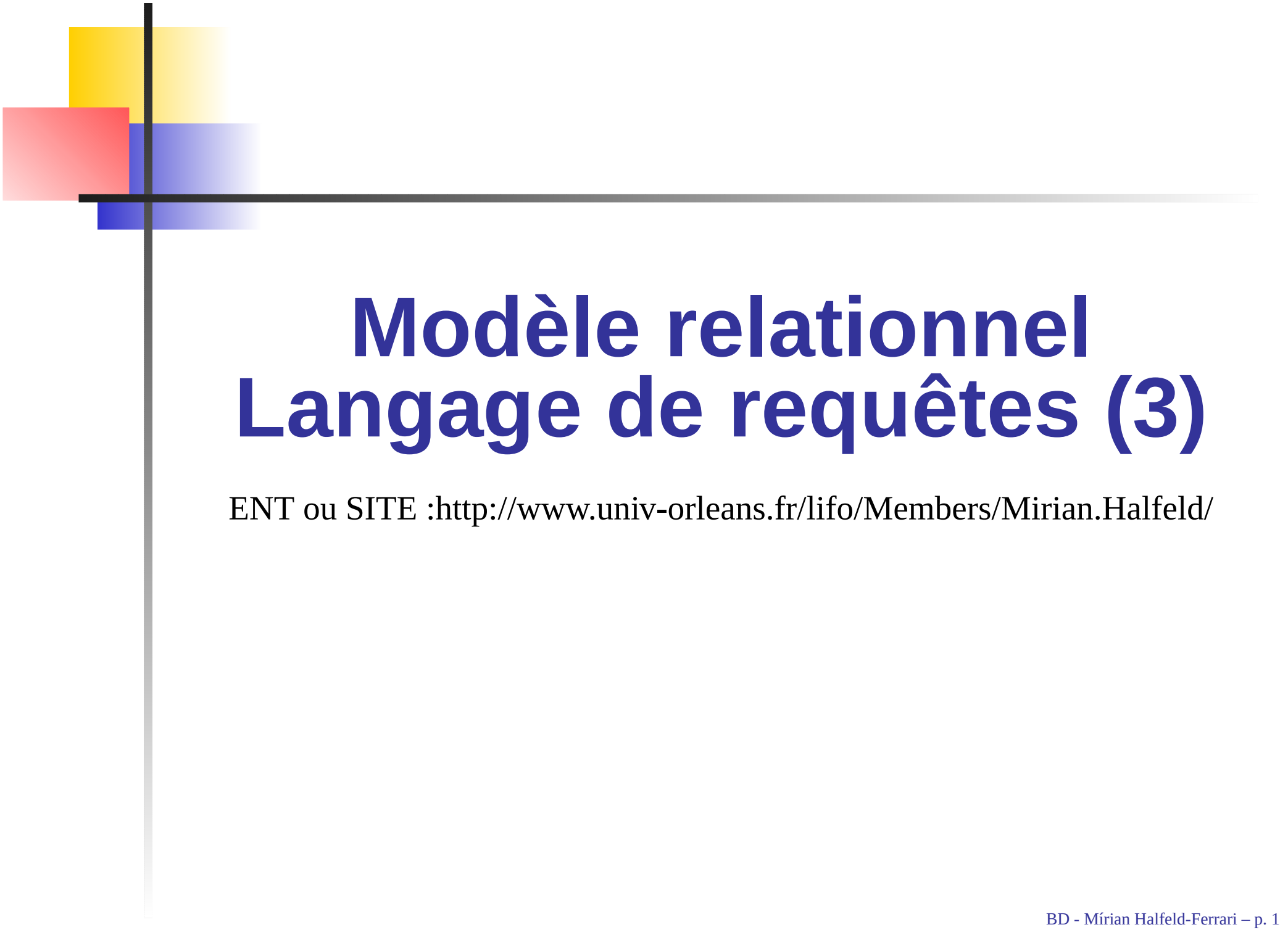

Modèle relationnel
Langage de requêtes (3)

