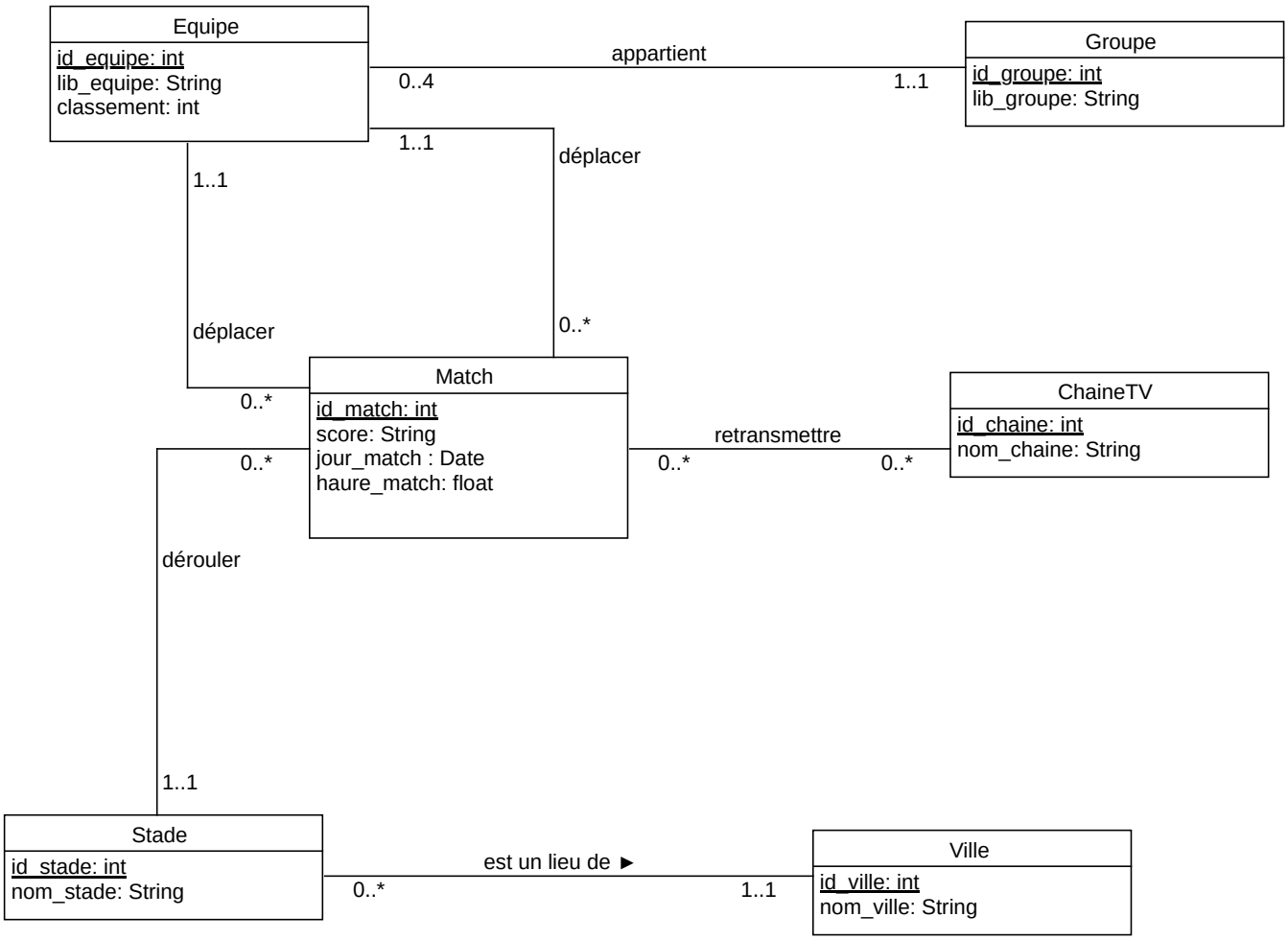
Correction exercices - Module 4: Modélisation

