF_02]`
    - Lien entre `NOMTABLE.Atti` et `Atti`
    - `ON UPDATE` et `ON DELETE` suivis par la politique de MAJ : `RESTRICT`, `CASCADE`,...

Expl. de créatº de table avec des contraintes

Exemple de création de la table ETUDIANT

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

Modification de la structure d'une table

3 types de modification de la structure d'une table

- l'ajout d'un attribut : `ADD`
- la modification d'un attribut existant : `MODIFY`
- la suppression d'un attribut : `DROP`

Modification de la structure d'une table

Syntaxe

`ALTER TABLE NOMTABLE instruction` : `instruction` est

- `ADD (Att1 Type1 [Contraintes], Att2 ...)` : ajout d'attributs (cf. `CREATE TABLE`)
- `MODIFY (Att1 Type1 [Contraintes], Att2 ...)` : modification du type et des contraintes
- `DROP Att1, DROP Att2...` : suppression des attributs

Exemples de modifications

- Ds INTERVENANT : +attribut pr. les heures de (module,enseignant)

```
ALTER TABLE INTERVENANT ADD (NbHeures INTEGER CHECK (NbHeures > 0))
```

- Modification d'ETUDIANT : prise en compte des noms longs (100)

```
ALTER TABLE ETUDIANT MODIFY NomEtudiant char(100) NOT NULL
```

- Ds ETUDIANT : suppression des dates de naissance

```
ALTER TABLE ETUDIANT DROP DateNaissanceEtudiant
```

Suppression d'une table

Syntaxe

```
DROP TABLE NOMTABLE
```

Remarque

- DROP d'une table avec un attribut référencé dans une autre : refusé

Exemple : suppression de la table INTERVENANT

```
DROP TABLE INTERVENANT
```

Plan

Index : motivations

Accès lent à certains enregistrements d'une table

```
SELECT * FROM ETUDIANT WHERE NomEtudiant = ``Smolinski''
```

$\rightsquigarrow$ parcours séquentiel du SGBD de tous les enregistrements jusqu'à `NomEtudiant = 'Smolinski'` ? Non !

Accès rapide à certains enregistrements d'une table

- Une structure stockant les adresses des enregistrements organisée en fonction des noms :
- Par dichotomie sur les noms triés par ordre alphabétique : en $O(\log_2(n))$:

Nb. tuples	Nb. itérations	Gain (%)
5	3	40
10	4	60
50	6	88
100	7	93
500	9	98,2
1 000	10	99
5 000	13	99,74
10 000	14	99,86

Index : définition et réflexions

Définition

Un index : une structure entretenue automatiquement, permettant de localiser facilement des enregistrements dans un fichier selon un (ensemble d') attribut(s).

Conséquences

- Lors d'un `CREATE TABLE` : déclaration des attributs composant l'index
- Si pas d'index déclaré : indexation selon la clef primaire
- A chaque `INSERT`, `UPDATE`, `DELETE` : mise à jour de l'index par le SGBD
- Requêtes `SELECT` : transparentes à l'existence d'un index
- Intéressant de créer un index si :
  - Sélections récurrentes sur un ensemble d'attributs
  - Jointures récurrentes entre deux tables p.r. à un ensemble d'attributs
- Contre-productif de créer un index si :
  - la table contient peu d'enregistrements
  - l'attribut ou l'ensemble d'attributs prennent peu de valeurs différentes

Index : syntaxe et exemple

Syntaxe : après la définition des attributs

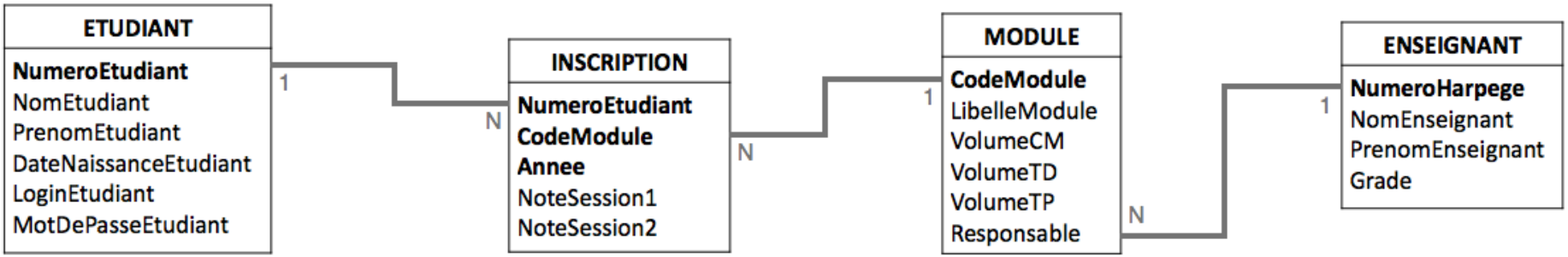
```
CREATE TABLE NOMTABLE
(Att1 Type1 [CA11 ... ],...,
[CONSTRAINT nom_contrainte_table_1] CONTRAINTE_TABLE_1,...,
INDEX [UNIQUE] nom_index (Attj,Attk,...),...)
```

