R3.07 - SQL dans un langage de programmation
Cours 1 - PL/SQL et déclencheurs

A. Ridard

Vannes
iut :
Université Bretagne Sud

A propos de ce document

- Pour naviguer dans le document, vous pouvez utiliser :
  - le menu (en haut à gauche)
  - l'icône en dessous du logo IUT
  - les différents liens
- Pour signaler une erreur, vous pouvez envoyer un message à l'adresse suivante : anthony.ridard@univ-ubs.fr

Plan du cours

Le langage SQL permet :

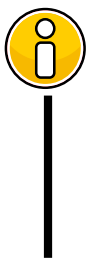
- de définir (**CREATE**), de modifier (**ALTER**) et de supprimer (**DROP**) des structures de données $\rightsquigarrow$ *LDD*
- d'ajouter (**INSERT**), de modifier (**UPDATE**), de supprimer (**DELETE**) et d'extraire (**SELECT**) des données $\rightsquigarrow$ *LMD*
- de contrôler l'accès des utilisateurs aux données $\rightsquigarrow$ *LCD*

Il permet aussi d'exprimer des contraintes (d'attribut ou de table) :

- de clé primaire (**PRIMAREY KEY**)
- de clé étrangère (**REFERENCES**)
- d'existence (**NOT NULL**)
- d'unicité (**UNIQUE**)
- de vérification (**CHECK**)

Mais il ne permet pas

- d'exprimer toutes les contraintes $\rightsquigarrow$ *Déclencheurs*
- d'appliquer des traitements « complexes » sur les données $\rightsquigarrow$ *PL/SQL*



PL/SQL

- Acronyme : Procedural Language / Structured Query Language
- Extension de SQL : les manipulations de données optimisées par le SGBDR cohabitent avec les éléments habituels de la programmation structurée

Les manipulations de données dans le développement d'une application

- Interface d'accès aux données : JDBC / PDO ⇝ ***R3.01***
- Patron de conception Data Access Object (DAO) ⇝ ***R4.01***
- Interface Object-Relational Mapping (ORM) : JPA / Doctrine

- Le coût de ces approches peuvent limiter les performances
- Le PL/SQL permet d'automatiser de manière efficace et transparente certains traitements grâce aux SGBDR, c'est un complément essentiel

Un programme est structuré en blocs d'instructions de 3 types :

- les procédures anonymes
- les procédures nommées
- les fonctions nommées

Structure d'un bloc

```
DECLARE
    -- définition des variables
BEGIN
    -- instructions à exécuter
    EXCEPTION
        -- récupération des erreurs
END;
```

- Les blocs, comme les instructions, se terminent par un « ; »
- Seuls **BEGIN** et **END** sont obligatoires

Il est évidemment possible (et recommandé) de commenter un programme.

Commentaires

```
-- Commentaire d'une (fin de) ligne

/*
Commentaire
de plusieurs
lignes
*/
```

Une variable possède un identificateur :

- 30 caractères au plus
- commence par une lettre
- peut contenir lettres, chiffres, _, $ et #
- pas sensible à la casse

Elle doit être déclarée (avant d'être utilisée) avec un type :

- type SQL2 : **INTEGER**, **NUMBER**, **VARCHAR2**, **DATE**, ...
- type d'une colonne d'une table : *table.colonne***%TYPE**
- type d'une ligne d'une table : *table***%ROWTYPE**
- type structuré [1] : **TABLE**, **RECORD**, ...

1. Non étudié dans ce cours

L'affectation de variable s'effectue :

- directement avec « := »
- via une requête **SELECT** avec la directive **INTO**

Conflits de noms

- Si une variable porte le même nom qu'une colonne, c'est la colonne qui l'emporte ce qui peut provoquer de graves erreurs
- Pour éviter cela, on peut préfixer par « **v_** » le nom d'une variable [a]

a. et par « **p_** » le nom d'un paramètre (bonne pratique pour les procédures et fonctions)

On dispose du :

branchement conditionnel

```
IF condition THEN
    instructions;
[ELSIF condition THEN
    instructions;]
[ELSE
    instructions;]
END IF;
```

Ainsi que de la :

boucle « générale »

```
LOOP
    instructions;
    EXIT [WHEN condition];
    instructions;
END LOOP;
```

boucle « tant que »

```
WHILE condition
    LOOP
        instructions;
    END LOOP;
```

