Rapport module SQL

Ce document présente les choix techniques, conceptuels et l'architecture de la base de données relationnelle conçue pour l'application de gestion de mosaïques. La base de données gère les clients, les commandes, la composition des produits (mosaïques), ainsi que les stocks de pièces élémentaires.

AISSYNE Amine, ALKHASTOV Zhabrail,
ESSAIDI Rayan, POLIN Ethan

1. Choix Techniques

1.1 Système de Gestion de Base de Données (SGBD)

- **SGBD :** MySQL / MariaDB.
- **Moteur de Stockage :** InnoDB.
    - *Justification :* InnoDB a été choisi pour sa gestion des transactions (ACID), essentielle pour garantir l'intégrité des commandes et des mouvements de stocks. Il supporte également les clés étrangères (FOREIGN KEY), assurant la cohérence référentielle des données.

1.2 Encodage

- **Jeu de caractères :** utf8mb4.
- **Interclassement :** utf8mb4_general_ci.
    - *Justification :* utf8mb4 est le standard moderne permettant de stocker l'ensemble des caractères Unicode (y compris les emojis et caractères spéciaux), contrairement à l'ancien utf8 de MySQL. Cela garantit une compatibilité internationale pour les données clients et les descriptions produits.

1.3 Conventions de Nommage

- **Tables :** PascalCase (ex: CustomerOrder, MosaicComposition). Cette convention permet de distinguer clairement les entités.
- **Colonnes :** Snake_case (ex: id_Customer, bank_name) pour une lisibilité accrue dans les requêtes SQL complexes.
- **Clés Primaires :** Préfixées par id_ suivi du nom de l'entité (ex: id_Item), sauf exceptions sémantiques.

2. Choix Conceptuels et Modélisation

2.1 Décomposition des Produits (Items)

Plutôt que d'avoir une table unique géante pour tous les types de pièces, nous avons opté pour une normalisation stricte (3NF) :

- La table Item représente une pièce générique.
- Les caractéristiques sont externalisées dans des tables de référence : Shapes (Formes) et Colors (Couleurs).
- *Avantage :* Évite la redondance de données textuelles (ex: répéter "Rouge" 1000 fois) et facilite la maintenance des attributs.

2.2 Gestion des Mosaïques (Composition)

La structure suit un modèle de "Nomenclature" (Bill of Materials) :

- **Mosaic** : L'objet fini vendu au client.
- **MosaicComposition** : Table d'association liant Mosaic à Item. Elle définit de quoi est faite une mosaïque (combien de pièces de quel type).
- *Contraintes :* ON DELETE CASCADE est appliqué sur la composition. Si une mosaïque est supprimée, sa recette (composition) l'est aussi, évitant les orphelins.

2.3 Gestion de l'Immuabilité (Commandes)

Un point critique dans la conception e-commerce est l'immuabilité des commandes passées.

- La table CustomerOrder enregistre l'entête de la commande.
- La table OrderItem lie la commande aux articles (Item).
- **Choix fort :** Bien que liée à Item, la table OrderItem doit conceptuellement "figer" l'état de la vente au moment T. Si le prix d'un Item change dans le futur, cela ne doit pas impacter les commandes passées.

2.4 Gestion des Stocks (Traçabilité)

L'existence de la table StockEntry indique un choix de gestion par "Journalisation" (ou *Event Sourcing* simplifié) plutôt que par simple compteur :

- Au lieu de modifier uniquement une colonne quantity dans Item, on peut enregistrer les mouvements dans StockEntry.
- *Avantage :* Permet une traçabilité totale (audit) de qui a ajouté/retiré du stock et quand.

3. Triggers

Nous avons choisi d'utiliser des déclencheurs (triggers) pour faciliter certaines opération et de garantir la sécurité

3.1 Automatisation : Génération de Facture Séquentielle

La numérotation des factures est généré ainsi : `FAC-{ANNEE}-{CLIENT}-{SEQ}`

Trigger : `before_invoice_insert`

- **Fonctionnement :** À chaque insertion d'une facture, le trigger :
  1. Récupère l'ID du client historique (**`SaveCustomer`**) via la commande.
  2. Construit le préfixe dynamique (ex: **`FAC-2024-42-`**).
  3. Recherche le dernier numéro de séquence utilisé pour ce client et cette année.
  4. Incrémente le compteur et formate le numéro final (ex: `FAC-2024-42-001`).
- **Intérêt :** Gestion parfaite de la concurrence (évite les doublons) et garantie de la conformité comptable.

3.2 Sécurité Anti-Fraude : Verrouillage des Commandes

Une commande est une entité vivante ses données financières sont immuables.

- **Trigger : `secure_order_update`**
- **Stratégie Hybride :**
  - **Autorisé :** La modification du statut (ex: passage de "Payée" à "Expédiée").
  - **Bloqué :** Le trigger lève une exception (**`SQLSTATE 45000`**) si l'on tente de modifier le **montant total**, la **date**, l'**identité du client** ou le **produit acheté**.
- **Intérêt :** Empêche de modifier le prix d'une commande après paiement.

