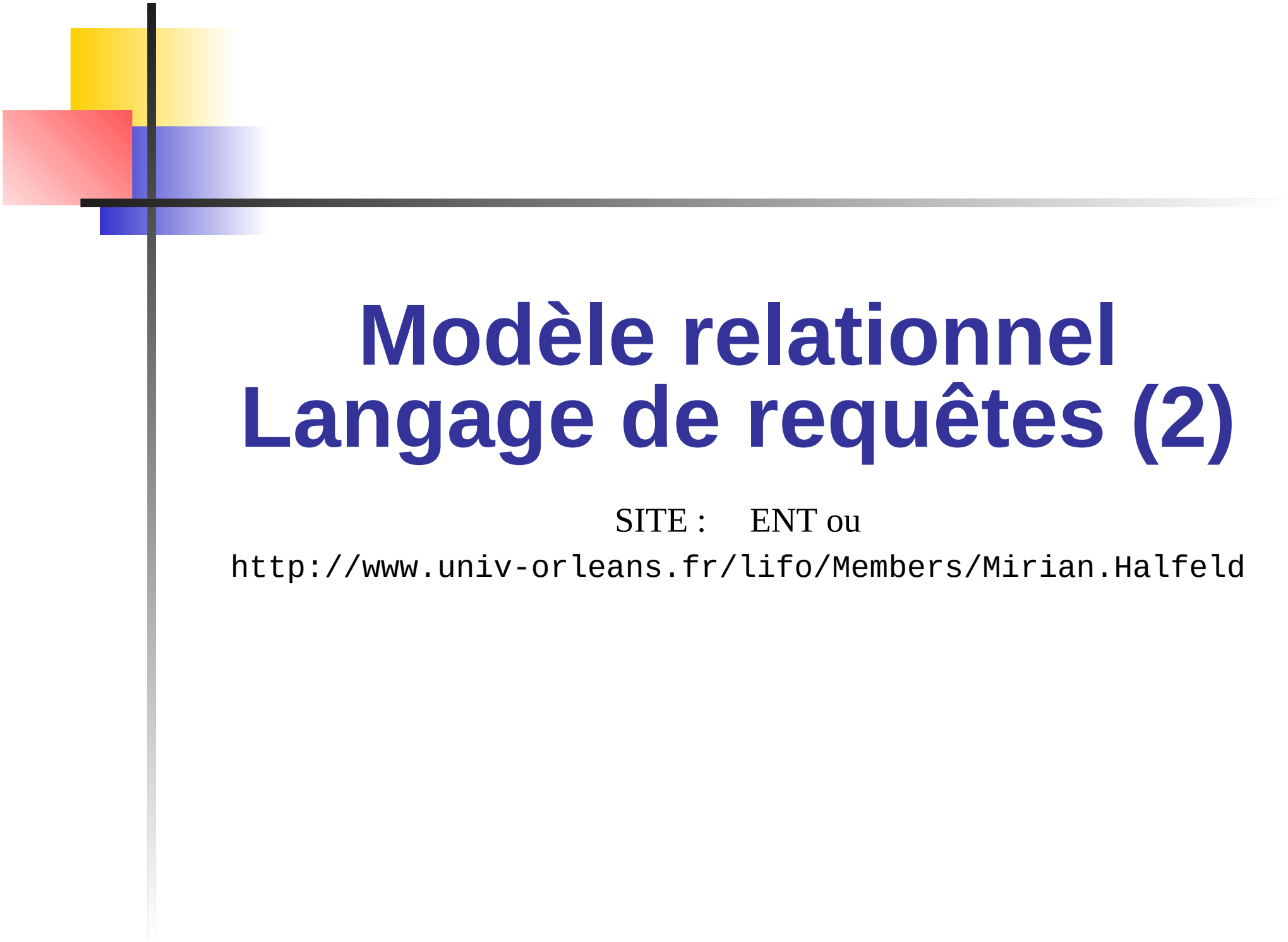

Modèle relationnel
Langage de requêtes (2)

SITE : ENT ou

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

Besoin d’augmenter le pouvoir d’expression

- Algèbre SPJR (ou SPC): possibilité d’écrire des requêtes conjonctives (très courantes).
- D’autres requêtes existent:

  Lister les films de Almodovar ou de Sergio Leone

  Pouvons nous écrire cette requête avec SPJR (ou SPC)?