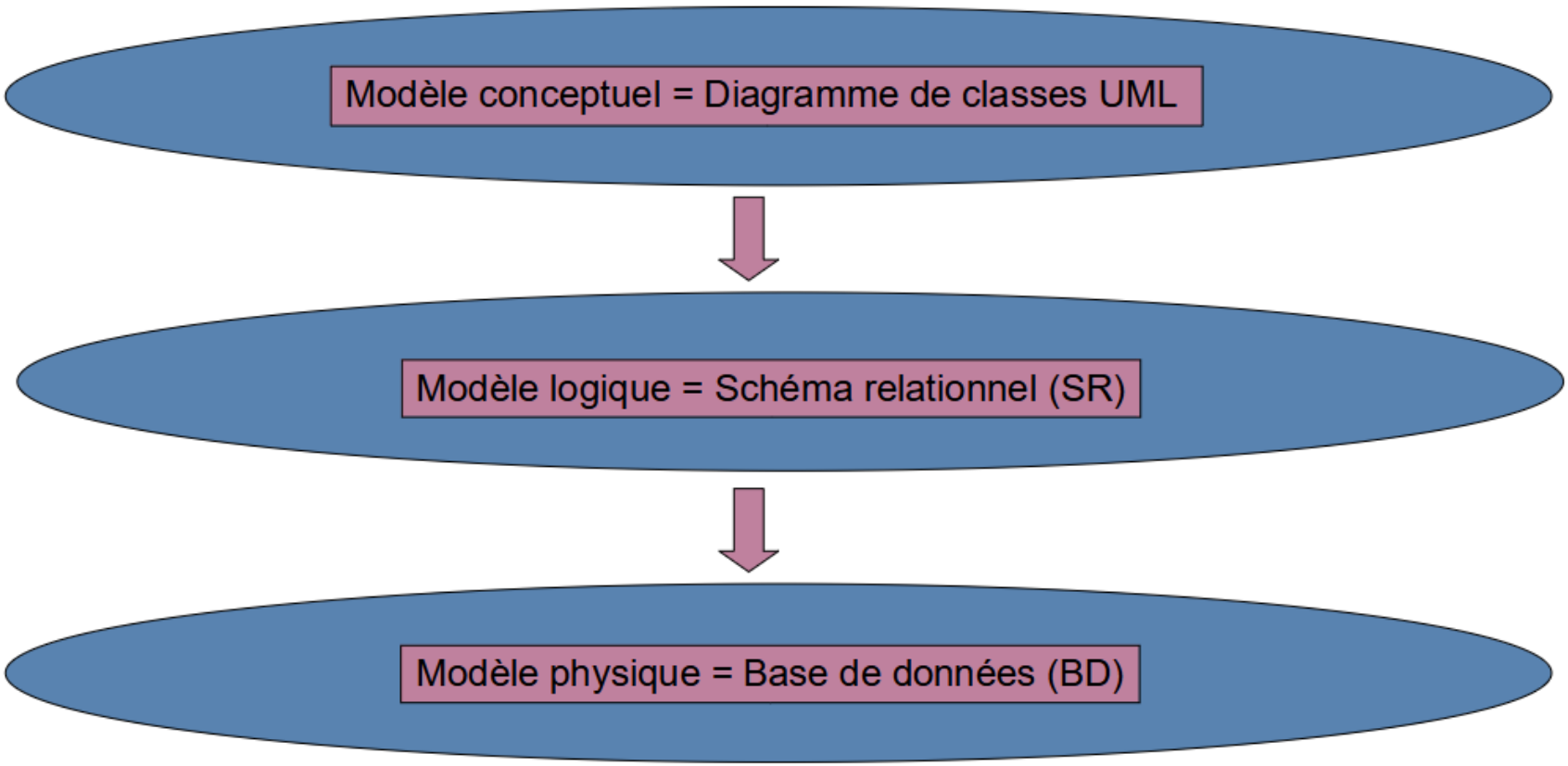
Sans oublier la :

boucle « pour »

