La Manipulation du Marché de l'Art

Rapport DSB
Groupe 2

Florian EPAIN
florian.epain@etudiant.univ-rennes1.fr
Thomas De Premorel
? ? ?@etudiant.univ-rennes1.fr

March 21, 2022

Résumé

C'est surement indigeste mais j'y ai mis mon coeur.

Table des matières

1 Introduction

Nous avons choisi de représenter la 'discrimination' des artistes dans le marché de l'art et le principe d'évaluation d'une oeuvre d'art.

Depuis l'aube des temps, les artistes ont eu du mal à se rémunérer et ce phénomène est d'autant plus fort dans ce monde d'hyper-concurence et néocapitaliste.

D'après les dits de cette video :

FIGURE 1 – The Art Market is a Scam (And Rich People Run It)

L'art n'a pas de valeur intrisèque. L'art se voit etre évalué par la somme *hypothétique* que les acheteurs *potentiels* sont prêt à dépenser pour l'acquérir. Il en va donc d'une enchère monétaire, de manipulation de critique, mais aussi de recherche et de travaux d'expert sur certaines oeuvres pour que celle-ci se voit identifiée, voire authentifiée, ce qui a pour cause de monter son prix.

Ce marché est très restreint, accessible uniquement par des strates très riches de nos sociétés. En effet, le prix d'entrée des marchés où se déroulent les ventes les plus 'lucratives' et onéreuses dépasse souvent le salaire moyen d'une vie d'un salarié (source dans un prochain épisode).

La grande majorité des ventes mondiales d'art sont concentrées à New york, Hong-kong et London.

Le problème se pose sur le caractère spéculatif (stéril et fictif), subjectif et concentré de ce système. Ce sont dans ce genre de petit monde où le monopole contrôle tout (les prix, quand pour les ventes, combien d'oeuvre, etc) -> l'exemple des oeuvres d'Andy Warhol et du presque unique acheteur Jose Mugrabi.

Ces mêmes conditions qu'a permit à Mugrabi de dominer le marché des oeuvres de Warhol, à son influence de fonctionner reflètent que la connivence et la corruption s'y confonde facilement.

Nous parlons même pour le marché de l'art d'une composition d'arnaques, par exemple :
Contre-intuitivement, d'ultra-riche individus peuvent gagner en rentabilité en procédant à des donations d'art.

Pour le cas des US -se trouve également en France-, lorsqu'un de ces individus opère une donation d'art à un musée -à but non-lucratif- il obtient une déduction de taxe.

Pour un don d'une oeuvre valant 10 millions de $ la déduction d'impôt s'applique pour 10 millions, qui en théorie devrait leur profiter de 4 millions. Etant donné la difficulté de determiner la valeur d'oeuvre, l'IRS (Internal Revenue Service : organisme de taxation des Etats Unis vis-à-vis de l'art) se voit surpayer 38% des déductions de taxes comparement à la valeur établie plus tard. (source manquante dans la vidéo)

En résumé, un ultra-riche pourrait acheter un pièce d'art pour 4millions. L'apprécier pendant quelque année, négocier pour une évaluation favorable, augmenter sa valeur, se baser sur le fait que l'IRS n'audite (ne contrôle) qu'une infime partie des oeuvres, faire la donation pour 10millions et il aura déjà atteint le seuil de rentabilité.

Ce don aura également pour cause, la présence d'une piece d'une même collection dans le musée, d'augmenter le prix des autres de la collection que ce créancier possède potiellement (puissance d'un monopole).

L'art n'est plus un exercice de compétence, c'est celui d'une marque.

2 Modèle conceptuel

Nous avons omis les conservateurs de musée, propriétaire de gallerie et d'autres acteurs dans l'équation de l'évaluation du prix d'une oeuvre. Les relations de connaisance entre les différents humains, leur contact, leur société écran, *etc.* Ce procédé reste complexe et non exhaustivement représenté par cette base de donnée. Il s'agit surtout d'une ébauche.

2.1 Description du modèle

L'art est au milieu de tout ce système. Ce marché se compose de plusieurs acteurs, chacun influence la valeur de l'oeuvre. Comme représenté sur ce shéma mocodo : 2

Une pièce d'oeuvre d'art possède :

- un numéro d'objet (idart unique) ;
- un titre (titre) ;
- un medium (typeArt) ;
- une cote (la valeur estimée de l'oeuvre).

Nous avons décidé de séparer le créateur de la pièce, étant donné que son créateur peut etre inconnu ou en avoir plusieurs.

Nombreuses tables représentent des humains, pour pouvoir les différencier nous les avons séparé en plusieurs tables. Ces humains possèdent tous ces *attributs fixes* :

- un identifiant unique* ;
- un nom ;
- une nationalité.

Certains d'entre eux peuvent avoir comme attribut :

- un capital ;
- une spécialité ;
- un site web ;
- une réputation.

Les attributs des artistes sont tous facultatifs sauf leur identifiant qui lui est unique. En effet, on va créer un profil artiste lorsqu'on retrouve une collection d'oeuvre à l'artiste inconnu, on les regroupe par un artiste 'fantôme'. Ils possèdent potentiellement un site-web -un contact- et une réputation. Cette réputation est positive, un artiste avec une réputation :

- $= 0$ est inconnu ;
- > 1000 est connu localement ;
- > 2000 est connu à l'internationale ;
- > 10000 est une figure historique.

On associe les artistes avec les artworks par la relation CREE. Un Artwork peut etre créé par personne ou plusieurs artistes. Et un artiste peut ne rien avoir créé ou avoir une multitude de création.

Les Mecenes sont peu majoritaire dans le marche de l'art, il s'agit surtout de créancier qui achetent directement à l'artiste. Ils possèdent les attributs humains fixes, une réputation (même ordre que celle des les artistes) et leur patrimoine supposé (capitalMecene). Ces Mecènes peuvent ou pas AIDE un ou des artistes, représenté par une somme d'argent : prixAide

Les Créanciers sont principalement des commerciaux plutôt que des collectionneurs d'art. Ils ont les attributs fixes et un capitalCreancier. Ils peuvent vendre de l'art, à un certain prix (prixVente), à un certain moment (dateFinPossede). Ils peuvent participer à un seul marché à la fois, dans lequel ils peuvent acheter/vendre des oeuvres. Leur passage est répertorié dans la relation PARTICIPE. Pour les achats de crénacier à créancier privé (sans marché public) on passe par possede.

Les Commissaires-priseurs ont seulement les attributs fixes. Ils peuvent diriger un marché.

Pour qu'une oeuvre monte en valeur, et s'affiche mieux dans un immense salon, les Restaurateurs travaillent avec les Musées, les galeries et les collectioneurs d'art (créanciers). Ils ont une spécialité (typeRestaurateur) sur laquelle ils travaillent un medium. Ils sont en contact direct avec les oeuvres (relation RESTAURE : pour un certain prix (prixRestaure)).

