Cours 2 : Langage de définition, manipulation et contrôle des données

Objets d’une base de données

- Dans un schéma
  - Tables, vues
  - Index, clusters, séquences, synonymes
  - Packages, procédures, fonctions, déclencheurs
- Hors schéma
  - Utilisateurs, profils, rôles
  - Tablespace

Langage de définition des données (LDD)

- Les trois fonctions de base du LDD
  - Créer :

    CREATE ***type-de-l'objet***
  - Modifier :

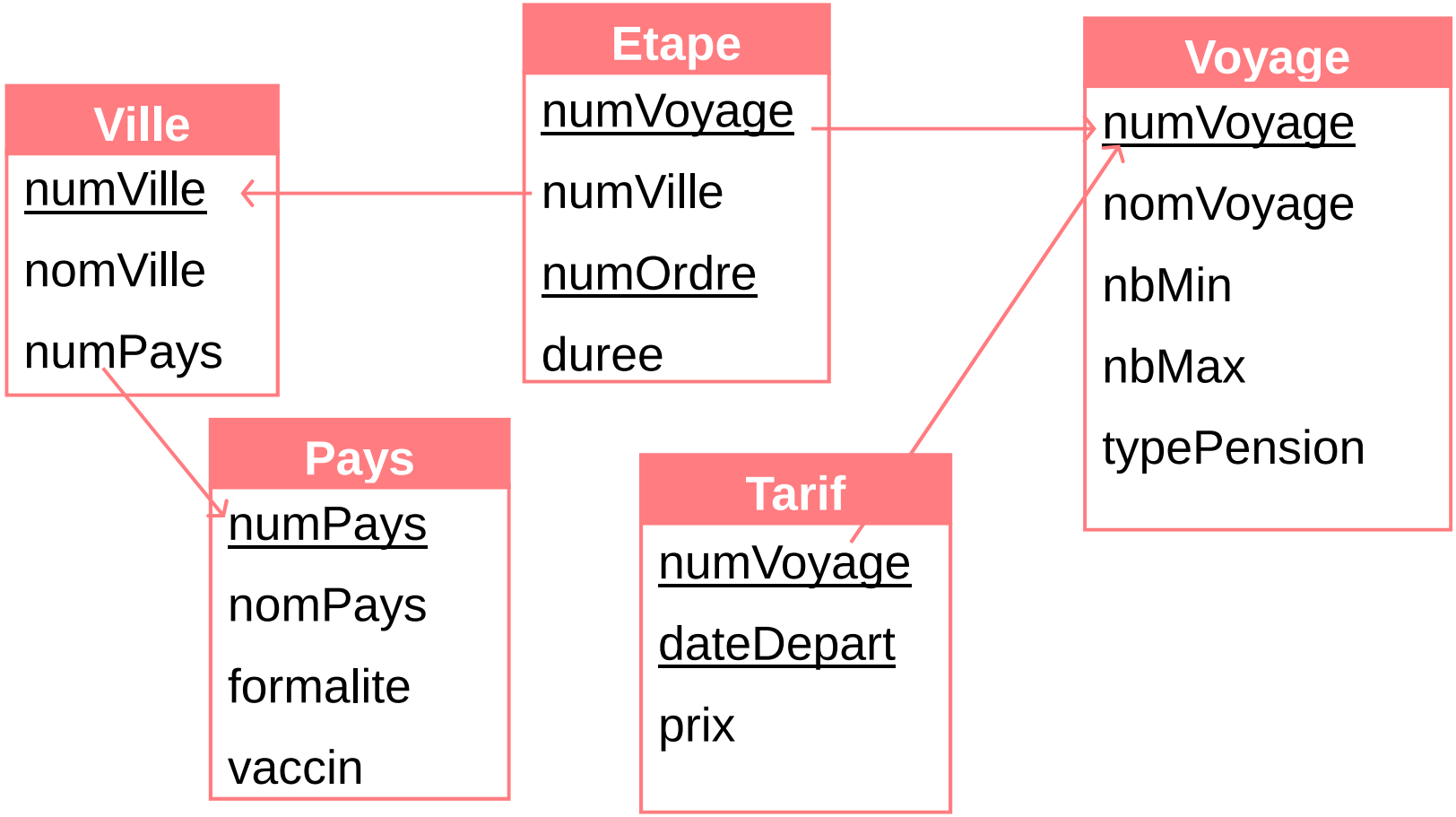
    ALTER ***type-de-l'objet***
  - Supprimer :

