**Université de Nantes, UFR sciences et techniques**

Projet de base de données interfacée web

**Système de Gestion d'Agence Photographique : PicturMe**





Safae LABEL, Sonhibou DIOUF

11/12/2024

Table des matières

[1. Présentation du Contexte 2](#_Toc184843507)

[2. Modélisation selon le Modèle Entité-Association 2](#_Toc184843508)

[3. Schéma Relationnel de la Base de Données 4](#_Toc184843509)

[4. Requêtes de création des différentes tables de la base de données (après optimisation) 5](#_Toc184843510)

[5. Requêtes SQL intéressantes pour interroger la base de données (après optimisation) 6](#_Toc184843511)

[6. Tables de données après amélioration (createTables.sql) 7](#_Toc184843512)

[Références : 9](#_Toc184843513)

# Présentation du Contexte

En tant que passionnés de photographie, on souhaite créer notre propre agence de photographie. C’est pourquoi on a choisi ce sujet pour notre projet. L'objectif de ce projet est de développer une application web interfacée à une base de données relationnelle pour une agence de photographie nommée **PicturMe**.

Cette application permettra de gérer efficacement plusieurs aspects de l'agence, notamment :

* La gestion des clients, qu'ils soient photographes ou entreprises.
* La gestion des événements et des séances photo.
* La gestion des photos, avec des fonctionnalités avancées pour le stockage, la recherche et la consultation.
* La gestion des paiements et des factures liées aux prestations de photographie.

Le contexte technologique choisi inclura une interface utilisateur intuitive, permettant aux photographes et aux clients d'interagir facilement, de réserver des sessions et de visualiser les résultats. L'agence souhaite également disposer d'une base de données bien structurée pour organiser et interroger facilement toutes les données liées aux événements et aux photos.

# Modélisation selon le Modèle Entité-Association

Dans ce modèle, plusieurs entités doivent être identifiées, chacune avec ses attributs et ses relations avec les autres entités. Voici les entités principales :

* **Client** : Les clients peuvent être des particuliers ou des entreprises qui sollicitent des services de photographie.
* **Photographe** : Ce sont les employés ou freelances qui réalisent les séances photo.
* **Événement/Séance** : Une séance photo est programmée à une date spécifique avec un photographe et un client.
* **Photo** : Les photos sont associées à une séance et stockées avec des métadonnées (résolution, format, etc.).
* **Facture/Paiement** : Les transactions financières relatives aux séances et aux ventes de photos.

Les relations entre ces entités sont comme suit :

* Un **client** peut réserver plusieurs **séances**.
* Un **photographe** peut être associé à plusieurs **séances**.
* Une **séance** produit plusieurs **photos**.
* Chaque **séance** peut générer une **facture**, et un **paiement** y est lié.

Avant optimisation :

Initialement, on avait deux tables distinctes : Facture et Paiement, avec une relation de type "un à plusieurs", c'est-à-dire qu'une facture pouvait avoir plusieurs paiements associés.

Dans ce modèle, la table Facture contient les informations liées à la facturation d'une séance (montant total, date), tandis que la table Paiement stocke chaque transaction de paiement liée à une facture.

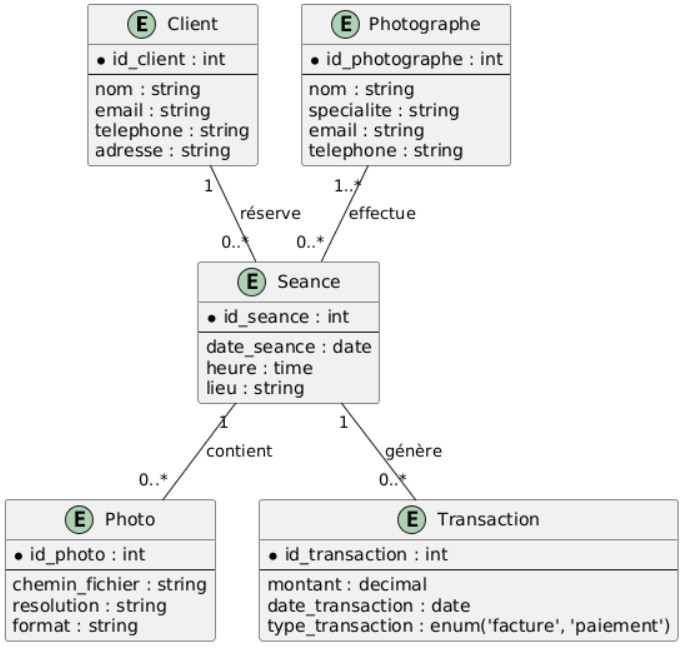
Après optimisation :