Les Critiques sont des personnes publiques émettant un jugement sur certaine oeuvre, mouvement d'art. Ils jouent un rôle important dans la monter de prix des oeuvres (relation JUGE : en donnant une cote -prixJUGE-).

Enfin les Experts sont des chercheurs, en Histoire, en Matière, en Technique. Leur collaboration avec le corps des galeries et créanciers est essentielle pour l'authentification d'oeuvre d'art. Prouver qu'une pièce est l'original peut faire grimper sa cote de plusieurs millions. [Vin]

Le marché devra être dirigé par un seul commissaire-priseur et peut avoir aucun ou plusieurs créanciers participant. Il s'agit de galerie privée, où le prix d'entrée (prixMarche) vise à resteindre les acheteurs (n'avoir que la même 'clientèle'). Ils sont unique et ne durent que le jour dateMarche*. Ces marchés peuvent être heberger par des organismes privés ou avoir des locaux fixes comme Christies's ou Sotheby's. Il se déroule à un certain moment (dateMarche), rendez vous à ne pas manquer pour repérer les fluctuations de prix et certaines oeuvres qui peuvent se rentabiliser.

Les Galeries et les Musées sont des organismes présentant des oeuvres (EXPOSE et PRET resp), elles ne peuvent pas ouvrir si ces organismes n'ont pas assez d'oeuvres à exposer. (cardinalité N N). Une oeuvre peut etre exposée / pretée plusieurs fois, pour garder l'historique. Les attributs datedebut / datefin (datedebutExpose/datefinExpose et datedebutPret/dureefinPret resp) correspondent à deux dates entre laquelle la galerie ou le musée dispose de l'oeuvre.

D'après l'ICOM, un musée est une institution permanente sans but lucratif au service de la société et de son développement ouverte au public, qui acquiert, conserve, étudie, expose et transmet le patrimoine matériel et immatériel de l'humanité et de son environnement à des fins d'études, d'éducation et de délectation. Le musée doit avoir une date de Creation : dateMusee* et une adresseMusee*.

Une galerie à une date de creation ou date d'evenement pour les galeries temporaires (dateGalerie). Un prix d'entré : prixGalerie, pouvant dépassant les centaines de milliers de dollars voire le million. Une galerie peut reverser les gains à une association. (smiley qui sourit) Une galerie peut être à l'extérieur, à l'intérieur dans des locaux fixes ou mobiles : adresseGalerie.

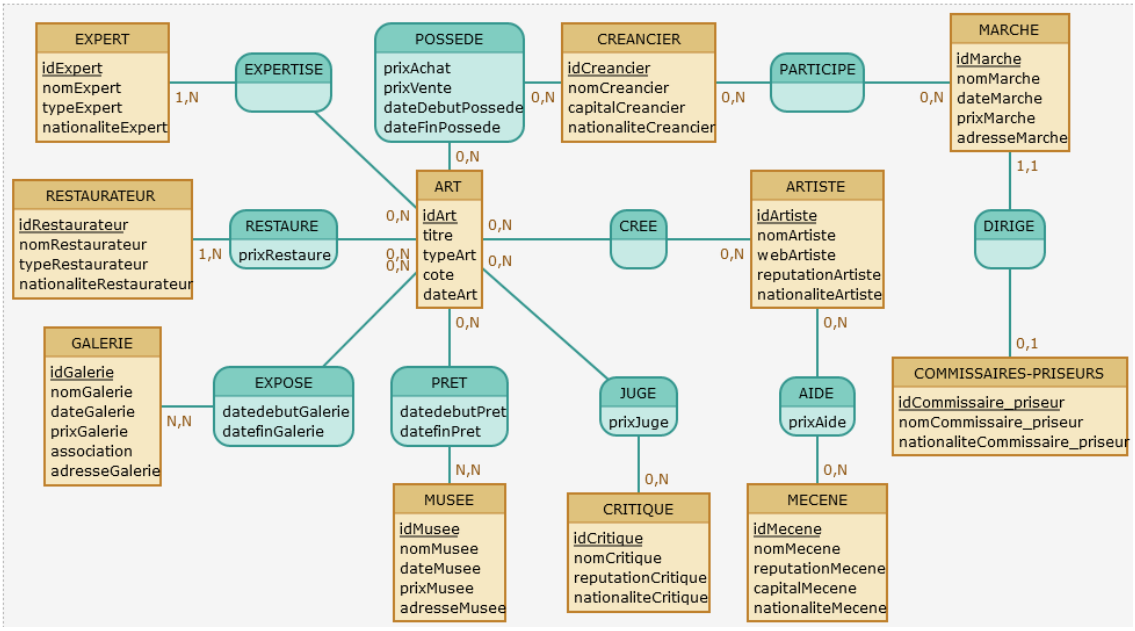

FIGURE 2 – Schéma Mocodo Revisité.

3 Modèle relationnel

3.1 Application règle 1

.
ART (idArt, titre, typeArt, cote, dateArt)
ARTISTE (idArtiste, nomArtiste, webArtiste, reputationArtiste, nationaliteArtiste)
CREANCIER (idCreancier, nomCreancier*, capitalCreancier, nationaliteCreancier*)
COMMISSAIRE-PRISEUR (idcommissaire_priseur, nomcommissaire_priseur*, nationalitecommissaire_priseur*)
RESTAURATEUR (idRestaurateur, nomRestaurateur*, typeRestaurateur*, nationaliteRestaurateur*)
MUSEE (idMusee, nomMusee*, dateMusee*, prixmusee, adresseMusee*)
GALERIE (idGalerie, nomGalerie*, dateGalerie*, prixGalerie, association, adresseGalerie*)
CRITIQUE (idCritique, nomCritique*, reputationCritique, nationaliteCritique*)
MARCHE (idMarche, nomMarche*, dateMarche*, prixMarche, adresseMarche*)
EXPERT (idExpert, nomExpert*, typeExpert*, nationaliteExpert*)
MECENE (idMecene, nomMecene*, reputationMecene, capitalMecene, nationaliteMecene*)

3.2 Application règle 2 et 2bis

Les 0 1 et 1 1 se transforment en clé étrangères et supprime la relation qui est alors inutile. Ici le marché étant unique, il n'est dirigé que par un seul commissaire-priseur, la relation dirige est alors incorporée dans la table marché.

