Modélisation d'une base de données pour des réservations de billets de concerts

Marie Chaneac, Marie-Lou Ribeiro Marquès

May 19, 2025

1 Explications du fonctionnement

Nous avons décidé de modéliser un site de réservations de tickets de concerts dont nous allons, dans cette section, détailler le fonctionnement global.

Si un client souhaite acheter des billets sur le site il doit d'abord se créer un compte. A la création de ce compte il a le choix de prendre un abonnement VIP ou non (pour nous cela est représenté simplement par un attribut statutUser qui peut valoir soit "VIP" soit "regular"). En fonction de ce statut il a accès ou non à certaines sessions de vente d'un événement.

Un événement génère des sessions de vente qui mettent en vente un nombre de billets predéfini et qui ont une date/heure de début et de fin. Il existe des sessions réservées aux utilisateurs VIP et d'autres non.

Les utilisateurs peuvent être placés dans une file d'attente dans le cadre d'une session de vente (la file d'attente est spécifique à chaque vente), pour accéder à l'étape du choix de leurs billets (l'utilisateur pourra choisir ses billets en fonction de leur catégorie, ou bien choisir leur numéro de siège). Un rang leur sera attribué dans la file en fonction de leur arrivée sur la page de la session de vente, rang qui évoluera en fonction de l'avancée de la vente des billets.

Une fois que l'utilisateur a fini de faire la queue, et que son tour est venu de choisir ses billets, il peut les placer dans son panier. Une fois qu'il a choisi tous ses billets (dans la limite du nombre de billets qu'il a le droit d'acheter par événement en fonction de son statut) il peut passer à la réservation de ses billets. Son panier est chronométré et au bout d'un certain temps il est supprimé, ses billets ne sont plus pré-réservés (plus dans le panier), et si celui-ci souhaite à nouveau mettre un billet dans son panier, il sera placé en fin de queue.

Une pré-réservation va donner lieu à une potentielle transaction, et si celle-ci est terminée, elle donne lieu à une réservation des billets concernés.

2 Modèle logique de données et contraintes

Billet(idBillet, statutBillet, #numSiège, #date, #idEvent, #idPanier, #idSession)

Grille(#idEvent, #idCategorie, prix)

CategorieSiege(idCategorie, nomCategorie, capaciteCategorie, #idLieu)

Siege(idSiege, numSiege, #idCategorie)

Lieu(idLieu, nomLieu, adresseLieu, capaciteAccueil)

AvoirLieu(#date, #idEvent, #idLieu, heure)

Calendrier(date)

Concert(idEvent, descriptionEvent)

Participe(#idEvent, #idArtiste)

Artiste(idArtiste, nomArtiste, styleMusique)

SessionVente(idSession, dateDebutSession, dateFinSession, nbBilletsMisEnVente, onlyVIP, nbMaxBilletsAchetesVIP, nbMaxBilletsAchetesRegular, #idEvent)
(CHECK: dateDebutSession avant dateFinSession, nbBilletsMisEnVente > 0)

FileAttente(idQueue, capaciteQueue, #idSessionVente)
(CHECK: capaciteQueue > 0)

Attendre(#idQueue, #idUser, rang)
(UNIQUE: (idQueue, rang), TRIGGER: ($0 <$ rang $<$ capaciteQueue))

PreReservation(idPanier, dateHeureCreation, #idUser, #idSession)

Reservation(idReservation, dateReservation, #idUser, #idTransaction)

Transaction(idTransaction, dateTransaction, montant, statutTransaction, #idPanier)

Utilisateur(idUser, nomUser, prenomUser, adresseUser, dateInscription, email, statutUser, ptsFidelite)