ENT ou SITE :http://www.univ-orleans.fr/lifo/Members/Mirian.Halfeld/

Les opérations d'aggrégation

- Besoin non seulement de récupérer des données mais aussi d'exécuter des calculs et de proposer une vue globale de certaines données.
- SQL propose **une extension de l'AR** avec une classe de construteurs capable de calculer des valeurs groupées.
- Une opération d'**aggregation** calcule une valeur à partir d'une liste de valeurs dans une colonne.
- SQL permet AUSSI de grouper des tuples d'une relation selon certains critères, comme la valeur d'une autre colonne.

Les opérations d'aggrégation

1. `COUNT [DISTINCT] A`: le nombre de valeurs associées à l'attribut A (dans la colonne A). Sans la clause DISTINCT les doublons sont aussi comptés
2. `SUM [DISTINCT] A`: la somme des valeurs associées à l'attribut A (dans la colonne A).
3. `AVG [DISTINCT] A`: la moyenne des valeurs associées à l'attribut A
4. `MIN A`: la plus petite valeur de A
5. `MAX A`: la plus grande valeur de A.

Ces opérateurs sont appliqués dans une clause SELECT

La moyenne

STUDENTS $[Number, Name, Address, Age]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Trouver la moyenne d'age des étudiants

```
SELECT  AVG (S.Age)
FROM STUDENTS  S  ;
```

Trouver la moyenne d'age des étudiants habitant Blois

```
SELECT  AVG (S.Age)
FROM STUDENTS  S
WHERE Address = 'Blois' ;
```

Les valeurs maximales et minimales

STUDENTS $[Number, Name, Address, Age]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Donner le nom et l'age de l'étudiant le plus agé

```
SELECT  S.Name, MAX (S.Age)
FROM STUDENTS  S  ;
```

ATTENTION: Réponse NON VALIDE!!!!!!!!!

- Un SELECT qui utilise une opération d'aggrégation:
  - Peut aussi contenir d'autres opérations d'aggrégation.
  - Ne peut pas contenir un attribut de la relation (sauf, s'il s'agit d'un attribut sur lequel la clause GROUP BY est utilisée).

Les valeurs maximales et minimales

STUDENTS $[Number, Name, Address, Age]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Donner le nom et l'age de l'étudiant le plus agé

Une requête imbriquée est donc nécessaire ici:

```
SELECT  S.Name, S.Age
FROM STUDENTS  S
WHERE  S.Age = (SELECT MAX (S2.age) FROM STUDENTS S2);
```

Plusieurs systèmes ne traitent pas la requête suivante:

```
SELECT  S.Name, S.Age
FROM STUDENTS  S
WHERE  (SELECT MAX (S2.age) FROM STUDENTS S2) = S.Age;
```

Compter des tuples

STUDENTS $[Number, Name, Address, Age]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Combien d'étudiants avons nous?

```
SELECT  count(*)
FROM STUDENTS  S  ;
```

- Nous comptons ainsi tous les lignes de la table STUDENTS.

Combien d'étudiants avons nous?

```
SELECT  count( DISTINCT S.name)
FROM STUDENTS  S  ;
```

- Nous comptons les étudiants en comptant dans la table ceux qui ont des noms différents.

Une alternative pour ANY et ALL

Les opérations d'aggrégation offrent une alternative pour ANY et ALL.

STUDENTS $[Number, Name, Address, Age]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Trouver les noms des étudiants plus agés que l'étudiant plus agé de Blois

```
SELECT  S.Name    FROM STUDENTS  S
WHERE  S.Age > ( SELECT  MAX(S2.Age)
                 FROM STUDENTS S2
                 WHERE S2.Address = 'Blois');
```

```
SELECT  S.Name    FROM STUDENTS  S
WHERE  S.Age > ALL ( SELECT  S2.Age
                     FROM STUDENTS S2
                     WHERE S2.Address = 'Blois');
```

Requêtes sur des groupes

Parfois nous voulons appliquer les opérations d'aggrégation sur chaque groupe parmi un nombre de groupes de tuples d'une relation. Le nombre de groupes dépend de l'instance de la relation et est connu d'avance.