3.3 Intégrité et Immuabilité (Data Preservation)

Certaines tables représentent des archives historiques qui ne doivent jamais être altérées.

Tables protégées (Delete/Update Interdits) :

- **`Invoice` :** Une facture émise est un document légal définitif.
- **`SaveCustomer` :** L'historique des informations client au moment de la commande ne doit jamais disparaître.
- **`FactoryOrder` & `Details` :** Assure la cohérence avec le module de production Java. Si une commande de réapprovisionnement est lancée, elle ne doit pas être supprimée en base sous peine de désynchroniser les stocks réels.

4. Procédures Stockées

L'utilisation de procédures stockées permet de centraliser les calculs lourds côté serveur SQL et d'offrir une interface simplifiée aux applicatifs (PHP, Java).

4.1 Vérification de Disponibilité (Encapsulation)

Une mosaïque est composée de milliers de pièces différentes. Vérifier si on peut la fabriquer ne doit pas se faire via des centaines de requêtes PHP.

- **Procédure : `check_mosaic_stock`**
- **Paramètres : `IN p_id_Mosaic`, `OUT p_is_available`**
- **Logique :**
  - La procédure compare **en une seule opération** la composition de la mosaïque (**`MosaicComposition`**) avec l'état actuel des stocks (via la vue **`View_StockStatus`**).
  - Elle renvoie un simple booléen (**`TRUE`**/**`FALSE`**).
- **Gain :** Performance drastique et simplification du code PHP qui appelle juste **`CALL check_mosaic_stock(?, @res)`.**

4.2 Agrégation des Stocks en Temps Réel

Le stock est la somme de tous les mouvements (Entrées - Sorties).

- **Procédure : `get_all_items_stock`**
- **Rôle :** Fournir au tableau de bord Admin une vue consolidée.
- **Logique :** Elle effectue les jointures nécessaires entre **`Item`**, **`Shapes`**, **`Colors`** et agrège dynamiquement la table **`StockEntry`** (**`SUM(quantity)`**).

4.3 Interface d'Export pour les Modules Externes (Java/C)

Les modules de traitement d'image (C et Java) ont besoin de connaître le catalogue de pièces pour réaliser la pixelisation (Pavage).

- **Groupe de Procédures :**
  - **`get_export_colors`** : Fournit les codes HEX pour l'algo de *Color Matching*.
  - **`get_export_shapes`** : Fournit les dimensions physiques pour l'algo de *Pavage*.
  - **`get_export_items_stock`** : Fournit l'état du stock pour l'algo d'*Optimisation*
- **Intérêt :** Découplage fort. Si la structure interne de la base change, on modifie juste ces procédures sans avoir à recompiler le code C ou Java.

5. Vues

Les vues ont été mises en place pour masquer la complexité du modèle de données (notamment le calcul de stock par historique de mouvements).

5.1 Tableau de Bord : Alerte Stock Critique

Le tableau de bord administrateur doit afficher immédiatement les problèmes sans charger tout l'inventaire.

- **Vue : `View_LowStockDetails`**
- **Rôle :** Filtrage préventif.
- **Logique :**
  - Cette vue effectue les jointures entre **`Item`**, **`Shapes`** et **`Colors`**.
  - Elle agrège dynamiquement les entrées/sorties (**`StockEntry`**) pour obtenir le stock réel.
  - **Filtre intégré :** Elle ne retourne *que* les lignes où **`current_stock < 50`**.
- **Intérêt :** Optimisation des performances. Le code PHP exécute un simple **`SELECT * FROM View_LowStockDetails`** au lieu de récupérer 10 000 articles et de les trier en mémoire.

5.2 Gestion de l'Inventaire : Champs Calculés

Pour la page de gestion des stocks, nous avons besoin d'afficher l'état complet du magasin avec une indication visuelle simple.

- **Vue : `View_StockStatus`**
- **Rôle :** Présentation et Logique conditionnelle.
- **Logique :**
  - Comme la précédente, elle calcule le stock en temps réel.
  - **Champ Calculé :** Elle intègre une colonne virtuelle **`alert_status`** générée par un **`CASE WHEN`**. Si le stock est inférieur à 50, la colonne vaut "OUI", sinon "NON".
- **Intérêt (Architecture) :** Centralisation de la règle métier. Si demain le seuil d'alerte passe à 100, on modifie uniquement la Vue SQL. Pas besoin de chercher et modifier cette condition dans tous les fichiers PHP du projet.

6. Sécurité et Droits

- **Utilisateur Applicatif :** L'application PHP/Java ne doit pas se connecter en root. Un utilisateur spécifique app_user doit être créé avec des droits restreints (SELECT, INSERT, UPDATE, DELETE) sur les tables de la base SAE_S3_BUT2_INFO uniquement.
- **Injection SQL :** L'utilisation de types stricts (int, date) dans la structure force une première validation des données, mais les requêtes préparées côté applicatif restent obligatoires.

7. MCD