    DROP ***type-de-l'objet***

Créer une table : syntaxe

CREATE TABLE ***nom-de-table*** **(**
***nom-attribut*$_1$** ***type-attribut*$_1$**
[DEFAULT ***expression*$_1$**] [*contrainte-attribut*$_1$**],**
… ,
***nom-attribut*$_n$** ***type-attribut*$_n$**
[DEFAULT ***expression*$_n$**] [*contrainte-attribut*$_n$**]**
[, ***contrainte*$_1$*-table*, … , ***contrainte*$_m$*-table*** **])**

Base « Voyages »



Exemple

```
CREATE TABLE voyage (
numVoyage NUMBER(8) PRIMARY KEY,
nomVoyage VARCHAR2(60) NOT NULL,
nbMin NUMBER(8),
nbMax NUMBER(8),
typePension CHAR(2) DEFAULT 'P'
CHECK(typePension IN ('P', 'DP')),
CONSTRAINT CHK_NB CHECK(nbMin <= nbMax)
);
```

Exemple (2)

```
CREATE TABLE voyage (
numVoyage NUMBER(8) CONSTRAINT PK_voyage  PRIMARY KEY,
nomVoyage VARCHAR2(60) CONSTRAINT VAL_nom NOT NULL,
nbMin NUMBER(8),
nbMax NUMBER(8),
typePension CHAR(2) DEFAULT 'P' CONSTRAINT CHK_pension
CHECK(typePension IN ('P', 'DP')),
CONSTRAINT CHK_NB CHECK(nbMin <= nbMax)
);
```

Exemple (3)

```
CREATE TABLE voyage (
numVoyage NUMBER(8),
nomVoyage VARCHAR2(60),
nbMin NUMBER(8),
nbMax NUMBER(8),
typePension CHAR(2) DEFAULT 'P‘,
CONSTRAINT CHK_pension CHECK(typePension IN ('P', 'DP')),
CONSTRAINT PK_voyage PRIMARY KEY(numVoyage),
CONSTRAINT VAL_nom nomVoyage NOT NULL,
CONSTRAINT CHK_NB CHECK(nbMin <= nbMax)
);
```

Rappel

- Nommage des tables sous Oracle :

  schéma.nom-table

- Si la table ***nom-table*** appartient à l'utilisateur (s'il en est le créateur), le nom de schéma est inutile, puisqu'il utilise «son» schéma par défaut

Rappel (2)

- Nommage des attributs sous Oracle :

  schéma.nom-table.nom-attribut

- Si la table ***nom-table*** appartient à l'utilisateur (s'il en est le créateur), le nom de schéma est inutile, puisqu'il utilise «son» schéma par défaut
- Quand il n'y a pas d'ambiguïté, ***schéma.nom-table*** peut être omis

Intérêt des contraintes

- Déclarées lors de la définition des tables
- Non normalisées avant SQL2 (92)
- Simplification du travail du programmeur
- Meilleur intégration des applications utilisant les mêmes données
- Dans un environnement client-serveur, gain substantiel au niveau du trafic réseau

Cinq types de contraintes

1. Valeur obligatoire (**NOT NULL**)
2. Unicité des lignes (**UNIQUE**)
3. Clé primaire (**PRIMARY KEY**)
4. Contrainte d'intégrité référentielles (CIR) (**FOREIGN KEY**, **REFERENCES**)
5. Contrainte de valeurs (**CHECK**)

1. Contrainte de valeur obligatoire

- La clause **NULL** autorise l'attribut à ne pas avoir de valeur pour certaines lignes
- La clause **NOT NULL** indique que l'attribut doit avoir une valeur pour toutes les lignes de la table
- Par défaut : **NULL**

2. Contrainte d’unicité

- La clause **UNIQUE** (associée à une colonne ou un ensemble de colonnes) oblige chaque ligne à avoir une valeur différente pour la (les) colonne(s)
- La (les) colonne(s) peut (peuvent) ne pas avoir de valeur
- La (les) colonne(s) peut (peuvent) être référencée(s) par une clé étrangère
- Un index est automatiquement crée. Il porte le même nom que la contrainte