- `Attj`, `Attk` sont les composants de l'index
- Option `UNIQUE` : valeurs uniques pour `(Attj,Attk,...)`

Index sur le nom et prénom des étudiants

```
CREATE TABLE ETUDIANT
  (NumeroEtudiant INTEGER NOT NULL,
   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',
   CONSTRAINT Pk_no_etudiant PRIMARY KEY (NumeroEtudiant),
   INDEX idx_nom_prenom (NomEtudiant,PrenomEtudiant))
```

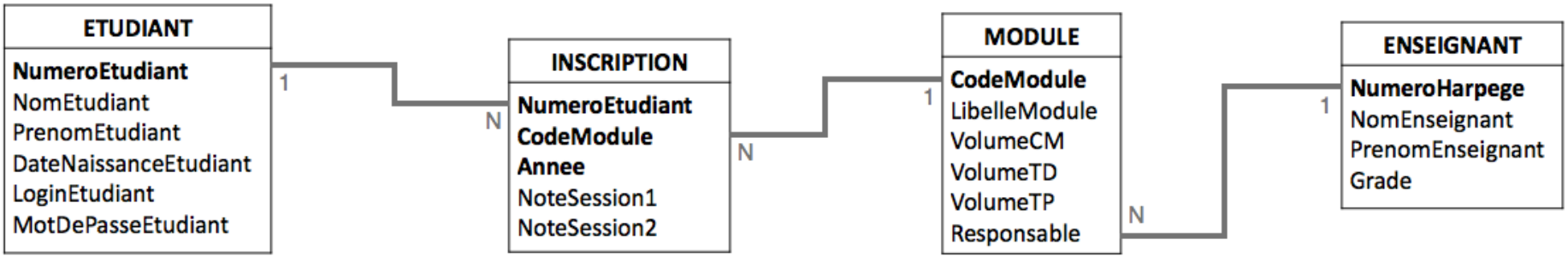
Création : à vous de jouer



Préliminaires

- Création ordonnées de :
- Index :
- Politique d'intégrité référentielle :

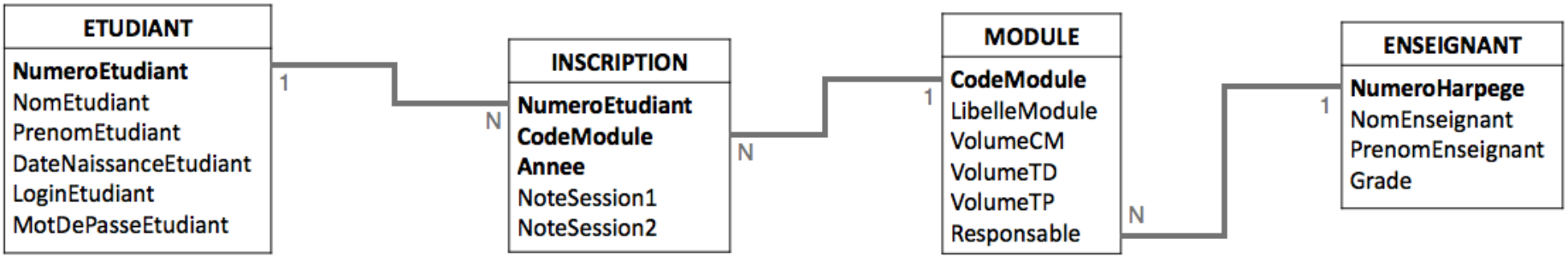
Création : à vous de jouer



Préliminaires

- Création ordonnées de : `ETUDIANT`, `ENSEIGNANT`, `MODULE`, `INSCRIPTION`
- Index :
- Politique d'intégrité référentielle :