En analysant les besoins, on a constaté que dans la plupart des cas, chaque facture correspondait à un unique paiement, ou que la séparation entre les deux informations n'apportait pas une valeur ajoutée suffisante.

En conséquence, on a fusionné les deux tables en une seule table appelée Transaction, qui contient à la fois les informations de la facture et du paiement. Cela permet de simplifier les relations et les requêtes SQL.

Voici les entités principales finales :

* **Client :** Représente les clients de l'agence, qui peuvent réserver des séances photo.
* **Photographe** : Représente les photographes employés par l'agence.
* **Séance** : Représente une session photo, où un photographe prend des photos pour un client.
* **Photo** : Représente les photos prises lors d'une séance.
* **Transaction** : Regroupe à la fois les informations sur les factures et les paiements, avec un champ pour distinguer les deux types de transactions.



# Schéma Relationnel de la Base de Données

Le schéma relationnel découle du modèle Entité-Association. Voici une ébauche du schéma relationnel avec des optimisations telles que les clés étrangères :

* Client (id\_client, nom, email, téléphone, adresse)
* Photographe (id\_photographe, nom, spécialité, email, téléphone)
* Séance (id\_seance, date\_seance, heure, lieu, id\_client, id\_photographe)
* effectue (id\_photographe REF Photographe, id\_seance REF Séance)
* Photo (id\_photo, chemin\_fichier, résolution, format, id\_seance)
* Transaction (id\_transaction, montant, date\_transaction, type\_transaction, id\_seance)

# Requêtes de création des différentes tables de la base de données (après optimisation)

Voici les requêtes SQL pour créer les différentes tables de la base de données après optimisation :

CREATE TABLE Client (

id\_client INT PRIMARY KEY AUTO\_INCREMENT, nom VARCHAR(100) NOT NULL,

email VARCHAR(100), telephone VARCHAR(15),

adresse TEXT

);

CREATE TABLE Photographe (

id\_photographe INT PRIMARY KEY AUTO\_INCREMENT, nom VARCHAR(100), specialite VARCHAR(100), email VARCHAR(100),

telephone VARCHAR(15)

);

CREATE TABLE Seance (

id\_seance INT PRIMARY KEY AUTO\_INCREMENT,

date\_seance DATE, heure TIME, lieu VARCHAR (255), id\_client INT, id\_photographe INT,

FOREIGN KEY (id\_client) REFERENCES Client(id\_client),

FOREIGN KEY (id\_photographe) REFERENCES Photographe(id\_photographe)

);

CREATE TABLE Photo (

id\_photo INT PRIMARY KEY AUTO\_INCREMENT,

chemin\_fichier VARCHAR (255), resolution VARCHAR (50), format VARCHAR (10), id\_seance INT,

FOREIGN KEY (id\_seance) REFERENCES Seance(id\_seance)

);

CREATE TABLE Transaction (

id\_transaction INT PRIMARY KEY AUTO\_INCREMENT,

montant DECIMAL(10, 2), date\_transaction DATE,

type\_transaction ENUM('facture', 'paiement'), id\_seance INT,

FOREIGN KEY (id\_seance) REFERENCES Seance(id\_seance)

);

**NOTE** : « effectue » est mentionné comme une relation entre les tables « Photographe » et « Séance ». Étant donné que la relation entre un photographe et une séance est déjà représentée par la clé étrangère id\_photographe dans la table Séance, il n'est pas nécessaire de créer une table séparée pour la relation effectue.