Trouver l'age de l'étudiant le plus jeune dans chaque ville.

- Si nous savons que 10 villes sont dans notre BD alors il est toujours possible d'écrire 10 requêtes du type

```
SELECT  MIN (S.Age)
FROM STUDENTS  S
WHERE S.Address = x);
```

  où x serait remplacé par les noms de villes.
- Les problèmes: tâche répétitive et ... nous ne savons pas forcement le nombre de villes qui sont dans la base, ni quelles sont ces villes.
- Pour écrire cette requête nous avons besoin d'une extension du SQL de base, à savoir, de la clause GROUP BY

Les clauses GROUP BY et HAVING

Format de base:

```
SELECT [DISTINCT] select-list
FROM from-list
WHERE  qualification
GROUP BY grouping-list
HAVING  group-qualification
```

- La `select-list` peut avoir:
  1. Des attributs **qui apparaîsent dans le GROUP BY**
     Chaque tuple dans la relation résultat correspond à un groupe (une collection des tuples).
  2. Une liste de termes dans le format `aggop (attribut-name) [AS new-name]` où `aggop` est un opérateur d'aggrégation.

Les clauses GROUP BY et HAVING

Format de base:

```
SELECT [DISTINCT] select-list
FROM from-list
WHERE  qualification
GROUP BY grouping-list
HAVING  group-qualification
```

- Les expressions dans `group-qualification` doivent avoir UNE SEULE valeur par groupe.
- Intuitivement, la clause `HAVING` determine si un tuple résultat doit être engendré pour un certain groupe.
- Seulement les attributs qui sont dans le `GROUP BY` ou qui sont dans une opérations d'aggrégation peuvent apparaître dans la clause `HAVING`.

Les clauses GROUP BY et HAVING

STUDENTS $[Number, Name, Address, Age]$

INSCRIPTION $[StudNb, CourseCode, Year, Time]$

COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Pour chaque ville ayant plus que 1 étudiant, trouver l'age de l'étudiant le plus jeune qui peut voter

```
SELECT S.Address, MIN (S.Age) as MinAGE
FROM STUDENTS S
WHERE S.age >= 18
GROUP BY S.Address
HAVING COUNT(*) > 1;
```

Example

STUDENTS	Number	Name	Address	Age
	10	John Dustin	Orléans	17
	11	Jean Dupont	Orléans	18
	21	Jean Martin	Orléans	20
	22	Jean Lubber	Orléans	22
	12	Mary Zorba	Paris	19
	13	Martin Dubois	Paris	22
	14	Patrick Lorent	Blois	22
	23	Martin Leblanc	Paris	20
	24	Marina Leblanc	Blois	17
	23	Paul Brutus	Orléans	20

Example

STUDENTS	Number	Name	Address	Age
	10	John Dustin	Orléans	17
	11	Jean Dupont	Orléans	18
	21	Jean Martin	Orléans	20
	22	Jean Lubber	Orléans	22
	12	Mary Zorba	Paris	19
	13	Martin Dubois	Paris	22
	14	Patrick Lorent	Blois	22
	23	Martin Leblanc	Paris	20
	24	Marina Leblanc	Blois	17
	23	Paul Brutus	Orléans	20

Applicant la clause WHERE : `WHERE S.age >= 18`

Example

Image du groupement	Address	Age	Name ...
	Orléans	18	Jean Dupont
	Orléans	20	Jean Martin
	Orléans	22	Jean Lubber
	Orléans	20	Paul Brutus
	Paris	19	Mary Zorba
	Paris	22	Martin Dubois
	Paris	20	Martin Leblanc
	Blois	22	Patrick Lorent

Comment le groupement va être fait...