3 Dépendances fonctionnelles

$DF = \{RidBillet \longrightarrow idSiege, date, idEvent, idSessionVente, idPanier, prix, statut$
$idEvent \longrightarrow desriptionEvent$
$idLieu \longrightarrow nomLieu, adresseLieu, capaciteAcceuil$
$idEvent, date \longrightarrow idLieu, heure$
$idSiege \longrightarrow idCategorie, numSiege, categorie$
$idCategorie \longrightarrow nomCategorie, capaciteCategorie, idLieu$
$idCategorie, idEvent \longrightarrow prix$
$idArtiste \longrightarrow nomArtiste, styleMusique$
$idSessionVente \longrightarrow dateDebutSession, dateFinSession, nbBilletsMisEnVente, idQueue, idEvent,$
$onlyVIP, nbMaxBilletsAchetesVIP, nbMaxBilletsAchetesRegular$
$idPanier \longrightarrow DateHeureCreation, idUser, idReservation, idSessionVente$
$idQueue \longrightarrow capaciteQueue$
$idQueue, idUser \longrightarrow rang$
$idQueue, rang \longrightarrow idUser$
$idUser \longrightarrow idPanier, nomUser, prenomUser, adresseUser, dateInscription, email, statutUser,$
$ptsFidelite$
$idReservation \longrightarrow dateReservation, idUser, idTransaction$
$idTransaction \longrightarrow dateTransaction, montant, statutTransaction, idPanier$

4 Triggers et fonctions d'automatisation

Utilisateur:

trigger_verrou_utilisateur et **verrouiller_modifications_utilisateur()** : Empêche la modification de colonnes sensibles (email, statut utilisateur, date d'inscription). Seule la mise à jour des points de fidélité est autorisée, et uniquement si la variable myapp.allow_pts_update est activée.

Concert:

trigger_verrou_modifications_concert et **verrouiller_modifications_concert()** : Empêche la modification de la colonne idEvent, garantissant que l'identifiant de l'événement reste inaltéré lors d'une mise à jour.

Lieu

trigger_verrou_modifications_lieu et **verrouiller_modifications_lieu()** : Empêche la modification de la colonne idLieu, garantissant ainsi que l'identifiant du lieu reste inchangé lors des mises à jour.

CategorieSiege:

trigger_verrou_modifications_categorie et **verrouiller_modifications_categorie()** : Empêche la modification des colonnes idCategorie et nomCategorie, assurant que ces identifiants restent constants lors des mises à jour.

Siege:

trigger_verrou_modifications_siege et **verrouiller_modifications_siege()** : Empêche la modification de la colonne idSiege, garantissant ainsi que l'identifiant du siège reste inchangé lors des mises à jour.

AvoirLieu:

trigger_verifier_horaire_al et **verifier_horaire_al()** : Empêche la création d'un enregistrement en doublon pour le même lieu en vérifiant que la combinaison dateEvent, idLieu et heure n'existe pas déjà avant insertion.

trigger_set_grille et **setGrillePrix()** : Calcule et insère automatiquement dans la table Grille les tarifs pour chaque catégorie de siège d'un événement, en attribuant des prix par défaut modulés par le coefficient de demande attendu de l'événement.

trigger_supprimer_dans_grille et **supprimer_dans_grille()** : Supprime automatiquement les entrées de la table Grille associées à un événement dès qu'une ligne correspondante est supprimée de la table AvoirLieu.

trigger_creer_session_vente et **creer_session_vente_par_defaut()** : (Commenté) Prévoit la création automatique d'une session de vente par défaut pour un événement et un lieu donnés, en fixant les périodes de vente et les capacités selon les informations du lieu, bien que ce code ne soit pas actuellement actif.

Grille:

trigger_genererBillets et **genererBillets()** : Automatise la création des billets en récupérant la capacité d'une catégorie depuis la table CategorieSiege et en parcourant toutes les dates d'un événement dans un lieu donné (via AvoirLieu). Pour chaque date, la fonction sélectionne les sièges disponibles pour la catégorie concernée et insère des billets (marqués "en vente" avec le prix indiqué), évitant ainsi les doublons.

Artiste:

trigger_verrou_modifications_concert et **verrouiller_modifications_artiste()** : Empêche la modification de la colonne idArtiste, garantissant ainsi que l'identifiant de l'artiste reste inchangé lors des mises à jour.

