- **L3 MIASHS/Ingémath/METIS**
- **Université Paris Cité**
- Année 2024-2025
- Course Homepage

- Moodle

🚲 Un service de vélos partagés permet aux abonnés d'utiliser des vélos mécaniques ou électriques. Chaque vélo porte un numéro. Un vélo entre en service à une date donnée, Il est retiré du service à une date donnée (pas connue à l'avance).

🚲 Chaque abonné souscrit un abonnement pour une durée d'un an à une date donnée. Chaque abonnement possède un numéro, et un titulaire qui possède un nom, un prénom, un âge et un sexe. Un abonnement n'est pas renouvelable. Cela n'empêche pas une personne de souscrire un autre abonnement.

🏁 Un abonné emprunte un vélo à une bornette à un instant de départ donné. Une fois le trajet effectué, l'abonné verrouille le vélo sur une bornette à l'instant d'arrivée.

🚵 Les bornettes sont situées sur des stations. Chaque station contient un nombre $n$ (qui peut varier d'une station à l'autre, mais est constant pour une station donnée) de bornettes numérotées de 1 à $n$. Les stations sont numérotées, elles ont un nom et une adresse. Chaque station a une longitude et une latitude.

**Question 1 (4pts)**

Proposer un diagrammme entité-association (EA) correspondant à cette modélisation.
On attend un dessin selon les conventions du cours, pas une énumération.

**Conseils**

- Distinguer entités fortes et faibles
- Distinguer associations fortes et faibles
- Pour chaque entité préciser l'identifiant (éventuellement relatif)
- Préciser les cardinalités pour chaque participation à une association

**Solution**

**Entités**

(1) `Abonné`
- `AbonnéID` (Identifiant)
- `Nom`
- `Prénom`
- `Age`
- `Genre`
- `DébutDate` (de souscription)

(2) `Velo`
- `VeloID` (PK)
- `Type` (Mecanique ou Electrique)
- `DebutServiceDate`
- `FinServiceDate`

(3) `Bornette` (entité faible)
- `BornetteID` (Identifiant relatif)

(4) `Station`
- `StationID` (Identifiant)
- `Nom`
- `Longitude`
- `Latitude`
- `Addresse`
- `#bornettes`

**Solution**

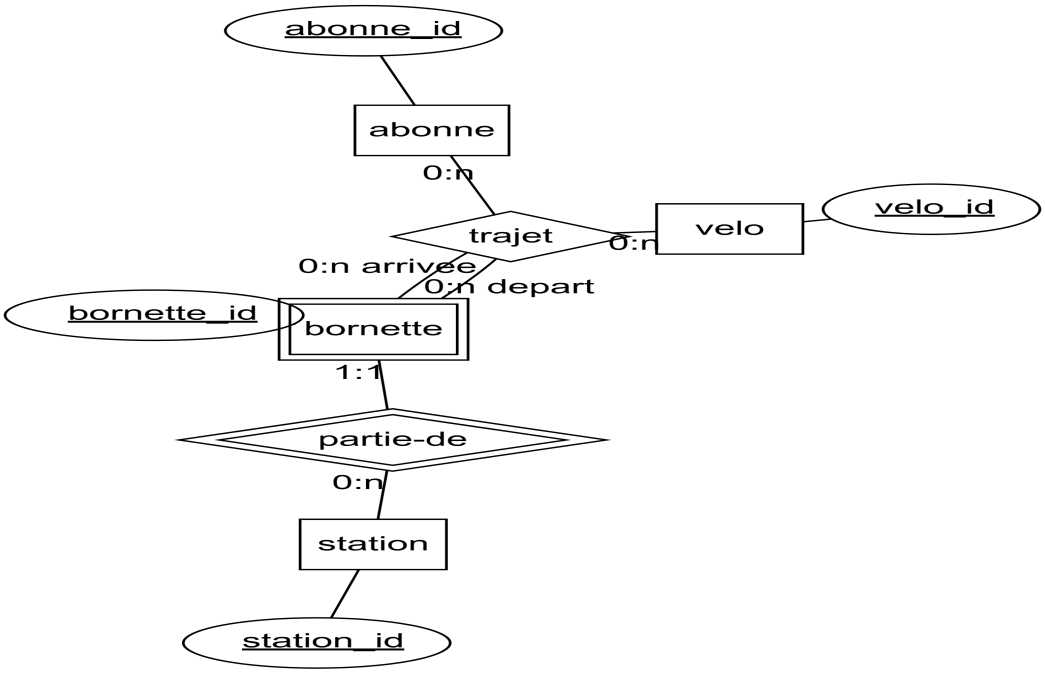
**Associations**

1. Trajet
   - `StartTime`
   - `EndTime`

   Entités participantes
   - `Abonné` 0 :n
   - `Vélo` 0 :n
   - `Bornette` (rôle : Départ) 0 :n
   - `Bornette` (rôle : Arrivée) 0 :n
2. Station–Bornette

   Entités participantes
   - `Bornette` 1 :1
   - `Station` 1 :n

C'est un lien *partie de* entre une entité faible (`Bornette`) et une entité forte (`Station`). Une station comporte plusieurs bornettes.