Cependant, nous souhaitons la représenter pour des raisons de normalisation, notamment si un photographe peut être associé à plusieurs séances de manière distincte. Voici les requêtes SQL correspondantes :

CREATE TABLE Effectue ( id\_photographe INT, id\_seance INT,

PRIMARY KEY (id\_photographe, id\_seance),

FOREIGN KEY (id\_photographe) REFERENCES Photographe(id\_photographe),

FOREIGN KEY (id\_seance) REFERENCES Seance(id\_seance)

# Requêtes SQL intéressantes pour interroger la base de données (après optimisation)

Après optimisation, voici 5 requêtes SQL utiles pour interagir avec la base de données et extraire des informations pertinentes.

1. Récupérer toutes les séances réservées par un client spécifique :

SELECT \* FROM Seance WHERE id\_client = ?;

1. Trouver toutes les photos prises par un photographe dans une période donnée :

SELECT Photo.chemin\_fichier, Seance.date\_seance

FROM Photo

JOIN Seance ON Photo.id\_seance = Seance.id\_seance

WHERE Seance.id\_photographe = ? AND Seance.date\_seance BETWEEN 'YYYY-MM-DD' AND 'YYYY-MM-DD';

1. Calculer le total des paiements reçus pour une séance spécifique :

SELECT SUM(Transaction.montant) AS total\_paiements

FROM Transaction

WHERE Transaction.id\_seance = ? AND Transaction.type\_transaction = 'paiement';

1. Obtenir la liste des séances pour lesquelles aucune transaction n'a été émise :

SELECT Seance.id\_seance, Seance.date\_seance

FROM Seance

LEFT JOIN Transaction ON Seance.id\_seance = Transaction.id\_seance

AND Transaction.type\_transaction = 'facture'

WHERE Transaction.id\_transaction IS NULL;

1. Récupérer les transactions (factures et paiements) pour un client spécifique, groupées par séance :

SELECT Seance.id\_seance, Transaction.type\_transaction, Transaction.montant,

Transaction.date\_transaction

FROM Seance

JOIN Effectue ON Seance.id\_seance = Effectue.id\_seance JOIN Transaction ON Seance.id\_seance = Transaction.id\_seance WHERE Effectue.id\_photographe = ?

ORDER BY Seance.date\_seance;

# Tables de données après amélioration (createTables.sql)

CREATE TABLE Client (

id\_client INT PRIMARY KEY AUTO\_INCREMENT,

nom VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL,

telephone VARCHAR(15),

adresse TEXT

);

CREATE TABLE Photographe (

id\_photographe INT PRIMARY KEY AUTO\_INCREMENT,

nom VARCHAR(100),

specialite VARCHAR(100),

email VARCHAR(100),

telephone VARCHAR(15)

);

CREATE TABLE Seance (

id\_seance INT PRIMARY KEY AUTO\_INCREMENT,

date\_seance DATE,

heure TIME,

lieu VARCHAR(255),

id\_client INT,

id\_photographe INT,

FOREIGN KEY (id\_client) REFERENCES Client(id\_client) ON DELETE CASCADE,

FOREIGN KEY (id\_photographe) REFERENCES Photographe(id\_photographe) ON DELETE CASCADE

);

CREATE TABLE Photo (

id\_photo INT PRIMARY KEY AUTO\_INCREMENT,

chemin\_fichier VARCHAR(255),

resolution VARCHAR(50),

format VARCHAR(10),

id\_seance INT,

FOREIGN KEY (id\_seance) REFERENCES Seance(id\_seance) ON DELETE CASCADE

);

CREATE TABLE Transaction (

id\_transaction INT PRIMARY KEY AUTO\_INCREMENT,

montant DECIMAL(10, 2),

date\_transaction DATE,

type\_transaction ENUM('facture', 'paiement'),

id\_seance INT,

FOREIGN KEY (id\_seance) REFERENCES Seance(id\_seance) ON DELETE CASCADE

);