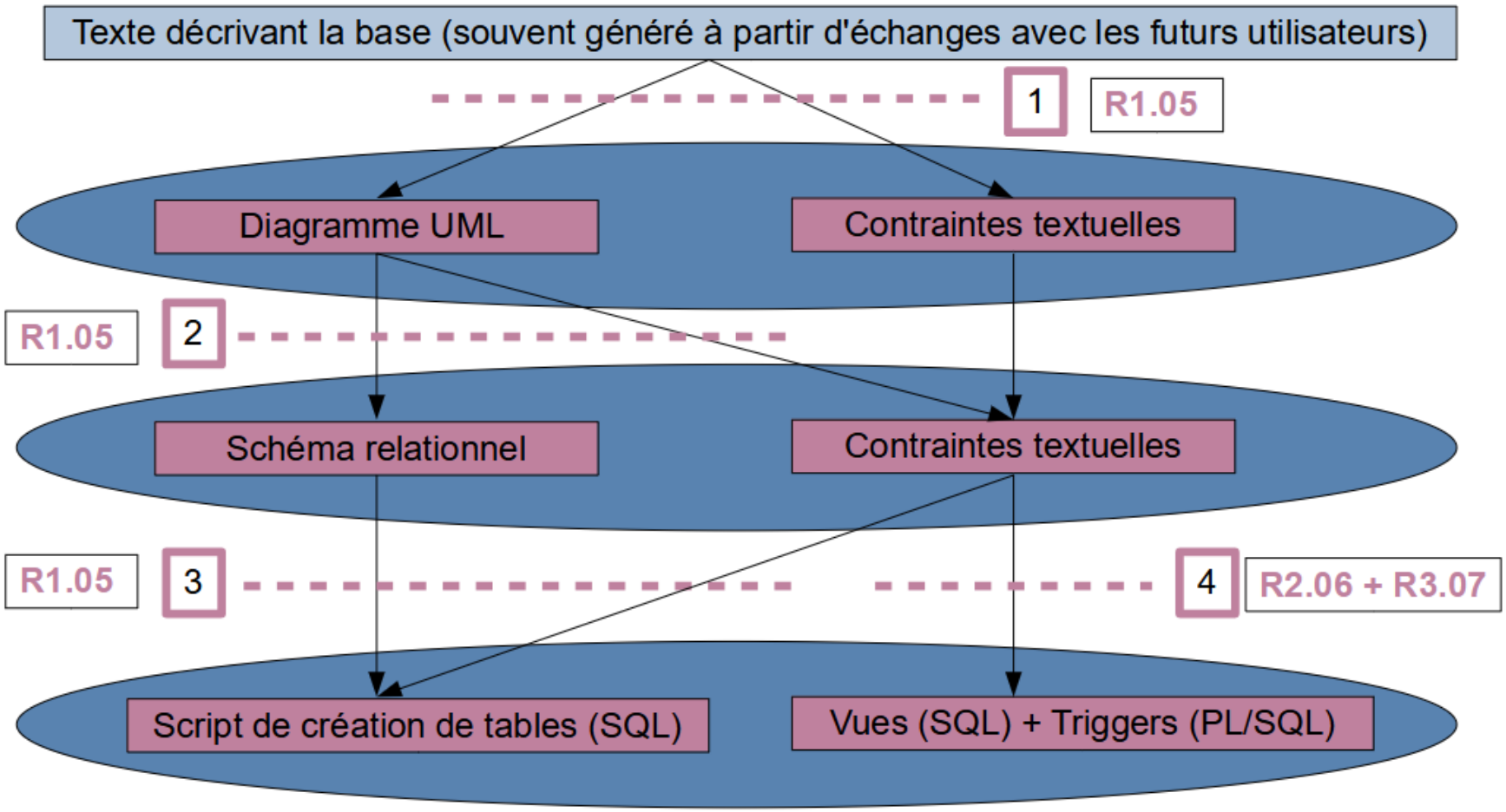
```
FOR compteur IN [REVERSE] inf..sup
    LOOP
        instructions;
    END LOOP;
```

Parmi toutes ces boucles, on utilisera principalement la boucle « pour », mais avec un curseur...