MARCHE (idMarche, nomMarche*, dateMarche*, prixMarche, adresseMarche*, idcommissaire)

3.3 Application règle 3

Ces relations deviennent des tables contenant des clés étrangères pour permettre à toutes informations de pouvoir être accèder sans ambiguité. Et les relations inutiles sont supprimées
PARTCIPE(idMarche*, idCreancier*)
POSSEDE(idCreancier*, idArt*, prixAchat, prixVente, dateDebutPossede*, dateFinPossede)

Les deux relation POSSEDE et VEND (creancierXart) sont fusionnée afin d'éviter les conflits Le prix peut etre privé (sans trop d'interet dans le milieu de la spéculation)

RESTAURE(idRestaurateur*, idArt*, prixRestaure)
PRET(idMusee*, idArt*, datedebutPret*, datefinPret*)
EXPOSE(idGalerie*, idArt*, datedebutExpose*, datefinExpose*)
JUGE(idCritique*, idArt*, prixJuge*)
EXPERTISE(idExpert*, idArt*)
CREE(idArtiste*, idArt*)
AIDE(idMecene*, idArtiste*, prixAide)

4 Schéma physique

```
CREATE DATABASE IF NOT EXISTS `MERISE`
DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci;
USE `MERISE`;

DROP TABLE IF EXISTS POSSEDE;
DROP TABLE IF EXISTS CREE;
DROP TABLE IF EXISTS PARTICIPE;
DROP TABLE IF EXISTS EXPOSE;
DROP TABLE IF EXISTS RESTAURE;
DROP TABLE IF EXISTS PRET;
DROP TABLE IF EXISTS JUGE;
DROP TABLE IF EXISTS EXPERTISE;
DROP TABLE IF EXISTS AIDE;
DROP TABLE IF EXISTS CRITIQUE;
DROP TABLE IF EXISTS EXPERT;
DROP TABLE IF EXISTS ARTISTE;
DROP TABLE IF EXISTS MECENE;
DROP TABLE IF EXISTS RESTAURATEUR;
DROP TABLE IF EXISTS MUSEE;
DROP TABLE IF EXISTS GALERIE;
DROP TABLE IF EXISTS ART;
DROP TABLE IF EXISTS CREANCIER;
DROP TABLE IF EXISTS MARCHE;
DROP TABLE IF EXISTS COMMISSAIRE_PRISEUR;

CREATE TABLE ART (
  idart INT NOT NULL,
  titre VARCHAR(1000),
  typeArt VARCHAR(1000),
  cote INT,
  dateArt DATE,
  PRIMARY KEY (idart)
);

CREATE TABLE ARTISTE (
  idartiste INT NOT NULL,
  nomartiste VARCHAR(1000),
  webartiste VARCHAR(1000),
  reputationartiste INT,
  nationaliteartiste VARCHAR(1000),
  PRIMARY KEY (idartiste)
);

CREATE TABLE COMMISSAIRE_PRISEUR (
  idcommissaire_priseur INT NOT NULL,
  nomcommissaire_priseur VARCHAR(1000) NOT NULL,
  nationalitecommissaire_priseur VARCHAR(1000) NOT NULL,
  PRIMARY KEY (idcommissaire_priseur)
);

CREATE TABLE MARCHE (
  idmarche INT NOT NULL,
  nommarche VARCHAR(1000) NOT NULL,
  datemarche DATE NOT NULL,
  prixmarche INT,
  adressemarche VARCHAR(1000) NOT NULL,
  idcommissaire_priseur INT NOT NULL,
  PRIMARY KEY (idmarche),
  FOREIGN KEY (idcommissaire_priseur)
  REFERENCES COMMISSAIRE_PRISEUR(idcommissaire_priseur)
);

CREATE TABLE RESTAURATEUR (
  idrestaurateur INT NOT NULL,
  nomrestaurateur VARCHAR(1000) NOT NULL,
  typerestaurateur VARCHAR(100) NOT NULL,
  nationaliterestaurateur VARCHAR(1000) NOT NULL,
  PRIMARY KEY (idrestaurateur)
);

CREATE TABLE MUSEE (
  idmusee INT NOT NULL,
  nommusee VARCHAR(1000) NOT NULL,
  datemusee DATE NOT NULL,
  prixmusee INT,
  adressemusee VARCHAR(1000) NOT NULL,
  PRIMARY KEY (idmusee)
);

CREATE TABLE GALERIE (
  idgalerie INT NOT NULL,
  nomgalerie VARCHAR(1000) NOT NULL,
  dategalerie DATE NOT NULL,
  prixgalerie INT,
  association VARCHAR(1000),
  adressegalerie VARCHAR(1000) NOT NULL,
  PRIMARY KEY (idgalerie)
);

CREATE TABLE CRITIQUE (
  idcritique INT NOT NULL,
  nomcritique VARCHAR(1000) NOT NULL,
  reputationcritique INT,
  nationalitecritique VARCHAR(1000) NOT NULL,
  PRIMARY KEY (idcritique)
);

CREATE TABLE EXPERT (
  idexpert INT NOT NULL,
  nomexpert VARCHAR(1000) NOT NULL,
  typeexpert VARCHAR(1000) NOT NULL,
  nationaliteexpert VARCHAR(1000) NOT NULL,
  PRIMARY KEY (idexpert)
);

CREATE TABLE MECENE (
  idmecene INT NOT NULL,
  nommecene VARCHAR(1000) NOT NULL,
  reputationmecene INT,
  capitalmecene INT,
  nationalitemecene VARCHAR(1000) NOT NULL,
  PRIMARY KEY (idmecene)
);

CREATE TABLE CREANCIER (
  idcreancier INT NOT NULL,
  nomcreancier VARCHAR(1000) NOT NULL,
  capitalcreancier INT,
  nationalitecreancier VARCHAR(1000) NOT NULL,
  PRIMARY KEY (idcreancier)
);
```

```
CREATE TABLE POSSEDE (
  idcreancier INT NOT NULL,
  idart INT NOT NULL,
  prixAchat INT,
  prixVente INT,
  datedebutPossede DATE NOT NULL,
  datefinPossede DATE,
  PRIMARY KEY (idcreancier, idart),
  FOREIGN KEY (idart)
  REFERENCES ART(idart),
  FOREIGN KEY (idcreancier)
  REFERENCES CREANCIER(idcreancier)
);

CREATE TABLE CREE (
  idartiste INT NOT NULL,
  idart INT NOT NULL,
  PRIMARY KEY (idartiste, idart),
  FOREIGN KEY (idart)
  REFERENCES ART(idart),
  FOREIGN KEY (idartiste)
  REFERENCES ARTISTE(idartiste)
);

CREATE TABLE PARTICIPE (
  idcreancier INT NOT NULL,
  idmarche INT NOT NULL,
  PRIMARY KEY (idcreancier, idmarche),
  FOREIGN KEY (idcreancier)
  REFERENCES CREANCIER(idcreancier),
  FOREIGN KEY (idcreancier)
  REFERENCES MARCHE(idmarche)
);

CREATE TABLE EXPOSE (
  idart INT NOT NULL,
  idgalerie INT NOT NULL,
  datedebutexpose DATE NOT NULL,
  datefinexpose DATE NOT NULL,
  PRIMARY KEY (idart, idgalerie),
  FOREIGN KEY (idart)
  REFERENCES ART(idart),
  FOREIGN KEY (idgalerie)
  REFERENCES GALERIE(idgalerie)
);
```

```
CREATE TABLE RESTAURE (
  idrestaurateur INT NOT NULL,
  idart INT NOT NULL,
  prixrestaure INT,
  PRIMARY KEY (idrestaurateur, idart),
  FOREIGN KEY (idart)
  REFERENCES ART(idart),
  FOREIGN KEY (idrestaurateur)
  REFERENCES RESTAURATEUR(idrestaurateur)
);

CREATE TABLE PRET (
  idart INT NOT NULL,
  idmusee INT NOT NULL,
  datedebutpret DATE NOT NULL,
  datefinpret DATE NOT NULL,
  PRIMARY KEY (idart, idmusee),
  FOREIGN KEY (idart)
  REFERENCES ART(idart),
  FOREIGN KEY (idmusee)
  REFERENCES MUSEE(idmusee)
);

CREATE TABLE JUGE (
  idcritique INT NOT NULL,
  idart INT NOT NULL,
  prixJuge INT NOT NULL,
  PRIMARY KEY (idcritique, idart),
  FOREIGN KEY (idart)
  REFERENCES ART(idart),
  FOREIGN KEY (idcritique)
  REFERENCES CRITIQUE(idcritique)
);

CREATE TABLE EXPERTISE (
  idexpert INT NOT NULL,
  idart INT NOT NULL,
  PRIMARY KEY (idexpert, idart),
  FOREIGN KEY (idart)
  REFERENCES ART(idart),
  FOREIGN KEY (idexpert)
  REFERENCES EXPERT(idexpert)
);

CREATE TABLE AIDE (
  idmecene INT NOT NULL,
  idartiste INT NOT NULL,
  prixAide INT,
  PRIMARY KEY (idmecene, idartiste),
  FOREIGN KEY (idartiste)
  REFERENCES ARTISTE(idartiste),
  FOREIGN KEY (idmecene)
  REFERENCES MECENE(idmecene)
);
```