3. Contrainte de clé primaire

- La clause **PRIMARY KEY** (associée à une colonne ou un ensemble de colonnes) oblige chaque ligne à avoir une valeur non nul différente pour la (les) colonne(s)
- La (les) colonne(s) peut (peuvent) être référencée(s) par une clé étrangère
- Un index est automatiquement crée. Il porte le même nom que la contrainte
- Il n'y a qu'une seule clé primaire par description de table

4. Contrainte d'intégrité référentielle

- Dans une table T, on appelle *clé étrangère* ou *clé externe* tout attribut (ou combinaison d'attributs) qui apparaît comme clé primaire dans une table de référence T_{ref}, appelée *table primaire*
- Les attributs de la clé étrangère et de la clé primaire doivent être définis avec le même type de données

4. Contrainte d'intégrité référentielle (2)

- Contrainte au niveau d'attribut

nom-attribut type-attribut
REFERENCES *table-ref* [(*clé-cand*)]
[ON DELETE CASCADE]

- ***table-ref*** est la table primaire T_{ref}
- ***clé-cand*** est la clé primaire de T_{ref} (dans ce cas, il est facultatif de la désigner) ou un attribut unique de T_{ref}

4. Contrainte d'intégrité référentielle (3)

- Contrainte au niveau de table

FOREIGN KEY (*attr*$_1$, … ,*attr*$_n$)
REFERENCES *table-ref* [(*clé-cand*)]
[ON DELETE CASCADE]

- ***clé-cand*** est la clé primaire de T_{ref} (dans ce cas, il est facultatif de la désigner) ou l'attribut (les attributs) unique(s) de T_{ref}
- ***attr*$_1$**, …, ***attr*$_n$** sont les colonnes de T qui composent la clé étrangère

4. Contrainte d'intégrité référentielle (4)

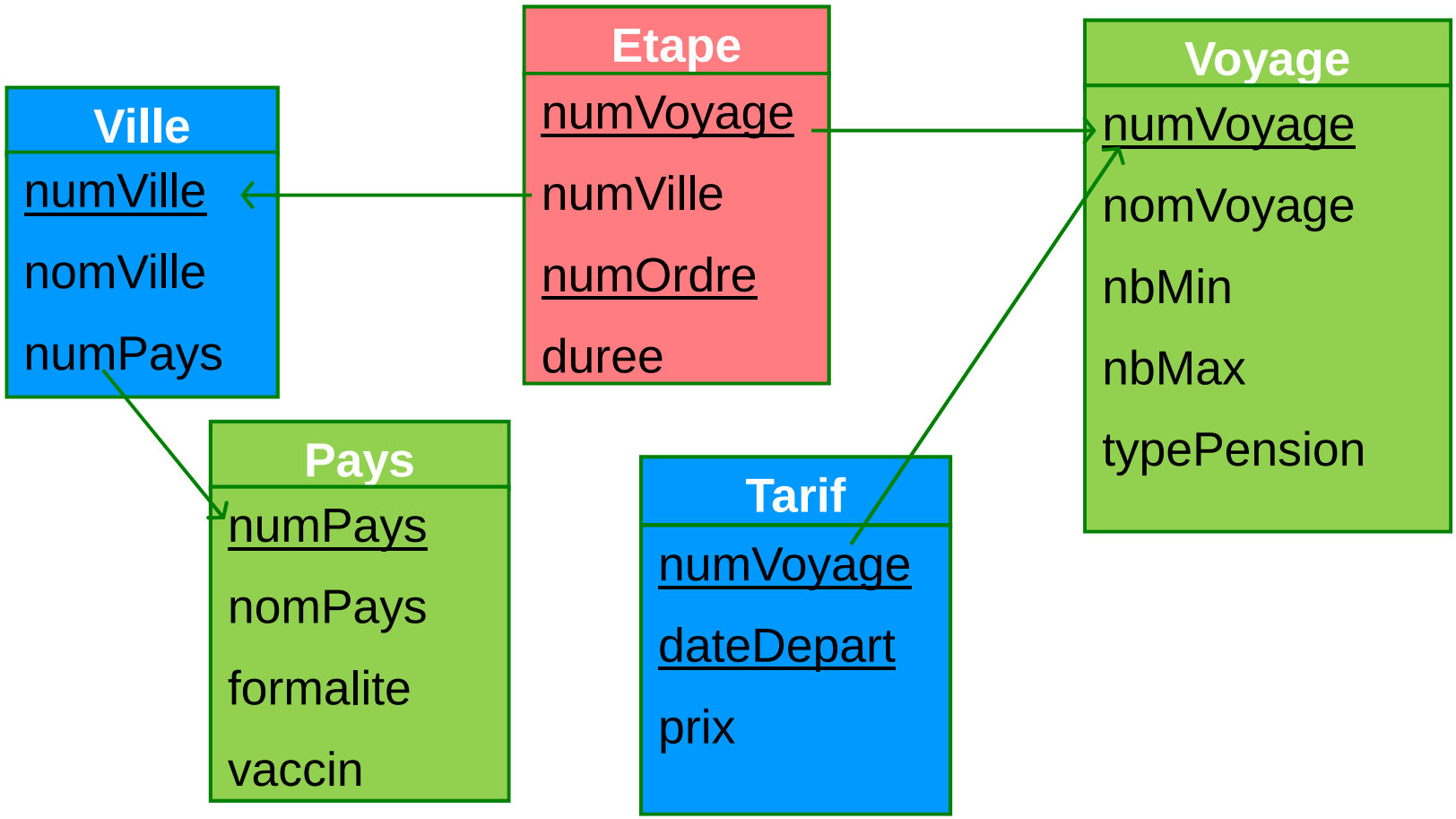
- Pour définir une contrainte **REFERENCES** sur une table T_{ref}, le créateur de la table T doit avoir le droit de créer des références sur cette table
- Lors d'un **INSERT** ou **UPDATE**, la valeur de la clé étrangère doit déjà exister dans la clé primaire de la table primaire. Sinon, l'instruction n'est pas effectuée