Modélisation des données



Processus de création d'une BD



UML vers SR

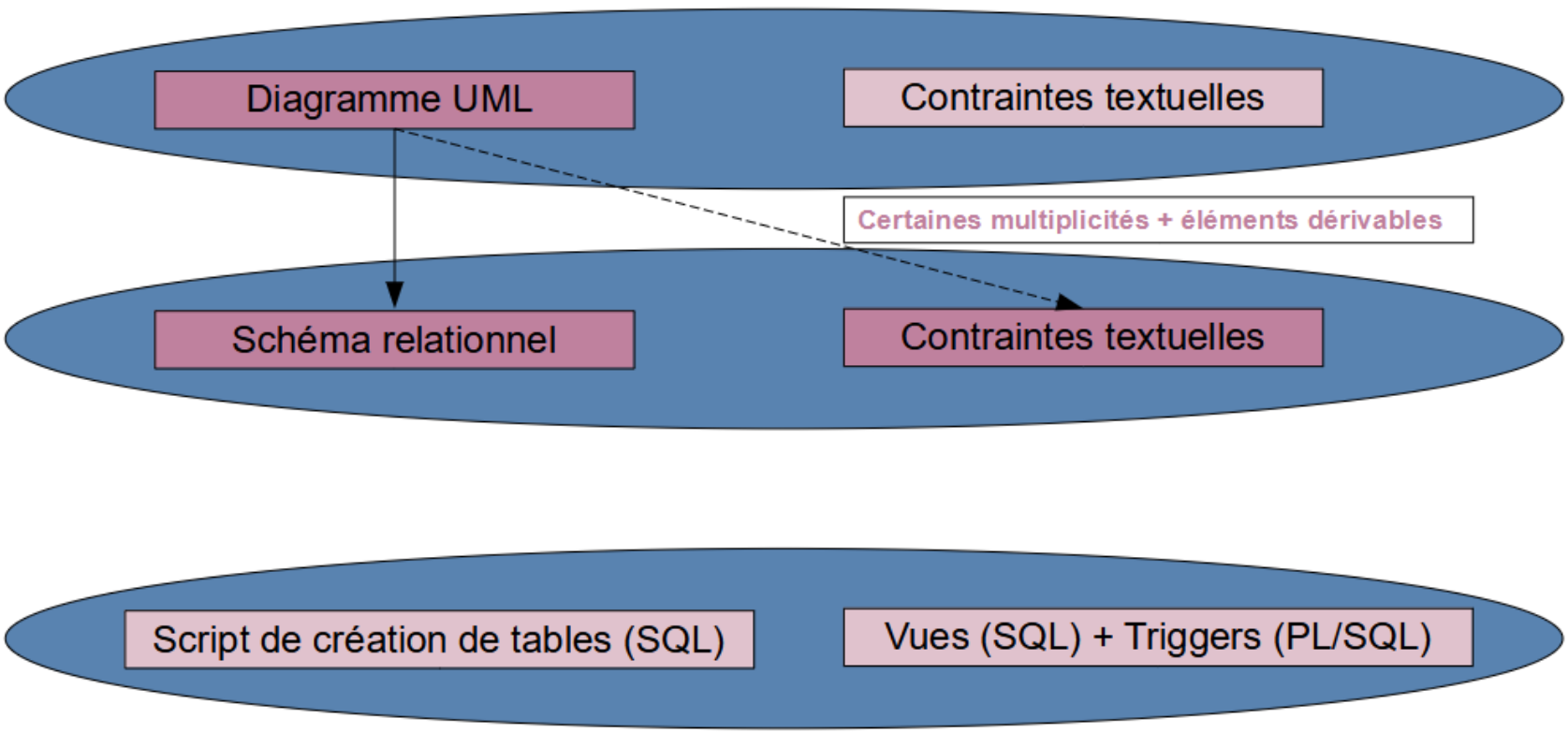


Diagramme UML
Contraintes textuelles
Certaines multiplicités + éléments dérivables
Schéma relationnel
Contraintes textuelles
Script de création de tables (SQL)
Vues (SQL) + Triggers (PL/SQL)