Example

Image du groupement	Address	Age	Name ...
	Orléans	18	Jean Dupont
	Orléans	20	Jean Martin
	Orléans	22	Jean Lubber
	Orléans	20	Paul Brutus
	Paris	19	Mary Zorba
	Paris	22	Martin Dubois
	Paris	20	Martin Leblanc
	Blois	22	Patrick Lorent

En applicant la clause HAVING `HAVING COUNT(*) > 1` quels sont les groupes (ci-dessus) qui ne seront pas pris en compte?

Example

Image du groupement	Address	Age	Name ...
	Orléans	18	Jean Dupont
	Orléans	20	Jean Martin
	Orléans	22	Jean Lubber
	Orléans	20	Paul Brutus
	Paris	19	Mary Zorba
	Paris	22	Martin Dubois
	Paris	20	Martin Leblanc
	Blois	22	Patrick Lorent

En applicant la clause HAVING `HAVING COUNT(*) > 1`

Example

RESULTAT	Address	MinAGE
	Orléans	18
	Paris	19

Après avoir calculé l'age minimal ...
le résultat de `SELECT S.Address, MIN (S.Age) as MinAGE` ...

Remarquer l'importance de l'évaluation du `WHERE` avant le `GROUP BY`: Si le `WHERE` n'avait pas été consideré, Blois aurait fait partie du résultat.

Les valeurs nulles

- Dans la pratique certaines valeurs peuvent ne pas être connus au moment de la saisie des informations.
- Exemple: Un nouveau cours est inséré dans la base, mais nous ne connaissons pas encore le nombre de crédits qui lui seront attribués...
- Un autre exemple serait un attribut comme *Service Militaire* valide seulement pour les garçons dans notre base. Quelle valeur aurait cet attribut pour les filles?
- Besoin ainsi, dans différents situations, d'une valeur *unconnu*.
- SQL propose une valeur `NULL` pour ces situations.
- **Remarquer que la valeur NULL peut avoir différentes significations: unconnu, inexistant, incohérent...**. Cela peut être la source de différentes anomalies ou des traitements spécifiques.
- L'introduction du `NULL` rend les choses plus compliquées... Il existe différents travaux sur le sujet.

Les valeurs nulles en SQL

- Supponsons donc le tuple suivant dans la table COURSE
  ⟨CourseCode: CS201, CourseTitle: Databases,CreditNb: null, HoursNb: 40⟩
- Comment évaluer la comparaison `CreditNb = 20` sur ce tuple?
  Le résultat est **unconnu**.
- Comment évaluer la comparaison `CreditNb = 20 OR HoursNb > 20`?
  Le résultat est **vrai**.
- La présence de la valeur nulle introduit le besoin d'une logique à trois valeurs

A	B	NOT A	A OR B	A AND B
unk	*unk*	*unk*	*unk*	*unk*
T	*unk*	F	T	*unk*
F	*unk*	T	*unk*	F

Le NULL en SQL

- Une opération arithmétique sur une valeur NULL donne NULL comme résultat.
- Une comparaison ($=$, $<$ ...) entre une valeur NULL et une autre valeur quelconque donne UNKNOWN (unconnu) comme résultat... et nous devons ensuite raisonner dans la logique à 3 valeurs.
- NULL n'est pas une constante: NULL ne peut pas être utilisé explicitement comme un opérand.
  - $x + 3$, si x a la valeur NULL l'expression a aussi la valeur NULL.
  - NULL$+3$ n'est pas une expression valide en SQL.

Remarques sur les valeurs nulles en SQL

- La clause `WHERE` va éliminer des tuples pour lesquels la condition de selection **n'est pas évaluée à VRAIE!**

  COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

```
SELECT CourseTitle
FROM COURSE c
WHERE  CreditNb > 6 or CreditNb <= 6;
```

  Si CreditNb est `NULL` ce tuple ne sera pas considéré.