CREATE TABLE Utilisateurs (

id\_utilisateur INT PRIMARY KEY AUTO\_INCREMENT,

email VARCHAR(255) NOT NULL UNIQUE,

mot\_de\_passe VARCHAR(255) NOT NULL,

role VARCHAR(100) NOT NULL DEFAULT("client")

);

CREATE TABLE Effectue (

id\_photographe INT,

id\_seance INT,

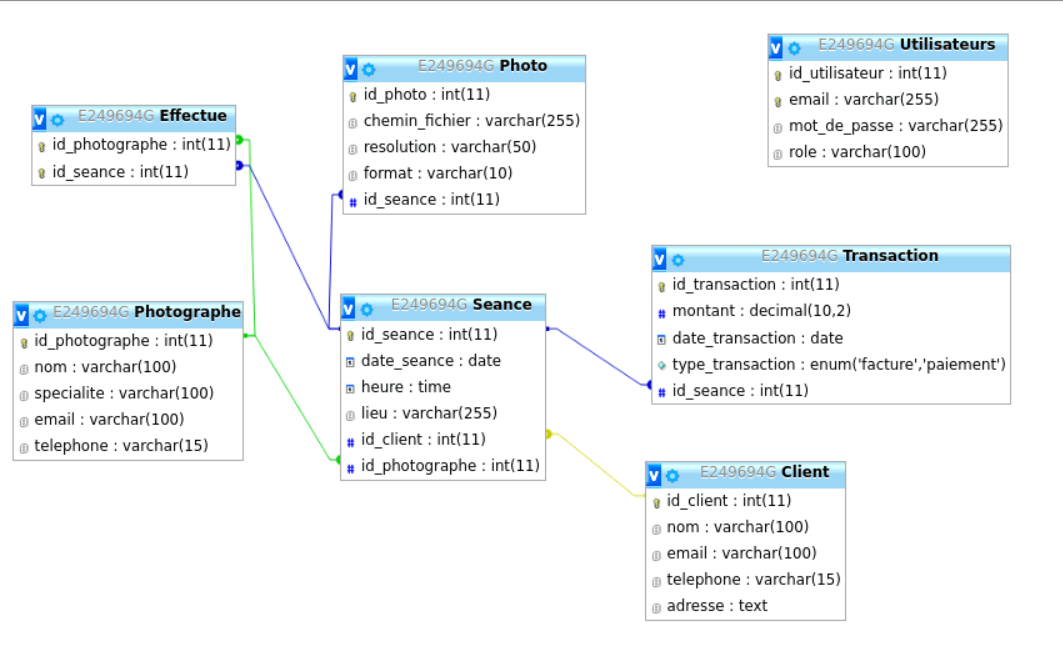
PRIMARY KEY (id\_photographe,id\_seance),

FOREIGN KEY (id\_photographe) REFERENCES Photographe(id\_photographe) ON DELETE CASCADE,

FOREIGN KEY (id\_seance) REFERENCES Seance(id\_seance) ON DELETE CASCADE

);

**Modèle Entité-Association après amélioration :**

****

**Modifications ajoutées :**

* **Client** : Ajout de la contrainte NOT NULL sur les champs email et nom.
* **Photographe** : Ajout des champs spécialité, email, téléphone pour plus de détails.
* **Séance** : Ajout des contraintes ON DELETE CASCADE sur les clés étrangères id\_client et id\_photographe pour gérer la suppression des données de manière intégrée.
* **Utilisateurs** : Ajout d'une nouvelle table Utilisateurs qui centralise la gestion des comptes et l'authentification des utilisateurs, tout en distinguant leurs rôles (client, photographe). Elle assure la sécurité des accès via des mots de passe hachés et facilite la gestion des droits dans le système.
* **Effectue** : Relation normalisée entre les photographes et les séances, avec la contrainte ON DELETE CASCADE.

# Références :

**W3Schools.** (n.d.). *SQL Tutorial*. Retrieved from W3Schools SQL Tutorial

**PlantUML**. (n.d.). *PlantUML Documentation*. Retrieved fro[m PlantUML](https://plantuml.com/)

**Audibert, L.** (2016). *Bases de données : de la modélisation au SQL*. Éditions Dunod