Élaboration d'un banc d'essai pour la comparaison de SGBDR de modèles différents

par

Aurélie Ottavi

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, avril 2012

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-88793-6

Our file Notre référence
ISBN: 978-0-494-88793-6

Le 10 avril 2012

le jury a accepté le mémoire de Madame Aurélie Ottavi
dans sa version finale.

Membres du jury

Professeur Luc Lavoie
Directeur de recherche
Département d'informatique

Professeure Bessam Abdulrazak
Membre
Département d'informatique

Professeur Marc Frappier
Président rapporteur
Département d'informatique

Sommaire

Ces dernières années, le domaine des bases de données a beaucoup évolué. De nombreux modèles ont émergé de part et d'autre : XML, ODBMS, Map Reduce... Ces modèles ont pour objectif de résoudre les problèmes existants du modèle relationnel. Et pourtant, aucun n'a réussi à détrôner SQL, la variante la plus connue du modèle relationnel. Cela révèle tout d'abord un problème vis-à-vis du modèle relationnel tel qu'implémenté par SQL qui ne s'adapte pas à toutes les applications. Cela révèle aussi l'incapacité des nouveaux modèles à pallier les lacunes de SQL tout en conservant ses forces : son expressivité et sa capacité à assurer l'intégrité des données. Ces caractéristiques restent primordiales pour de nombreuses applications et ne peuvent donc pas être ignorées. Notre intuition est donc qu'il ne faut pas totalement mettre de côté le modèle relationnel de Codd [10], mais plutôt l'améliorer. Étant donné que seul le langage relationnel SQL a su s'imposer, l'évaluation des SGBD relationnels s'est toujours faite en ne considérant que celui-ci. De plus, les bancs d'essai existants considèrent le facteur de l'efficience comme le plus important à évaluer, probablement car c'est le critère qui dépend davantage de l'implémentation du langage faite par les SGBD que des variantes dialectiques du langage. Nous nous sommes donc donnée pour objectif de définir un banc d'essai plus large permettant d'évaluer des modèles relationnels différents en prenant en compte des critères plus variés et tout aussi importants tels que la validité, la disponibilité, la sécurité et l'expressivité. Le banc d'essai ainsi défini a été implémenté pour deux SGBDR basés sur SQL : Oracle et PostgreSQL. L'implémentation pour des SGBDR basés sur des modèles différents étant laissée pour des travaux futurs. Les résultats obtenus nous ont tout d'abord permis de démontrer la pertinence de notre banc d'essai. Malgré le fait que les deux SGBD soient basés sur le langage SQL, le banc d'essai a révélé des différences pour la majorité des critères définis. Ainsi, Oracle se démarque en ce qui concerne la validité des données. Contrairement à Postgresql, il ne présente pas de lacunes pouvant compromettre l'intégrité des données. Oracle se démarque aussi en ce qui

concerne la sécurité. Les mécanismes présents y sont plus complets et plus flexibles. PostgreSQL, lui, se démarque par rapport à l'expressivité du dialecte SQL utilisé. Des fonctionnalités primordiales sont présentes tandis qu'elles sont absentes sous Oracle. Au niveau de l'efficience, nos résultats révèlent une différence quant à la gestion des optimisations, car selon les fonctionnalités testées, le SGBD le plus efficient est différent. Cependant, aucun des SGBD ne se démarque. Pour finir, en ce qui concerne le critère de disponibilité des données, les résultats sont identiques pour les deux SGBD. Cette identification des lacunes propres à ces deux SGBD nous permettra par la suite de tirer des enseignements de ces problèmes de manière à ne pas les reproduire pour notre nouveau modèle.

Mots clés : SGBD, banc d'essai, modèle relationnel, efficience, expressivité, sécurité, transactionnel, validité

Remerciements

En préambule à ce mémoire, je souhaite adresser ici tous mes remerciements aux personnes qui m'ont apporté leur aide et qui ont ainsi contribué à l'élaboration de ce mémoire.

Tout d'abord, je remercie Luc Lavoie, mon directeur de recherche, pour tout ce qu'il a pu m'apprendre sur le domaine, sur la méthodologie et la rigueur propre à un travail de recherche. Je le remercie aussi d'avoir été un directeur d'une grande gentillesse, toujours disponible et d'un grand soutien.

Je remercie ensuite mon collègue et ami Zouhir Abbouadaoui avec qui il fut un plaisir de partager cette première expérience de la recherche et qui m'a aidée à persévérer dans les moments de doute.

J'exprime ma gratitude à Mario Paquet et à Jean Goulet de m'avoir aidée à obtenir et à mettre en place le matériel dont j'avais besoin malgré les conditions difficiles dues à la grève.

Je remercie aussi Christina Khnaisser, Leonardo Coll, Pierre-Yves Nivollet et Mohamed Ali Ayed qui dans le cadre d'un de leurs cours ont accepté de travailler sur un projet utile à la réalisation de mon mémoire.

Enfin, j'adresse mes plus sincères remerciements à tous mes proches et amis qui m'ont soutenue et encouragée tout au long de la réalisation de ce mémoire et tout particulièrement Arnaud.

Table des matières

Liste des abréviations

BD	Base de Données
DBA	Administrateur de base de données (« *DataBase Administrator* »)
SGBD	Système de Gestion de Base de Données
SGBDR	Système de Gestion de Base de Données Relationnelle
SQL	« *Structured Query Langage* »
UCT	Unité centrale de traitement (« *CPU* »)

Liste des tableaux

Liste des figures

Introduction

Contexte

SQL, le langage le plus utilisé dans le domaine des bases de données est fondé sur le modèle relationnel. C'est ce qui a fait très vite son succès, le modèle relationnel apportant énormément d'avantages notamment au niveau de l'expressivité et de l'affranchissement de la représentation physique. Cependant, la communauté d'utilisateurs qui utilisait ce langage pour des applications concrètes a soulevé plusieurs problèmes. Par exemple, il était impossible de stocker des tuples dont l'un des attributs était inconnu au moment de l'insertion. Le langage SQL a donc été amené à évoluer au fur et à mesure que des problèmes étaient soulevés par cette communauté. Cette évolution s'est faite de manière hâtive en ajoutant des couches par-dessus un modèle initial émergent qui n'était pas tout à fait maîtrisé, le tout en étant confronté à la pression des éditeurs de SGBDR qui doivent faire face à plusieurs contraintes (rétrocompatibilité, maintien de parts de marché, etc.). La conséquence inévitable de cette évolution est un langage dont la norme trop volumineuse est difficile à maîtriser et comportant des incohérences. Si on reprend l'exemple du problème de la modélisation des informations manquantes, SQL propose de le résoudre à l'aide du marqueur NULL. La gestion du NULL, telle que faite par SQL, présente toutefois un comportement incohérent dans certains cas particuliers [12].

Bien qu'étant le langage le plus largement utilisé dans le monde des bases de données, SQL n'est donc pas sans comporter quelques lacunes.

Des langages basés sur d'autres modèles commencent à voir le jour tel que les modèles orientés objet ou le modèle XML. Ces modèles semblent avoir du mal à se faire une place sur le marché. Peut-être en raison du fait qu'ils excluent totalement le modèle relationnel qui malgré ses défauts a tout de même fait ses preuves.

C'est dans ce contexte que le projet DOMINUS a vu le jour. Le projet consiste à reprendre les bases du modèle relationnel puis à l'étendre de manière réfléchie pour résoudre les problèmes existants de SQL, dont les NULL, les doublons, l'ordonnancement des tuples dans la relation, l'ordonnancement des attributs dans les tuples. On désire appuyer le modèle relationnel sur un système de typage fort, flexible et adapté aux problèmes du XXI^{e} siècle. Il s'ensuit la proposition d'un modèle relationnel fondé sur une solide théorie du typage conforme à l'approche de Cardelli [8] et intégrant les bases de la vérification axiomatique sous forme de contrats mis de l'avant par Meyer [34]. Un langage issu de ce modèle, Discipulus, est en cours d'implantation.

Objectifs

Dans le cadre du projet DOMINUS, nous voulons pouvoir comparer les SGBDR mettant en œuvre Discipulus avec les SGBDR existants (tels que Oracle, PostgreSQL, MySQL, DB2, Rel, Microsoft Office Access...) de manière à valider la pertinence des résultats obtenus. Or, à ce jour, il n'existe pas de bancs d'essai nous permettant de comparer différents SGBDR entre eux s'ils sont basés sur un autre langage que SQL (ce qui est le cas de Rel qui implantent le langage Tutorial D et de Microsoft Office Access qui implémente son propre langage graphique). L'objectif premier de ce mémoire est donc de développer un banc d'essai permettant de comparer des SGBDR de modèles et de langages différents.

En plus de ce contexte, le banc d'essai ainsi développé pourra aussi servir à comparer des SGBDR commerciaux en regard de critères plus larges, reflétant mieux la problématique des applications actuelles de bases de données que ceux des suites de tests existantes. C'est le deuxième objectif de ce mémoire.

Méthodologie

Nous avons tout d'abord réalisé une première phase de recherche bibliographique qui nous a permis d'établir les critères à retenir pour évaluer un SGBD ainsi que les critères important pour établir un bon banc d'essai. À partir de ces critères, nous avons ensuite analysé des bancs d'essai existants permettant de comparer les différentes variantes de SQL. Cette

analyse nous a permis de nous rendre compte de ce qui est déjà fait et des problèmes qui restent à résoudre. Cela nous a amenées à proposer des solutions aux problèmes identifiés.

Une fois ces différentes phases terminées, nous sommes passées à la mise en œuvre d'un banc d'essai conforme intégrant l'essentiel de nos propositions. Cette mise en œuvre a été découpée en trois étapes. Tout d'abord, nous avons établi la spécification des essais et des tests qui les composent. Une fois les essais bien définis, nous avons été en mesure de définir nos besoins de soutien à l'exécution de ceux-ci et d'en dégager la spécification d'un outil d'automatisation des essais. Nous avons ensuite développé cet outil qui nous a permis d'exécuter facilement et systématiquement un grand nombre des tests spécifiés.

Pour terminer, nous avons procédé aux essais eux-mêmes sur un petit échantillon de SGBDR et fait l'analyse des résultats dans le but de démontrer la pertinence et l'utilité de notre travail.

Résultats

Nos résultats nous ont avant tout permis de démontrer la faisabilité de notre banc d'essai. Ils ne révèlent cependant pas toute la puissance du banc d'essai développé, car les SGBDR que nous avons testés sont basés sur des dialectes SQL, une même variante du modèle relationnel.

Certains points restent à résoudre pour pouvoir implémenter le banc d'essai pour des SGBDR basés sur d'autres modèles. Le plus important étant l'implémentation d'une interface de programmation (un pilote JDBC, par exemple) sans quoi l'utilisation de l'outil d'automatisation des essais ne pourra se faire.

De plus, il reste des améliorations à faire pour avoir un banc d'essai plus complet et plus objectif.

Nous avons exécuté notre banc d'essai sur les SGBD Oracle 10g Enterprise Edition (Release 10.2.0.1.0) et PostgreSQL 9.1.0. Cela nous a permis de révéler qu'Oracle est meilleur sur la plupart des points, mais possède tout de même quelques lacunes gênantes. Cependant PostgreSQL se défend bien compte tenu de sa gratuité.