SR vers BD

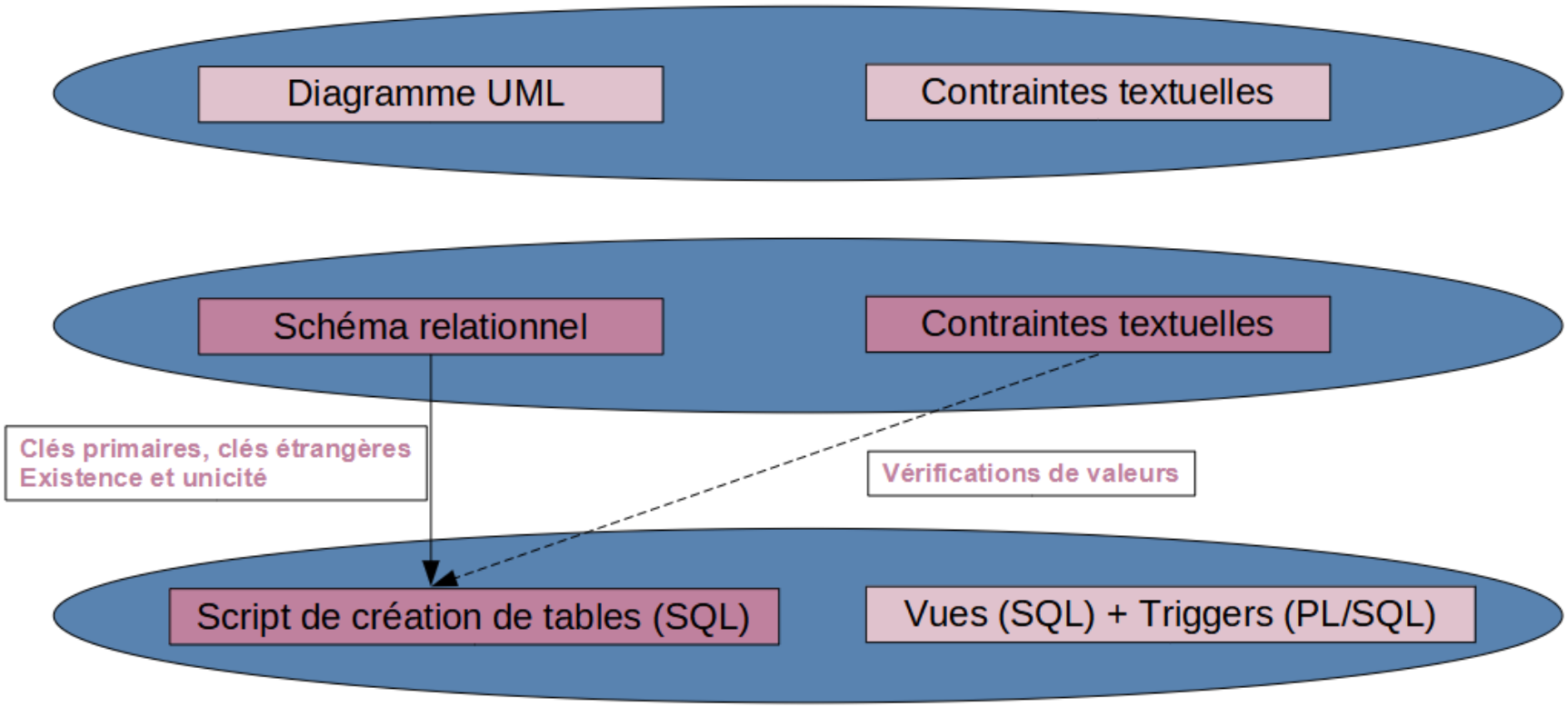
