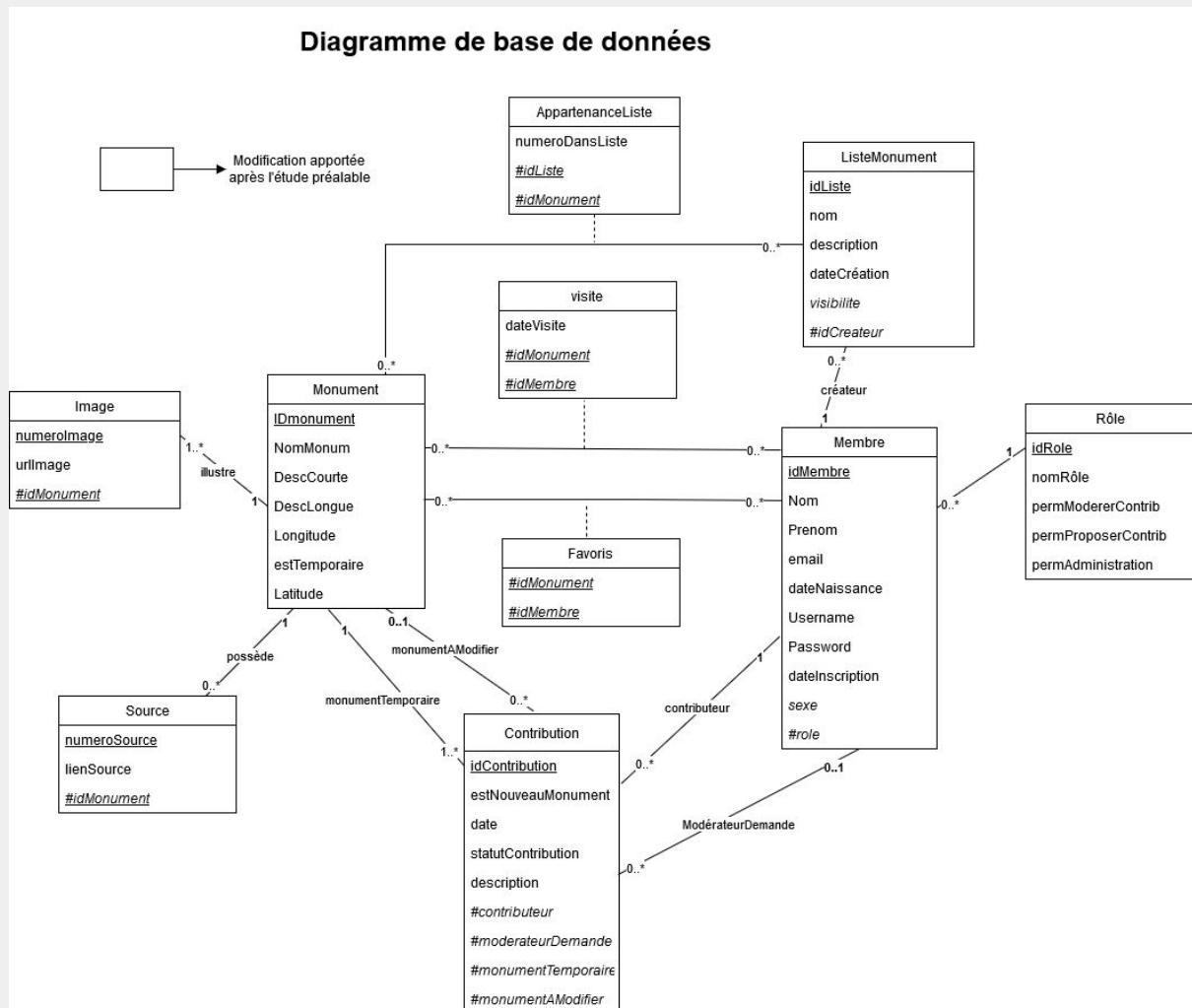


Travail sur la base de données

itération 2

Diagramme de synthèse à l'issue de l'itération 1 :



Tokens pour les urls

1 - Conception / Explication

Avec la base actuelle il est impossible de générer des urls "non devinables" pour les monuments, les listes de monuments et les membres.

Pour cette raison nous avons décidé d'ajouter une colonne TOKEN de type varchar(20) aux trois tables susmentionnées qui serviront à générer des urls et liens de partage indépendamment de leur clé primaire dans la table.

