OZAR FORMATION	**BTS SIO** **Services Informatiques aux Organisations**	YHC Développeur
	Option SLAM	
	Session **2025/26**	

Nom et prénom : Costantini Yaacov-Haï	**Activité professionnelle N°**	3

Intitulé de l'activité	**Conception et modélisation de la base de données d'une application de chat instantané**
Contexte	Dans le cadre de mon projet de développement d'une application de messagerie instantanée, j'ai conçu et mis en œuvre une base de données relationnelle permettant de gérer les utilisateurs et leurs échanges de messages. L'application a été hébergée sur un serveur applicatif dédié, la base de données étant quant à elle déployée sur un serveur de base de données distinct sous PostgreSQL.
Objectifs	Modéliser de manière rigoureuse la base de données de l'application en suivant la démarche Merise, depuis l'analyse des besoins jusqu'à la création physique des tables, afin d'assurer la cohérence, l'intégrité et la maintenabilité des données.
Lieu de réalisation	CFA OZAR

SOLUTIONS ENVISAGEABLES

Plusieurs approches auraient pu être retenues pour la gestion des données :

- Stocker les messages dans des fichiers plats (solution simple mais non adaptée à une montée en charge et sans intégrité des données)
- Utiliser une base de données NoSQL de type document (MongoDB), adaptée aux échanges mais moins structurée
- Utiliser une base de données relationnelle (MySQL, PostgreSQL), offrant intégrité référentielle, typage strict et performances adaptées à ce type d'application

DESCRIPTION DE LA SOLUTION RETENUE

Conditions initiales	L'application de chat est fonctionnelle et hébergée. La base de données PostgreSQL est déjà créée et opérationnelle sur un serveur dédié. Il s'agit ici de reconstituer et formaliser la démarche de conception qui a guidé sa création.
Conditions finales	La base de données est modélisée selon la méthode Merise (MCD, MLD, MPD), documentée et illustrée par des diagrammes clairs et un script SQL de création des tables adapté à PostgreSQL.
Outils utilisés	• Looping (looping-mcd.fr) pour la réalisation du MCD et la génération du MLD • dbdiagram.io pour la représentation visuelle du schéma physique • PostgreSQL comme SGBD cible • PHP pour les requêtes côté backend

CONDITIONS DE REALISATION	
Matériels	Poste de travail personnel, serveur applicatif, serveur de base de données
Logiciels	PostgreSQL, PHP, Looping, dbdiagram.io, navigateur web
Durée	
Contraintes	Respecter la séparation serveur applicatif / serveur BDD, assurer l'intégrité référentielle entre les tables, sécuriser le stockage des mots de passe.

COMPETENCES MISES EN OEUVRE POUR CETTE ACTIVITE PROFESSIONNELLE	
Code	Intitulé
B2.1	Concevoir ou adapter une solution applicative
B2.2	Concevoir une base de données
B3.1	Exploiter les ressources du système d'information

DEROULEMENT DE L'ACTIVITE

Liste des étapes :

1. **Recueil des besoins**
2. **Dictionnaire de données**
3. **MCD**
4. **MLD**
5. **MPD /Script SQL**

---

1. Recueil des besoins

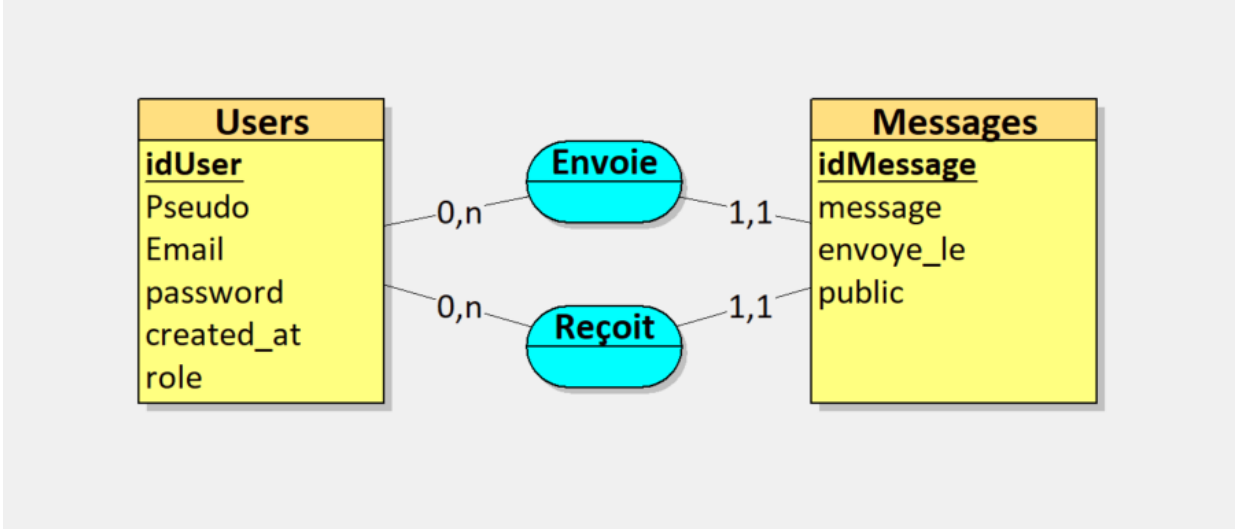
L'application doit permettre à un utilisateur de s'inscrire et de se connecter avec un identifiant et un mot de passe. Une fois connecté, il peut envoyer des messages à d'autres utilisateurs. Chaque message est horodaté et rattaché à un expéditeur et un destinataire. Ces besoins font apparaître deux entités principales : les utilisateurs et les messages.