Le besoin de l'union

CINEMA[Film [Title, Director, Year], Theater [Title, Director, TheaterName]]

- Pour écrire certaines requêtes nous avons besoin de la disjonction ou de l'union comme opérateur de l'algèbre.
- Soit la requête:

Lister les film de Almodovar ou de Sergio Leone

  - Si nous avons la disjonction dans la condition de sélection:

$$\sigma_{Director=Almodovar \vee Director=Leone} Film$$

  - Si nous avons l'opération d'union dans l'algèbre:

$$\sigma_{Director=Almodovar} Film \cup \sigma_{Director=Leone} Film$$

Exemple

Soit le schéma de relation:

$$FActors[Director, Title, Actor]$$

- Construire la table *EquipeFilm* contenant le noms de toutes les personnes travaillant dans un film donné: les acteurs et les metteurs en scène. La table doit avoir un seul attribut MEMBER.
- Est il possible d'écrire cette requête avec SPJR en ajoutant la disjonction dans la condition de sélection?
- Avec l'opération d'union

$$\delta_{Director \rightarrow Member}(\pi_{Director}(\sigma_{Title=toto} FActors)) \cup \delta_{Actor \rightarrow Member}(\pi_{Actor}(\sigma_{Title=toto} FActors))$$

Algèbre: Ajouter l'union

- **Algèbre SPCU**: extension de la définition de l'algèbre SPC pour inclure l'opérateur d'union
  - Soient I_1 et I_2 deux relations **de même arité**.
  - Comme en mathématiques, $I_1 \cup I_2$ est une **relation possédant cette arité et contenant l'union de 2 ensembles**.
- **Algèbre SPJRU**: obtenue de la même façon, sauf que **l'union peut seulement être appliquée à des expressions de même sort**.

$$R \cup S = \{t \mid t \in I(R) \vee t \in I(S) \text{ où } sort(R) = sort(S)\}$$

Encore d'autres types de requêtes ...

Comment écrire la requête suivante:

Quels sont les films de Hitchcock dans lesquels il ne joue pas?

- Étapes:
  1. Trouver les films de Hitchcock (comme metteur en scène) (R1).
  2. Trouver les films de Hitchcock dans lequel il joue (R2).
  3. Nous voulons les films qui sont dans R1 et qui ne sont pas dans R2. DONC.... Supprimer du résultat R1, le résultat de R2.
- Pour écrire cette requête nous avons besoin de la négation.

$$\pi_{Title}(\sigma_{Director=" Hitchcock"}(Films)) \quad \setminus$$
$$\pi_{Title}(\sigma_{Director=" Hitchcock" \wedge Actor=" Hitchcock"}(Films))$$

L’opérateur de différence

La négation en algèbre correspond à l’opérateur de la **différence ensembliste**.

- Utilisée sur des relations de même sort (étiqueté)
- Utilisée sur des relations de même arité (non étiqueté)

$$R \setminus S = \{t \mid t \in I(R) \wedge t \notin I(S) \text{ où } sort(R) = sort(S)\}$$

L’algèbre relationnelle

Algèbre relationnelle étiquetée

- SPJRU + opérateur de différence (il est classique d’y ajouter aussi l’opérateur d’intersection).
- L’union est présente DONC les relations constantes non réduites à un singleton peuvent être utilisées dans cette algèbre
- Opérateur de sélection est étendu pour permettre la négation.

Algèbre relationnelle non étiquetée

- SPCU + opérateur de différence (il est classique d’y ajouter aussi l’opérateur d’intersection).
- L’union est présente DONC les relations constantes non réduites à un singleton peuvent être utilisées dans cette algèbre.
- Opérateur de sélection est étendu pour permettre la négation.

Langages de requêtes équivalents

Soient Q_1 et Q_2 deux langages de requête (avec leurs sémantiques associées).

- $Q_1 \sqsubseteq Q_2$: Le langage Q_1 est dominé par Q_2 ou Q_1 est plus faible que Q_2, si pour toute requête q_1 de Q_1 il existe une requête q_2 de Q_2 telle que $q_1 \equiv q_2$.
- $Q_1 \equiv Q_2$: Q_1 et Q_2 sont équivalents si $Q_1 \sqsubseteq Q_2$ et $Q_2 \sqsubseteq Q_1$.
- **Pouvoir d'expression des algèbres: Les algèbres relationnelles non étiquetées et étiquetées ont des puissances d'expression équivalentes.**