**Question 2 (2pts)**

Lister les contraintes externes

- Un vélo ne peut pas être emprunté simultanément par deux abonnés
- Le nombre de bornettes rattachées à une station ne peut excéder la limite fixée pour la station
- Un vélo ne peut effectuer de trajets qu'entre sa date de début de service et sa date de retrait de service.
- Un abonné ne peut pas effectuer de trajet avant la date de début ou après la date de fin d'abonnement.
- Une bornette ne peut pas être occupée par deux vélos simultanément.

**Question 3 (2pts)**

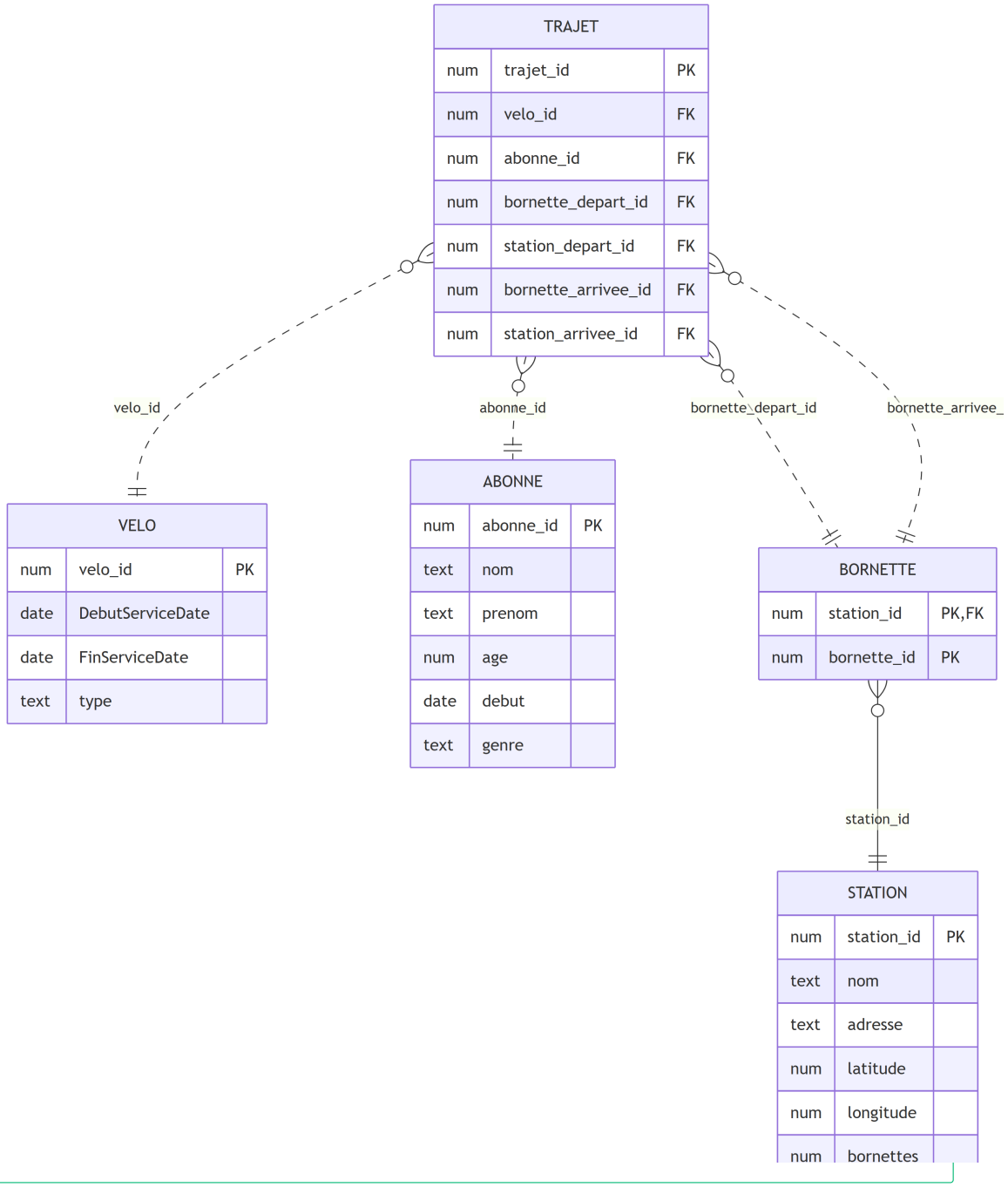
Proposer une traduction en pattes de corbeau du diagramme EA proposé en réponse à la première question.