SessionVente:

trigger_verrou_modification_sv et **verrouiller_modification_sv()** : Empêche la modification d'une session de vente en vérifiant que la variable de configuration `myapp.allow_modify_sv` est activée. Sinon, la modification est bloquée par une exception.

trigger_verrou_creation_sv et **verrouiller_creation_sv()** : Empêche la création d'une nouvelle session de vente hors des contextes autorisés en se basant sur la variable `myapp.allow_create_sv`.

trigger_verifier_et_attribuer_session et **verifier_et_attribuer_session_vente()** : Automatise l'attribution de l'identifiant de session aux billets d'un événement pour un lieu donné. La fonction compte les billets disponibles et, si leur nombre est suffisant, met à jour leur champ `idSession` avant insertion, sinon elle lève une exception.

trigger_nettoyer_billets_apres_annulation et **nettoyer_billets_session_annulee()** : Nettoie automatiquement le champ `idSession` des billets en le passant à `NULL` lorsque la session de vente associée est supprimée.

trigger_creer_file_attente et **creer_file_attente()** : Crée automatiquement une file d'attente dans la table `FileAttente` pour la session de vente, avec une capacité déterminée par le nombre de billets mis en vente (doublé, plafonné à 1000), afin de gérer l'afflux des demandes.

FileAttente:

trigger_verrou_modification_file et **verrouiller_modification_file()** : Empêche la modification des enregistrements dans la table FileAttente en s'assurant que la variable `myapp.allow_modify_file` est activée avant toute mise à jour.

trigger_verrou_creation_file et **verrouiller_creation_file()** : Interdit la création de nouvelles entrées dans FileAttente en vérifiant que la variable `myapp.allow_create_file` est activée; sinon, une exception est levée.

Attendre:

trigger_verrou_modification_attendre et **verrouiller_modification_attendre()** : Empêche la modification non autorisée dans Attendre en vérifiant que la variable de configuration `myapp.allow_modify_attendre` est activée. Sinon, la tentative de mise à jour est bloquée par une exception.

trigger_verrou_creation_attendre et **verrouiller_creation_attendre()** : Interdit la création de nouvelles entrées dans Attendre en s'assurant que `myapp.allow_create_attendre` est activée avant l'insertion.

trg_check_rang et **check_rang_valid()** : Vérifie que le rang attribué à un utilisateur se situe dans l'intervalle [0, capaciteQueue-1] défini pour la file (récupérée via `idQueue`). Une valeur hors de ce domaine lève une exception.

trigger_verifie_periode_session et **verifie_periode_session()** : Assure que l'insertion dans la file d'attente se fait durant la période d'ouverture de la session de vente associée, en comparant le timestamp courant aux dates de début et de fin de session. Si l'insertion intervient en dehors de cette période, elle est refusée.

trg_shift_queue_after_exit et **shift_queue_after_exit()** : Après la suppression d'un utilisateur de la file, cette fonction décale le rang de tous les utilisateurs situés derrière celui supprimé en décrémentant leur valeur de rang, pour maintenir l'ordre de la file.

trg_before_insert_attendre et **trg_combine_capacity_and_rank()** : Avant l'insertion, la fonction vérifie que la file n'est pas pleine en comparant le nombre d'utilisateurs déjà présents à la capacité maximale de la file. En cas de place disponible, elle attribue au nouvel utilisateur un rang positionné à la fin (le rang maximum existant + 1).

SAS:

trigger_verrou_modification_sas et **verrouiller_modification_sas()** : Empêche la modification des enregistrements dans la table SAS en s'assurant que la variable de configuration `myapp.allow_modify_sas` est activée. Sinon, toute tentative de mise à jour génère une exception.

trigger_verrou_creation_sas et **verrouiller_creation_sas()** : Empêche la création de nouvelles entrées dans la table SAS en vérifiant que `myapp.allow_create_sas` est activée, et lève une exception en cas de non-activation.

trigger_annuler_pre_resa_if_sas_expelled et **annuler_pre_reservation_if_user_lost_SAS()** : Annule automatiquement la pré-réservation d'un utilisateur (en supprimant la ligne correspondante dans `PreReservation`) lorsque son statut dans SAS passe de "en cours" à "expulse". La fonction récupère l'identifiant de session associé via la table `FileAttente` et procède ensuite à l'annulation, tout en affichant une notification.

trigger_creer_panier_prereservation et **creer_prereservation()** : Crée automatiquement un panier de pré-réservation pour un utilisateur dès son insertion dans la table SAS. La fonction récupère l'identifiant de session associé à partir de `FileAttente`, insère la pré-réservation dans la table `PreReservation` et notifie l'utilisateur via l'appel à `informerUtilisateurOptionsAchat`.