Intersection: un opérateur simulé par les opérateurs primitifs

- $R \cap S$
- Conditions: $sort(R) = sort(S)$
- Résultat sur $sort(R)$
- $R \cap S = \{t \mid t \in R \text{ et } t \in S\}$
- L'intersection peut être exprimée à partir des opérations de base:
  $R \cap S = R \setminus (R \setminus S) = S \setminus (S \setminus R)$

Exercice

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

1. Trouver les étudiants inscrits dans le cours de *bases de données*.
2. Trouver les étudiants inscrits dans le cours de *bases de données* ou le cours de *génie logicielle*.
3. Trouver les étudiants inscrits dans le cours de *bases de données* et le cours de *génie logicielle*.
4. Trouver les étudiants inscrits dans le cours de *bases de données* et non inscrits dans le cours de *génie logicielle*.
5. Trouver les étudiants inscrits *seulement* dans le cours de *bases de données*.

SQL: Union

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

Trouver les étudiants inscrits dans le cours de *bases de données* ou le cours de *génie logicielle*

```
SELECT  StudNb
FROM   INSCRIPTION NATURAL JOIN  COURSE
WHERE  CourseTitle = 'bases de données'
UNION
SELECT  StudNb
FROM   INSCRIPTION NATURAL JOIN  COURSE
WHERE  CourseTitle = 'genie logicielle';
```

Une autre possibilité

```
SELECT  StudNb
FROM   INSCRIPTION NATURAL JOIN  COURSE
WHERE  CourseTitle = 'bases de données'
OR CourseTitle = 'genie logicielle';
```

SQL: Intersection

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

Trouver les étudiants inscrits dans le cours de *bases de données* et le cours de *génie logicielle*

```
SELECT  StudNb
FROM   INSCRIPTION NATURAL JOIN  COURSE
WHERE  CourseTitle = 'bases de données'
INTERSECT
SELECT  StudNb
FROM   INSCRIPTION NATURAL JOIN  COURSE
WHERE  CourseTitle = 'genie logicielle';
```

ATTENTION: Que fait la requête suivante?

```
SELECT  StudNb
FROM   INSCRIPTION NATURAL JOIN  COURSE
WHERE  CourseTitle = 'bases de données'
AND CourseTitle = 'genie logicielle';
```

SQL: Différence

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

Trouver les étudiants inscrits dans le cours de *bases de données* et non inscrits dans le cours de *génie logicielle*.

```
SELECT  StudNb
FROM   INSCRIPTION NATURAL JOIN  COURSE
WHERE  CourseTitle = 'bases de données'
MINUS
SELECT  StudNb
FROM   INSCRIPTION NATURAL JOIN  COURSE
WHERE  CourseTitle = 'genie logicielle';
```

Remarques

- UNION, INTERSECTS and MINUS peuvent être utilisés entre deux relations ayant le même nombre de colonnes, dans le même ordre, ayant le même domaine.
- Opérations ensemblistes (sans doublons):
  - Si les types sont les mêmes. Par exemple VARCHAR avec VARCHAR.
  - INTERSECT ne fonctionne pas entre VARCHAR et CHAR.
  - Les opérations UNION et MINUS entre VARCHAR et CHAR: bag!

Sous-requêtes, requêtes imbriquées

- Une requête imbriquée contient une autre requête qui nous appelons sous-requête.
- En écrivant une requête nous avons souvent le besoin d'exprimer une condition qui fait référence à une relation qui doit aussi être calculée.
- Une requête auxiliaire, utilisée pour calculer cette relation, est une sous-requête.
- Il existe un certain nombre d'opérateurs qui peuvent être appliqués à une relation R (calculée dans une sous-requête) et produire un résultat boolean.

Exemple

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

Trouver les étudiants inscrits dans le cours dont le code est de *BD1-0910*.

```
SELECT  S.Name
FROM    STUDENTS  S
WHERE   S.Number IN ( SELECT I.StudNb
                      FROM   INSCRIPTION I
                      WHERE  I.CourseCode = 'BD1-0910');
```

s *IN* R: vrai si et seulement si s est identique à **une** des valeurs dans R.
s *NOT IN* R: vrai si et seulement si s n'est identique à **aucune** valeur dans R.
Dans ce cas, R est une relation unaire.

Comment cela marche?

- Fonctionnement comme de boucles imbriquées.
- Pour chaque tuple dans la table STUDENTS la sous-requête est évaluée.