version: 1.0

Exercice 1



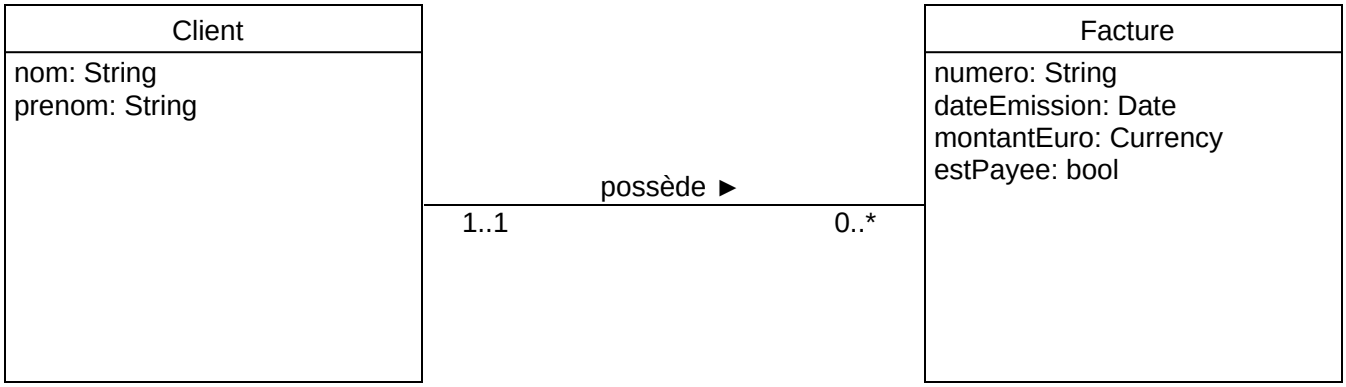
Remarques:

- il est préférable d'utiliser *deux associations* pour modéliser une rencontre entre 2 équipes qu'une seule association plusieurs à plusieurs (*many-to-many*). Cela permet de passer plus facilement du modèle conceptuel ou modèle relationnel. Ici l'entité `Match` est une *classe-association* (ou *table relationnelle* dans la couche relationnelle)

- il est préférable de relier `Stade` à `Ville` plutôt que `Match` à `Ville`. Cette information est déduite sans ambiguïté des associations `dérouler` et `est un lieu de`.

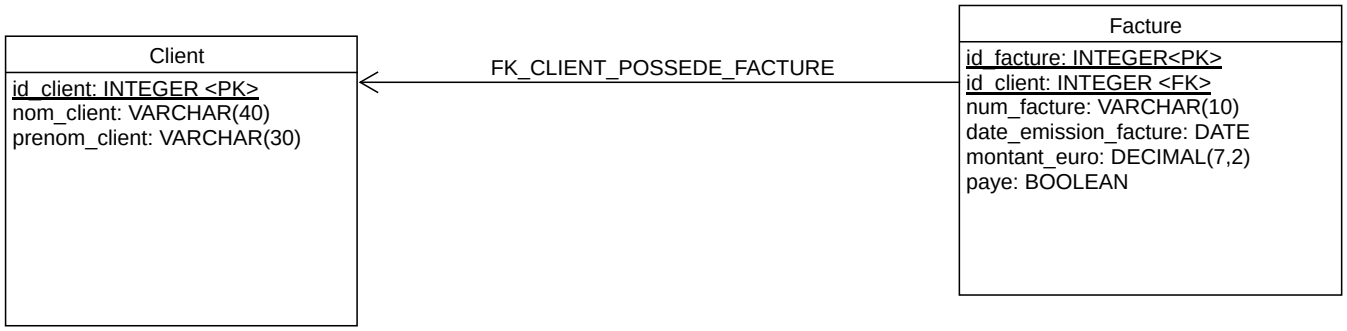
Exercice 2

1. Le diagramme UML avec les associations



2. Passage du modèle conceptuel (diagramme de classes UML) au modèle relationnel.
    1. Une classe => une relation
    2. Chaque relation doit avoir une clé primaire (clé système) => attribut `id INTEGER`
    3. Association *un à plusieurs* => contraintes de **clé étrangère** du côté `plusieurs` (donc la relation Facture)
    4. Types:
        1. String => VARCHAR
        2. Date => DATE (ou TIMESTAMP)
        3. Currency => DECIMAL (le type `DECIMAL` enregistre des valeurs exactes). Ici on va mettre `DECIMAL(7,2)`, ce qui nous permet d'aller de -99999,99 à 99999,99 EUROS