5 Peuplement des tables uwu

Github: Museum of Modern Art

MoMA est un musée dans New York city, il nous partage sa base de données. Nous avons utilisé seulement quelque informations contenues dans la base de données MoMA :
Pour les artistes :

- DisplayName (leur nom) ;
- Nationality (leur origine).

Pour les oeuvres d'art :

- Title ;
- Artist ;
- Medium ;
- Date de création.

Toutes les autres informations contenues dans notre base de données n'est que pure supposition et imagination (cote d'une oeuvre / reputation d un artiste / *etc*)

5.1 Artistes

On importe depuis le .json du MoMA, la table artiste contenant (id max : 133038) 15223 individus/organismes. On utilise le code convert_artists() (section 7.1). Deux exemples d'artiste :

```
{
  "constituent_id": 7626,
  "display_name": "Patrick Ireland",
  "artist_bio": "Irish, 1972-2008",
  "nationality": "Irish",
  "gender": "Male",
  "begin_date": 1972,
  "end_date": 2008,
  "wiki_qid": "Q3777954",
  "ulan": "500079431"
},

{
  "constituent_id": 7636,
  "display_name": "IBM Corporation",
  "artist_bio": "American, founded 1911",
  "nationality": "American",
  "gender": null,
  "begin_date": 1911,
  "end_date": 0,
  "wiki_qid": null,
  "ulan": null
},
```

5.2 Artworks

On importe depuis le .json du MoMA, la table artwork contenant (ifd max : 419289) 139904 pièces d'art. On utilise le code convert_artworks() (section 7.2) Toutes les dates vont etre à 0 ou null car la base de donnée MoMA insere des phrases (Option<String>) dans l'attribut date de l'artwork.

Un exemple d'artwork :


```
{
  "title": "Zamaul' III",
  "artist": [
    "Various artists",
    "Vladimir Burliuk",
    "Pavel Filonov",
    "Aleksei Kruchenykh",
    "Il'ia Rogovin"
  ],
  "constituent_id": [
    24409,
    12001,
    12124,
    3263,
    23615
  ],
  "artist_bio": [
    "Ukrainian, 1886-1917",
    "Russian, 1883-1941",
    "Russian, 1886-1969"
  ],
  "nationality": [
    "",
    "Ukrainian",
    "",
    "Russian"
  ],
  "begin_date": [
    0,
    1886,
    1883,
    1886,
    0
  ],
  "end_date": [
    0,
    1917,
    1941,
    1969,
    0         ],
  "gender":   [
    "",
    "Male",
    "Male",
    "Male",
    "Female" ],
  "date": "1919",
  "medium": "Book with five hectographed illustrations,
  letterpress and rubber-stamped typographic designs, and hectographed manuscript text and designs",
  "dimensions": "page (irreg.): 6 1/2 x 5 13/16\ (16.5 x 14.8 cm)",
  "credit_line": "Gift of The Judith Rothschild Foundation",
  "accession_number": "114.2001",
  "classification": "Illustrated Book",
  "department": "Drawings & Prints",
  "date_acquired": "2001-01-24",
  "cataloged": "Y",
  "object_id": 11384,
  "url": "http://www.moma.org/collection/works/11384",
  "thumbnail_url": "http://www.moma.org/media/W1siZiIsIjIxMjY5NCJdLFsicCIsImNvbnZlcnQiLCItcmVzaXplIDMwMHgzMDB
  "height_cm": 16.5,
  "width_cm": 14.8
},
```

5.3 Humains

À partir d'ici, nous n'avons plus de base de donnée à importer alors nous allons créer des humains.

On créé deux tableaux contenant des prénoms et noms.

```
static LIST_FIRST_NAME: &'static [&str] = &["Adrien", "Nelson", "Benoit", "Morgan",
"Florian", "Thomas", "Maeto", "Clementine", "Stephane", "Otto", "Jan", "Patrick",
"Rudolf", "Pietra", "Lex", "Nancy", "Waltercio", "Peter", "George", "Jotaro",
"Joseph", "Jonathan", "Joe", "Maria", "Laurence", "Emil", "Usul", "Wout", "Montse",
"Bent", "Nobert", "Marcel", "Arian", "Hella"];

static LIST_LAST_NAME: &'static [&str] = &["Strömholm", "Strömholm", "Rusten",
"Russomagno", "Biddle", "Burckhardt", "Martin", "Pinney", "Neusüss", "Goldes",
"Charlesworth", "Bernsten", "Appelt", "Cohen", "McCarthy", "Wagner", "Van Toorn",
"Mol", "Lynch", "Bell", "Cassell", "Schönthal", "Kammerer", "Hoppe", "De Vringer",
"Faydherbe", "Marqués", "Nobert", "Smith", "Knoll", "Rizzatto", "Tetrarc",
"Wanders", "van der Meulen", "Joestar", "Abitbol", "Zepelli", "Speedwagon"];
```