Structure du mémoire

Le document est structuré de la manière suivante :

- Le Chapitre 1 est consacré à la définition des critères d'un bon SGBDR et d'une bonne suite d'essais. Une critique de bancs d'essai existants est ensuite effectuée sur la base de ces critères.
- Le Chapitre 2 présente les problèmes à résoudre pour mener à bien notre travail ainsi que des pistes de solution.
- Le Chapitre 3 spécifie les stratégies adoptées pour évaluer chaque critère d'un bon SGBDR.
- Le Chapitre 4 décrit l'outil d'automatisation des essais mis au point dans le but d'exécuter à moindre effort les essais.
- Le Chapitre 5 explicite les résultats obtenus lors de la mise en œuvre du banc d'essai pour les SGBD Oracle 10g Enterprise Edition (Release 10.2.0.1.0) et PostgreSQL 9.1.0.
- L'Annexe A présente un ensemble de modèles de sécurité existants.
- L'Annexe B présente différents types de conflits transactionnels possibles.
- L'Annexe C présente la liste des fonctionnalités exprimables adaptées à un langage relationnel.
- L'Annexe D présente l'adaptation à des langages relationnels du calcul de métriques de complexité retenues.
- L'Annexe E présente le manuel de référence de l'outil d'automatisation implémenté pour faciliter l'exécution de notre suite de tests.
- L'Annexe F présente les résultats d'efficience obtenus lors de l'exécution sur le SGBD Oracle.
- L'Annexe G présente les résultats d'efficience obtenus lors de l'exécution sur le SGBD PostgreSQL.
- L'Annexe H présente les points qui n'ont pas été traités dans le cadre du mémoire à chaque étape de la mise en œuvre.
- L'Annexe I présente une liste de définitions utiles à la compréhension du mémoire.

Chapitre 1

État de l'art

1.1 Objectifs historiques des SGBD

James P. Fry et H. Sibley [17] proposent un état de l'art des objectifs originels de la mise en place des SGBD. La notion de SGBD est apparue pour résoudre les problèmes liés à la centralisation des données. Au départ, la gestion des données n'était pas centralisée. Chaque application devait s'occuper de cette gestion selon ses besoins. Ceci entraînait inévitablement une redondance et des risques d'incohérence, car plusieurs applications peuvent avoir à effectuer le même traitement sur les données, il fallait alors que chacune d'entre elles fournisse un moyen pour le faire. L'idée consistait donc à centraliser le traitement des données pour un gain de performance et de coût. McGee est le premier à proposer des idées qui correspondent à cette définition. Son système permet de créer des routines capables de trier des fichiers en fonction de leur contenu. Le but originel qui a motivé la création de SGBD est donc l'optimisation du coût et du temps nécessaire pour le traitement de grands volumes de données accessibles (éventuellement concurremment) par plusieurs applications.

D'autres objectifs très importants découlent des problèmes engendrés par la centralisation du stockage des données et de leur traitement :

L'accès aux données

La centralisation des données pose un problème quant à l'accessibilité des données. Les données centralisées sont partagées entre plusieurs applications. Or pour que chaque application puisse avoir accès aux données, il faut que leur description puisse être comprise

par toutes les applications. Cette description doit donc être indépendante du support physique sur lequel sont stockées les données.

L'article ne le mentionne pas, mais la centralisation des données peut engendrer d'autres problèmes pour garantir l'accès aux données en tout temps. Tout d'abord, il est possible que l'on ait deux applications différentes qui désirent accéder à une même donnée au même moment. L'application doit alors pouvoir accéder aux données de la même manière que si les données n'étaient pas partagées entre plusieurs applications. De plus, puisque les données et l'application ne sont pas forcément sur le même support physique, il est possible que l'endroit où sont stockées les données soit défaillant. Le système doit alors parvenir à rétablir l'accès aux données dans un temps acceptable pour minimiser les conséquences.

La qualité des données et l'intégrité du système

Le terme qualité des données désigne ici ce que subséquemment la littérature scientifique a plutôt désigné comme l'intégrité des données. Les données peuvent être altérées pour plusieurs raisons possibles : une erreur humaine, une défaillance de la machine, un conflit d'accès aux données entre plusieurs applications. La redondance du traitement des données implique un plus grand risque de compromettre l'intégrité des données. De plus, comme l'accès aux données est centralisé, une application quelconque a moins de contrôle sur les données. Le SGBD a une vision globale du traitement des données, il est donc logique de lui déléguer la tâche d'assurer l'intégrité et la cohérence des données. Le SGBD doit donc fournir des moyens de détecter les erreurs, voire de les corriger. Ce n'est pas mentionné dans l'article, mais d'un point de vue externe, on peut ajouter que le SGBD doit minimiser, voire empêcher, la corruption des données. Pour cela, on peut s'assurer que les données peuvent être modifiées uniquement par des personnes dignes de confiance. On peut aussi définir des règles qui permettent de déterminer si la donnée est considérée comme intègre ou non et refuser son insertion dans la base lorsqu'elle ne l'est pas.

Sécurité et confidentialité

Étant donné que les données sont centralisées, les risques d'accès non autorisé à des données sensibles sont augmentés. En effet, bien que les données soient centralisées, il est possible

que certaines d'entre elles ne doivent pas être accessibles à toutes les applications qui se connectent à la BD pour des raisons de confidentialité. Encore une fois, puisque le SGBD a une vision globale des données qu'il permet de gérer, il est normal que ce soit lui qui s'occupe d'en assurer les droits d'accès.

L'indépendance des données

Les structures logiques et physiques doivent être séparées pour assurer l'indépendance des données. L'utilisateur ne s'occupe alors pas de la structure physique de la base qu'il ne connaît pas. Cela permet à l'utilisateur d'utiliser simplement les données et aussi qu'il soit indépendant de l'implémentation choisit pour la base. Si l'administrateur change la structure physique de la base, les programmes qui utilisent la base ne devront pas être modifiés. Cette séparation permet aussi de protéger les données, la définition de la base étant commune à plusieurs programmes, plusieurs programmeurs, celle-ci ne peut être modifiée par l'un d'entre eux sans l'accord des autres.

Ce modèle bischématique peut aussi être étendu à un modèle trischématique qui rajoute un niveau d'indépendance de manière à ce que les utilisateurs ne voient que la structure logique des données dont ils ont besoin. Cet objectif fut confirmé par l'apparition du modèle relationnel qui permet d'obtenir cette indépendance. L'utilisateur doit pouvoir spécifier ce qu'il veut et non la manière dont il le veut [2].

Il est à noter que le concept de centralisation présenté dans cet article a quelque peu évolué depuis. De nos jours, on continue de vouloir centraliser la gestion des données, mais le stockage des données lui-même n'est pas toujours centralisé (les données sont très souvent réparties ou distribuées sur différents supports physiques). En fait, la centralisation des données est virtuelle, le SGBD sert d'intermédiaire entre les données et les applications pour que ces dernières puissent manipuler les données en faisant abstraction de leur emplacement physique.

1.2 Caractéristiques définissant un bon SGBD

De cette analyse des objectifs, on peut en déduire les caractéristiques par rapport auxquelles nous comparerons les différents SGBDR. Dans un premier temps, nous nous concentrons sur les caractéristiques de tout bon SGBD, les caractéristiques spécifiquement relationnelles seront traitées par la suite.

1.2.1 Efficience

L'efficience est un des objectifs originels de la mise en place de SGBD. Ce critère est resté primordial, car, malgré l'amélioration incessante des capacités de traitement, il y a toujours plus de données à traiter, notamment en ce qui concerne le web.

L'efficience est traditionnellement exprimée en termes de temps de réponse du système. C'est-à-dire le temps écoulé entre le moment où le programme fait appel au SGBD avec une requête caractéristique et le moment où le résultat de la requête est obtenu par le programme [5]. Plus ce temps est faible, plus le SGBD est efficient. Cependant, on peut aussi effectuer les mesures d'efficience en termes de mémoire utilisée. La mémoire est une ressource consommée au même titre que le temps, il convient donc de vérifier que son usage est minimisé. De plus, cela nous permet de relativiser les mesures en termes de temps, car elles sont dépendantes. Par exemple, la pagination des systèmes d'exploitation peut avoir une influence sur le temps de réponse et sur la faisabilité pratique de certains types de requêtes en l'absence de traitement massivement parallèle. Mais ce n'est pas le SGBD qui est en cause, on ne veut donc pas que cela soit pris en compte.

Plusieurs paramètres peuvent avoir une influence sur la mesure d'efficience en terme de temps ou de mémoire. C'est d'ailleurs en jouant sur le choix de leurs valeurs que les vendeurs de SGBD s'appuient souvent pour affirmer que leur système est le meilleur [7]. Il nous faut donc bien identifier ces paramètres pour pouvoir interpréter les résultats de manière cohérente. Nous pourrons ensuite choisir de fixer ou de faire varier ces paramètres pour tester l'efficience dans différentes conditions et voir les implications réelles.

Voici les paramètres influents identifiés [5, 36, 45] :

- Matériel :
  - vitesse du processeur,
  - nombres de processeurs,
  - quantité de mémoire primaire,
  - quantité de mémoire secondaire,
  - type de mémoire (temps d'accès différencié en lecture et en écriture, taille de la plus petite unité adressable, rémanence...)
- Système d'exploitation :
  - type de système d'exploitation,
  - occupation du système d'exploitation à d'autres tâches
- Caractéristiques des données de la base :
  - volume des données présentes dans la base,
  - type des données présentes dans la base,
  - taille des objets manipulés,
  - taille de chaque tuple (ou taille des résultats),
  - structure des données (schémas de la BD)
  - utilisation d'index,
  - types d'index
- Application utilisant les données :
  - type de l'application : « *read only* », « *read mostly* », « *read write* », « *write mostly* » ou « *write only* »
  - répartition de l'accès aux données (pourcentage de données auxquelles l'application accède par rapport au volume de donnée total présent dans la base de données),
  - facteur de sélectivité (combien de résultats ont été retournés par la requête),
  - taille des tampons mémoires
- Communication réseau :
  - nombre de connexions simultanées à la base de données,
  - débit nominal des connexions,

- encombrement du réseau,
- latence du réseau,
- protocole de communication

1.2.2 Coût

Nous ne prendrons pas en compte le critère de coût qui était pourtant défini comme étant un des objectifs primordiaux des SGBD et qui est très important dans la pratique, car il va permettre à une entreprise de décider si elle a les moyens de mettre en place le SGBD et de l'entretenir. Cependant, les coûts sont liés à l'efficience. En effet, il est possible de calculer les coûts en fonction de trois catégories de paramètres : l'efficience, les fonctionnalités offertes, la valeur marchande. Le problème de ce critère est donc qu'il évolue sans cesse en fonction de la conjoncture économique qui dicte la valeur marchande. On ne peut donc pas établir un coût hors du temps, il faut refaire l'évaluation au moment de la prise de décision. Dans le cadre de ce mémoire, nous nous contenterons donc d'évaluer l'efficience en fonction des fonctionnalités offertes[1]. Chacun pourra ensuite analyser les coûts qui en découlent.

1.2.3 Validité des données

Ce critère prend sa source dans l'objectif de qualité des données et d'intégrité du système. C'est la mesure de la conformité des données en regard de ce qu'elles représentent réellement. L'adéquation requiert deux types de conformité : la conformité de représentation et la conformité d'interprétation (l'interprétation étant le résultat de l'exécution d'une fonction).

Validité de représentation du monde réel

Pour cela, la donnée doit tout d'abord être conforme à ce qu'elle représente dans le monde réel. Dans l'idéal, à n'importe quel instant dans le temps, les données enregistrées dans la base doivent représenter la réalité. C'est donc les changements dans la réalité qui ont un impact direct, instantané sur la validité de la base de données. Or cela n'est pas vérifiable, car

---

[1]En pratique, les fonctionnalités seront regroupées en termes d'applications typiques ciblées.

les modifications dépendent de l'extérieur du SGBD. Nous ne pouvons donc pas assurer si les données que l'on manipule sont bien valides et conformes relativement au monde réel.

Validité du langage

D'un autre côté, il faut s'assurer de la conformité des données en regard des manipulations prescrites par le langage de manipulation des données. On s'assure ainsi que si la donnée entre dans le SGBD, elle ne sera pas corrompue par une mauvaise exécution/définition des opérations du langage.