Ce qui donne



3. À présent qu'on a un modèle relationnel satisfaisant, on peut passer à son implémentation en SQL (qu'on appelle *modèle physique* des données). Voici le contenu du script `ddl_modelisation.sql` pour créer les relations.

```sql
USE modelisation;

CREATE TABLE IF NOT EXISTS Client(
    id_client INTEGER PRIMARY KEY,
    nom_client VARCHAR(40) NOT NULL,
    prenom_client VARCHAR(30) NOT NULL
);

CREATE TABLE IF NOT EXISTS Facture(
    id_facture INTEGER PRIMARY KEY,
    id_client INTEGER,
    num_facture VARCHAR(10),
    date_emission DATE,
    montant_euro DECIMAL(7,2),
    paye BOOLEAN DEFAULT false COMMENT 'Par défaut: impayé',
    CONSTRAINT fk_client_possede_facture FOREIGN KEY(id_client) REFERENCES
        Client(id_client)
);

-- Décrire les tables (DESC est l'abréviation de DESCRIBE)
DESC Client;
DESC Facture;
```

4. Pour insérer les données, on peut au choix:
    1. insérer le client *avant* ses factures (a cause de la contrainte FK)
    2. insérer les clients et les factures dans n'importe quel ordre et ajouter les contraintes FK *après* avec ALTER TABLE Client ADD CONSTRAINT

On va choisir l'option 1. Ici, on va créer les id manuellement et non en utilisant `AUTO_INCREMENT`. Si on laissait le SGBD gérer la génération séquentielle des id on pourrait utiliser dans le script la fonction MySQL `LAST_INSERT_ID()` pour récupérer le dernier id généré.

```sql
-- Créer la base de données avec l'user root et donnez tous les privilèges
        dessus à l'user dev:
-- CREATE DATABASE modelisation;
-- GRANT ALL PRIVILEGES ON modelisation.* TO 'dev'@'localhost';

USE modelisation;

-- inline constraint (attention, primary key ne fait pas AUTO_INCREMENT par
        défaut)
-- On peut ajouter ici un AUTO_INCREMENT
CREATE TABLE IF NOT EXISTS Client(
    id_client INTEGER PRIMARY KEY,
    nom_client VARCHAR(40) NOT NULL,
    prenom_client VARCHAR(30) NOT NULL
);
```

```
        CREATE TABLE IF NOT EXISTS Facture(    id_facture INTEGER PRIMARY KEY,
    id_client INTEGER,    num_facture VARCHAR(10),    date_emission DATE,
    montant_euro DECIMAL(7, 2),
    paye BOOLEAN DEFAULT false COMMENT 'Par défaut: impayé',
    CONSTRAINT fk_client_possede_facture FOREIGN KEY(id_client) REFERENCES
        Client(id_client)
);              -- Décrire les tables (DESC est l'abréviation de DESCRIBE)
-- DESC Client;-- DESC Facture;        -- Insertion des données
-- INSERT INGORE permet d'ignorer les insert qui échoue (ex: contrainte violée
        sur PK car la donnée a déjà été insérée) et de continuer la requête
        malgré les erreurs (pratique pour debug).

-- Comme pas d'AUTO_INCREMENT sur la clef primaire, on doit l'insérer
        manuellement
INSERT IGNORE INTO    Client (id_client, nom_client, prenom_client)VALUES
    (1, 'John', 'Doe'),    (2, 'Jane', 'Doe');
-- DATE attend les dates au format YYYY-MM-DD
-- Comme pas d'AUTO_INCREMENT sur la clef primaire, on doit l'insérer
        manuellement
INSERT IGNORE INTO     Facture (        id_facture,        id_client,
        num_facture,        date_emission,        montant_euro,        paye
    )        VALUES    (1, 1, 'F-900-08', '2022-12-12', 120.5, FALSE),
    (2, 1, 'F-500-02', '2023-13-01', 90, TRUE),
    (3, 2, 'Z-500-03', '2023-01-01', 1000, FALSE),
    (4, 2, 'J-400-02', '2023-09-11', 800, TRUE),
    (5, 2, 'F-434-04', '2023-12-22', 400, FALSE);
    (6, 2, 'J-333-05', '2023-02-07', 1255, FALSE);
```

