

Usine à Jeux de Donnée

Tests Simultanés



Gérer des Slices avec Data Slicer

Les enjeux liés à l'émergence de l'usine à test logiciel

La montée en puissance des architectures orientées services augmente le coût des tests. Pour les entreprises, il est urgent de mobiliser les équipes autour d'expertises, d'outils et d'objectifs communs.

"L'émergence de l'usine à test logiciel va courir sur les dix années à venir, selon la maturité et l'organisation des directions logicielles impliquées", estime Xavier Flez, Directeur d'Yphise. Pour les entreprises, il y a urgence à explorer de nouvelles pistes. Selon les mesures d'Yphise, l'industrialisation croissante dans le développement et l'utilisation des architectures orientées services ou des progiciels, font augmenter fortement le coût des tests. En effet, la plupart des projets aujourd'hui ne partent pas d'une "feuille blanche", et s'intègrent plutôt à un existant, réutilisent des composants, etc.

Un surcoût de 10%

Le résultat est une montée en flèche des coûts de tests de non régression, répartis entre la maîtrise d'œuvre et la maîtrise d'ouvrage. Selon ces calculs, un projet reprenant un fort existant coûte environ 10 % de plus qu'un projet "feuille blanche". Si cela ne suffisait pas, la part des tests s'y élève à 43 % du total (contre 23 en feuille blanche). En plus de cette inflation déjà préoccupante, d'autres phénomènes se renforcent, comme la tendance à déporter la vérification sur la maîtrise d'ouvrage. Elle est due à la concurrence qui fait rage entre prestataires pour décrocher, souvent après rabais, les projets de développement. Il ne reste tout simplement plus assez d'argent et de temps, en fin de processus, pour effectuer correctement les tests.

La nécessaire industrialisation des tests

Autre phénomène inquiétant : non seulement la part des tests de non régression augmente, mais ils alourdissent aussi le processus, parfois pour des apports métiers nuls aux utilisateurs. Il y a donc un hiatus qui apparaît entre la nécessité de produire des applications de qualité, et l'urgence de leur mise en place. Pour toutes ces raisons, Yphise milite aujourd'hui pour l'industrialisation

des tests. L'enjeu est clair : *"il y va de leur maîtrise des coûts, de la réactivité métier et finalement du risque opérationnel"*. Le concept d'usine à tests présente dans ce contexte l'intérêt de focaliser les énergies, autour d'expertises, d'outils et d'objectifs communs. Les maîtres mots en seront Automatisation et Référentiels.

Gérer des Slices avec Data Slicer

Data Slicer et **Data Compare** est une solution d'outillage des activités de test des applications :

Elle permet aux programmeurs, testeurs des applications et aux administrateurs de base de données d'avoir leur propre environnement de test avec des données privées. **Slice/Compare** élimine les conflits entre plusieurs personnes et groupes de personnes qui essaient de se partager un environnement de test. Des conflits peuvent intervenir quand les données manipulées pendant des tests sont écrasées lors de rafraîchissements de celles-ci ou lors d'une manipulation pour d'autres opérations de test. D'autres manifestations de conflits interviennent et impactent les équipes de test lors de la planification séquencée des tests pour éviter les chevauchements et les contentions d'accès aux données « deadlocks ».

Les organisations sont obligées de multiplier les sous-systèmes et les environnements pour fournir un niveau suffisant d'autonomie aux personnes impliquées dans les tests. Les opérations de tests nécessitent un contrôle maîtrisé des données manipulées.

Slice/Compare solutionne ces problèmes sans modifier les programmes sources. Le code testé est celui qui sera utilisé en production ultérieurement.

Le Problème :

Lors des phases de développement et de test d'une application utilisant une base de données relationnelle, plusieurs utilisateurs ont besoin d'accéder aux tables avec des buts différents. Tester des applications, quelles soit « petites » ou « grandes » en nombre de tables, peut devenir très onéreux. Par exemple, pendant les tests unitaires, le développeur D1 veut insérer et mettre à jour la ligne de la table « A », pendant qu'au même moment le développeur D2 veut supprimer des lignes de la table « A ». La fonction de partage des données entre plusieurs utilisateurs (Base de données relationnelle) devient une contrainte forte en environnement de développement et de test. En effet, ces tests ne peuvent pas se faire sur les mêmes données simultanément. De plus, les développeurs D1 et D2 vont générer des contentions d'accès à la table « A » induites par des locks. Les développeurs s'organisent et coordonnent leurs scénarii de test.

Tentative de Solutions dégradées :

Pour éviter ce problème, D1 et D2 consentent à utiliser différentes lignes pour toutes les tables modifiées (ou valeurs de clé primaire) pour réaliser leurs tests. Ceci à pour effet de reporter le problème, puisque dans plusieurs cas de figure les programmes du D2 doivent utiliser les données insérer et mises à jour par le D1. Si parfois, ceci leur permet d'effectuer leurs tests unitaires, cela ne leur permet de pas de partager facilement leur cas de test.