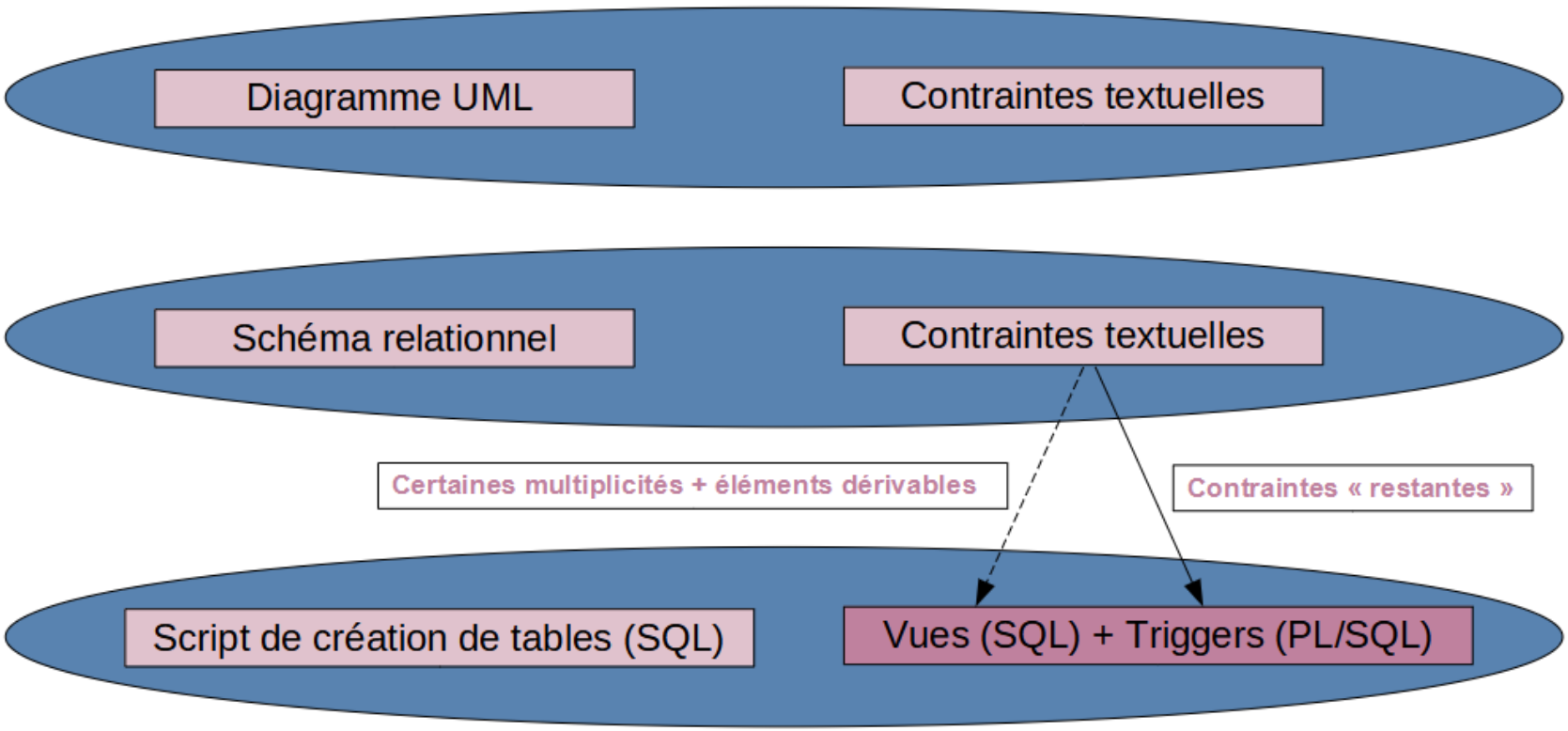
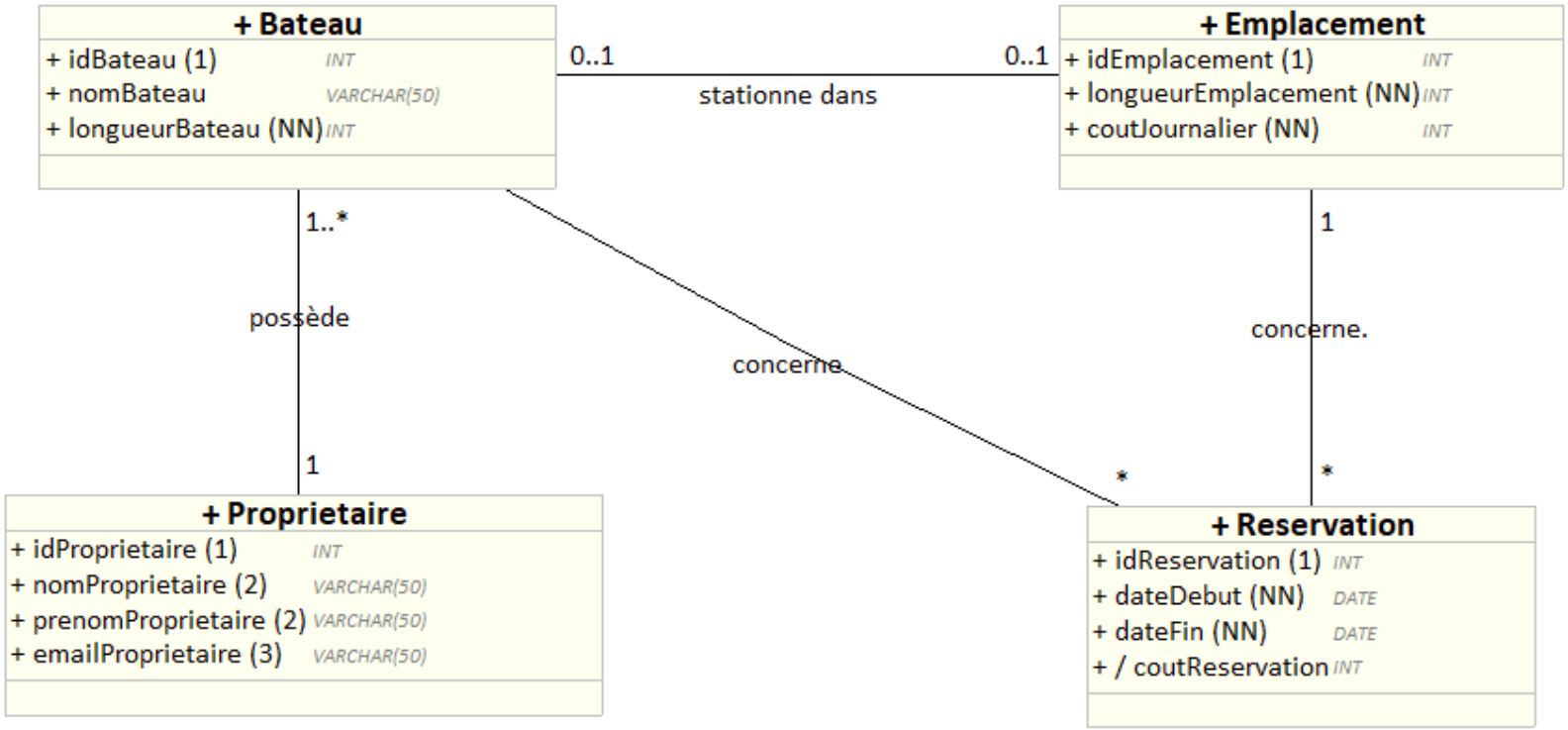