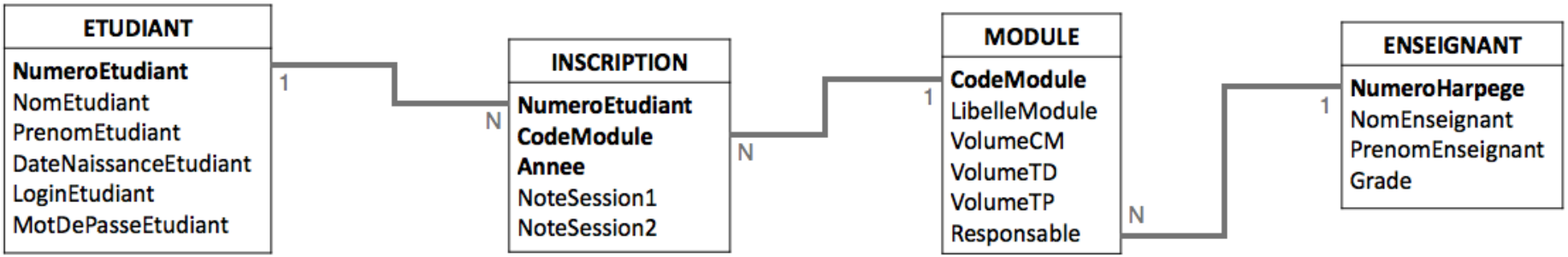
Création : à vous de jouer



Préliminaires

- Création ordonnées de : `ETUDIANT`, `ENSEIGNANT`, `MODULE`, `INSCRIPTION`
- Index : `ETUDIANT(NomEtudiant,PrenomEtudiant)` `ENSEIGNANT(NomEnseignant,PrenomEnseignant)`, `MODULE(LibelleModule)`
- Politique d'intégrité référentielle :

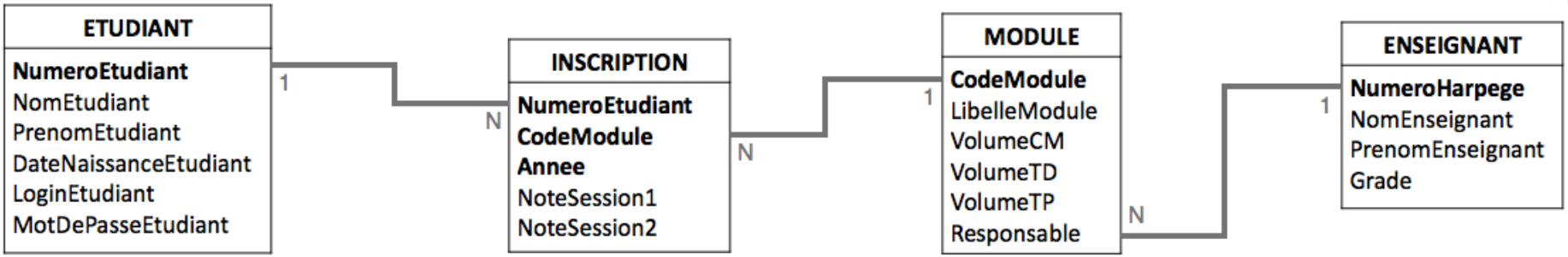
Création : à vous de jouer



Préliminaires

- Création ordonnées de : `ETUDIANT`, `ENSEIGNANT`, `MODULE`, `INSCRIPTION`
- Index : `ETUDIANT(NomEtudiant,PrenomEtudiant)` `ENSEIGNANT(NomEnseignant,PrenomEnseignant)`, `MODULE(LibelleModule)`
- Politique d'intégrité référentielle : `CASCADE` sur tous les liens