On importe depuis le .json artistes un tableau pour l'origine (de leur attribut nationality) :

```
static LIST_NATIONALITY: &'static [&str] = &["nationality unknown", "American",
"Spanish", "Danish", "Italian", "French", "Estonian", "Mexican", "Swedish",
"Israeli", "British", "Finnish", "Polish", "Palestinian", "Japanese", "Guatemalan",
"Colombian", "Romanian", "Russian", "German", "Argentine", "Kuwaiti", "Belgian",
"Dutch", "Norwegian", "Chilean", "Swiss", "Costa Rican", "Czech", "Brazilian",
"Austrian", "Canadian", "Australian", "Ukrainian", "Hungarian", "Haitian",
"Congolese", "Bolivian", "Cuban", "Yugoslav", "Portuguese", "Indian", "Peruvian",
"Icelandic", "Irish", "Guyanese", "Uruguayan", "Slovak", "Croatian", "Greek",
"Chinese", "Venezuelan", "Turkish", "Panamanian", "Algerian", "Ecuadorian",
"South African", "Iranian", "Korean", "Canadian Inuit", "Paraguayan",
"Luxembourgish", "Nicaraguan", "Zimbabwean", "Moroccan", "Slovenian", "Tanzanian",
"Bulgarian", "Tunisian", "Sudanese", "Taiwanese", "Ethiopian", "Scottish",
"Latvian", "Senegalese", "Thai", "New Zealander", "Lithuanian", "Pakistani",
"Bahamian", "Bosnian", "Malian", "Czechoslovakian", "Georgian", "Egyptian",
"Kenyan", "Emirati", "Nigerian", "Cypriot", "Albanian", "Azerbaijani", "Ivorian",
"Malaysian", "Serbian", "Singaporean", "Namibian", "Cambodian", "Ghanaian",
"Afghan", "Native American", "Lebanese", "Kyrgyzstani", "Vietnamese", "Ugandan",
"Cameroonian", "Welsh", "Macedonian", "Puerto Rican", "Catalan", "Filipino",
"Sahrawi", "Bangladeshi", "Coptic", "Persian", "Burkinabe", "Beninese",
"Sierra Leonean", "Salvadoran"];
```

On créé un tableau pour les différents types globaux de medium.

```
static LIST_ARTWORK_TYPE: &'static [&str] = &["Sculture", "Paint", "Ceramics",
"Graphic Art", "Illuminated Manuscripts", "Jewellery Art", "Metalwork Art",
"Mosaic Art", "Photography", "Architecture", "Religious Art", "Rock Art",
"Stained Glass Art"];
```

On crée une multitude de methodes pour décrire tous les attributs des humains (section 7.3). On crée une méthode pour générer et insérer des humains (section 7.4).

Un exemple d'appel pour créer amounth_of_each créanciers :

```
pub fn create_requests(amount_of_each: i32) // -> Result<()>
{
    let mut request: String = "".to_string();

    let mut number_of_creation: i32 = 0;
```

```
    let creancier = create_insert_humans("creancier", amount_of_each, false, false, true);
    number_of_creation +=amount_of_each;
    request.push_str(&creancier);

    println!("humans created: {}", number_of_creation);
    println!("--------create_.txt---------");

    fs::write("E:/Code/projects_rust/art-manipulation/RENDU/humans.txt", request)
        .expect("Unable to write file");
```

5.4 Organismes

Pour les organisations : Marché, Galerie, Musée. Nous utilisons les mêmes méthodes attributs (section 7.3).

De plus, le tableau (vide) qui peut contenir des associations (pour attr de galerie) :

```
static LIST_ASSOCIATION: &'static [&str] = &["", ""];
```

Pour créer la requête INSERT des organismes on utilise le code en section 7.5

5.5 Relations

On utilise les codes de la Section 7.6 pour générer des fausses relations.

Deux exemple de création de relations :

```
pub fn create_requests(amount_of_each: i32) // -> Result<()>
{
    let mut request: String = "".to_string();

    let mut number_of_creation: i32 = 0;

    //--POSSEDE-----------------------------------------------------------------

    let own = create_insert_relations("possede".to_string(), "creancier".to_string(),
                              "art".to_string(), amount_of_each, 2, true, true);
    number_of_creation +=amount_of_each;
    request.push_str(&own);

    //--RESTAURE----------------------------------------------------------------
    let restore = create_insert_relations("restaure".to_string(), "restaurateur".to_string(),
                                "art".to_string(), amount_of_each, 18, true, false);
    number_of_creation +=amount_of_each;
    request.push_str(&restore);

    println!("inserts created: {}", number_of_creation);
    println!("--------create_.sql---------");

    fs::write("E:/Code/projects_rust/art-manipulation/RENDU/insertsFictiv.sql", request)
        .expect("Unable to write file");
}
```

6 Requetes SQL

6.1 Projection

Les noms des oeuvres d'art des Artistes avec un grand A (reputation > 1200) avec les artistes en questions.
Algèbre relationnel :
$\pi_{nomArtiste,titre}(\sigma_{reputationArtiste>1200}(((\textbf{P1\_ARTISTE})$ NATURAL JOIN $(\textbf{P1\_CREE}))$ NATURAL JOIN $(\textbf{P1\_ART})))$
SQL :

```
SELECT P1_ARTISTE.nomartiste, P1_ART.titre
FROM (P1_CREE JOIN P1_ARTISTE on P1_ARTISTE.idartiste = P1_CREE.idartiste )
              JOIN P1_ART on P1_ART.idart = P1_CREE.idart
WHERE reputationArtiste > 1200
;
```

6.2 Jointure

Les mecènes et les collectionneurs d'art (créanciers) qui partagent le même nom et prénom.
Algèbre relationnel :
$(\textbf{P1\_MECENE})$ $JOIN_{nomMecene=nomCreancier}$ $(\textbf{P1\_CREANCIER})$
SQL :

```
SELECT *
FROM P1_MECENE Join P1_CREANCIER on P1_CREANCIER.nomCreancier=P1_MECENE.nomMecene
;
```

6.3 Moyenne sur l'integralite d'un attribut

La moyenne du capital supposé des créanciers.
Algèbre relationnel :
AVG $(\textbf{P1\_CREANCIER})$.capitalCreancier
SQL :

```
SELECT AVG(capitalCreancier) as moy_capitalcreancier
FROM P1_CREANCIER
;
```

6.4 Regroupement par calcul

Les prix d'entree au Marche d'art les plus bas, actuel (qui ne sont pas expirée)
Algèbre relationnel :
$\pi_{(MIN(prixMarche)}(\sigma_{dateMarche>dateActuelle}(\textbf{P1\_MARCHE})))$
SQL :

```
SELECT MIN(prixMarche)
FROM P1_MARCHE
WHERE dateMarche > CURDATE()
;
```

Les Marches d'art dont leur billet d'entree sont en dessous de la moyenne
Algèbre relationnel :
RPrixMoyen : AVG(prixMarche)(**P1_MARCHE**)
$\pi_{nomMarche}(\sigma_{dateMarche>dateActuelle}(\textbf{P1\_MARCHE})$ AND $\sigma_{prixMarche<RPrixMoyen})$
SQL :

```sql
SELECT nomMarche
FROM P1_MARCHE
WHERE dateMarche > CURDATE() AND
      prixMarche < (SELECT AVG(prixMarche)
                           FROM P1_MARCHE
                           -- WHERE dateMarche > CURDATE()
                    )
;
```