Plusieurs sous-requêtes

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

Trouver les noms des étudiants inscrits dans le cours dont le code est de *Bases de données*.

```
SELECT S.Name
FROM   STUDENTS  S
WHERE  S.Number IN
        ( SELECT I.StudNb
          FROM   INSCRIPTION I
          WHERE  I.CourseCode IN
             (SELECT C.CourseCode
              FROM   COURSE C
              WHERE  C.CourseTitle = 'Bases de donnees'));
```

Que fait la requête?

STUDENTS $[Number, Name, Address]$

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

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

```
SELECT S.Name
FROM   STUDENTS  S
WHERE  S.Number NOT IN
        ( SELECT I.StudNb
          FROM   INSCRIPTION I
          WHERE  I.CourseCode IN
             (SELECT C.CourseCode
              FROM   COURSE C
              WHERE  C.CourseTitle = 'Bases de donnees'));
```

Sous-requêtes en corrélation

La sous-requête peut dépendre du tuple qui est en train d'être examiné dans la requête externe.

STUDENTS $[Number, Name, Address]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$

Trouver les noms des étudiants inscrits dans le cours ayant code *BD1-0910*.

```
SELECT  S.Name    FROM STUDENTS  S
 WHERE  EXISTS ( SELECT *   FROM INSCRIPTION I
         WHERE  I.CourseCode = 'BD1-0910' AND S.Number = I.StudNb)
```

EXISTS R: vraie si et seulement si R **n'est pas vide**.
NOT EXISTS R: vraie si et seulement si R **est vide**.

- Pour chaque tuple S de STUDENTS, nous testons si l'ensemble contenant les tuples I de INSCRIPTION tels que `I.CourseCode = 'BD1-0910'` et `S.Number = I.StudNb` n'est pas vide.
- La requête interne dépend de la valeur actuelle de S et doit être réévaluée pour chaque tuple dans STUDENTS.

Que fait la requête?

STUDENTS $[Number, Name, Address]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$

Trouver les noms des étudiants NON inscrits dans le cours ayant code *BD1-0910*.

```
SELECT  S.Name    FROM STUDENTS  S
 WHERE    NOT EXISTS ( SELECT *   FROM INSCRIPTION I
         WHERE  I.CourseCode = 'BD1-0910' AND S.Number = I.StudNb)
```

Opérateurs de comparaison des ensembles

1. $s\ op\ ANY\ R$: vraie si et seulement si s est op **qu'au moins une valeur dans la relation unaire** R.
2. $s\ op\ ALL\ R$: vraie si et seulement si s est op **que toutes les valeurs dans la relation unaire** R.
3. op est un des opérateurs dans $\{<, >, <=, <>, >=, =\}$
4. Les négations sont aussi possibles: $NOT\ ALL$, $NOT\ ANY$

ANY

STUDENTS $[Number, Name, Address]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$

Trouver les codes des étudiants inscrits dans une année postérieur à l'année d'inscription de l'étudiant *Paul*.

```
SELECT  I.StudNb    FROM INSCRIPTION I
WHERE  I.Year > ANY (SELECT I2.Year
                FROM INSCRIPTION I2, STUDENTS S
                WHERE I2.StudNb = S.Number AND S.Name= 'Paul');
```

- La requête trouve les étudiants dont l'année d'inscription est postérieur à celle d'AU MOINS UN étudiant dont le nom est Paul (il peut avoir plusieurs Paul).
- S'il n'existe AUCUN étudiant Paul, alors la comparaison $I.Year > ANY$ est fausse.
- Raisonnement: $I.Year$ est comparé à chaque résultat obtenue via la sous-requête. La sous-requête doit retourner un tuple permettant que la condition $I.Year > ANY$ soit vraie.

ALL

STUDENTS $[Number, Name, Address]$
INSCRIPTION $[StudNb, CourseCode, Year, Time]$

Trouver les codes des étudiants inscrits le plus récemment.

```
SELECT  I.StudNb    FROM INSCRIPTION I
WHERE  I.Year >= ALL (SELECT I2.Year
                FROM INSCRIPTION I2);
```

Opérations sur des tuples

- Une liste SQL est représenté par des éléments entre parenthèses.
- Exemple:
  - **Tuple de valeurs** $(123,' foo')$
  - **Tuple d'attributs** $(Name, Address, Networth)$

  Nous avons un **tuple de valeurs** $(123,' foo')$, tuple d'attributs $(Name, Address, Networth)$
- Si un tuple a les mêmes composants d'une relation R, alors de comparaisons sont possibles
  $t\ IN\ R$ ou $t <> ANY\ R$
- L'ordre doit être respecté