2. Dictionnaire de données

Donnée	Format	Longueur	Contrainte	Entité
id_user	Entier	-	PK, auto-incrémenté	users
pseudo	Chaîne	50	NOT NULL, UNIQUE	users
email	Chaîne	100	NOT NULL, UNIQUE	users
password	Chaîne	255	NOT NULL (hashé)	users
created_at	Date/heure	-	DEFAULT NOW()	users
role	Chaîne	10	NOT NULL, DEFAULT 'user '	users
id_message	Entier	-	PK, auto-incrémenté	messages
message	Texte	-	NOT NULL	messages
envoyé_le	Date/heure	-	DEFAULT NOW()	messages
Id_expediteur	Entier	-	FK → users(id)	messages
Id_destinataire	Entier	-	FK → users(id)	messages
public	Booleen			

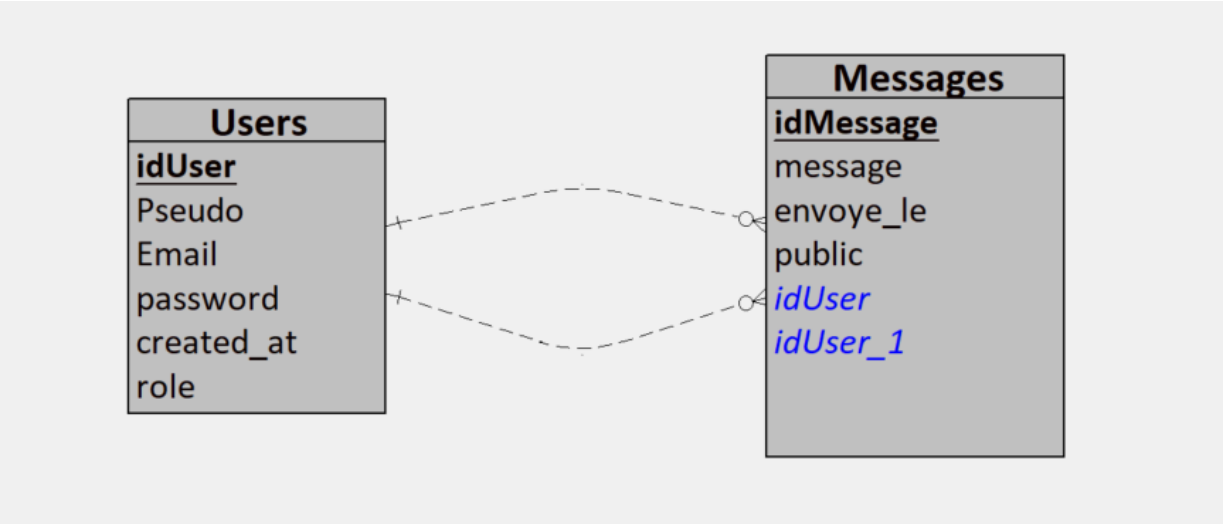
3. MCD (Modèle Conceptuel des Données)

Le MCD a été réalisé avec l'outil Looping (looping-mcd.fr). Il met en évidence deux entités (USER et MESSAGE) reliées par deux associations : ENVOIE (un utilisateur envoie un ou plusieurs messages) et REÇOIT (un utilisateur reçoit un ou plusieurs messages). Les cardinalités sont les suivantes : un utilisateur peut envoyer 0 à N messages, chaque message est envoyé par exactement 1 utilisateur. La même logique s'applique pour la réception.



4. MLD (Modèle Logique de Donnés)

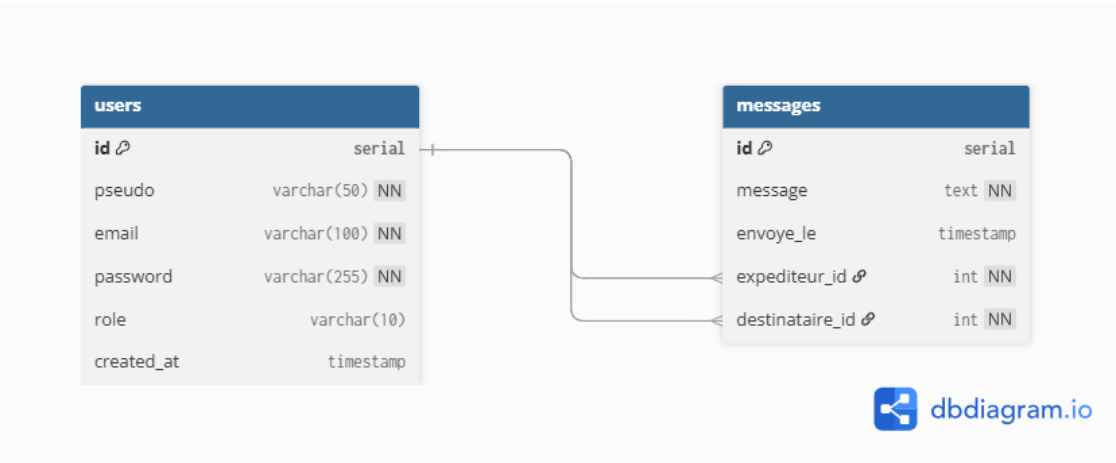
Le MLD est généré automatiquement depuis Looping. Les deux associations se traduisent par deux clés étrangères dans la table messages :