6.5 Différence

Les artistes qui n'ont rien créé
Algèbre relationnel :
$\pi_{nomArtiste}$((**P1_ARTISTE**) - (**P1_ARTISTE**) NATURAL JOIN (**P1_CREE**)))
SQL :

```sql
SELECT nomArtiste -- DISTINCT non nécéssaire
FROM P1_ARTISTE
WHERE idartiste NOT IN (SELECT DISTINCT idartiste
                        FROM P1_CREE)
;
```

6.6 Division

Nom et ID des Creanciers ayant possedé toutes les oeuvres d'art.
Algèbre relationnel :
$\pi_{idCreancier,nomCreancier}$((**P1_CREANCIER**) DIVIDE (**P1_POSSEDE**))
SQL :

```sql
SELECT idcreancier, nomCreancier
FROM P1_CREANCIER
WHERE (SELECT DISTINCT COUNT(*)
       FROM P1_POSSEDE
       WHERE P1_POSSEDE.idcreancier = P1_CREANCIER.idcreancier)
          -- AND P1_POSSEDE.idart = P1_ART.idart -- P1_POSSEDE.#idArt
      = (SELECT DISTINCT COUNT(*) from P1_ART)
;
```

Nom et ID des Musées ayant disposé de toutes les oeuvres d'art.
Algèbre relationnel :
$\pi_{idMusee,nomMusee}$((**P1_MUSEE**) DIVIDE (**P1_PRET**))
SQL :

```sql
SELECT idmusee, nommusee
FROM P1_MUSEE
WHERE (SELECT DISTINCT COUNT(*)
       FROM P1_PRET
       WHERE P1_PRET.idmusee = P1_MUSEE.idmusee)
      = (SELECT DISTINCT COUNT(*) from P1_ART)
;
```

6.7 Group By

Nombre d'artiste par nationalité. On pourrait regrouper : '' et 'nationality unknown'.

```
SELECT nationaliteArtiste, COUNT(*)
FROM P1_ARTISTE
GROUP BY nationaliteArtiste
ORDER BY COUNT(*) DESC
;
```

7 Code

7.1 convert_artists()

On instaure une Structure json d'un artiste. Dans la fn convert_artists() : On stocke un tableau de cette structure (artists : Vec<Artist>) grâce à serde_json import rust. On créé le début de la requête SQL INSERT, avec les attributs artiste.

Pour chaque artiste dans le Vec<Artist> :

- on créé une ligne de valeur ;
- on y place des apostrophes : '*attribut à remplacer*' si la valeur est une date ou un varchar ;
- on retire les apostrophes, les espaces, les virgules du nom pour le siteWeb
- on place à la virgule à la fin (on peut la placer lors du premier point mais la derniere doit etre retirer, on choisit donc cette ordre par logique)

On insère un point-virgule tout en retirant la denrière virgule. On écrit le résultât foo dans un fichier.

```
#[derive(Serialize, Deserialize, Debug)]
struct Artist {
        constituent_id: i128,
        display_name: String,
        artist_bio: Option<String>,
        nationality: Option<String>,
        gender: Option<String>,
        begin_date: i16,
        end_date: i16,
        wiki_qid: Option<String>,
        ulan: Option<String>
}

fn convert_artists() -> Result<()>
{
    let path = "E:/Code/projects_rust/MoMA/Artworks-reformed.json";

    let content = fs::read_to_string(path)
        .expect("Unable to read file");

    println!("-----read_.json-----");

    let artists: Vec<Artist> = serde_json::from_str(&content).unwrap();

    println!("--------create_request---------");

    let mut foo =
    "INSERT INTO P1_ARTISTE
    (idartiste, nomartiste, webartiste, reputationartiste, nationaliteartiste)
    \n VALUES ".to_string();

```

```
    for artist in artists
    {
        let artist_nationality: &str=
                match &artist.nationality {
                        Some(s) => s,
                        None => "",
                };


        let mut artist_web = artist.display_name
            .replace(" ", ".")
            .replace("'", "")
            .replace(",","");
        artist_web.push_str(".org");

        let foobar =
        "\n (id,'display_name','site', reputation,'nationality')";
        let mut artist_n = foobar.replace("id", &artist.constituent_id.to_string());
        artist_n = artist_n.replace("display_name", &artist.display_name.replace("'", " "));
        artist_n = artist_n.replace("site", &artist_web);
        artist_n = artist_n.replace("reputation",
                            &create_fictional_human::create_reputation(0,0).to_string());
        artist_n = artist_n.replace("nationality", &artist_nationality.replace("'", " "));
        foo.push_str(&artist_n);

        // have to remove the last one
        foo.push(',');
    }

    // rm the last , and add a ; to end the SQL request
    foo.push_str(";END");
    foo = foo.replace(",;END",";");

    println!("--------create_.sql---------");

    fs::write("E:/Code/projects_rust/art-manipulation/RENDU/insert_artists.sql", foo)
        .expect("Unable to write file");

    Ok(())
}
```

7.2 convert_artworks()

On instaure une Structure json d'un artwork. On ignore les dimensions. Dans la fn convert_artworks() : On stocke un tableau de cette structure (artists : Vec<Artwork>) grâce à serde_json import rust. On créé le début de la requête SQL INSERT, avec les attributs art. On ignore la date (à cause des phrases contenues dedans).

Pour chaque oeuvre d'art dans le Vec<Artwork> :
On créé la ligne de valeur. On y incorpore les attributs, en retirant les caractères génants (apostrophe, vrigule, backslash).