Pour cela, il faut tout d'abord s'assurer que les opérations exprimables grâce au langage sont toujours définies correctement par rapport à leur modèle mathématique. Il faut aussi s'assurer que le compilateur est capable de traduire un langage en langage machine sans introduire d'erreurs. Le mieux pour vérifier cela est de prouver de manière formelle l'exactitude du compilateur.

1.2.4 Disponibilité

Ce critère est lié à l'objectif d'accessibilité des données. C'est la capacité selon laquelle les données sont accessibles même en cas de conflit d'accès (plusieurs accès concurrents à la même donnée), d'erreur ou de défaillance, tout en continuant d'assurer leur intégrité. Nous nous placerons dans le contexte transactionnel, car les mécanismes qui permettent d'assurer la disponibilité en cas d'accès concurrent sont des mécanismes transactionnels. De plus, puisque toute requête fait implicitement partie d'une transaction (même si elle est seule), si la disponibilité est vérifiée dans le cadre de transaction elle l'est donc aussi dans le cadre d'une requête simple.

La disponibilité des données doit être évaluée pour chacun des contextes suivants :

- **Contexte normal** : dans le cadre d'une utilisation normale, c'est-à-dire sans conflits, pannes ou anomalies, il faut s'assurer que les données sont accessibles en tout temps et que leur manipulation ne dégrade pas leur intégrité.
- **Contexte avec conflit** : lorsque plusieurs utilisateurs accèdent à une même ressource au même moment, l'intégrité des données peut être menacée si au moins

un accès a pour effet de modifier une donnée (voir l'annexe B pour une présentation détaillée des différents types de conflits transactionnels).

- **Contexte d'une erreur logicielle interne** : Si le SGBD plante à cause d'une erreur provoquée par lui-même.
- **Contexte d'une erreur logicielle externe** : Si un programme externe envoie des données erronées au SGBD et le fait planter.
- **Contexte d'une défaillance matérielle interne** : Si, par exemple, le disque sur lequel les données sont stockées est endommagé.
- **Contexte d'une défaillance matérielle externe :** Si par exemple la carte réseau de l'ordinateur qui envoie les données est défaillante et donc que les données envoyées sont corrompues.

Dans chacun de ses contextes, le système doit assurer deux choses :

- la tolérance aux pannes qui permet de continuer à assurer le service en cas de problème sans risque de corrompre l'intégrité des données ;
- la robustesse qui assure la possibilité de restaurer les données à un point dans le temps au cas où le service ne peut plus être assuré.

En plus de cela, il est important de s'assurer que le temps de gestion des conflits/erreurs/défaillances n'est pas trop élevé, car les utilisateurs du système n'ont plus accès aux données concernées pendant ce temps-là.

1.2.5 Expressivité

L'expressivité désigne la facilité avec laquelle l'utilisateur peut exprimer ce qu'il veut faire, mais aussi la facilité pour un autre utilisateur de comprendre cette expression. Cette caractéristique est liée à l'objectif d'indépendance des données. En effet, une des conséquences de fournir un haut niveau d'abstraction pour que l'utilisateur puisse accéder aux données est que le langage sera plus aisé à comprendre et à manipuler.

Il est important de distinguer la compréhensibilité du langage qui fait que le langage est simple à manipuler pour l'utilisateur et la compréhensibilité des programmes écrits dans ce langage qui assure que suffisamment d'informations sont données aux lecteurs du programme pour comprendre facilement les intentions du programmeur. Les deux sont très importants et ne sont pas forcément liés. Par exemple, le langage APL possède une syntaxe très simple,

mais il est réputé pour être particulièrement difficile à comprendre, car il utilise une notation propre, apparentée, mais différente de celle des mathématiques, ce qui rend les programmes très concis, mais sujets à de subtiles difficultés d'interprétation. On appelle cela un « *write-only language* » [32].

Pour qu'un langage soit simple à manipuler et à comprendre, il faut :

- **Une couverture fonctionnelle.** C'est-à-dire que le langage fournit le moyen d'exprimer l'ensemble des fonctionnalités pertinentes, que la sémantique soit complète et assez flexible pour répondre aux besoins de l'utilisateur (aussi variés qu'ils soient). C'est le critère le plus important, même si la façon dont on va ensuite exprimer ces fonctionnalités est aussi importante. Plus le langage offre de fonctionnalités pertinentes, plus il sera facile d'exprimer ce que l'on veut faire.
- **Une syntaxe simple et concise.** Selon les langages, on peut exprimer une même chose en plus ou moins de lignes de code. Or plus le code est long, plus il est difficile de s'y retrouver. On perd donc en lisibilité.
- **Un haut niveau d'abstraction.** C'est-à-dire s'éloigner le plus possible du langage machine constitué de 0 et de 1 pour se rapprocher d'un langage plus compréhensible pour un être humain. Il peut être caractérisé par trois caractéristiques du langage : le modèle de données, le modèle de typage et le modèle algorithmique.
- **Un vocabulaire expressif.** C'est la manière de choisir les mots clés ou l'interface graphique du langage. Il faut que ces derniers soient évocateurs pour l'utilisateur. Pour cela, il faut que le langage soit basé sur des codes connus de tous et qu'il n'y ait pas d'ambiguïté possible.
- **Une redondance adéquate.** Il ne faut pas que la redondance soit trop grande pour simplifier l'utilisation du langage qui sera alors plus facile à apprendre et portera moins à confusion. D'un autre côté, il faut tout de même une certaine redondance pour vérifier que le programmeur écrit des programmes cohérents [57]. En effet, si l'on n'impose pas une certaine redondance, il suffit que le programmeur se trompe une fois dans l'expression de ses intentions pour que le programme soit faux, alors que si le langage impose au programmeur de la redondance, c'est-à-dire d'exprimer plusieurs fois, mais de manière différente, ce qu'il veut, alors il est plus difficile pour lui de se tromper. S'il se trompe à un endroit, alors il sera aisé de le savoir, car les deux expressions ne seront pas compatibles. Par exemple, si l'utilisateur déclare une variable de type **Int** et qu'il l'utilise ensuite à un endroit où seul un string peut être utilisé, on peut déterminer que le programme comporte une

erreur. Si la déclaration de type n'était pas nécessaire, on n'aurait pas pu détecter l'erreur. Une certaine redondance permet donc d'améliorer la fiabilité.

1.2.6 Sécurité

C'est la capacité à prévenir ou à faire face à une utilisation non autorisée de la base de données. Dans un SGBD, un ensemble de mécanismes permet de gérer la sécurité aux données. Il faut tout d'abord un mécanisme d'authentification pour pouvoir identifier l'utilisateur qui veut accéder à la base. Une fois l'entité identifiée, le mécanisme de gestion des autorisations permet de vérifier les droits qui lui sont attribués pour savoir quelles sont les actions qu'elle peut exécuter. Finalement, un mécanisme d'audit doit permettre d'assurer la traçabilité de la sécurité des données, en particulier au cas où la gestion des autorisations ne fonctionnerait pas bien ou serait contournée. On doit être en mesure de garder une trace des accès et des opérations qui ont été effectuées afin de pouvoir identifier l'auteur de l'attaque, mais aussi pour pouvoir restaurer l'intégrité de la base en défaisant les modifications non autorisées qui auraient pu être effectuées.

Pour terminer, un dernier point est essentiel afin de pouvoir assurer la sécurité : l'étanchéité de l'accès à ces trois mécanismes de sécurité. En effet, si un individu mal intentionné peut accéder à la structure physique où sont situés ces mécanismes, il pourra s'introduire dans le système sans mal et, vraisemblablement, sans laisser de trace.

On peut se demander si la gestion de l'authentification est bien le rôle du SGBD. L'omettre et se reposer sur un système d'authentification fourni par le système d'exploitation permet de réduire la mise en œuvre du SGBD de façon significative. Or, l'authentification peut très bien être définie de manière indépendante de la structure de la BD. Il est donc acceptable de soulager le SGBD de cette tâche. Cependant, dans tous les cas, le SGBD se doit de vérifier qu'un mécanisme d'authentification existe, sans quoi la sécurisation du système ne pourra pas fonctionner.

Au contraire, au niveau de la gestion des autorisations, il est logique que ce soit le rôle du SGBD, car l'on a besoin du schéma de la BD pour définir des droits sur les objets présents

dans le SGBD. De même, la gestion de l'audit doit aussi être intégrée au SGBD, car elle se sert du schéma de la BD pour garder une trace des opérations et recouvrer des données.

Pour que les mécanismes de contrôle d'accès présentés ci-dessus ne soient pas inutiles, il faut que le niveau d'abstraction auquel se trouvent ces derniers ne soit pas accessible directement (de même que les sous-niveaux). Ainsi, on ne peut pas contourner le système de sécurité en accédant directement à la structure physique des données sans avoir auparavant eu besoin de s'authentifier. Pour évaluer ce critère, il faut donc identifier le niveau d'accès le plus bas et intégrer des tests l'utilisant.

1.3 Caractéristiques définissant un bon banc d'essai

Avant d'analyser des bancs d'essai existant, nous avons défini les critères qui nous permettront de justifier la qualité d'un banc d'essai.

1.3.1 Objectivité

C'est la mesure dans laquelle on donne une image non déformée des choses, ou de ce qui les décrit et les juge, sans parti pris. De façon opératoire, l'objectivité est définie comme le degré de concordance entre plusieurs observations.

L'objectivité est une caractéristique primordiale que devra avoir notre banc d'essai. Notre but est de pouvoir comparer différents SGBD, pour que les résultats soient interprétables, il faut donc le faire en toute impartialité et en faisant attention de ne pas mettre en avant un SGBD particulier. Les résultats doivent être indépendants et incontestables.

Pour avoir un banc d'essai objectif, il nous faudra tout d'abord démontrer notre bonne volonté. Cela n'est cependant pas suffisant, car il est possible d'introduire un biais sans le vouloir. Nous utiliserons les caractéristiques présentées par la suite pour nous assurer d'une certaine objectivité.

1.3.2 Complétude

C'est la mesure dans laquelle un test ou une suite de test traite toutes les exigences définies pour un système donné ou un composant [23].

La complétude nous permettra d'assurer l'objectivité de nos résultats. Une couverture incomplète peut entraîner un biais dans les résultats, car les portions omises pourraient révéler les faiblesses d'un SGBD. Il est cependant impossible de prétendre à une couverture totale, car les cas de test seraient trop nombreux.

Pour s'assurer d'une couverture suffisante, il nous faudra tout d'abord proposer au moins une stratégie pour l'évaluation de chacun des critères d'évaluation d'un SGBD définis précédemment. Pour chacune de ces stratégies, il nous faudra ensuite isoler au mieux les cas de test représentatifs et motiver les cas de test non pris en compte.

1.3.3 Traçabilité

C'est la mesure dans laquelle chaque élément d'un projet de développement logiciel décrit et motive sa raison d'être [23].

La traçabilité permet tout d'abord la transparence de notre banc d'essai ce qui renforce encore une fois le critère d'objectivité. En plus de cela, elle nous sera utile pour faire évoluer facilement le banc d'essai.

Pour s'assurer d'une bonne traçabilité, la documentation qui accompagne notre suite d'essai doit être complète. Pour cela, chaque cas de test, test, essai, stratégie, choix d'implémentation doit être décrit, expliqué et motivé. La documentation doit aussi être claire, compréhensible et ne pas laisser libre cours à l'interprétation.

1.3.4 Représentativité

C'est la mesure dans laquelle les items du test échantillonnent suffisamment l'essentiel du contenu (au sens large) à mesurer [41].

Dans notre cas, il est important de mettre en avant les résultats représentatifs d'application réelle et non pas de mettre sur le même plan tous les résultats obtenus alors qu'en pratique, certaines fonctionnalités du SGBD sont rarement utilisées. Il faut tester ces fonctionnalités pour respecter le principe de complétude, mais leur impact devra être minimisé.

Pour satisfaire ce principe, nous utiliserons des jeux de données issus d'applications réelles pour compléter nos tests.