PreReservation:

trigger_remettre_billets_en_vente et **remettre_billets_en_vente()** : Automatise la remise en vente des billets associés à un panier supprimé dans PreReservation. Lors de la suppression, la fonction met à jour les entrées dans Billet en définissant leur statut à "en vente" et en réinitialisant le champ idPanier, après avoir temporairement activé les permissions nécessaires via set_config.

trigger_verrou_modification_prereservation et **verrouiller_modification_prereservation()** : Empêche les modifications non autorisées dans la table PreReservation en vérifiant que la variable de configuration `myapp.allow_modify_prereserv` est activée, sinon l'opération est bloquée par une exception.

trigger_verrou_creation_prereservation et **verrouiller_creation_prereservation()** : Interdit la création de nouvelles entrées dans PreReservation en s'assurant que la variable `myapp.allow_create_prereserv` est activée, levant une exception en cas de non-conformité.

Transaction:

trigger_verrou_modification_transaction et **verrouiller_modification_transaction()** : Empêche la modification des enregistrements dans la table Transac en vérifiant que la variable `myapp.allow_modify_transac` est activée, sinon une exception est levée.

trigger_verrou_creation_transaction et **verrouiller_creation_transaction()** : Interdit la création de nouvelles transactions hors des contextes autorisés en s'assurant que la variable `myapp.allow_create_transac` est activée avant l'insertion.

calculer_reduction(idPanierInput) : Calcule le pourcentage de réduction applicable à une transaction en se basant sur le nombre de points de fidélité de l'utilisateur associé au panier, avec des paliers à 300, 200 et 100 points.

creer_transaction_avec_montant_calcule(idPanierInput) : Calcule le montant total du panier en sommant les prix des billets, applique la réduction déterminée par `calculer_reduction()` et insère une transaction avec le montant final et le statut 'en attente'.

trigger_creer_reservations_apres_validation et **creer_reservations_depuis_transac()** : Dès que le statut d'une transaction passe de 'en attente' à 'validé', crée automatiquement une réservation liée en récupérant l'identifiant de l'utilisateur à partir du panier associé.

trigger_billets_vendus et **marquer_billets_vendus()** : Lors de la validation d'une transaction, met à jour le statut des billets associés au panier pour les marquer comme 'vendu', assurant ainsi la cohérence de la vente.

trigger_maj_statut_sas et **maj_statut_sas_apres_validation_transac()** : Après validation d'une transaction, met à jour le SAS correspondant à l'utilisateur en passant son statut à 'termine' et en enregistrant l'heure de sortie, indiquant ainsi la finalisation de la session.

Reservation:

trigger_verrou_modification_reservation et **verrouiller_modification_reservation()** : Empêche la modification des enregistrements dans la table Reservation en vérifiant que la variable de configuration `myapp.allow_modify_reserv` est activée. Sinon, une exception est levée et la modification est bloquée.

trigger_verrou_creation_reservation et **verrouiller_creation_reservation()** : Interdit la création de nouvelles entrées dans Reservation en s'assurant que la variable `myapp.allow_create_reserv` est activée, sinon l'insertion est rejetée.