4. Contrainte d'intégrité référentielle (5)

- Il n'est pas possible de supprimer des lignes par un ordre **DELETE** dans la table primaire T_{ref} tant qu'il existe dans la table secondaire des lignes ayant une clé étrangère égale à la valeur de la clé primaire de T_{ref}
- Avec la clause **ON DELETE CASCADE**, dans le cas d'une suppression dans la table primaire, les occurrences liées sont automatiquement supprimées dans la table secondaire

Ordre de création des tables :

1 2 3



5. Contrainte de valeurs

[CONSTRAINT *nom-contrainte*]
CHECK (*condition*)

La condition ***condition*** doit être vérifiée à tout moment par toutes les lignes de la table

Exemple

```
CREATE TABLE pays (
numpays NUMBER(8) PRIMARY KEY,
nompays VARCHAR2(20) NOT NULL,
formalite CHAR(2)
CHECK(formalite IN ('CI','P','PV')),
vaccin CHAR(3)
CHECK(vaccin IN ('Oui','Non'))
);
```

Options de création de tables

CREATE TABLE ***nom-table*** **[(*attr*$_1$, …, *attr*$_n$)]**
AS ***requête***

Exemple

```
CREATE TABLE prixVoyage (num,
  nom, type, dep, prix) AS
SELECT v.numVoyage, nomVoyage,
typePension, dateDepart, prix
FROM voyage v, tarif t
WHERE v.numVoyage = t.numVoyage
```

Modification de la définition d'une table

- **ALTER TABLE *nom-table***
  ADD (*$attr_1$ $type_1$*, … , *$attr_n$ $type_n$*)
- **ALTER TABLE *nom-table***
  ADD *définition-contrainte*
- **ALTER TABLE *nom-table***
  RENAME COLUMN *old_attr* TO *new_att*

Modification de la définition d'une table (2)

- **ALTER TABLE *nom-table***
  MODIFY(*attr*$_1$ *type*$_1$, … , *attr*$_n$ *type*$_n$)
- Permet de modifier :
  - Le type
  - La taille
  - La valeur par défaut
  - La contrainte **NOT NULL**

Suppression d'une table

DROP TABLE *nom-table* [CASCADE CONSTRAINTS]