1.3.5 Reproductibilité

C'est la mesure dans laquelle le même résultat est produit à chaque fois que le test est effectué [23].

Encore une fois, cela est très lié à l'objectivité. Si les tests sont reproductibles, cela veut dire que l'on maîtrise bien les conditions d'exécution. Les résultats obtenus seront alors fiables et non fluctuants si on les exécute plusieurs fois.

Pour respecter ce principe, il nous faudra identifier les conditions qui peuvent faire varier les résultats et les fixer. Dans certains cas, il nous sera impossible de contrôler entièrement ces conditions. On pourra alors établir des mesures de stabilité pour s'assurer que la mesure est suffisamment fiable pour être interprétée.

1.3.6 Utilisabilité

Elle caractérise l'aisance avec laquelle un utilisateur peut apprendre à mettre en œuvre, préparer les jeux de données et interpréter les résultats d'un système ou d'un composant [23].

Cette caractéristique est importante dans notre cas, car pour valider l'impartialité de nos résultats, il faudra qu'ils soient reproduits dans des conditions similaires par d'autres personnes. On doit donc faciliter la reproduction des résultats pour qu'ils puissent être corroborés.

Plusieurs critères pourront nous permettre de valider l'utilisabilité d'un banc d'essai. Parmi eux on peut citer le temps d'obtention, la portion de tests automatisés, la facilité d'utilisation de l'outil d'automatisation

1.4 Analyse de suites de tests existantes

1.4.1 Présentation des différentes suites de tests analysées

Ensemble des suites de tests du TPC [48-50]

TPC (Transaction Processing Performance Council) est une organisation à but non lucratif fondée en 1988 dans le but de remédier à l'absence de références pour l'évaluation des SGBD qui prévalait alors. Au final, il a réussi à s'imposer comme un standard. Plusieurs suites de tests ont été mises au point par cette organisation. Les versions les plus récentes sont les suivantes :

- TPC-C permet l'évaluation d'un SGBDR transactionnel. Il se base sur une application représentant une entreprise quelconque qui doit gérer, vendre et distribuer un produit.
- TPC-E est une version plus récente pour l'évaluation d'un SGBDR transactionnel. Il se base sur une application réaliste d'un système d'information d'une entreprise de courtage. L'application est plus complexe que dans la version de TPC-C. Elle représente le système d'information d'une entreprise de courtage.
- TPC-H qui permet l'évaluation d'un SGBDR qui effectue des requêtes « *ad hoc* » complexes et possiblement concurrentes sur un large volume de données. L'application utilisée pour les tests a été choisie de manière à être représentative d'une quelconque application industrielle.

AS3AP [51]

AS3AP (« *Ansi SQL Standard Scalable and Portable* ») est une suite de tests conçue pour évaluer les SGBDR qui utilisent le langage SQL tel que défini par l'ANSI. Elle a été mise au point en 1991 par des chercheurs de l'Université de l'Illinois à Chicago.

OSDB [42]

OSDB (« *Open Source Database Benchmark* ») est une suite de tests mise au point en 2001 par une communauté indépendante de développeurs sur internet. Le projet est à code ouvert et en constante évolution. Chacun peut y apporter sa contribution. Son but est de fournir une suite de tests complète accessible à chacun, car les autres suites sont le plus souvent très dispendieuses. Elle est basée sur AS3AP, mais diffère en quelques points. Notamment, OSDB permet l'évaluation de différents SGBD relationnels basés sur un langage SQL qui ne serait pas complètement implémenté ou même sur un langage non SQL.

Suite de test SQL du NIST [39]

Elle a été mise au point en 1987 par le NIST (« *National Institute of Standards Technology* ») qui est une agence du ministère du Commerce des États-Unis. Cette suite de tests est accès sur la conformité du langage avec ses spécifications. La dernière version (v6) date de 1996 et permet de tester la conformité avec les standards suivants : ISO/IEC 9075:1992, ANSI X3.135-1992, FIPS 127-2, et X/Open XPG4 SQL.

Suites de tests du CIS [47]

CIS (« *Center for Internet Security* ») est une entreprise à but non lucratif qui s'occupe de développer des guides de sécurité qui regroupent les meilleures pratiques pour le déploiement, la configuration et l'utilisation de systèmes variés (système d'exploitation, SGBD, navigateurs internet, machine virtuelle...). Ces guides sont mis au point par le consensus d'une centaine d'experts en sécurité. En ce qui concerne les SGBD, CIS propose des essais pour évaluer les systèmes suivants : MySQL, MS SQL, Oracle, Sybase. Nous nous contenterons arbitrairement d'analyser celle d'Oracle pour avoir une idée de la stratégie utilisée.

Suite de tests intégrée à MySQL [58]

Elle est à code ouvert et évolue au fur et à mesure. Une version est fournie avec chaque nouvelle version de code du SGBD MySQL. Elle permet de tester l'efficience des différentes opérations de MySQL. Elle comporte aussi le scénario « *crash-me* » qui permet de comparer les fonctionnalités présentes dans les différents SGBD.

Suite de test intégrée à PostgreSQL [44]

Elle est à code ouvert et évolue au fur et à mesure. Une version est fournie avec chaque nouvelle version de code du SGBD PostgreSQL. Elle permet de tester l'efficience des différentes opérations de PostgreSQL. Par défaut, le scénario de test utilisé est inspiré de celui de TPC-B.

1.4.2 Critique globale de la qualité

De manière globale, plusieurs lacunes ont été recensées quant à la qualité des essais analysés.

Tout d'abord, on remarque l'absence d'état de la couverture des tests dans tous les essais analysés. Il est important d'identifier la couverture des tests, car cela nous donne une indication sur la qualité de l'essai et donc sur la confiance que l'on peut y accorder.

Pour ce qui est de la représentativité, seuls les bancs d'essai de TPC se basent sur un cas d'utilisation réel représentatif d'une catégorie d'application. Les autres suites de test utilisent des jeux de données sans signification ou bien très simplistes. Or, bien qu'il soit utile d'utiliser des jeux de données spécialement adaptés au banc d'essai (notamment pour assurer une couverture de test complète), il est important d'avoir aussi des jeux de données plus proches de la réalité d'utilisation du SGBD. Par exemple, si un opérateur est très rarement utilisé en pratique et qu'un test montre qu'il est mis en œuvre de façon peu efficiente, l'impact de ce résultat doit alors être minime.

Au niveau de la documentation, elle est bien souvent incomplète. Notamment pour les suites intégrées de MySQL et PostgreSQL, la documentation est pratiquement inexistante, il faut donc analyser le code pour comprendre les cas de tests. Cela rend difficile la compréhension de la pertinence des tests et corollaire l'appréciation de leur couverture.

Pour les essais intégrés à MySQL et PostgreSQL, le fait que les développeurs de la suite d'essais soient aussi des développeurs d'un SGBD amène une présomption de manque d'objectivité. Même avec de la bonne volonté, le fait qu'ils connaissent le fonctionnement de leurs SGBD risque de les amener à le mettre en avant.

Pour finir, un point sur lequel nous allons devoir apporter une grande contribution est l'absence de flexibilité par rapport aux modèles. Les essais analysés se limitent à la comparaison de SGBDR basés sur une implémentation de SQL. Or, nous rappelons que notre objectif est de comparer ces langages basés sur SQL à d'autres langages relationnels. Cela nous demandera donc une adaptation et une complétion des essais analysés. Nous devrons pour cela spécifier les essais à un plus haut niveau pour que tous les langages s'y retrouvent malgré les différences de modèle qui les séparent.

1.4.3 Critique par critère d'évaluation

1.4.3.1 Efficience

L'évaluation de l'efficience est la préoccupation principale des suites de test du marché. Pour l'évaluer, ils proposent d'exécuter un certain nombre de transactions représentatives du langage et de mesurer le temps que cela a pris. Le résultat est le plus souvent donné en nombre de transactions par unité de temps. Aucun des essais analysés ne propose de mesures de la mémoire utilisées, seules des mesures de temps sont utilisées, ce qui est moins fiable, car trop dépendant des conditions de test. En effet, si l'on mesure le temps total pour exécuter les transactions, il est possible que pendant cette période il y ait des moments où le processeur ne soit pas occupé à l'avancement de l'exécution de la transaction, mais à une autre tâche qui ne nous intéresse pas. Il serait donc plus judicieux de mesurer le temps de l'effort, c'est-à-dire uniquement le temps pendant lequel le processeur est occupé à la tâche du SGBD.

Presque tous les essais analysés proposent des tests pour ce critère (tous sauf ceux de CIS et du NIST). Cependant, ils sont plus ou moins bons. La différence entre chacun des essais provient notamment de la couverture des tests. Ainsi, le banc d'essai de PostgreSQL ne propose pas une bonne couverture des opérations qu'il est possible d'utiliser. Il permet de tester uniquement les opérateurs **SELECT**, **UPDATE**, **INSERT**, même si ce sont les plus utilisés en pratique, cela n'est pas suffisant pour atteindre une couverture de test adéquate. Il suffit qu'une autre opération moins utilisée soit considérablement moins efficiente pour que son

influence sur les résultats ne soit pas négligeable. Le banc d'essai OSDB est le plus complet en ce qui concerne la couverture des différentes fonctionnalités des langages testés. La suite de tests du NIST, plus axée sur la validité du langage est elle aussi intéressante en ce qui concerne la couverture des fonctionnalités du langage. Les bancs d'essai de TPC, eux, utilisent des transactions inspirées d'applications réelles. C'est une approche qui permet une couverture de tests plus proche de l'utilisation réelle des fonctionnalités.

1.4.3.2 Adéquation du langage

Parmi les bancs d'essai analysés, seul celui du NIST permet réellement de tester la conformité du langage par rapport à ses spécifications. Les autres sont surtout axés sur l'évaluation de l'efficience, mais pourraient être adaptés à l'évaluation de ce critère. En effet, pour tester la validité du langage, il faut tester toutes les opérations du langage et vérifier que les résultats obtenus sont les bons. On peut donc utiliser les requêtes proposées pour évaluer l'efficience des opérations du langage pour atteindre notre but. Surtout dans le cas où les tests d'efficience sont partitionnés par fonctionnalités.

L'essai du NIST est sans surprise le plus complet, il comporte environ 900 cas de tests pour couvrir l'ensemble des opérations des différents langages dérivés de SQL.

1.4.3.3 Disponibilité des données

Les bancs d'essai TPC-H, TPC-C et TPC-E sont les seuls qui proposent des tests permettant de s'assurer que la disponibilité des données est bien gérée.

Leur approche est de tester que les propriétés ACID sont bien respectées lors de l'exécution de transactions. Ces propriétés ont été définies par Jim Gray [18] pour caractériser la validité du mécanisme transactionnel.

Voici leur signification :

- **A pour atomicité** : Soit la transaction est effectuée dans son ensemble, soit aucune modification n'est faite sur la BD.

- **C pour cohérence :** Une transaction amène la BD d'un état cohérent vers un autre.
- **I pour isolation :** Plusieurs transactions ne doivent jamais interférer entre elles, ni même agir selon le fonctionnement de chacune.
- **D pour durable** : Si une transaction s'est terminée correctement, toutes les opérations de cette transaction doivent persister (être considérée comme ayant eu lieu) même si le système plante.

Cette approche permet une bonne couverture du critère. Il y a cependant quelques améliorations possibles. Tout d'abord pour ce qui est de la propriété d'isolation qui permet de tester un contexte avec conflit d'accès, TPC-C n'inclut pas les cas des erreurs d'écriture impropre et d'agrégation (voir Annexe B). En plus de cela, TPC-C ne mentionne pas les différents types de défaillance à tester pour la durabilité.

En plus de cela, des mesures du temps de récupération du système sont effectuées de manière à tester si le temps de non-disponibilité des données est tolérable (entre 1 et 3 secondes selon les transactions). Trois mesures sont suggérées par TPC : « *Database recovery* », « *Application recovery* », « *Business recovery* ». Soit le temps pour que les données redeviennent accessibles, puis deux mesures de temps pour que le système atteigne un certain niveau d'efficience.