- Anomalie: pour l'éliminition des doublons (dans ce cas `NULL = NULL` est vrai!) Deux tuples sont identiques si les attributs correspondants ont la même valeur ou s'ils sont tous les deux `NULL`

Remarques sur les valeurs nulles en SQL

- `COUNT (*)` compte les valeurs `NULL`.
- Les autres opérateurs d'aggrégation (`COUNT, AVG, MIN, MAX, SUM`) ne considèrent pas les valeurs `NULL` (mais les doublons OUI).
- SQL propose un opérateur de comparaison pour les valeurs `NULL`: `IS NULL` ou `IS NOT NULL`

Encore un peu d'AR: la division

- $R \div S$
- Conditions: $sort(S) \subset sort(R)$
- Résultat sur $sort(R) \setminus sort(S)$
- $R \div S = \{t \mid t \in \pi_{sort(R) \setminus sort(S)}(R)$ et $(\{t\} \bowtie S) \subseteq I(R)\}$
- La division peut être exprimée à partir des opérations de base:
- Notons que si R et S ont des instances vides, ou encore si $sort(R) = sort(S)$ alors $R \div S = \emptyset$.

R

A	B	C
a1	b1	c1
a1	b2	c2
a2	b1	c1
a2	b3	c3

S

B	C
b1	c1
b2	c2

$R \div S$

A
a1

Division: un opérateur simulé par les opérateurs primitifs

STUDENTS $[Number, Name, Address, Age]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

- La division est en général utilisée pour exprimer des requêtes du type "trouver les étudiants inscrits dans **tous** les cours".
- La requêtes en AR:

$$(\pi_{StudNb,CourseCode} INSCRIPTION) \div (\pi_{CourseCode} COURSE)$$

En SQL: une solution pas très bonne

INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Trouver les étudiants inscrits dans *tous* les cours

```
select StudNb from INSCRIPTION
group by StudNb
having count(*) = (select count (*) from Course);
```

En SQL: une solution pas très bonne

- Que se passe t'il lorsqu'il y a une redondance de données dans la table diviseur COURSE ou lorsqu'il y a un cours dans la table INSCRIPTION qui n'est pas dans la table COURSE?

  La requête SQL en question ne marche pas correctement!!!!

- Important: Au sens de Codd, l'introduction d'un tuple dans la relation INSCRIPTION faisant référence à un cours qui n'est pas dans la table COURSE **ne fait pas obstacle ni à la définition de la division; ni au résultat que nous attendons**. Nous voulons savoir quels étudiants sont inscrits dans les cours dans COURSE mais si certains étudiants font encore plus de cours ...

- De manière générale il ne faut JAMAIS supposer que les tables que nous manipulons sont conhérentes car nous ne savons pas si les bonnes contraintes ont été implémentées.

En SQL: une bonne solution - version 1

INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Trouver les étudiants inscrits dans *tous* les cours

Trouver les étudiant pour lesquels **il n'existe pas un cours dans lequel il n'est pas inscrit.**

```
select I1.StudNb
from INSCRIPTION I1
where not exists
  (select *
  from COURSE C1
  where C1.CourseCode not in
     (select I2.CourseCode from INSCRIPTION I2
      where I1.StudNb = I2.StudNb
      and I2.CourseCode = C1.CourseCode));
```

En SQL: une bonne solution - version 2

INSCRIPTION $[StudNb, CourseCode, Year, Time]$
COURSE $[CourseCode, CourseTitle, CreditNb, HoursNb]$

Trouver les étudiants inscrits dans *tous* les cours

Trouver les étudiant pour lesquels **il n'existe pas un cours dans lequel il n'est pas inscrit.**

```
select I1.StudNb
from INSCRIPTION I1
where not exists
  (select *
  from COURSE C1
  where not exists
     (select * from INSCRIPTION I2
      where I1.StudNb = I2.StudNb
      and I2.CourseCode = C1.CourseCode));
```