**Conseil**

Précisez

- une clé primaire pour chaque table,
- les tables dites intermédiaires,
- pour les liens matérialisant les contraintes référentielles, préciser s'ils sont identifiant ou non.

**Solution**




**Question 4 (1pt)**

Proposer un mécanisme pour mettre en place les contraintes externes en SQL lorsque c'est possible sans utiliser les gachettes (`TRIGGER`).

On écrit ici en SQL, le schéma correspondant,

**Solution**

```sql
CREATE TABLE Abonné (
    AbonnéId INT PRIMARY KEY,
    Nom VARCHAR(100),
    Prénom VARCHAR(100),
    Age INT,
    Genre CHAR(1),
    DébutDate DATE
);
```

```sql
CREATE TABLE Velo (
    VéloId INT PRIMARY KEY,
    Type ENUM('Mecanique', 'Electrique'),

);
```

```sql
CREATE TABLE Station (
    StationID INT PRIMARY KEY,
    Nom VARCHAR(100),
    Longitude DECIMAL(9,6),
    Latitude DECIMAL(9,6),
    Adresse VARCHAR(255),
    `#Bornettes` INT
);
```

```sql
CREATE TABLE Bornette (
  StationID INT,
  BorneID INT,
  PRIMARY KEY (StationID, BorneID),
  FOREIGN KEY (StationID) REFERENCES Station.StationID
)
```

```sql
CREATE TABLE Trajet (
    TrajetID INT PRIMARY KEY,
    DébutTS TIMESTAMP,
    FinTS TIMESTAMP,
    AbonnéID INT,
    VéloID INT,
    BornetteDépartID INT,
    BornetteArriveeID INT,
    StationDepartID INT,
    StationArriveeID INT,
    FOREIGN KEY (AbonnéID) REFERENCES Abonné(AbonnéID),
    FOREIGN KEY (VeloID) REFERENCES Velo(VeloID),
    FOREIGN KEY (StationDepartID, BornetteDépartID)
      REFERENCES Bornette(StationID, BornetteID),
    FOREIGN KEY (StationArriveeID, BornetteArriveeID)
      REFERENCES Bornette(StationID, BornetteID)
);
```

**Solution**

On suppose que le schéma est muni des dépendances fonctionnelles déduites de la question précédente et de celles qui se déduisent des contraintes de clé primaire. On note cet ensemble de dépendances fonctionnelles $\Sigma$.

**Question 5 (1pt)**

Votre schéma est-il en Forme Normale de Boyce-Codd ?

**Solution**

**Question 6 (2pt)**

Quelles actions faut-il effectuer sur votre base pour traduire les événements suivants :

a. Souscription d'un abonnement.
b. Mise en service d'un vélo.
c. Retrait de service d'un vélo.
d. Décrochage d'un vélo.
e. Accrochage d'un vélo.

👉 On n'attend pas du code. Juste une phrase qui décrit l'opération à effectuer.

**Solution**

a. Insertion d'un nouveau tuple dans la table `ABONNE`
b. Insertion d'un nouveau tuple dans la table `VELO` avec une date de retrait de service `NULL` ou `'infinity'::date`
c. Mise à jour d'un tuple dans la table `VELO`, `RetraitServiceDate` est affecté de la date courante
d. Insertion d'un nouveau tuple dans la table `TRAJET`. `FinTS`, `BornetteArriveeID` `StationArriveeID` sont `NULL`
e. Mise à jour d'un tuple dans la table `TRAJET` (le tuple correspondant au trajet en cours du vélo raccroché), modification de `FinTS`, `BornetteArriveeID` `StationArriveeID`.