Aucune explication n'est donnée sur la manière dont le temps tolérable de non-disponibilité des données est défini. Il ne sert à rien de donner un seuil pour évaluer ce temps, car il varie en fonction des exigences des applications utilisant la BD. En effet, si une application s'occupe de fournir des informations sur la santé des malades dans un hôpital, alors le temps acceptable de non-disponibilité des données sera très faible. D'un autre côté, si l'on est dans le cas d'une application d'achat en ligne de matériel informatique, il est plus tolérable que les données soient inaccessibles pendant vingt-quatre heures.

1.4.3.4 Expressivité

Aucun des bancs d'essai analysés ne prend en compte l'expressivité. Cela peut s'expliquer par le fait que tous les essais que nous avons trouvés se contentent de comparer des SGBDR

qui utilisent l'un des langages dérivés de SQL. Il n'y a donc pas de différences fondamentales d'expressivité entre les différents SGBD comparés.

1.4.3.5 Sécurité

Seul le banc d'essai de CIS prend en compte l'aspect sécurité. Cependant, il ne permet pas de comparer différents SGDB quant à leurs capacités à assurer une sécurité maximum si les besoins de l'application le nécessitent. CIS se concentre sur l'évaluation d'une politique mise en place avec un SGBD particulier. Ainsi, il propose une liste de vérification (qui dépend du SGBD et du système d'exploitation) pour vérifier que la politique mise en place par l'administrateur ne comporte pas de faille. Ce n'est pas une approche qui nous intéresse, car ce que nous voulons c'est comparer différents SGBD entre eux. Or, le banc d'essai de CIS évalue surtout une implémentation particulière de politique de sécurité. Pour ce faire, il évalue entre autres la sécurité au niveau de l'accès au système d'exploitation ainsi que la capacité de l'administrateur à implémenter une politique. Or, ce sont des choses indépendantes du SGBD que nous ne voulons pas prendre en compte.

Chapitre 2

Problématique

À partir de notre analyse des bancs d'essai existants, nous pouvons maintenant cerner l'envergure du travail à mener. Nous discutons dans ce chapitre des éléments que nous réutiliserons et proposons des solutions aux éléments manquants pour avoir une évaluation satisfaisante de chaque critère.

2.1 Évaluation du critère d'efficience

Dans un premier temps, nous nous baserons sur l'approche de TPC-C. Nous avons choisi d'utiliser TPC-C plutôt que TPC-E, car c'est une suite de tests plus éprouvée. En effet, TPC-C a été mis au point en 1991, mais n'a eu de cesse d'évoluer. C'est donc un banc d'essai éprouvé qui continue à être une référence. De plus, TPC-E étant trop récent, aucune implémentation à code ouvert n'existe pour l'instant. Il nous sera donc plus facile de reprendre la suite de tests si l'on peut se baser sur une implémentation déjà existante.

Le jeu de requêtes de TPC-C comporte des opérations relationnelles de base qui seront a priori communes à tous les langages que nous testerons. Nous aurons donc uniquement besoin de traduire celles-ci pour adapter le banc d'essai à des langages relationnels de natures différentes.

Nous réutiliserons aussi les bancs d'essai d'OSDB et du NIST pour nous permettre de spécifier les fonctionnalités des langages. Cependant, puisqu'ils sont basés tous deux sur SQL, nous les utiliserons uniquement pour vérifier qu'il n'y a pas de trous dans notre couverture de test. Pour spécifier les fonctionnalités qui devraient être communes à tous les

langages que nous voulons testés, nous nous baserons sur la définition du modèle relationnel tel que défini par Codd [10]. Pour couvrir la totalité des langages, il nous faudra aussi éplucher les documents de spécifications des langages relationnels existants pour ajouter les fonctionnalités spécifiques qu'ils ont pris la liberté d'ajouter au modèle relationnel.

Pour ce qui est des mesures, nous rappelons qu'une mesure de temps seule ne nous satisfait pas, car elle est trop dépendante de la plateforme de test et de paramètres incontrôlables tels que l'occupation de l'UCT à d'autres tâches que l'exécution du SGBD. Pour minimiser cela et pouvoir interpréter de manière plus précise les mesures de temps obtenues, voici les mesures complémentaires que nous avons choisies de prendre :

- la taille de l'ensemble des pages (virtuelle, mémoire vive) au début et à la fin de la transaction,
- la taille des données sur le disque,
- le nombre d'échanges de l'ensemble des pages,
- le nombre d'échange en lecture et en écriture sur le disque,
- le temps d'utilisation de l'UCT par le SGBD,
- les caractéristiques de l'UCT (fréquence et nombre de cœurs).

2.2 Évaluation du critère de validité des données

Pour évaluer ce critère, il nous faut nous assurer d'une couverture totale des langages. La stratégie à appliquer pour évaluer ce critère est similaire à celle définie pour l'évaluation du critère d'efficience. Il nous faut pouvoir spécifier toutes les fonctionnalités des langages en factorisant ce qui peut l'être. Nous combinerons donc l'évaluation de ces deux critères. Dans le cadre du présent mémoire, nous nous occuperons uniquement des tests concernant les fonctionnalités liées au relationnel. Pour les autres tests, nous présumerons qu'ils font partie de la suite de tests du SGBDR lui-même. En effet, les techniques pour ce faire sont éprouvées depuis longtemps et ne nécessitent pas une approche particulière aux SGBDR, ce qui les place hors de la portée du présent mémoire. De plus, notre objectif n'est pas à proprement parler de valider les SGBDR, mais de les comparer et on peut supposer que les fonctionnalités basiques ont été validées lors de leur mise en œuvre.

2.3 Évaluation du critère de disponibilité

Nous reprendrons l'approche de TPC-C qui semble tout à fait adaptée. Nous ajouterons cependant quelques améliorations.

Tout d'abord pour ce qui est de la propriété d'isolation, nous ajouterons les cas ou le conflit d'accès risque d'engendrer une erreur d'agrégation ou une erreur d'écriture sale.

Pour la propriété de durabilité, voici les différents cas d'erreurs que nous avons définis :

- arrêt du processus du SGBD par un programme externe,
- arrêt du processus du SGBD à cause d'une erreur interne,
- défaillance du disque sur lequel est physiquement stockée la BD,
- défaillance de l'alimentation du serveur sur lequel tourne le SGBD,
- réamorçage du serveur sur lequel tourne le SGBD,
- coupure de la connexion réseau entre le client et le serveur.

Pour l'évaluation du temps de récupération du SGBD, il nous paraît plus objectif de l'évaluer comme une fonction qui dépend :

- du volume de données représenté par les transactions interrompues ;
- du nombre de transactions en cours au moment de l'arrêt.

En effet, ce sont deux paramètres influents que nous devons fixer pour avoir des résultats comparables.

De plus, nous ne définirons pas quel est le temps acceptable pour récupérer d'une transaction représentant un certain volume de données. En effet, nous n'avons pas trouvé de solution pour le définir de manière objective. Pour comparer les SGBD sur cette caractéristique, il faudra donc se contenter de déterminer si la différence de temps observée est significative.

2.4 Évaluation du critère d'expressivité

Les solutions apportées dans la littérature pour évaluer ce critère sont peu satisfaisantes. Le seul aspect de l'expressivité abordé est la comparaison de l'étendue des fonctionnalités

exprimables [1, 31, 35]. Or cela ne correspond pas à notre problème, car nous voulons définir une méthode plus générique pour comparer l'ensemble des caractéristiques déterminant l'expressivité. Par ailleurs, il serait souhaitable que les caractéristiques de l'expressivité puissent être évaluées, le plus possible, par des mesures quantitatives.

Tout d'abord pour évaluer la pertinence des fonctionnalités fournies par le langage, on peut faire une évaluation comparative des fonctionnalités exprimables au sein d'un langage relativement à ce qu'elles apportent réellement.

Pour évaluer la simplicité de la syntaxe, on peut utiliser les métriques de Halstead [52]. Elles permettent d'évaluer la complexité du code d'un programme en analysant les opérateurs et les opérandes utilisés. On peut ainsi déduire les métriques suivantes :

- longueur du programme,
- taille du vocabulaire,
- volume du programme,
- niveau de difficulté du programme.

La mesure de complexité cyclomatique [33] peut aussi être utilisée pour évaluer la complexité d'un programme. Cette mesure comptabilise le nombre de « chemins » possibles au travers d'un programme.

Pour compléter la mesure de complexité de McCabe, nous pouvons aussi calculer la taille fonctionnelle de Wang et Shao [55]. Celle-ci se base aussi sur la complexité des structures conditionnelles, mais prend en compte des paramètres tels que la complexité du type de structure (**IF** moins complexe qu'un **WHILE**), l'imbrication des structures, et le nombre d'entrées-sorties des différents modules.

À partir du volume de Halstead et de la complexité cyclomatique, nous pourrons aussi calculer l'indice de maintenabilité [11] qui est une mesure synthétique prenant aussi en compte le nombre de lignes et le nombre de commentaires. Nous mettrons de côté le terme de l'expressivité des commentaires, car nous voulons isoler l'expressivité liée au langage.

Ces différentes mesures ont été définies de manière générale pour les langages de programmation. Cependant, il nous faudra les définir de manière plus spécifique dans le cadre de langages relationnels pour éviter de laisser libre cours à l'interprétation.

D'autres mesures existent dans la littérature.

La mesure du KLCID de Klemola et Riling [27] qui définit la complexité en fonction du nombre d'identificateurs présents dans le programme, un identificateur étant tout élément défini par le programmeur. Le nombre d'identificateurs contenus dans un programme à une grande influence sur l'expressivité, car pour un lecteur, ce n'est en général pas les mots clés du langage qui sont difficiles à comprendre, mais bien les identificateurs, car il doit comprendre à partir du nom de l'identificateur les intentions du programmeur, ce qui n'est pas toujours aisé. On peut donc considérer qu'un langage qui réduit le besoin d'identificateur est plus facile à comprendre.

La mesure CICM de Kushwaha et Misra [28] qui est un mélange de toutes les autres mesures. Elle prend en compte les identificateurs et opérandes, le nombre de lignes, les structures de contrôle et leurs poids.

Nous éliminerons ces deux mesures, car elles sont quelque peu redondantes avec les autres et que nous préférons isoler les moyens d'évaluation dans des mesures propres. Ces mesures seraient néanmoins intéressantes à prendre pour améliorer la couverture et vérifier leur cohérence avec les résultats obtenus pour les mesures précédentes. Mais, étant donné que nous n'avons pas d'outils déjà implémentés pour prendre ces mesures, cela demanderait du temps de développement supplémentaire.

D'autres aspects pourront seulement être évalués de manière qualitative.

Tout d'abord, pour évaluer le niveau d'abstraction du système de typage, on peut se baser sur la classification de Cardelli et Wegner [8] qui propose une classification pour les systèmes de typage qui peut nous permettre de déduire des niveaux d'abstraction. Nous présenterons plus en détail les niveaux d'abstraction au prochain chapitre.

On peut ensuite évaluer le niveau d'abstraction du langage de calcul. On peut ainsi définir trois niveaux d'abstraction :

- **Axiomatique** : on décrit les caractéristiques du résultat que l'on veut, mais on ne dit aucunement comment l'obtenir. C'est donc le niveau le plus abstrait. Cela assure une syntaxe simple et concise ainsi que l'indépendance vis-à-vis de la plateforme matérielle [21]. Par exemple le langage Prolog.
- **Fonctionnel** : on donne les fonctions (au sens mathématique) à appliquer pour arriver au résultat. Par exemple Lisp, Skin, ML, CAML.
- **Procédural** : on détaille l'algorithme du calcul. Par exemple FORTRAN, COBOL, Algol, Pascal, Modula-2, Ada, Eiffel, C, C++, Java, Python.