On insère un point-virgule tout en retirant la denrière virgule. On écrit le résultât foo dans un fichier.

```rust
#[derive(Serialize, Deserialize, Debug)]
struct Artwork {
  title: String,
  artist: Vec<String>,
  constituent_id: Vec<i32>,
  artist_bio: Vec<String>,
  nationality: Vec<String>,
  begin_date: Vec<i32>,
  end_date: Vec<i32>,
  gender: Vec<String>,
  date: Option<String>,
  medium: Option<String>,
  dimensions: Option<String>,
  credit_line: Option<String>,
  accession_number: Option<String>,
  classification: Option<String>,
  department: Option<String>,
  date_acquired: Option<String>,
  cataloged: Option<String>,
  object_id: i32,
  url: Option<String>,
  thumbnail_url: Option<String>,
//   height_cm: f32,
//   width_cm: f32
}

fn convert_artworks() -> Result<()>
{
    let path = "E:/Code/projects_rust/MoMA/Artworks-reformed.json";

        let content = fs::read_to_string(path)
                .expect("Unable to read file");

        println!("-----read_.json-----");

    let artworks: Vec<Artwork> = serde_json::from_str(&content).unwrap();

        // println!("{:?}", artists);

        println!("--------create_request---------");

    let mut foo =
    "INSERT INTO P1_ART (idart, titre, typeArt, cote)
    \n VALUES ".to_string();

    //, dateArt)

    for artwork in artworks
    {
```

```
50          let artwork_medium: &str =
51          match &artwork.medium {
52              Some(s) => s,
53              None => "",
54          };
55
56          let artwork_date: &str =
57          match &artwork.date {
58              Some(s) => {
59                  s.to_string().push_str("-01-01");
60                  s
61              },
62              None => "",
63          };
64
65          let artwork_title = artwork.title.replace("'", " ");
66
67          let foobar =
68          "\n (id,'title', 'medium', cote)";
69          //, date)";
70          let mut artwork_n = foobar.replace("id", &artwork.object_id.to_string());
71          artwork_n = artwork_n.replace("title", &artwork_title.replace("\\", ""));
72          // how on earth you can accept '\' in a user insertion...
73          artwork_n = artwork_n.replace("medium", &artwork_medium.replace("'", " ").replace("\\", ""
74          artwork_n = artwork_n.replace("cote", &create_fictional_human::create_reputation(0,0).to_s
75
76          // artwork_n = artwork_n.replace("date", &artwork_date.to_string());
77
78          foo.push_str(&artwork_n);
79
80          // have to remove the last one
81          foo.push(',');
82      }
83
84      // rm the last , and add a ; to end the SQL request
85      foo.push_str(";END");
86      foo = foo.replace(",;END",";");
87
88      println!("--------create_.sql---------");
89
90          fs::write("E:/Code/projects_rust/art-manipulation/RENDU/insert_artworks.sql",
91                    foo)
92                  .expect("Unable to write file");
93
94      Ok(())
95  }
```

7.3 create all kind of attributes

```
pub fn create_reputation(_number_of_creation: i32, _base_reput: i32)->i32 {
    let mut rng = thread_rng();
    return rng.gen_range(1..2000);
}

fn create_name() -> String {
    let mut rng = thread_rng();

    let mut first_name: String =
        match &LIST_FIRST_NAME.choose(&mut rng) {
        Some(n) => n.to_string(),
            None => "Foo".to_string()
        };
    let last_name: String =
        match &LIST_LAST_NAME.choose(&mut rng) {
            Some(n) => n.to_string(),
            None => "Bar".to_string()
        };

    first_name.push_str(" ");
    first_name.push_str(&last_name);

    first_name
}

fn create_nationality() -> String {
    let mut rng = thread_rng();

    let nationality =
        match &LIST_NATIONALITY.choose(&mut rng) {
            Some(n) => n,
            None => "nationality unknown"
        };

    nationality.to_string()
}

fn create_type() -> String {
    let mut rng = thread_rng();

    let art_type: String =
        match &LIST_ARTWORK_TYPE.choose(&mut rng) {
            Some(n) => n.to_string(),
            None => "Undefined".to_string()
        };

    art_type
}

fn create_capital() -> i128 {
    let mut rng = thread_rng();

    let capital: i128 = rng.gen_range(1..5000000)+1000000;

    capital
}

/**
```

```rust
 * turn every number less than 10 to two digit number
 * skip other number
 * cause 2016-5-12 or 2032-01-2
 *      isn't accepted in SQL format (TODO: change date format sql ?)
 */
fn convert_to_twodigit(number: i32) -> String {
    let n = number.to_string();
    let mut temp = "".to_string();
    if n.len() == 1 {
        temp = "0".to_string();
    }
    temp.push_str(&n);

    temp
}

/**
 * @param past : 'a future date' -> false | 'a past date' -> true
 * format : AAAA-MM-JJ
 */
fn create_date(past: bool) -> String{
    let mut rng = thread_rng();

    let mut date = "year-month-day".to_string();
    // dc about 31/02 or 31/04
    let day = convert_to_twodigit(rng.gen_range(0..31));
    // let day_reformed = (day).to_string().padStart(2,0);

    let month = convert_to_twodigit(rng.gen_range(0..12));
    let year; //= 1445;
    if past {
        year = rng.gen_range(2003..2021);
    }
    else {
        year = rng.gen_range(2022..2030);
    }

    date = date.replace("day", &day);
    date = date.replace("month", &month);
    date = date.replace("year", &year.to_string());
    date.to_string()
}

fn create_price() -> i32{
    let mut rng = thread_rng();
    // placeholder
    let price: i32 = rng.gen_range(0..50000);
    price
}

fn create_association() -> String{
    let mut rng = thread_rng();

    let association: String =
        match &LIST_ASSOCIATION.choose(&mut rng) {
            Some(n) => n.to_string(),
            None => "Undefined".to_string()
        };
    association
}
```

7.4 create_insert_humans()

```rust
/**
 * create a sql request  to insert a good amount of data in your database
 * respect my current sql structure
 * should be great to import a example of struct with all table to auto all of this
 */
fn create_insert_humans(table_name: &str, amount: i32, art_type: bool,
                  reputation: bool, capital: bool) -> String
{
    let mut request: String =
    "INSERT INTO P1_NAME (idname, nomname, "
        .to_string();
    if art_type {
        request.push_str("typename, ");
    }
    if reputation {
        request.push_str("reputationname, ");
    }
    if capital {
        request.push_str("capitalname, ");
    }

    request.push_str("nationalitename) \n VALUES");
    request = request
        .replace("NAME", &table_name.to_uppercase())
        .replace("name", &table_name.to_lowercase());

    for i in 0..amount {

        let name = create_name();

        let nationality = create_nationality();

        let foobar =
        "\n (id, 'display_name', ".to_string();

        let mut human_n = foobar.replace("id", &i.to_string());
        human_n = human_n.replace("display_name", &name);

        if art_type{
            human_n.push_str("'type', ");
            human_n = human_n.replace("type", &create_type());
        }
        if reputation{
            human_n.push_str("reputation, ");
            human_n = human_n.replace("reputation", &create_reputation(0,0).to_string());
        }
        if capital{
            human_n.push_str("capital, ");
            human_n = human_n.replace("capital", &create_capital().to_string());
        }

        human_n.push_str("'nationality'),");
        human_n = human_n.replace("nationality", &nationality);

        request.push_str(&human_n);

    }
    request.push_str(";END");
    request = request.replace(",;END","; \n \n");

    request
}
```