5. MPD (Modèle Physique des Données) / Script SQL

Le schéma physique a été visualisé avec dbdiagram.io, le script de création des tables en MySql pour le test local, et l'adaptation à PostgreSQL pour la version serveur :

a) Schema physique (Diagramme) :



b) Script MySql :

db_chat.sql

```sql
CREATE DATABASE IF NOT EXISTS `db_chat`
DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;
USE `db_chat`;

--
-- Structure de la table `t_messages`
--

DROP TABLE IF EXISTS `t_messages`;
CREATE TABLE IF NOT EXISTS `t_messages` (
  `id` int NOT NULL AUTO_INCREMENT,
  `expediteur` int NOT NULL,
  `message` longtext NOT NULL,
  `date` datetime DEFAULT CURRENT_TIMESTAMP,
  `destinataire` int DEFAULT NULL,
  `is_public` tinyint(1) DEFAULT '0',
  PRIMARY KEY (`id`),
  KEY `fk_id_dest` (`destinataire`) USING BTREE,
  KEY `fk_id_expe` (`expediteur`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

--
-- Structure de la table `t_users`
--

DROP TABLE IF EXISTS `t_users`;
CREATE TABLE IF NOT EXISTS `t_users` (
  `id` int NOT NULL AUTO_INCREMENT,
  `pseudo` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `role` varchar(10) DEFAULT 'user',
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `pseudo` (`pseudo`),
  UNIQUE KEY `email` (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

--
-- Contraintes pour la table `t_messages`
--
ALTER TABLE `t_messages`
  ADD CONSTRAINT `t_messages_ibfk_1` FOREIGN KEY (`destinataire`)
  REFERENCES `t_users` (`id`) ON DELETE SET NULL ON UPDATE SET NULL,
  ADD CONSTRAINT `t_messages_ibfk_2` FOREIGN KEY (`expediteur`)
  REFERENCES `t_users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;
COMMIT;

```

c) Script PostgreSQL :

db_chat_pgsql.sql

```sql
-- Créer la base de données
CREATE DATABASE db_chat;

-- Se connecter à la nouvelle base
\c db_chat

-- Suppression des tables si elles existent (ordre important à cause des contraintes)
DROP TABLE IF EXISTS t_messages CASCADE;
DROP TABLE IF EXISTS t_users CASCADE;

-- Table t_users
CREATE TABLE t_users (
  id SERIAL PRIMARY KEY,
  pseudo VARCHAR(255) NOT NULL UNIQUE,
  email VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  role VARCHAR(10) DEFAULT 'user'
);

-- Index sur pseudo et email (déjà créés automatiquement via UNIQUE)
CREATE INDEX idx_users_pseudo ON t_users(pseudo);
CREATE INDEX idx_users_email ON t_users(email);

-- Table t_messages
CREATE TABLE t_messages (
  id SERIAL PRIMARY KEY,
  expediteur INTEGER NOT NULL,
  message TEXT NOT NULL,
  date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  destinataire INTEGER NOT NULL,
  CONSTRAINT fk_expediteur FOREIGN KEY (expediteur)
    REFERENCES t_users(id) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_destinataire FOREIGN KEY (destinataire)
    REFERENCES t_users(id) ON DELETE CASCADE ON UPDATE CASCADE
);

-- Index pour améliorer les performances
CREATE INDEX idx_messages_expediteur ON t_messages(expediteur);
CREATE INDEX idx_messages_destinataire ON t_messages(destinataire);
CREATE INDEX idx_messages_date ON t_messages(date);

-- Afficher les tables créées
\dt
```

CONCLUSION

La modélisation de cette base de données m'a permis de structurer rigoureusement les données de mon application de chat en suivant la démarche Merise. Le principal défi technique a été de modéliser la double relation entre les utilisateurs et les messages (expéditeur et destinataire), qui se traduit par deux clés étrangères pointant vers la même table. Le choix de PostgreSQL, conseillé pour sa robustesse et sa conformité aux standards SQL, s'est avéré pertinent dans un contexte de séparation des serveurs. *(Compléter avec les difficultés que tu as réellement rencontrées)*démarches et solutions.

EVOLUTION POSSIBLE

Plusieurs améliorations pourraient être envisagées : l'ajout d'une table conversations pour gérer des groupes de discussion, l'intégration d'un champ *is_read* dans la table messages pour gérer les accusés de lecture, ou encore l'ajout d'une table *attachments* pour permettre l'envoi de fichiers joints.