Billet:

trigger_avant_insert_billet et **avant_insert_billet()** : Vérifie que le nombre total de billets pour un événement à une date donnée ne dépasse pas la capacité du lieu. La fonction récupère la capacité via la table `Lieu` (jointure avec `AvoirLieu`) et compte les billets déjà créés pour l'événement à la date donnée. Si le total est supérieur ou égal à la capacité, une exception est levée.

trigger_avant_insert_statut_billet et **verifier_statut_billet()** : Garantit que le statut initial d'un billet est fixé à `"en vente"` lors de son insertion. Toute valeur différente déclenche immédiatement une exception.

trigger_verrou_modifications_billet et **verrouiller_modifications_billet()** : Empêche toute modification non autorisée d'un billet. La fonction bloque toute mise à jour si le billet est déjà vendu et interdit de changer le prix, le statut, `idSession` ou `idPanier` sauf si les permissions correspondantes (via les variables de configuration) sont activées. De plus, elle interdit la modification des champs critiques `idSiege`, `dateEvent` et `idEvent`.

trigger_mis_panier_billet et **mis_panier_billet()** : Lorsqu'un billet passe du statut `"en vente"` à `"dans un panier"`, cette fonction autorise le changement en activant temporairement la permission de modification du statut via `set_config`.

trigger_vendre_billet et **vendu_billet()** : Permet de modifier le statut d'un billet de `"dans un panier"` à `"vendu"` en activant temporairement la permission nécessaire pour cette mise à jour.

trigger_autoriser_changement_idsession et **autoriser_changement_idsession()** : Contrôle la modification du champ `idSession` d'un billet. La modification est autorisée uniquement dans les deux cas suivants : passage de `NULL` à une valeur non nulle si la session existe dans `SessionVente`, ou passage d'une valeur non nulle à `NULL`. Pour tout autre cas, une exception est levée.

4.1 Index

idx_avoirlieu_date_lieu_heure (**utilisé dans** `verifier_horaire_al()`) : Accélère la recherche des événe-ments ayant lieu à une date et un endroit spécifique.

idx_billet_statut_event_session (**utilisé dans** `verifier_et_attribuer_session_vente()`) : Optimise la récupération des billets en fonction de leur statut et de leur événement ou session de vente.

idx_billet_panier (**utilisé dans** `remettre_billets_en_vente()`, `preReserver()`) : Facilite l'accès aux billets associés à un panier donné, accélérant les opérations de pré-réservation.

idx_categorie_idlieu (**utilisé dans** `prereserver_demande()`) : Permet une identification rapide des catégories de sièges disponibles dans un lieu donné.

idx_prereservation_user (**utilisé dans** `afficher_billets_par_email()`, `preReserverAvecEmail()`) : Améliore l'accès aux pré-réservations effectuées par un utilisateur spécifique.

idx_billet_statut (**utilisé dans** `afficher_billets_par_email()`, `prereserver_demande()`) : Accélère le filtrage des billets par statut (`en vente`, `vendu`, etc.).

idx_billet_idsiege (**utilisé dans** `prereserver_demande()`) : Optimise les recherches de billets en fonction de leur siège attitré.

idx_transac_panier (**utilisé dans** `afficher_billets_par_email()`) : Facilite la récupération des transactions associées à un panier.

idx_sas_status (**utilisé dans** `verifierExpulsionsSAS()`) : Améliore le suivi des utilisateurs en fonction de leur statut dans le SAS.

idx_fileattente_sessionvente (**utilisé dans** `preReserver()`) : Accélère la recherche des files d'attente liées à une session de vente spécifique.

4.2 Scénario : Simulation complète du processus de réservation

- **Création des utilisateurs** : Ajout de plusieurs utilisateurs dans la base avec leurs informations personnelles, leur email et leur statut (`VIP` ou `regular`).
- **Ajout des artistes et création des événements** : Enregistrement des artistes et des concerts auxquels ils participent.
- **Organisation des concerts** : Association des concerts à des lieux, dates et heures précises.
- **Simulation de l'entrée dans une file d'attente** : Inscription des utilisateurs dans une file en tenant compte des restrictions (période ouverte, capacité maximale).
- **Sortie de file et entrée dans le SAS** : Passage des utilisateurs en tête de file dans le SAS et expulsion automatique après expiration du délai de 2 minutes.
- **Pré-réservation des billets** : Vérification des demandes de billets et assignation des billets disponibles dans le panier de l'utilisateur.
- **Validation des transactions** : Simulation du paiement en fonction du montant fourni et validation des billets achetés.
- **Visualisation des billets achetés** : Affichage des billets réservés et achetés pour un utilisateur spécifique.