Réaliser des scripts

2.1

Si un utilisateur modifie les photos associées à un logement, on peut réaliser un trigger BEFORE UPDATE qui va récupérer l'ancienne valeur de l'image pour l'insérer dans une table photo_to_remove (à créer avant) par exemple. Une fois l'insert effectué, il peut remplacer l'ancienne valeur de l'image par la nouvelle dans la table photos.

Ainsi, un script pourra venir récupérer régulièrement les image (1 fois par jours, la nuit par exemple) dans la table photo_to_remove et les supprimer.

2.2

Lors d'un import, d'une création, d'une mise à jour d'un propriétaire, pour s'assurer qu'il possède au moins une information de contact on peut réaliser deux nouveau trigger BEFORE UPDATE et BEFORE INSERT.

Pour éviter de répéter le code et devoir modifier à deux endroits si nécessaire, on peut effectuer la vérification dans une procédure qui sera appelé dans les deux triggers.

2.3

Pour l'historisation du prix d'un logement, on va devoir créer une nouvelle table : logement_historique_prix. Ensuite, on réalise deux triggers, un pour l'insert d'un logement et un autre pour l'update. A chaque ajout, modification d'un logement, si le prix change on l'enregistre dans la table avec l'id du logement et la date du changement.

Pour l'historisation du statut, on réalise la même opération. Pour être plus simple on le fait dans le même trigger mais pas l'insert dans la même table (créer une nouvelle comme logement_historique_statut) que le prix avec un appel d'une procédure pour éviter d'avoir du copié collé.

Optimisez votre base de données

4

En fonction des demandes du clients et des requêtes qui pourraient amener à être régulières, on devrait rajouter des Index sur les champs qui seraient sollicité régulièrement. Ne pas mettre d'Index partout, on risque d'avoir l'effet inverse !

Activer le cache peut permettre d'améliorer les performances suivant la machine sur laquelle se trouve la base de données.

Utiliser EXPLAIN, (ANALYSE, VACCUM pour postgreSQL) (en fonction du SGBD) pour pouvoir analyser des requêtes, tables et pouvoir ainsi les optimiser et obtenir de meilleures performances. SQL_NO_CACHE peut-être très utile aussi pour pouvoir réaliser des requêtes sans cache et ainsi s'apercevoir du temps de chargement de la requête et travailler sur l'optimisation.

Dans les requêtes, il faut mieux éviter le SELECT * qui est assez gourmand et risque dans certaines requêtes avec des joins de retourner énormément de colonne et mettre du temps à charger. On peut aussi faire un LIMIT 1 si on attend qu'un seul résultat pour optimiser une requête.

Pour terminer sur l'optimisation, on peut utiliser des scripts comme MySQLTuner qui permet de suggérer des optimisations, des ajustements pour améliorer les performances et la stabilité.

Il ne faut pas hésiter à « purger » régulièrement la base des données qui ne sont plus utilisés voir inutile. Voir pourquoi archiver les tables d'historisation au bout d'un certain temps.

Au vu du nombres d'informations et de leurs importances, il faudra réaliser une tâche CRON régulière pour faire une sauvegarde de la base de données.
Une tâche CRON pour supprimer les anciennes photos serait aussi à mettre en place.

Il faudra aussi mettre en place des rôles sur la base de données pour éviter de donner accès à la suppression de table à tout le monde et aussi sécuriser les Delete avec des Triggers.

En fonction de la demande du client et de ses employés, il faudra prévoir la création de procédure pour des requêtes régulières (comme une moyenne des ventes sur une période donnée par exemple)