Supposons qu'un autre développeur D3 utilise la même table "A" pour tester une opération de chargement. D3 doit maintenant se coordonner avec D1 et D2. Ce nouveau développeur D3 a besoin d'un accès exclusif à la table « A ». Il ne peut tout simplement pas tester pendant que D1 et/ou D2 travaillent.

Une autre solution au problème est la gestion de plusieurs copies/sauvegarde de la même table. Une copie pour chaque développeur. Cette solution fonctionne, seul le DBA responsable de la création des tables est directement impacté (trois fois plus de sauvegarde en espace physique et en gestion).

L'impact sur la gestion est important. Cela peut entraîner des erreurs quand une table est oubliée dans un changement, comme le changement d'une colonne de SMALLINT en INTEGER.

S'il existe un programme partagé par les trois développeurs, il doit être « Bindé » pour les trois tables de chacun des développeurs. A chaque changement de la définition d'une table ou du code d'un programme, il doit être communiqué à tous ses utilisateurs impactés pour effectuer a nouveau les « bind » pour permettre l'exécution du programme.

Gérer des Slices

La mise en opération des données de test ou jeux de test doit prendre en compte les difficultés organisationnelles ou fonctionnelles. Ces contraintes sont directement liées à la communication entre les développeurs qui augmente de manière exponentielle, tout en étant impacté par le nombre croissant des personnes impliquées dans les phases de test. L'organisation doit intégrer des principes de communication, de duplication, de publication pour éviter des collisions de traitement des données.

Les départements études doivent faire des choix qui ne sont pas très confortables :

- Ne pas être en mesure d'ajouter plus de développeurs pour optimiser et fiabiliser la durée des tests
- Donner plus de travail aux DBAs pour la gestion du nombre croissant de tables
- Standardiser les données ou jeux de tests pour éviter les collisions
- Implémenter des files d'attente pour éviter les collisions

La Solution Data Slicer

Data Slicer offre d'autres alternatives :

Avec quelques conventions simples de programmation, la solution permet aux développeurs de partager leurs données de test dans un ensemble unique de tables dans leur environnement de test pour éviter les collisions pendant les phases de test. Ces techniques peuvent réduire le besoin d'avoir des centaines de copies des mêmes tables et permettre des tests simultanés avec des données partagées.

Les exemples cités jusqu'à présent concernaient les tests unitaires, **Slice** simplifie aussi :

- les tests de non régression
- les tests d'intégration
- les recettes applicatives
- la présentation pour démonstration des résultats (rapports)

Pour outiller et mettre en œuvre ces fonctionnalités, l'administrateur de **Slice** utilise l'utilitaire fourni par la solution « POWER UTILITY » pour réaliser la mise à jour automatique du catalogue (définition des tables). **Slice** nécessite que chaque ligne d'une table slicée soit identifiée par un identifiant unique pour chaque utilisateur. Une vue DB2 est utilisée pour accéder à ces lignes associées à l'identifiant.

Pour plus de souplesse, chaque utilisateur peut avoir plusieurs identifiants. Les identifiants peuvent avoir n'importe quelle valeur à condition de gérer les autorisations primaires ou secondaires. **Slice** s'appuie sur les autorisations DB2 standard. Pour utiliser un identifiant différent, l'utilisateur doit avoir l'autorisation d'exécuter un « SET CURRENT SQLID » avec la valeur souhaitée. Les utilisateurs qui veulent télécharger des données avec **Slice/Compare** doivent avoir au moins l'autorisation « SELECT » pour chaque table à télécharger. Ils doivent avoir l'autorisation « INSERT » sur la vue pour charger un Slice. Toutes ces autorisations et modifications dans le catalogue sont gérées par l'administrateur et mise en place grâce au « POWER UTILITY ».

Le diagramme suivant illustre le concept de Slice :

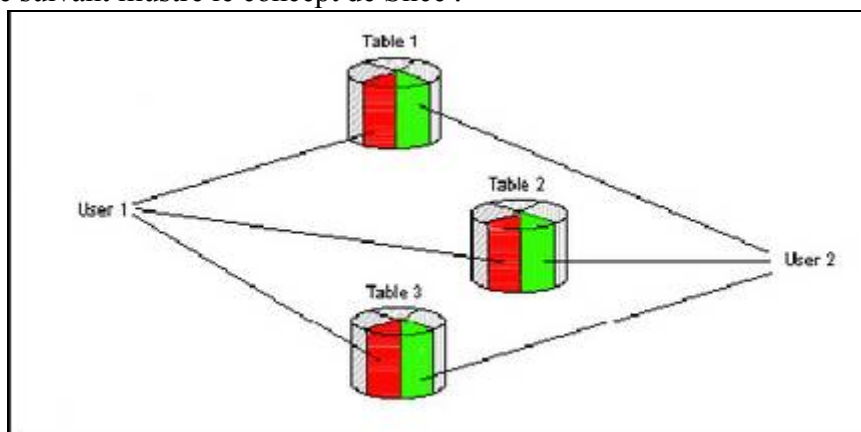


Figure 1 : Chaque utilisateur possède son propre jeu de données « Slice » au sein de la même base de données.