À noter qu'un même langage peut être composé d'un mélange de ces trois niveaux. Par exemple, SQL est à la fois fonctionnel et axiomatique avec l'opérateur **SELECT ... WHERE,** mais il est aussi procédural, car il est possible de définir des procédures avec PL/SQL.

En ce qui concerne le niveau d'abstraction du modèle de donnée, étant donné que nous voulons uniquement comparer des langages basés sur le modèle de données relationnel, nous ne le prendrons pas en compte.

Les méthodes cognitives peuvent aussi être envisagées pour évaluer l'expressivité des langages. L'étude de la manière dont le cerveau apprend et utilise un langage pourrait permettre de déduire les caractéristiques nécessaires pour que le langage soit simple à utiliser et à comprendre. Une telle évaluation nécessite une évaluation expérimentale. Par exemple, on pourrait donner à un échantillon de programmeur un programme à écrire dans deux langages différents et on observe lesquels sont les mieux écrits et le plus rapidement. Cependant ce type d'évaluation est souvent critiquable, car il est difficile d'identifier et d'isoler tous les facteurs influents.

En ce qui concerne l'évaluation de la signification des symboles, plus particulièrement de la charge d'apprentissage et de mémorisation induite, nous ne voyons pas de solution satisfaisante. En effet, elle est très complexe à déterminer étant donné qu'elle dépend du

vocabulaire connu par l'utilisateur du langage et donc varie selon les langues qu'il maîtrise, son expérience des langages de programmation, ses antécédents personnels...

2.5 Évaluation du critère de sécurité

L'article de Marco Vieira et Henrique Madeira [53] propose un modèle pour analyser la sécurité d'un SGBD. Il permet de définir des niveaux de sécurité en fonction des fonctionnalités de sécurité implémentées par le SGBD. Puis un système de poids sur chacune des fonctionnalités permet de raffiner l'évaluation en donnant un score. Les fonctionnalités qui doivent être présentes pour avoir le plus haut score sont les suivantes :

- **Système d'authentification :** il est présent, seul le DBA peut modifier la table des données d'identification, le mot de passe est crypté durant les communications et lors du stockage.
- **Privilèges pour les utilisateurs :** le DBA peut donner ou reprendre les droits des utilisateurs, une liste de privilèges nécessaires est proposée, les utilisateurs peuvent donner ou reprendre des droits et autoriser la propagation des autorisations, seul le DBA peut avoir des droits d'administration, seul le DBA a un accès complet à la table « *data dictionary* ».
- **Cryptage des communications :** présent, ne cause pas trop de perte d'efficience lors du chiffrement.
- **Cryptage des données stockées :** présent, ne cause pas trop de perte d'efficience lors du chiffrement et du déchiffrement.
- **Audit :** présent pour identifier l'accès des utilisateurs aux tables de la BD.

Le problème de ce modèle est qu'il est très incomplet. Il servira cependant de base pour notre travail.

Pour avoir une meilleure idée de comment caractériser la qualité de la sécurité dans un SGBD, nous avons fait des recherches approfondies en ce qui concerne la sécurité informatique. Se référer à l'Annexe A pour une présentation des trois mécanismes indispensables : authentification, gestion des autorisations et audit. À partir de cette analyse, nous pouvons déduire des critères pour évaluer ces différents mécanismes.

Ainsi, pour évaluer le mécanisme d'authentification, on peut faire ressortir trois critères :

- Facilité de vol ou falsification du moyen d'authentification.
- Fiabilité de la technique d'authentification (faux positifs et faux négatifs).
- Redondance de sécurité : la superposition de plusieurs techniques d'authentification améliore la sécurité.

Au niveau de la gestion des autorisations, nous utiliserons trois critères pour la caractériser : la simplicité avec laquelle une politique de sécurité peut être définie et vérifiée, la flexibilité avec laquelle une politique adaptée à des problèmes particuliers peut être définie et pour finir la fiabilité découlant le de la mise en œuvre des politiques, car le système doit fournir un moyen de valider les politiques mises en place.

La simplicité peut être évaluée par :

- la possibilité de définir de l'héritage de droits sur les sous-objets, cela peut éviter d'avoir à définir des droits sur tous les objets ;
- la possibilité de regrouper des droits et des utilisateurs, on défini alors les règles pour le groupe et non un par un pour chaque individu ;
- le nombre de droits et de rôles prédéfinis ;
- le niveau d'abstraction fourni pour utiliser les mécanismes, ne pas avoir à donner tous les détails pour définir une politique ;
- la possibilité de définir des droits nominatifs positifs (on accorde l'accès) et négatifs (on refuse l'accès).

La flexibilité peut être évaluée par :

- la possibilité de créer des politiques personnalisées ;
- l'étendue des « objets » auxquels les droits peuvent être appliqués. On peut par exemple, non seulement définir des droits sur les attributs, tuples et relations, mais aussi sur les méthodes (des classes). On gagne alors en flexibilité ;
- la présence de règles d'attribution des droits, pour que les utilisateurs qui en ont le droit puissent donner des droits à d'autres ;
- la mesure dans laquelle pour chaque objet on peut définir un droit différent pour chaque utilisateur.

La fiabilité peut être évaluée par :

- la présence d'un système de résolution des conflits qui permet de ne pas laisser au hasard le droit qui prédominera ;
- la présence d'un système de vérification des politiques pour s'assurer que la politique définie par l'utilisateur se comporte comme prévu, c'est-à-dire que les bonnes autorisations sont attribuées aux bons utilisateurs en tout temps.

Le critère de fiabilité est le plus important, car il est inutile de définir une politique de sécurité si l'on ne peut pas s'assurer qu'il n'existe aucun moyen pour la contourner. La flexibilité et la simplicité sont liées et feront l'objet d'un compromis. En effet, lorsque l'on gagne en flexibilité, il est alors plus difficile de définir une politique.

Pour finir, pour évaluer le système d'audit, on peut utiliser les différents types de réaction à une intrusion présentée en annexe A (A3), car chacun offre un niveau de sécurité plus ou moins élevé. Le niveau de sécurité le plus bas est atteint avec une réaction de type déclenchement d'alarme, car ce type de réaction ne permet pas d'empêcher le vol ou la corruption de données. Ensuite, on trouve les réactions de type action correctives qui permettent de réparer les dégâts en cas de corruption des données, mais n'ont aucun effet en cas de vol de données. Puis viennent les réactions de type défensives qui permettent de lutter contre la corruption des données et le vol de données. En mettant en place différents scénarios d'attaque, on peut évaluer le pourcentage d'attaques qui ont été traitées avec chacune de ces techniques.

À cela, on peut ajouter un autre critère d'évaluation qui est le temps de réaction entre le début de l'attaque et le déclenchement de la réaction. Plus ce temps de réaction est faible, plus la sécurité du système est assurée, car on peut alors empêcher l'intrus de poursuivre son action dans le cas d'une action défensive et on réduit le temps pendant lequel les données sont non intègres dans le cas d'une action corrective.

Chapitre 3

Stratégies d'essai

Ce chapitre présente les stratégies d'essai retenu dans le cadre de ce mémoire pour évaluer chacun des critères. Nous justifions la pertinence de chacune de ces stratégies et décrivons tous les éléments nécessaires à leur implémentation.

3.1 Validité des données

Nous avons choisi de restreindre le critère de validité du langage établi dans la section 1.2.3 du Chapitre 1 à la validité du langage relationnel. La validité du langage transactionnel et du langage de sécurité seront respectivement évalués avec les critères de disponibilité et de sécurité. Comme stipulé précédemment, la validité du langage algorithmique et du langage de typage ne seront évalués que d'un point de vue d'intégration au relationnel, le reste étant laissé de côté pour des raisons de temps.

Pour évaluer le critère de fiabilité et de validité du langage relationnel, nous avons choisi de découper les fonctionnalités du langage en quatre stratégies : « FonctionnalitesTypage », « FonctionnalitesAlgorithmiques », « FonctionnalitesRelationnelles », « FonctionnalitesCombinees ». Ces quatre stratégies devront être effectuées dans un ordre particulier, car elles dépendent les unes des autres.

La stratégie « FonctionnalitesTypage » comprend les fonctionnalités de typage du langage relationnel. C'est la première qui doit être testée, car c'est la base du langage, toutes les autres fonctionnalités étant basées sur la manipulation de variables typées.

La stratégie « FonctionnalitesAlgorithmiques » comprend le langage algorithmique permettant d'ajouter des routines au langage relationnel. Elle doit être testée en deuxième, car une fois le système de typage validé, il est possible de la tester indépendamment du reste du langage.

La stratégie « FonctionnalitesRelationnelles » comprend le langage relationnel en lui-même. Il doit être testé en troisième, car il ne peut être dissocié du système de typage et du langage algorithmique.

La stratégie « FonctionnalitesCombinees » comprend des fonctionnalités plus complexes qui combinent plusieurs fonctionnalités qui ont été testées de manière isolée auparavant dans les autres stratégies. Cette stratégie nous permet de combler les lacunes de couverture des autres stratégies. Elle est logiquement à mettre en place en dernier, car il faut valider le fonctionnement de la fonctionnalité seule avant de vérifier si cela peut poser problème de la combiner avec d'autres.

Cette approche permet d'illustrer et vérifier la couverture du langage sur la base des spécifications du langage.

Pour chacune de ces stratégies, les résultats de tous les cas de tests d'une fonctionnalité doivent être bons pour considérer la fonctionnalité comme présente et totalement opérationnelle.

Si l'exécution d'un test entraîne un résultat faux, cela peut être pour deux raisons :

- Soit c'est une erreur d'exécution, alors cela peut signifier qu'il y a un problème dans la description du test ou bien que la fonctionnalité n'est pas présente pour le langage. Si le test n'est pas passé à cause d'un problème d'exécution engendré par une erreur dans le test, il faut corriger le problème et recommencer le test. Sinon, le résultat n'est utile que pour évaluer l'expressivité du langage et n'a aucune incidence sur la fiabilité et la validité du langage.
- Soit c'est une erreur au niveau des conditions de passage du test, on peut alors en déduire que la fonctionnalité n'est pas valide, car elle ne produit pas le résultat

attendu. Une analyse plus approfondie de l'erreur retournée nous permettra alors d'expliquer le problème de comportement identifié.

3.1.1 Stratégie « FonctionnalitesTypage »

L'objectif de cette stratégie est d'évaluer le bon fonctionnement de chaque fonctionnalité du langage concernant le système de typage. Pour cela il nous faut isoler chaque fonctionnalité dans un essai qui lui est propre puis évaluer la fonctionnalité avec différents cas de test. Cette stratégie sera combinée avec la stratégie de fonctionnalité du critère d'efficience qui, elle aussi, requiert l'isolation des différentes fonctionnalités.

Les fonctionnalités de typage que nous avons identifiées sont les suivantes :

- création d'un nouveau type ou d'une extension d'un type,
- création d'une relation utilisant un type créé ou un type prédéfini,
- insertion de valeurs dans une relation typée,
- modification d'un type créé,
- suppression d'un type créé

3.1.2 Stratégie « FonctionnalitesAlgorithmiques »

L'objectif de cette stratégie est d'évaluer le bon fonctionnement de chaque fonctionnalité du langage concernant les opérations algorithmiques. Pour cela, il nous faut isoler chaque fonctionnalité dans un essai qui lui est propre puis évaluer la fonctionnalité avec différents cas de test. Cette stratégie sera combinée avec la stratégie de fonctionnalité du critère d'efficience qui, elle aussi, requiert l'isolation des différentes fonctionnalités.