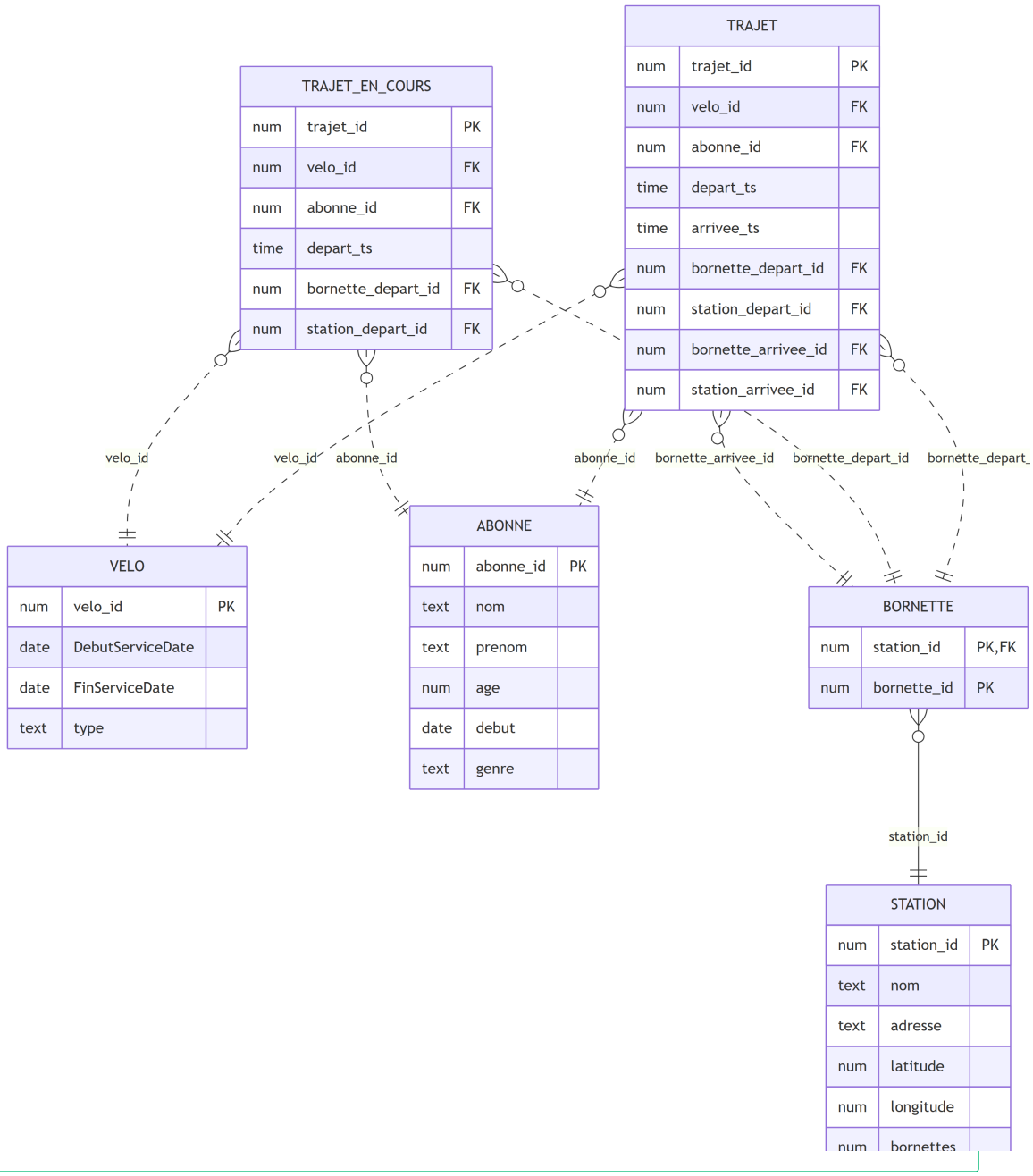
On note que l'accrochage d'un vélo, supposera la recherche du trajet en cours dans la table `TRAJET`, puis la mise à jour du tuple concerné.
Une partie des requêtes qui suivent sera compliquée à écrire, parce qu'avec cette modélisation, les trajets en cours sont noyés dans la table `TRAJET`. Pour les repérer, il faut examiner la vaneur de `FinTS` (`NULL` ou `'infinity'::timestamp`).
Pour se faciliter la vie, nous allons retoucher la modélisation présentée plus haut, introduire une nouvelle table `TRAJET_EN_COURS`.