Ce diagramme montre les Slices (rouge et vert) utilisés par chacun des utilisateurs (user-1 et user-2). Ils sont indépendants l'un de l'autre. Les utilisateurs peuvent créer plusieurs versions de leur Slice, ayant ainsi la capacité de tester les images avant/après lors de comparaisons de données de test pour vérifier exactement ce qui a changé. Cette fonctionnalité est indispensable lors des tests de non régression.

Ci-après sont précisés des principes fonctionnels importants et pris en charge par l'outillage de Slice :

- Les versions de Slice, qui ont été téléchargées avec la solution **Slice**, peuvent être rechargées vers le même ensemble de tables DB2 ou bien des tables différentes ou encore des Slices différents.
- Quand une colonne change dans la base de données, **Slice** reconnaît la modification (exemple de « decimal » à « float »), formate et ajuste automatiquement les données de tests concernées.
- Si des colonnes ont changé de nom, l'utilisateur peut très facilement appliquer le changement en utilisant le panel fourni par le produit.
- Les versions de Slice sont rechargées en séquence pour se conformer à l'intégrité référentielle des tables cibles. La récursivité est supportée.
- Les modifications dans les intégrités relationnelles sont également prises en compte.

Les typologies de phases de test suivantes précisent les cas d'utilisations possibles :

Tests Unitaires : les utilisateurs peuvent partager ou isoler des données de test unitaires dans un ensemble unique de table DB2.

Ceci permet à chaque développeur/testeur de gérer un ou plusieurs slices avec leurs propres données de test. Chaque Slice est taillé pour répéter des tests pour un ou plusieurs programmes donnés. Les données peuvent aussi être copiées et partagées par les programmeurs. Pour vérifier le bon fonctionnement des programmes, cette fonction élimine ainsi des tâches ennuyeuses et répétitives de création et de recréation des cas de

test. Une copie des Slices peut être générée et gelée pour servir de référence, dans ce cas elle pourra être partagée mais non modifiable. Les tables chargées à partir d'un Slice sont elles modifiables.

Tests d'intégration: Les tests d'intégration ou plans de tests sont organisés et cadencés en phases ou en journées de test.

Chaque phase ou journée se déroule en séquence, la deuxième après la première, etc... Les tests d'intégration sont pour la plus part linéaires. Les tests de la première journée doivent être terminés pour commencer ceux de la journée suivante. Avec la solution **Slice**, les tests peuvent être réalisés de manière concurrente ou simultanée. Non seulement le premier jour de test peut être fait en même temps que le »deuxième«, mais ils peuvent être faits sur le même jeu de tables DB2 par un ou plusieurs utilisateurs.

Tests de non Régression : Le test de non régression est une méthode complexe pour la création et la maintenance des données de test, elle est donc peu utilisée.

Slice/Compare simplifie la maintenance et la vérification des étapes nécessaires aux tests de non régression. Un test de non régression vérifie que les changements faits dans un ensemble de programmes impactent **Uniquement** les données qui doivent être changées par les modifications apportées par les programmes. Pour vérifier que les modifications produisent les bons effets, les modifications de données sont comparées manuellement aux résultats escomptés.

Voici l'illustration de étapes typiques pour réaliser des tests de non régression :

1. Sauvegarder une version Slice non modifiable (Référentiel : Base line Slice)
2. Exécuter les tests avec la version non modifiée du programme
3. Sauvegarder le Slice version 1
4. Recharger la version Base line du slice
5. Re-exécuter les tests avec la version modifiée du programme
6. Sauvegarder le Slice version 2
7. Comparer Slice version 1 et 2 pour détecter les différences

Les étapes de la comparaison répondent aux questions suivantes :

1. Est-ce qu'un changement a eu lieu?
2. Si un changement s'est produit, est-il correct ?

Recette Usine : La MOE valide la recette fonctionnelle des produits, elle est responsable de la qualité technique de la solution. Elle doit, avant toute livraison au MOA, procéder aux vérifications nécessaires (« usine recette »).

Le personnel réalisant ces tâches trouvent très souvent des erreurs dans les programmes qui doivent être revus et corrigés par le développement. Les vérifications se font alors à l'aide des jeux d'essais utilisés précédemment par le développement et des jeux d'essais utilisés par la MOE. Il est donc nécessaire de pouvoir conserver l'historique des différentes versions des jeux d'essais. Or, ceci est très souvent compliqué voir impossible avec des organisations classiques.

Slice résout ce problème en donnant la possibilité de Sauvegarder/Restaurer les différentes versions des Slices



Conversion des données : La conversion des données peut poser des problèmes même avec des fichiers de tests préparés avec toutes les précautions nécessaires. Même un simple changement d'une colonne peut nécessiter des heures de travail manuel pour apporter des corrections. **Slice** automatise la plus grande partie du processus, en fournissant des conversions de données automatiques pour les changements suivants :

- Changement de longueur de colonne
- Changement de type de colonne
- Changement du nom de la colonne
- Changement dans l'intégrité référentielle