Diagramme UML
Contraintes textuelles
Schéma relationnel
Contraintes textuelles
Clés primaires, clés étrangères
Existence et unicité
Vérifications de valeurs
Script de création de tables (SQL)
Vues (SQL) + Triggers (PL/SQL)

Gestion des contraintes « restantes »

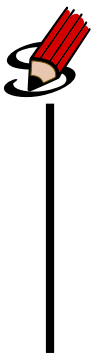


Considérons le **diagramme de classes UML** ci-dessous :



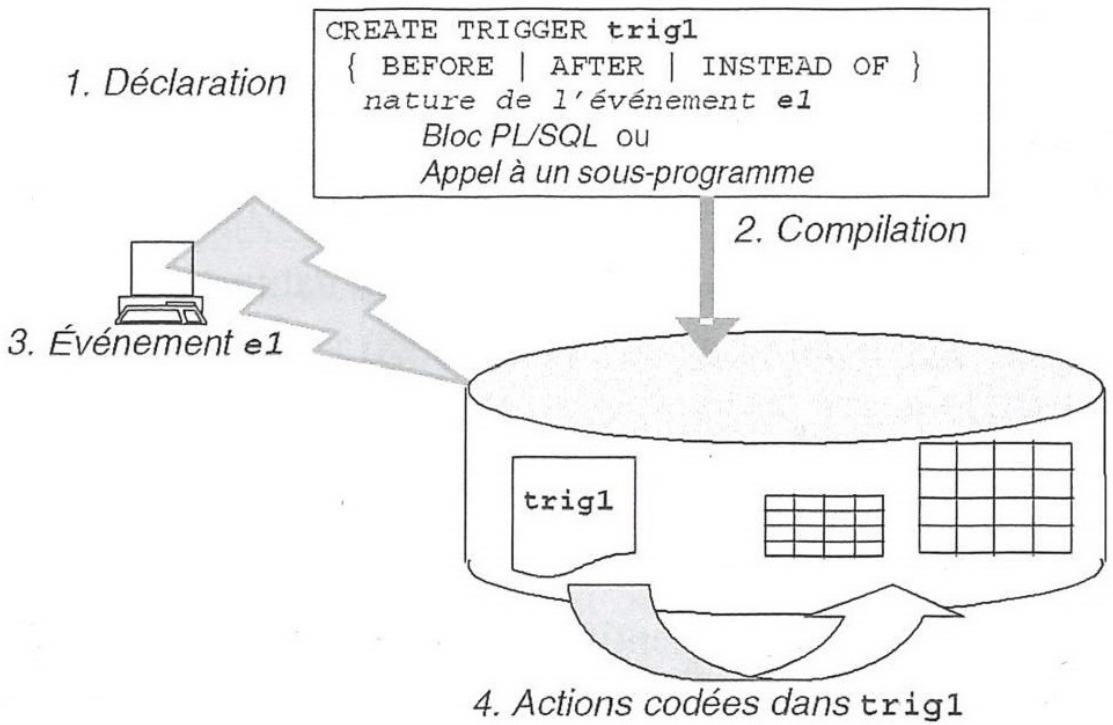
Complété par les **contraintes textuelles** suivantes :