- Permet d'enlever une table de la base de données :
  - élimination des lignes
  - suppressions de la définition de la table du dictionnaire de données
  - suppression des index et des droits associés
  - récupération de l'espace alloué
  - **CASCADE CONTRAINTS** permet de supprimer la table même si elle est référencée par d'autres tables

Langage de modification des données (LMD)

- Les trois fonctions de base du LMD
  - Insertion de nouvelles lignes :

    INSERT
  - Modification de lignes existantes :

    UPDATE
  - Suppression de lignes :

    DELETE

Insertion de données externes

INSERT INTO *nom-table* [(*attr*$_1$, … , *attr*$_n$)]
VALUES (*val*$_1$, … , *val*$_n$)

Lorsque la liste des colonnes à renseigner est mentionnée, seules les colonnes pouvant ne pas être renseignées (valeur **NULL**) ou ayant une valeur par défaut peuvent être omises

Exemple

```
INSERT INTO pays
 VALUES (1, 'Grèce', 'CI', 'Non');

INSERT INTO pays (nomPays, numPays)
 VALUES ('Hongrie', 2);
```

Insertion de données internes

INSERT INTO ***nom-table*** **[(*attr*$_1$, … , *attr*$_n$)]**

requête

Exemple

```
CREATE TABLE ville (
numVille NUMBER(8) PRIMARY KEY,
nomVille VARCHAR2(20) NOT NULL,
numPays NUMBER(8) NOT NULL
  REFERENCES pays
);

INSERT INTO ville SELECT * FROM ffioren.ville;
```

Modification des lignes d'une table

- La commande est composée de deux parties :
  - Sélection des lignes concernées par la mise à jour
  - Sélection des colonnes à modifier pour chaque ligne sélectionnée et mode d'obtention des nouvelles valeurs

Modification des lignes d'une table : syntaxe

UPDATE *nom-table*

SET *attribut*$_i$ **=** *expression* **|** *requête*

...

attribut$_j$ **=** *expression* **|** *requête*

...

(*attribut*$_h$ **...** *attribut*$_k$ **) =** *requête*

[WHERE *condition-modification***]**

Exemple

```
UPDATE tarif
SET prix = prix – 10
WHERE prix > 1000
```

Exemple (2)

```
UPDATE tarif
SET prix = prix – 10
WHERE numVoyage IN
        (SELECT numVoyage
         FROM voyage
         WHERE typePension = 'P')
```

Exemple (3)

```
UPDATE tarif t
SET prix =
 (SELECT 2.15 * SUM(duree)
  FROM etape
  WHERE numVoyage = t.numVoyage)
```

Cela corresponds à affecter comme prix à chaque voyage 2,15 € par jour d'étape

Suppression des lignes d'une table

DELETE FROM *nom-table*

[WHERE *condition*]

Exemple

```
DELETE FROM tarif
WHERE dateDepart BETWEEN
   '01-JAN-19' AND '31-DEC-19'
```

Langage de contrôle des données (LCD)

- Gestion des transactions
  - Transaction : unité logique de traitement qui regroupe un ensemble de commandes SQL et qui est
    - indivisible (atomicité de la transaction)
    - cohérente (toutes les contraintes de la base restent vérifiées après une transition)

Problèmes à gérer

- Cohérence des données
- Accès concurrents
- Reprise après panne

Début d’une transaction

- Pas de commande spécifique
- Première commande SQL exécutée après la fin de la transaction précédente (ou au démarrage de la connexion)

Fin d’une transaction

- Fin explicite
  - **COMMIT** : validation de la transaction
  - **ROLLBACK** : annulation de la transaction
- Fin implicite (Oracle)
  - Exécution d’une commande du LDD : **COMMIT** avant et après la commande

Rollback implicite d'une commande

- Si une commande échoue, tous les effets de la commande sont annulés
  - Commande non LDD : n'affecte pas les autres commandes de la transition qui continue
  - Commande LDD
    - **COMMIT** implicite avant et après la commande
    - Début d'une nouvelle transaction

Contrôle des transactions sous SQL Developer

- Mode transactionnel : est à l'utilisateur de gérer les fins de transactions
- Mode à validation automatique : cocher *Autocommit in SQL Worksheet* dans *Worksheet Parameters* (un **COMMIT** est alors fait automatiquement après chaque commande SQL)