7.5 create_insert_organisations()

```rust
/**
 * if creation_date and rdv_date are true only creation_date will effect
 * you can't bassicaly have those two attributs
 */
fn create_insert_organisations(table_name: String, amount: i32, creation_date: bool,
                        rdv_date: bool, price: bool, association: bool) -> String
{
    let mut request: String =
    "INSERT INTO P1_NAME (idname, nomname, "
        .to_string();

    // careful about this condition which can conflict
    // with future change on mocodoStructre
    if creation_date | rdv_date{
        request.push_str("datename, ");
    }
    if price {
        request.push_str("prixname, ");
    }
    if association {
        request.push_str("association, ");
    }
    request.push_str("adressename");

    let mut rng = thread_rng();

    if table_name.to_lowercase() == "marche"{
        request.push_str(", idcommissaire_priseur");
    }

    request.push_str(") \n VALUES");
    request = request
        .replace("NAME", &table_name.to_uppercase())
        .replace("name", &table_name.to_lowercase());

    for i in 0..amount {

        let name = create_name();

        let country = create_nationality();

        let foobar =
        "\n (id, 'display_name', ".to_string();

        let mut orga_n = foobar.replace("id", &i.to_string());
        orga_n = orga_n.replace("display_name", &name);
        if creation_date | rdv_date {
            //if rdv_date means !creation_date

            orga_n.push_str("'date', ");   // same care here
            orga_n = orga_n.replace("date", &create_date(creation_date));
        }
        if price{
            orga_n.push_str("price, ");
            orga_n = orga_n.replace("price", &create_price().to_string());
        }
        if association{
            orga_n.push_str("'association', ");
```

```
            orga_n = orga_n.replace("association", &create_association());
        }


        orga_n.push_str("'country'");

        if table_name.to_lowercase() == "marche"{
            orga_n.push_str(", idcommissaire_priseur");
            // instead of a random just pick into commissaire table
            orga_n = orga_n.replace("idcommissaire_priseur", &rng.gen_range(0..amount).to_string()
        }

        orga_n.push_str("),");
        orga_n = orga_n.replace("country", &country);

        request.push_str(&orga_n);
    }

    request.push_str(";END");
    request = request.replace(",;END","; \n \n");

    request
}
```

7.6 create_insert_relations()

```
/**
 * frequence: if 0 -> Err
 *            if 1 -> no condition whatsoever (no filter),
 *            if 8 -> condition 1/8 (only 1/8 will be associate)
 * don't handle insert attribute for PARTICIPE
 */
fn create_insert_relations(relation_name:String, table_name1: String,
                    table_name2: String, amount: i32, frequence: i32,
                    price: bool, duree: bool) -> String
{
    println!("{}", relation_name);
    let mut request: String =
    "INSERT INTO P1_NAME (idname1, idname2".to_string();

    request = request
        .replace("NAME",  &relation_name.to_uppercase())
        .replace("name1", &table_name1.to_lowercase())
        .replace("name2", &table_name2.to_lowercase());
    if price {

        if relation_name=="possede" {
            request.push_str(", prixAchat");
            request.push_str(", prixVente");
        }else {
            request.push_str(", prixNAME");
        }
    }
    if duree {
        request.push_str(", datedebutNAME, datefinNAME");
    }
    if relation_name.to_lowercase()=="marche"{
        request.push_str(", idcommissaire_priseur");
    }
```

```
34      request.push_str(") \n VALUES");
35      request = request.replace("NAME", &relation_name.to_lowercase());
36
37      // TODO: import thoses two elsewhere
38      let mut artists:Vec<Artist>=Vec::new();
39      let mut artworks:Vec<Artwork>=Vec::new();
40      if table_name2.to_lowercase() == "artiste" {
41          println!("--read artists-reformed.json--");
42          artists = import_artists();
43      }else if table_name2.to_lowercase() == "art" {
44          println!("--read artworks-reformed.json--");
45          artworks = import_artworks();
46      }
47
48      for i in 0..amount {
49
50
51
52          let mut foo =
53          "\n (idname1, idname2".to_string();
54
55
56          if price {
57              if relation_name=="possede" {
58                  foo.push_str(", prixAchat, prixVente");
59                  foo = foo.replace("prixAchat", &create_price().to_string());
60
61                  // I use the frequence to also choose whenever a art is still owned or selled
62                  // atm : 1/1
63                  if i%(frequence)==0 {
64                      foo = foo.replace("prixVente", &create_price().to_string());
65                  }else {
66                      foo = foo.replace("prixVente", " ");
67                  }
68              }
69              else {
70                  foo.push_str(", prix");
71                  foo = foo.replace("prix", &create_price().to_string());
72              }
73
74          }
75          if duree {
76              foo.push_str(", 'dateD', 'dateF'");
77              foo = foo.replace("dateD", &create_date(true).to_string());
78              if relation_name=="possede" {
79
80                  if (i%frequence)==0 {
81                  // 1/1 des oeuvres ont déja été vendu car tjrs vraie
82                  // (volontaire ou presque)
83
84                      // make it after dateD but not in the FUTURE
85                      foo = foo.replace("dateF", &create_date(true).to_string());
86                  }else {
87                      foo = foo.replace("dateF", " ");
88                  }
89              }else {
90                  foo = foo.replace("dateF", &create_date(false).to_string());
91              }
92          }
93
94
```

```
95
96          // We're selecting directly from .json
97
98          let mut rng = thread_rng();
99          let mut foobar: String;
100
101         // Select randomly a constituent_id which EXIST
102         if table_name2.to_lowercase() == "artiste" {
103             foobar = foo.replace("idname1", &i.to_string());
104             foobar = foobar.replace("idname2", &artists[rng.gen_range(0.. artists.len())]
105                                                 .constituent_id
106                                                 .to_string());
107
108         }else if table_name2.to_lowercase() == "art" {
109             foobar = foo.replace("idname1", &i.to_string());
110
111             // if you can't import 400K artwork into sql (weird.............)
112             // switch artworks.len() to the max number-1 of artwork you imported
113             // to avoid error on import of relation :
114             //      foreign key (on id which didn't exists)
115             foobar = foobar.replace("idname2", &artworks[rng.gen_range(0.. artworks.len())]
116                                                 .object_id
117                                                 .to_string());
118
119         }else {
120             foobar = foo.replace("idname1", &i.to_string());
121             foobar = foobar.replace("idname2", &i.to_string());
122         }
123
124
125
126         foobar.push_str("),");
127
128
129         // After creating the whole line, we decide if we keep it, based on the frequency
130         // only 1/frequence of all name1 and name2 will be associate
131         if (i%frequence)==0 {
132             request.push_str(&foobar);
133         }
134     }
135     // end the request with a ';' and
136     // remove the last ',' which cause error
137     request.push_str(";END");
138     request = request.replace(",;END","; \n \n");
139
140     request
141 }
```

Références

[Vin] Da Vinci. En prenant l'exemple du portrait du christ par Léonard de Vinci.
Les experts ont determiné en analysant les différentes couches de la peinture, jusqu'à l'esquice du portrait où la position du pouce diffère de la version finale, ce qui est stupide de représenter dans un brouillon lors de la création d'une copie.
Ils ont donc determiné qu'il s'agissait de l'original.
La peinture a explosé de 200$ à 450millions$
L'oeuvre est passée dans les mains d'expert, de restaurateurs, de critique et de nombreux créanciers.