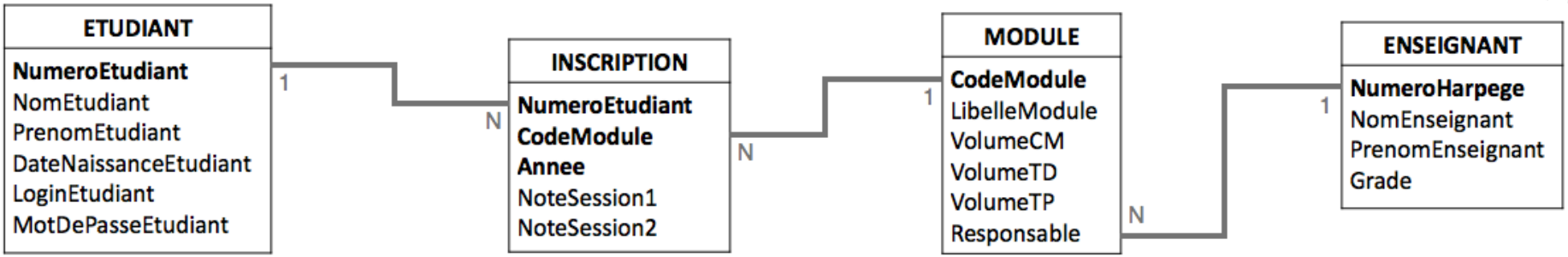
Création : à vous de jouer



ENSEIGNANT : attributs, clef, index

```
CREATE TABLE ENSEIGNANT
  (NumeroHarpege INTEGER NOT NULL,
   NomEnseignant CHAR(30) NOT NULL,
   PrenomEnseignant CHAR(50) NOT NULL,
   Grade ENUM('ext', 'MCF', 'PU', 'PRAG', 'PRCE', 'MCFPH','PUPH'),
   CONSTRAINT Pk_no_harpege PRIMARY KEY (NumeroHarpege),
   INDEX idx_nom_prenom (NomEnseignant,PrenomEnseignant))
```

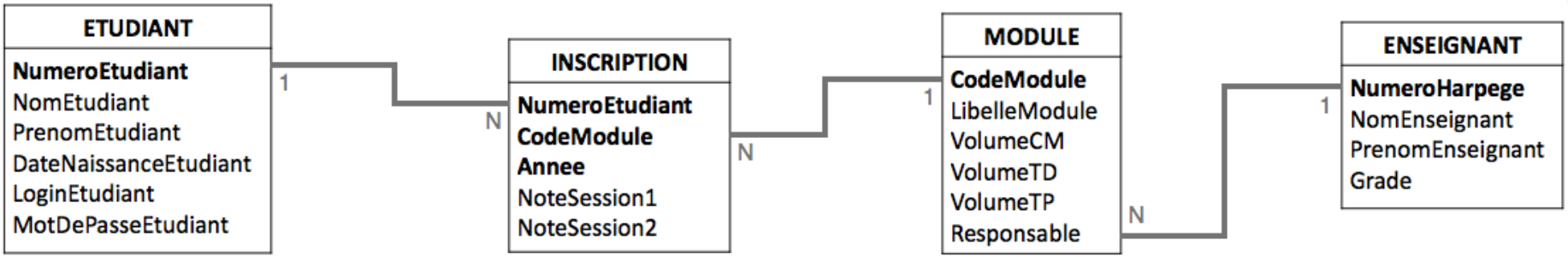
Création : à vous de jouer



MODULE : attributs (avec contraintes), clefs (avec politique), index

```
CREATE TABLE MODULE
  (CodeModule CHAR(10) NOT NULL,
   LibelleModule CHAR(30) NOT NULL,
   VolumeCM INTEGER DEFAULT 0 CHECK (VolumeCM < 100),
   VolumeTD INTEGER DEFAULT 0 CHECK (VolumeTD < 100),
   VolumeTP INTEGER DEFAULT 0 CHECK (VolumeTP < 100),
   Responsable INTEGER NOT NULL,
   CONSTRAINT Pk_code_module PRIMARY KEY (CodeModule),
   CONSTRAINT Fk_resp_no_harpege FOREIGN KEY (Responsable)
                              REFERENCES ENSEIGNANT(NumeroHarpege)
                              ON DELETE CASCADE ON UPDATE CASCADE,
   INDEX idx_libelle (LibelleModule))
```