1. la longueur du bateau ne doit pas dépasser 10 m
2. la syntaxe de l'email doit être valide
3. dateDebut $<$ dateFin
4. coutReservation $=$ coutJournalier $\times$ nbJours
5. un emplacement ne peut être réservé pour un bateau dépassant sa longueur
6. un stationnement doit faire l'objet d'une réservation
7. un emplacement ne peut être réservé par deux bateaux différents le même jour



1. Quelle est la contrainte textuelle issue du diagramme de classes ?
2. Quelles contraintes sont implantées dans le script de création de tables ?
3. Quelles contraintes sont gérées par des vues ?
4. Comment peut-on programmer les (trois) contraintes restantes ?

Voici le mécanisme général des déclencheurs :



Un événement déclencheur peut être :

- une instruction INSERT, UPDATE, DELETE
  $\rightsquigarrow$ *déclencheur LMD*
- une instruction CREATE, ALTER, DROP ou une prérogative GRANT, REVOKE
  $\rightsquigarrow$ *déclencheur LDD*
- une erreur spécifique, une connexion ou déconnexion d'un utilisateur
  $\rightsquigarrow$ *déclencheur d'instance*

Seuls les déclencheurs LMD seront traités dans ce cours.

Un déclencheur LMD est un bloc PL/SQL qui s'exécute automatiquement lorsque les données sont manipulées (et leur cohérence remise en question).

Il existe en fait trois sortes de déclencheurs LMD :

- les déclencheurs de lignes
- les déclencheurs d'état
- les déclencheurs *INSTEAD OF*

- les déclencheurs *INSTEAD OF* ne seront pas traités dans ce cours
- les autres nous permettront d'implanter les contraintes « restantes »

Voici la syntaxe générale :

Déclencheur LMD

```
CREATE [OR REPLACE] TRIGGER nom_trigger
{BEFORE|AFTER} {INSERT|UPDATE|DELETE} ON nom_table_mutante
[FOR EACH ROW [WHEN (condition)]]
DECLARE
    ...
BEGIN
    ...
END;
/
```

- L'événement déclencheur peut être « complexe » : INSERT **OR** UPDATE
- Le « / » après le « END ; » assure la compilation du programme

- C'est un déclencheur déclaré **avec** la directive **FOR EACH ROW**
- Il est exécuté autant de fois qu'il y a de lignes concernées par l'événement
- La référence aux valeurs manipulées se fait avec les pseudo-tables **:NEW** et **:OLD**
- La condition d'exécution peut être précisée avec l'option **WHEN (condition [2])**

Interdiction de requêter la table « mutante »

Requêter la table « mutante » provoque une erreur qui malheureusement ne se voit pas à la compilation, mais seulement à l'exécution

2. Expression SQL avec éventuellement NEW ou OLD (sans les « : »), mais ne pouvant inclure ni requêtes ni fonctions PL/SQL

- C'est un déclencheur déclaré **sans** la directive **FOR EACH ROW**
- Il est exécuté une seule fois quelque soit le nombre de lignes concernées
- Il est possible de requêter la table « mutante »

- Les pseudo-tables **:NEW** et **:OLD** ne sont plus accessibles
- L'option **WHEN (condition)** n'est plus disponible

Définir les déclencheurs permettant de gérer les contraintes 5, 6 et 7.

TP 0

1. Exécuter le script de création de tables disponible sur Moodle.
2. Créer les vues utiles (cf. diapo 30 - question 3).
3. Définir les déclencheurs permettant de gérer les contraintes 6, 7 et 8.
4. Écrire un script de tests pour les vues et les déclencheurs.

TP 1

Le sujet est disponible sur Moodle.