Système de monument privé

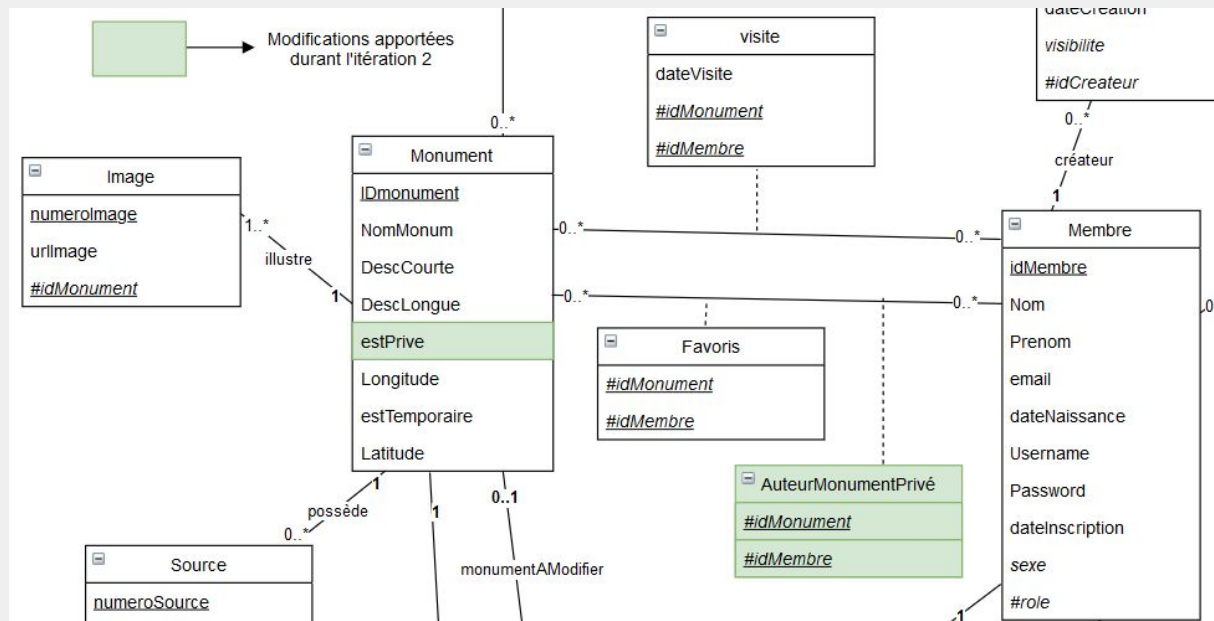
1 - Conception / Explication

Il est ressorti de la soutenance de l'itération 1 la nécessité de concevoir un système de monument privé.

Un utilisateur pourra donc au moment de l'ajout d'un monument choisir de mettre celui-ci en privé. Le monument sera alors uniquement accessible par son créateur ou tout autre personne possédant un lien vers le monument.

Un monument privé est automatiquement validé : il ne rentre pas dans le système de contribution.

D'un point de vue conception il faut ajouter un attribut booléen "estPrivé" à la table Monument et ajouter une table AuteurMonumentPrivé pour enregistrer l'auteur de chaque monument privé (nécessaire car on ne passe pas par le système de contribution classique) :



2 - SQL

```

CREATE TABLE `monument` (
  -- ...
  -- ...
  `estPrivé` BOOLEAN NOT NULL DEFAULT FALSE ,
  PRIMARY KEY (`idMonument`)
) ENGINE = InnoDB;

```

```

CREATE TABLE `auteurMonumentPrivé` (
  `idMonument` INT UNSIGNED NOT NULL ,
  `idMembre` INT UNSIGNED NOT NULL ,
  PRIMARY KEY (`idMonument`, `idMembre`)
) ENGINE = InnoDB;

```

Système d'amis

1 - Conception / Explication

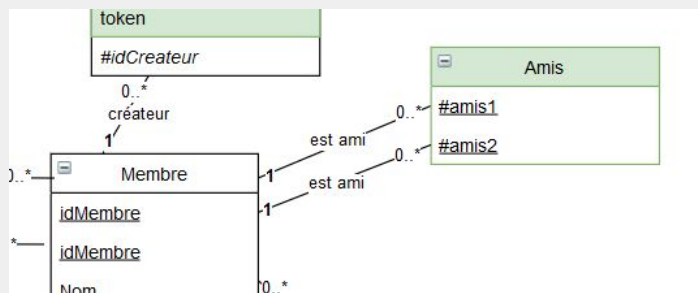
Après discussions avec monsieur Gueneguo et madame Debled-Renesson il a été convenu que l'ajout d'un système d'amis dans notre projet est pertinent.

Un des objectifs est de donner la possibilité de partager une liste privé de monument à un amis et que celui-ci soit automatiquement notifié lors de l'ajout d'un nouveau monument.

D'un point de vue conception on a la table :

Amis(#Amis1, #Amis2)

On convient arbitrairement que lors d'une nouvelle relation d'amitié amis1 sera celui qui aura le id le plus petit.



2 - SQL

```
CREATE TABLE `amis` (  
  `amis1` INT UNSIGNED NOT NULL COMMENT 'on convient arbitrairement que  
amis1 est celui qui a l id le plus petit',  
  `amis2` INT UNSIGNED NOT NULL ,  
  PRIMARY KEY (`amis1`, `amis2`)  
) ENGINE = InnoDB;
```

Et pour les clés étrangères :

```
ALTER TABLE amis  
  ADD CONSTRAINT fk_amis_membre1  
  FOREIGN KEY (amis1)  
  REFERENCES membre(idMembre);
```

```
ALTER TABLE amis
ADD CONSTRAINT fk_amis_membre2
FOREIGN KEY (amis2)
REFERENCES membre(idMembre);
```

3 - Trigger

Dans le cas d'un ajout dans le mauvais ordre des amis le trigger rétablit l'ordre croissant. En assurant que l'ordre est respecté cela garantit aussi qu'il n'y a pas deux lignes qui indiqueraient la même relation d'amis mais dans deux sens.

```
DELIMITER |
CREATE OR REPLACE TRIGGER `trigger_amis_ajoutOrdre`
BEFORE INSERT ON `amis`
FOR EACH ROW
BEGIN
    DECLARE tmp INTEGER;
    IF NEW.amis1 > NEW.amis2
    THEN
        SET tmp := NEW.amis1;
        SET NEW.amis1 = NEW.amis2;
        SET NEW.amis2 = tmp;
    END IF;
END |
DELIMITER ;
```

Synthèse :

Diagramme de base de données