Création : à vous de jouer



INSCRIPTION : attributs, clefs (avec politique)

```
CREATE TABLE INSCRIPTION
(NumeroEtudiant INTEGER NOT NULL,
  CodeModule CHAR(10) NOT NULL,
  Annee INTEGER NOT NULL,
  NoteSession1 INTEGER DEFAULT NULL,
  NoteSession2 INTEGER DEFAULT NULL,
  CONSTRAINT Pk PRIMARY KEY (NumeroEtudiant,CodeModule,Annee),
  CONSTRAINT Fk_no_etudiant FOREIGN KEY (NumeroEtudiant)
                            REFERENCES ETUDIANT(NumeroEtudiant)
                            ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT Fk_code_module FOREIGN KEY (CodeModule)
                            REFERENCES MODULE(CodeModule)
                            ON DELETE CASCADE ON UPDATE CASCADE)
```

Plan

Data Control Language

Bilan sur ce qui est (ou devrait être) su

- Création/modification de la structure de données : `CREATE TABLE`, `INDEX`, `DROP TABLE`...
- Manipulation des données : `SELECT`, `UPDATE`, ...

Objectif : contrôler les accès à une base de données

- Définir des utilisateurs
- Leur attribuer/retirer des droits

Plan

Utilisateurs : motivations

Définir des utilisateurs en SQL

- pour autoriser/interdire l'accès à certaines bases
- pour autoriser/interdire certaines opérations sur une base précise
- à partir d'un super-utilisateur `root` avec tous les droits : créer d'autres utilisateurs et leur affecter des droits

Exemple : nécessité de contrôler les utilisateurs

- En TP : accès à la base commune du `CABINETMEDICAL` à partir d'un utilisateur commun
  - autorisé à sélectionner, ajouter, modifier des données
  - non autorisé à modifier la structure
- En L2 Info dans le module "Langage du Web" : accès individuel (un étudiant $\equiv$ un utilisateur MySQL) à une base de données MySQL commune avec droits/interdits comme ci-dessus

Utilisateurs : création, modifications

Syntaxe de création

```
CREATE USER 'Nom' IDENTIFIED BY 'MotDePasse';...
```

Exemple : création de l'utilisateur *SecretaireScol* avec mdp 'scol@ufrST2122 !'

```
CREATE USER 'SecretaireScol' IDENTIFIED BY 'scol@ufrST2122!'
SELECT User, Host, Password FROM mysql.user
  WHERE mysql.user.User='SecretaireScol'
```

←T→	User	Host	Password
Edit Inline Edit Copy Delete	SecretaireScol	%	*F2B8A4D70CA562E39C10CBBB7CB98E2367980D5F

⇝ utilisateur ajouté, stockage d'une version hachée du mot de passe

Syntaxe de renommage

```
RENAME USER 'AncienNom' TO 'NouveauNom'
```

Syntaxe de changement de mot de passe

```
SET PASSWORD FOR 'Nom' = PASSWORD('NouveauMotDePasse')
```

Plan

Attribuer des droits : GRANT...ON...TO...

Syntaxe générale

```
GRANT TypePrivilege1, TypePrivilege2, ... ON NiveauPrivilege TO 'NomUtilisateur
```

- NiveauPrivilege :
  - * : application des droits sur toutes les bases.
  - *.* : application des droits sur toutes les tables du SGBD
  - NomBase.* : application des droits sur toutes les tables de NomBase et elle-même
  - NomBase.NomTable : application des droits sur NomBase.NomTable
- TypePrivilege : ensemble d'autorisations parmi
  - CREATE, ALTER, DROP : modifications possibles de la structure de base(s)
  - SELECT, INSERT, UPDATE, DELETE : manipulation possibles des données de table(s) des données ...

Exemple : ajout de droits de manipulation de données pour 'SecretaireScol' sur toutes les tables de la base 'APOGEE'

```
GRANT SELECT, INSERT, UPDATE, DELETE ON APOGEE.* TO 'SecretaireScol'
```

Retirer des droits

Syntaxe générale

```
REVOKE TypePrivilege1,... ON NiveauPrivilege FROM 'NomUtilisateur'
```

NiveauPrivilege et TypePrivilege : comme dans l'attribution de droits

Exemple : modification structure de la base de données PROJET plus autorisée pour l'étudiante de L2 Ibrahim

```
REVOKE CREATE, ALTER, DROP ON PROJET.* FROM 'Ibrahim'
```

Exemple : interdire l'écriture dans la table PROJET.LOG à l'étudiante Ibrahim

```
REVOKE INSERT, UPDATE, DELETE ON PROJET.LOG FROM 'Ibrahim'
```