Les fonctionnalités algorithmiques que nous avons identifiées sont les suivantes :

- création d'une routine (fonction, procédure, méthode),
- utilisation d'une routine dans le langage relationnel,
- modification d'une routine,
- suppression d'une routine

3.1.3 Stratégie « FonctionnalitesRelationnelles »

L'objectif de cette stratégie est d'évaluer le bon fonctionnement de chaque fonctionnalité du langage concernant le langage relationnelle. Pour cela il nous faut isoler chaque fonctionnalité dans un essai qui lui est propre puis évaluer la fonctionnalité avec différents cas de test. Cette stratégie sera combinée avec la stratégie de fonctionnalité du critère d'efficience qui elle aussi requiert l'isolation des différentes fonctionnalités. Nous nous baserons sur la description de l'algèbre relationnelle faite par ElMasri [16] pour spécifier des tests génériques de fonctionnalités qui devrait être présentent dans tous les langages (puisque nous avons prévu de tester uniquement des langages relationnels). Puis nous nous baserons sur les documentations respectives des différents langages que nous testerons pour définir des tests spécifiques si les langages fournissent des fonctionnalités supplémentaires ou nécessitent des cas de tests différents.

Les fonctionnalités relationnelles que nous avons identifiées sont les suivantes :

- sélection de tuples,
- projection sur des colonnes,
- union de deux relations avec élimination des doublons,
- union de deux relations avec conservation des doublons,
- intersection de deux relations avec élimination des doublons,
- intersection de deux relations avec conservation des doublons,
- différence entre deux relations avec élimination des doublons,
- différence entre deux relations avec conservation des doublons,
- division entre deux relations avec élimination des doublons,
- division entre deux relations avec conservation des doublons,
- produit cartésien entre deux relations,
- jointure thêta entre deux relations,
- jointure naturelle entre deux relations,
- jointure externe à gauche entre deux relations,
- jointure externe à droite entre deux relations,
- jointure externe totale entre deux relations,
- renommage d'une relation ou d'une colonne,
- comptage du nombre de tuples (opérateur d'agrégation),
- moyenne d'un ensemble de tuples (opérateur d'agrégation),

- somme d'un ensemble de tuples (opérateur d'agrégation),
- minimum d'un ensemble de tuples (opérateur d'agrégation),
- maximum d'un ensemble de tuples (opérateur d'agrégation),
- groupement de tuples pour une opération d'agrégation,
- tri ascendant sur une colonne,
- tri descendant sur une colonne,
- fermeture transitive sur une relation,
- création d'une relation,
- ajout d'une colonne à une relation,
- suppression d'une colonne dans une relation,
- modification du nom d'une colonne,
- ajout d'un tuple dans une relation,
- modification d'un tuple dans une relation,
- suppression d'un tuple dans une relation,
- définition d'une contrainte de clé sur une relation,
- définition d'une contrainte d'intégrité référentielle

3.1.4 Stratégie « FonctionnalitesCombinees »

L'objectif de cette stratégie est d'évaluer le bon fonctionnement de la combinaison de fonctionnalités testées au préalable de manière indépendante dans une stratégie précédente. Pour cela il nous faut isoler chaque fonctionnalité dans un essai qui lui est propre puis évaluer la fonctionnalité avec différents cas de test. Cette stratégie sera combinée avec la stratégie de fonctionnalité du critère d'efficience qui elle aussi requiert l'isolation des différentes fonctionnalités.

L'étendue de la combinaison des fonctionnalités est très vaste. Nous ne prétendrons donc pas à une couverture de test complète. Nous avons choisi deux exemples pour illustrer l'intérêt de cette stratégie :

- ajout dans une relation de tuples sélectionnés dans une autre relation,
- combinaison d'une projection et d'une sélection sur une relation

Le premier illustre la capacité de créer un tuple du même type qu'une relation existante. Or c'est une fonctionnalité que l'on ne peut avoir qu'en combinant les deux fonctionnalités d'ajout de tuples et de sélection de tuples.

Le deuxième illustre une utilisation plus réaliste des opérateurs de sélection et de projection, car en pratique ils sont le plus souvent combinés.

3.2 Disponibilité

Les stratégies choisies pour tester la disponibilité sont basées sur les tests de TPC-C. Il s'agit de : « Atomicite », « Coherence », « Isolation », « Durabilite ». TPC-C est avant tout une suite de tests qui permet d'évaluer l'efficience des SGBD. Mais étant donné que les tests proposés utilisent le mécanisme transactionnel et que l'efficience est très liée à la gestion du transactionnel, TPC-C stipule que les SGBD testés doivent pouvoir prouver leur validité transactionnelle pour que les résultats d'efficiences soient interprétables. Les tests proposés par TPC-C consistent tout simplement à vérifier que les propriétés ACID telles qu'elles ont été définies par Jim Gray [18] sont bien vérifiées. Cette approche nous permet par la même occasion d'avoir une couverture complète du langage transactionnel. Cependant, comme nous l'avons vu lors de l'analyse préliminaire des tests définis par TPC-C, ils comportent quelques lacunes. Les stratégies que nous définissons ci-dessous sont donc une amélioration des essais décrits par TPC-C.

Nous avons choisi de mettre de côté l'évaluation du temps de récupération. En effet, cette stratégie n'apporterait pas grand-chose de nouveau à notre banc d'essai et demanderait du temps à implémenter et surtout à exécuter.

3.2.1 Stratégie « Atomicite »

L'objectif de cette stratégie est de vérifier la propriété d'atomicité des transactions. Celle-ci stipule qu'une transaction doit être effectuée dans son ensemble ou bien aucune modification n'est faite sur la BD.

Dans cette stratégie, nous testerons la propriété d'atomicité dans des conditions d'exécution normales (sans pannes) et séquentielles de manière à pouvoir isoler les problèmes d'atomicité des problèmes de durabilité et d'isolation que nous testerons par la suite.

De plus, le fait de tester l'atomicité sur une transaction particulière est suffisant, car le mécanisme qui permet de définir une transaction ne dépend a priori pas de ce que contient la transaction, mais du comportement lorsqu'elle est interrompue. Il faut cependant bien choisir cette transaction. Pour être représentative, elle doit effectuer des modifications sur la BD (mise à jour ou insertion) pour que l'on puisse détecter si les changements ont effectivement été appliqués.

Le contenu de la transaction devra être contrôlé pour pouvoir déterminer l'état final attendu de la BD. Les paramètres d'appel ne seront donc pas générés aléatoirement.

Pour tester cela, nous réutiliserons le jeu de données initial de TPC-C. Les tests consistent à exécuter une transaction de paiement que nous annulerons avant la fin et nous vérifierons qu'aucun changement n'est présent dans la BD. Puis, nous exécuterons la même transaction sans l'annuler et vérifierons que cette fois-ci tous les changements sont présents dans la BD.

3.2.2 Stratégie « Coherence »

L'objectif de cette stratégie est de vérifier la propriété de cohérence de la base de données après l'exécution d'un lot de transactions.

La propriété de cohérence indique qu'une transaction doit amener la BD d'un état cohérent vers un autre. La cohérence de la BD est définie par un certain nombre de contraintes définies par l'utilisateur pour représenter son système qui doivent être vraies en tout temps où les données sont observables.

Pour définir la cohérence de la BD, nous utiliserons les douze contraintes définies par TPC-C en adéquation avec le modèle de données utilisé. Ces contraintes sont des relations entre les différentes tables ou les différents tuples d'une même table et elles doivent être respectées pour s'assurer de la validité des données présentes dans la base. Ces contraintes dépendent du sens que l'on donne aux données par rapport à l'application pour laquelle elles sont utilisées.

À l'aide de ces contraintes, nous vérifierons tout d'abord la cohérence de notre BD après initialisation de la BD pour nous assurer que le problème ne vient pas d'une mauvaise initialisation et non de l'exécution des transactions.

Nous exécuterons ensuite un grand nombre de transactions aléatoires choisies parmi celles de TPC-C. Les transactions de TPC-C comportent des opérations de modification de la BD telles que des insertions, des suppressions ou des modifications. Elles sont donc bien adaptées à la situation, car les opérations de lecture ne faisant aucune modification sur la BD, les chances de l'amener vers un état incohérent sont nulles.

Puisque l'on veut vérifier des propriétés et non l'état exact de la BD, on peut se satisfaire de générer des appels de procédure avec des paramètres aléatoires. Car l'on n'a pas besoin de savoir l'état de la BD pour vérifier les propriétés.

Nous testerons la propriété de cohérence dans deux contextes différents. Le premier est celui d'une exécution séquentielle de transactions. Le second est celui d'une exécution concurrente de transactions. De cette manière, nous pourrons tester le maintien de la cohérence par le SGBD en temps normal ainsi qu'en mode transactionnel. En effet, dans TPC-C, on se contente de le vérifier dans un contexte transactionnel, cela est suffisant, mais ne permet pas de définir précisément la cause du problème. Pour le deuxième contexte, nous devons aussi prendre en compte le niveau d'isolation des transactions de manière à pouvoir définir si celui-ci à une influence sur la cohérence de la BD.

Cette stratégie pourra être combinée avec la stratégie TPC-C pour mesurer le critère d'efficience de manière à éviter la redondance d'exécution des tests qui sont communs aux deux stratégies.

La création du jeu de données initial doit s'exécuter correctement pour pouvoir interpréter les résultats. Si celui-ci échoue, il faut alors identifier la source de l'erreur et relancer les tests. Une fois le jeu de données mis en place, il faut que les deux tests (avec annulation et avec validation des modifications) soient valides pour que la propriété d'isolation soit vérifiée.

Pour pouvoir interpréter les résultats, il faut que l'initialisation et la vérification de la cohérence après l'initialisation soient valides.

Pour valider la propriété de cohérence dans chaque contexte, il faut tout d'abord que la BD soit cohérente juste avant l'exécution des transactions dans ce contexte. Si ce n'est pas le cas, les résultats ne peuvent pas être interprétés. Il faut ensuite que l'exécution des transactions dans ce contexte s'exécute sans erreur et que les résultats de cohérence après l'exécution soient tous valides.

Si la propriété de cohérence est valide dans tous les contextes alors on peut en conclure que le mécanisme transactionnel implémenté par le SGBD testé respecte cette propriété.

3.2.3 Stratégie « Isolation »

L'objectif de cette stratégie est de vérifier la propriété d'isolation des transactions.

La propriété d'isolation indique que plusieurs transactions ne doivent jamais interférer entre elles, ni même agir selon le fonctionnement de chacune. On peut identifier cinq types d'erreurs (TPC-C n'en prend en compte que quatre) qui peuvent survenir à cause de l'enchevêtrement des transactions :

- l'écriture impropre,
- la lecture impropre,
- la lecture non reproductible,
- la lecture fantôme,
- l'agrégation incorrecte

Celles-ci sont définies dans l'Annexe B.

TPC-C ne définit pas comment l'enchevêtrement des transactions doit être contrôlé par le testeur. Nous proposons deux méthodes pour le faire.

La première consiste à tester le comportement de la BD si les transactions concurrentes utilisent des verrous sur les données forçant l'échec de la transaction de façon déterministe si

on a affaire à l'un des cas définis ci-dessus. Cela nous permettra de vérifier que le SGBD détecte bien tous les cas et que l'utilisation de sémaphores fonctionne.

Ensuite, nous provoquerons un enchevêtrement contrôlé des transactions en y introduisant des délais. Cette méthode nous permettra de vérifier le comportement du SGBD lorsqu'il n'est pas aidé par le programmeur pour détecter et résoudre les problèmes d'isolation. Les délais devront être suffisamment conséquents pour s'assurer de pouvoir reproduire le schéma d'exécution voulu.

Ces deux méthodes devront être appliquées séparément pour chaque type d'erreur possible afin d'isoler les différentes sources d'erreur. Nous vérifierons alors que les modifications qui ont été effectuées sur la BD sont celles attendues.

Dans TPC-C, les transactions utilisées pour les tests d'isolation sont pour la plupart celles définies pour les tests d'efficience. Pour pouvoir faciliter la vérification des résultats, nous nous contenterons d'une version simplifiée de ces transactions constituée uniquement de requête qui permette réellement d'obtenir le conflit. Cela sera moins représentatif d'une application réelle, mais nous permettra tout de même de représenter des cas de figure problématiques qui pourraient se produire avec l'utilisation complète des transactions de TPC-C.