**Solution**

```
CREATE TABLE Trajet (
    TrajetID INT PRIMARY KEY,
    DébutTS TIMESTAMP,
    FinTS TIMESTAMP,
    AbonnéID INT,
    VéloID INT,
    BornetteDépartID INT,
    BornetteArriveeID INT,
    StationDepartID INT,
    StationArriveeID INT,
    FOREIGN KEY (AbonnéID) REFERENCES Abonné(AbonnéID),
    FOREIGN KEY (VeloID) REFERENCES Velo(VeloID),
    FOREIGN KEY (StationDepartID, BornetteDépartID)
      REFERENCES Bornette(StationID, BornetteID),
    FOREIGN KEY (StationArriveeID, BornetteArriveeID)
      REFERENCES Bornette(StationID, BornetteID)
);
```

```
CREATE TABLE Trajet_en_cours (
    TrajetID INT PRIMARY KEY,
    DébutTS TIMESTAMP,
    AbonnéID INT UNIQUE,
    VéloID INT  UNIQUE,
    BornetteDépartID INT,
    StationDepartID INT,
    FOREIGN KEY (AbonnéID) REFERENCES Abonné(AbonnéID),
    FOREIGN KEY (VeloID) REFERENCES Velo(VeloID),
    FOREIGN KEY (StationDepartID, BornetteDépartID)
      REFERENCES Bornette(StationID, BornetteID),
);
```

**Solution**




**Solution**

Ce modification du schéma, facilite la mise en place de certaines contraintes.

- Pour les tuples de la table `TRAJET_EN_COURS`, les contraintes d'exclusions se réduisent à des contraintes d'unicité sur `velo_id` et `abonne_id`.
-

**Attention**

Dans la suite, vous formulerez les requêtes dans le schéma relationnel défini par votre schéma en pattes de corbeau.
🎓 : 1 point par requête

**Requête 1**

Liste des trajets en cours à un instant donné

**Solution**

**Requête 2**

Liste des vélos en trajet depuis plus d'une heure.

**Solution**

**Requête 3**

Nombre de trajets initiés durant chaque heure de la journée pendant le mois de juin 2024.

**Solution**

**Requête 4**

Liste des velos qui ont participé à un trajet commencé et achevé dans la même station pendant la dernière semaine.

**Solution**

**Remarque**

**Requête 5**

Lister d'éventuels couples de trajets suspects impliquant le même vélo à un même instant.

**Solution**

**Requête 6**

Liste des stations qui ont été vides ou pleines pendant la semaine écoulée.

**Solution**

**Requête 7**

Liste des vélos en service qui n'ont pas roulé depuis plus d'un mois.

**Solution**

**Requête 8**

Pour chaque couple de stations, durée moyenne des trajets entre la station de départ et la station d'arrivée.

**Solution**

**Requête 9**

Lister pour chaque station le nombre de bornettes occupées à l'instant courant.

**Solution**

**Requête 10**

Lister les trajets suspects dont le vélo n'est pas en service.

**Solution**

**Solution**

En PostgreSQL, pour définir un intervalle à l'aide de deux dates `debut` et `fin`, il suffit d'écrire `(début, fin)`. L'intervalle ne contient pas la date de fin.
Pour tester l'intersection/le recouvrement de deux intervalles, on utilise l'opérateur `OVERLAPS`

```
bd_2023-24=# SELECT
  ('2025-01-03'::date, '2025-01-10'::date) OVERLAPS
  ('2025-01-10'::date, '2025-01-15'::date)  ;
 overlaps
----------
 false
(1 row)

SELECT
  ('2025-01-03 20:26:00'::timestamp, '2025-01-03  21:31:01'::timestamp) OVERLAPS
  ('2025-01-03 20:50:04'::timestamp, '2025-01-03 21:45:00'::timestamp)  ;
 overlaps
----------
 t
(1 row)
```

Pour spécifier un intervalle semi-infini (dont la fin n'est pas connue), on peut utiliser `'infinity'::timestamp` pour décrire la borne supérieure.

```
bd_2023-24=#  SELECT
  ('2025-01-03 20:26:00'::timestamp, 'infinity'::timestamp) OVERLAPS
  ('2025-01-03 21:32:04'::timestamp, '2025-01-03 21:45:00'::timestamp)  ;
 overlaps
----------
 t
(1 row)
```

En PostgreSQL, `current_timestamp` s'évalue à l'instant courant (de type `timestamp` avec `timezone`).
En PostgreSQL, pour extraire le mois d'un objet `dd` de type `date`, vous pouvez utiliser `EXTRACT(MONTH FROM dd)`. Le résultat est un entier entre 1 et 12, 1 pour janvier, …

```
postgres=# SELECT
  current_timestamp AS instant,
  EXTRACT( MONTH FROM current_timestamp::date) AS le_mois ;
           instant            | le_mois
------------------------------+---------
 2025-06-05 20:26:12.556256+02 |       6
```

Pour définir un intervalle de temps, il suffit de décrire l'intervalle par une chaine de caratères et de convertir le résultat en type `interval`

```
bd_2023-24=# select '2025-06-05 21:10:38.732237+02'::timestamp - '7 days'::interval ;
         ?column?
----------------------------
 2025-05-29 21:10:38.732237
```