5. Voici les requêtes (DQL) pour récupérer les résultats attendus

```
-- 1. Lister toutes les factures impayées
SELECT * FROM Facture WHERE paye;

-- 2. Lister tous les numéros de facture impayée dont le montant est supérieur
        à 500 EUROS
SELECT num_facture FROM Facture WHERE NOT paye AND montant_euro > 500;

-- 3. Lister toutes les factures de 2022
SELECT num_facture FROM Facture WHERE YEAR(date_emission) = 2022;

-- 4. Lister toutes les factures impayées de 2023
SELECT num_facture FROM Facture WHERE YEAR(date_emission) = 2023 AND NOT paye;
```

Exercice 3 : manipulation des données temporelles

```
-- 1. La date du 20/01/2023 dans 31 jours
SELECT ADDDATE('2023-01-20', 31) "Date dans 31 jours" ;
-- 2. La date du 20/01/2023 (date+temps) à 23 heures - 1 microseconde dans 1
        jour et 1 microseconde.
SELECT ADDTIME('2023-01-20 22:59:59.999999','1 0:0:0.000001') "exemple
        ADDTIME";
-- 3. La date dans 4 mois
```

```
SELECT DATE_ADD(CURRENT_TIMESTAMP, INTERVAL '4' MONTH) "Rendez-vous dans 4
        mois";
-- 4. La date dans 7 jours et 1h30
SELECT DATE_ADD(CURRENT_TIMESTAMP, INTERVAL '7 01:30:00' DAY_SECOND) "RDV 1sem
        1h30";
-- 5. La date courante, mais en anglais
SELECT DATE_FORMAT(SYSDATE(), '%W %d %M %Y') %Y') "Today in English";
```

Exercice 4

```
--1 Type du poste p8
SELECT nPoste, typePoste
  FROM Poste WHERE nPoste = 'p8';

--2 Noms des logiciels UNIX
SELECT nomLog
  FROM Logiciel
  WHERE typeLog = 'UNIX';

--3 Nom, adresse IP, numéro de salle des postes de type UNIX ou PCWS.
SELECT nomPoste, indIP, ad, nSalle
  FROM poste
  WHERE typePoste = 'UNIX'
  OR    typePoste = 'PCWS';

--4 Même requête pour les postes du segment 130.120.80 triés
--par numéro de salle décroissant
SELECT nomPoste, indIP, ad, nSalle
  FROM poste
  WHERE (typePoste = 'UNIX'
  OR    typePoste = 'PCWS')
  AND   indIP = '130.120.80'
    ORDER BY nSalle DESC;

--5  Numéros des logiciels installés sur le poste p6.
SELECT  nLog
  FROM  Installer
  WHERE nPoste = 'p6';

--6 Numéros des postes qui hébergent le logiciel log1.
SELECT  nPoste
  FROM  Installer
  WHERE nLog = 'log1';

--7 Nom et adresse IP complète (ex : 130.120.80.01) des postes de type TX
SELECT  nomPoste, CONCAT(indIP,'.',ad)
  FROM  Poste
  WHERE typePoste = 'TX';


--8
SELECT nPoste, COUNT(nLog)
```

```
  FROM installer   GROUP BY (nPoste);         --9
SELECT nSalle, COUNT(nPoste)  FROM Poste  GROUP BY (nSalle)  ORDER BY 2;
        --10     SELECT nLog, COUNT(nPoste)  FROM Installer   GROUP BY (nLog);
        --11     SELECT AVG(prix)  FROM Logiciel  WHERE typeLog = 'UNIX';
        --12     SELECT MAX(dateAch)  FROM Logiciel;         --13
SELECT nPoste FROM Installer  GROUP BY nPoste  HAVING COUNT(nLog)=2;
--14     SELECT COUNT(*) FROM
  (SELECT nPoste FROM Installer GROUP BY nPoste HAVING COUNT(nLog)=2) T;
        --15     SELECT DISTINCT typeLP FROM Types
 WHERE typeLP NOT IN (SELECT DISTINCT typePoste FROM Poste);         --16
SELECT DISTINCT typeLog FROM Logiciel
  WHERE typeLog IN (SELECT typePoste FROM Poste);         --17
SELECT DISTINCT typePoste FROM Poste
  WHERE typePoste NOT IN (SELECT typeLog FROM Logiciel);
```

Exercice 5

Correction à venir.

Exercice 6

Correction à venir.

Références utiles

- Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC
- Date and Time Functions