En SQL, le comportement du mécanisme transactionnel vis-à-vis de ces erreurs peut être modifié en changeant le niveau d'isolation des transactions. Il nous faut donc tester pour chaque niveau d'isolation que les erreurs qui ne doivent pas se produire ne se produisent pas.

Si l'exécution d'un des tests d'isolation échoue, car le SGBD a détecté qu'un problème d'isolation pouvait se produire, mais qu'il est incapable de trouver une solution pour l'empêcher, alors son comportement est considéré comme valide, car l'intégrité des données est bien conservée. Cependant, il faudra approfondir la cause de l'échec, car cela peut révéler un problème au niveau de l'algorithme de sérialisation qui n'a pas trouvé de solution alors qu'il en existait une. Si l'exécution échoue pour une autre raison, alors il faut trouver la cause du problème et relancer les tests. Si l'un des tests échoue, car il ne respecte pas les conditions

de validité alors c'est que le SGBD n'a pas détecté la possibilité d'une erreur. Il ne respecte donc pas la propriété d'isolation des transactions.

3.2.4 Stratégie « Durabilite »

L'objectif de cette stratégie est de vérifier la propriété de durabilité des transactions.

La propriété de durabilité indique que toutes les transactions validées par le système doivent persister même s'il y a une panne. La panne peut être interne, soit provoquée par le SGBD ou externe, soit provoquée par un composant logiciel ou matériel externe au SGBD. Nous devrons donc tester ces deux cas.

Les mécanismes importants que le SGBD doit fournir pour assurer la durabilité des données sont les suivants : la journalisation, les points de reprise (« *check-points* ») et les copies de sécurité (« *backup* »). Nous nous contenterons de vérifier le bon fonctionnement de la journalisation et des points de reprise, car une bonne gestion des copies de sécurité dépend plus des politiques établies par le DBA que du SGBD.

Pour tester cela, nous provoquerons plusieurs types de panne pendant l'exécution des transactions. Puis nous vérifierons qu'après récupération du système, les modifications qui avaient été introduites par les transactions qui ont fini leur exécution sont bien présentes tandis que les modifications des transactions interrompues par la panne ne le sont pas.

Pour pouvoir contrôler le moment où l'on provoque la panne, nous exécuterons tout d'abord une transaction dans son ensemble dont on sait les résultats attendus puis nous exécuterons une transaction composée d'une boucle infinie pendant laquelle nous provoquerons la panne. Les modifications engendrées par la première transaction devront alors être présentes tandis que les modifications de la longue transaction ne devront pas avoir été effectuées.

Les transactions que nous utiliserons devront donc être principalement constituées d'instructions de modifications de la BD pour que l'on puisse voir si les changements ont effectivement été appliqués sur la BD.

Étant donné que l'enchevêtrement des transactions ne devrait pas avoir d'influence sur le processus de récupération, les transactions seront exécutées séquentiellement pour mieux contrôler les modifications sur la BD et ainsi vérifier plus simplement leurs présences.

Voici une liste des pannes à tester et de comment les simuler :

- Défaillance du disque : déconnexion à chaud en cours d'exploitation.
- Défaillance du processeur : réamorçage du serveur.
- Défaillance d'alimentation : débranchement du serveur.
- Défaillance du processus du SGBD : mise à mort du processus.
- Défaillance de la connexion réseau : interrompre la connexion réseau.
- Arrêt programmé du SGBD : Le SGBD peut fournir la possibilité d'arrêter le processus de manière plus propre qu'avec une mise à mort (« *kill* »). Par exemple, il peut attendre que les transactions en cours s'arrêtent ou non, les annulées ou non, que les utilisateurs connectés se déconnectent ou non. Ces tests dépendront donc du SGBD.
- Arrêt non programmé provoqué par une défaillance interne du SGBD : pas de méthode pour tester cela pour l'instant.

Ces différentes pannes devront être testées une par une dans des séances de test différentes de manière à isoler les pannes qui peuvent causer un problème de durabilité.

Les tests de durabilité doivent être lancés sur un jeu de données cohérent pour pouvoir interpréter les résultats. De même, la première transaction doit s'exécuter sans erreurs, si ce n'est pas le cas, il faut réparer le problème et relancer les tests. Si les tests permettant de vérifier la présence des modifications effectuées par la première transaction échouent ou bien si les tests permettant de vérifier que les modifications effectuées par la transaction infinie ne sont pas présentes échouent, alors c'est que le SGBD ne gère pas correctement la durabilité. Selon la gravité des erreurs engendrées par la simulation d'une défaillance, il est possible que le SGBD ait besoin de l'intervention d'un opérateur pour rejouer les transactions et remettre la BD en état. Dans ce cas, le SGBD devra tout de même indiquer que la BD est dans un état incohérent et que cette opération doit être effectuée. Les tests de vérification de l'état de la BD seront alors lancés après cette récupération manuelle pour valider le comportement du SGBD.

3.3 Sécurité

Nous avons choisi d'utiliser deux stratégies pour évaluer la sécurité : « FonctionnalitesSecurite » et « ScenarioIntrusion ». La stratégie « FonctionnalitesSecurite » consiste en une évaluation qualitative des différentes fonctionnalités fournies par le système. La stratégie « ScenarioIntrusion » est un complément à l'évaluation qualitative du mécanisme d'audit, elle consiste à lancer un lot de scénario d'attaque et évaluer le comportement du système à l'aide de plusieurs métriques. Cela nous permet une évaluation plus fine des mécanismes et donc améliore notre couverture de test.

3.3.1 Stratégie « FonctionnalitesSecurite »

Cette stratégie est basée sur les fonctionnalités liées à la sécurité que le SGBD fournit. La stratégie s'occupera tout d'abord de valider le fonctionnement de ces fonctionnalités puis d'évaluer leur niveau de fiabilité, tout en prenant en compte en second plan la simplicité et la flexibilité d'utilisation qu'elles amènent.

La stratégie sera composée de plusieurs séances chacune évaluant l'un des mécanismes nécessaires pour assurer la sécurité d'un SGBD.

Le premier est le mécanisme d'authentification. Il permet d'identifier l'utilisateur (processus, programme, personne) qui veut accéder à la base.

Voici les fonctionnalités le concernant :

- présence d'un mécanisme d'authentification intégré au SGBD ;
- possibilité de remplacer le mécanisme d'authentification interne par un externe ;
- ajout d'un utilisateur ;
- modification d'un utilisateur ;
- suppression d'un utilisateur ;
- algorithme de validation des données d'authentification ;
- mécanismes de renforcement des mots de passe (pertinent dans le cas d'une authentification par mot de passe) ;
- cryptage des données d'authentification ;

- redondances des mécanismes d'authentification.

Ensuite, le mécanisme de gestion des autorisations permet d'attribuer à l'utilisateur des droits pour savoir quelles sont les actions qu'il peut exécuter.

Voici les fonctionnalités le concernant :

- mécanisme de résolution des conflits ;
- mécanisme de vérification de politique ;
- héritage de droits ;
- regroupement de droits ;
- regroupement d'utilisateurs ;
- droits prédéfinis ;
- groupe de droits prédéfinis ;
- politique prédéfinie ;
- attribution de droits ;
- politiques personnalisables ;
- granularité de la personnalisation (définition d'un droit pour un utilisateur pour un objet) ;
- droits positifs et négatifs.

Puis, le mécanisme d'audit permet d'assurer la sécurité des données même dans le cas où la gestion des autorisations et de l'authentification échoue. On peut alors garder une trace des modifications qui ont été effectuées pour pouvoir tout d'abord identifier l'auteur de l'attaque, mais aussi pour pouvoir assurer l'intégrité de la base en défaisant les modifications non autorisées qui ont pu être appliquées.

Voici les fonctionnalités le concernant :

- présence d'un mécanisme d'audit ;
- type de déclenchement de l'algorithme de détection d'intrusion ;
- algorithme de détection d'intrusion ;
- algorithme de réaction

Pour finir, il faut s'assurer de l'étanchéité de l'accès à ces mécanismes. Les informations contenues dans la BD ne doivent pas pouvoir être accédées sans passer par les mécanismes de

sécurité décrits précédemment. Ainsi, les fichiers associés aux bases de données ne doivent pouvoir être accédés que par l'entremise du SGBD ou, à tout le moins, ne pouvoir être déchiffrés ou modifiés.

Cette approche permet de s'assurer que les mécanismes indispensables à tout système pour assurer la sécurité sont couverts.

Le résultat de cette stratégie sera une analyse qualitative de la simplicité, la fiabilité et la flexibilité de ces trois mécanismes à partir du recensement des fonctionnalités présentes.

3.3.2 Stratégie « ScenarioIntrusion »

Cette stratégie consiste à mettre au point un lot de scénarios d'attaques représentatives et d'évaluer les réactions du système. Par représentativité, nous entendons que les scénarios d'attaque doivent être variés et inspirés de cas réels. Il faudra aussi veiller à ce que les deux types d'attaque mentionnés dans l'annexe A.3 soient représentés (attaques connues suivant un scénario défini et attaque nouvelle ne suivant aucun scénario connu).

Cette stratégie permet uniquement d'évaluer le mécanisme de détection, elle est donc à appliquer uniquement si la stratégie « FonctionnalitesSecurite » n'a pas révélé un mécanisme de détection inexistant.

L'évaluation de la réaction du système se fera à l'aide des métriques suivantes :

- rapport entre le temps d'analyse pendant une intrusion et le temps d'analyse alors qu'aucune intrusion n'est en cours ;
- pourcentage de faux positif et non-détection d'intrusion ;
- temps de détection moyen des intrusions ;
- pourcentage de réussite d'une action corrective ;
- pourcentage de réussite d'une action défensive.

Nous utiliserons ensuite ces métriques pour comparer les mécanismes de détection des différents systèmes.

3.4 Efficience

L'efficience sera évaluée à l'aide de trois stratégies : « TPCC », « FonctionnalitesIsolees » et « ApplicationExistante ».

La stratégie « TPCC » se base sur le banc d'essai TPC-C qui est la référence en termes d'évaluation de l'efficience des SGBD. Elle nous permettra de couvrir le type d'applications bien particulier des applications commerciales. Ce type d'applications est intéressant, car bien souvent elle demande une grande efficience du fait de la quantité élevée de données qui est manipulée et du haut taux d'accès en lecture et en écriture. De plus, une telle application permet de tester l'influence du transactionnel sur l'efficience.

La stratégie « Fonctionnalite » évalue l'efficience des fonctionnalités relationnelles du langage. Cela permet une grande couverture du langage, car on peut alors affirmer que chaque fonctionnalité est testée au moins une fois. Cependant, au niveau de la réalité de l'utilisation des bases de données, c'est beaucoup moins pertinent, car les fonctionnalités ne sont que très rarement totalement isolées.

La troisième stratégie « ApplicationExistante » consiste à évaluer l'efficience d'un lot d'applications réelles qui sont peut-être moins représentatives, car moins exigeantes en terme d'efficience, mais qui vont nous permettre de combler les lacunes de notre couverture de test.

Ces trois stratégies sont indépendantes et peuvent donc être exécutées dans n'importe quel ordre.

3.4.1 Stratégie « TPCC »

L'objectif de cette stratégie est d'évaluer l'efficience du SGBD dans les conditions réelles d'une application commerciale. En effet, le cybercommerce nécessite un type d'applications où l'efficience est primordiale, car énormément de transactions peuvent être effectuées en parallèle et le système doit être capable de réagir rapidement malgré tout. Pour cela, nous nous basons sur la suite de tests TPC-C [48] qui est la référence en termes d'évaluation

d'efficience sur les SGBD. TPC-C définit un jeu de données qui représente une compagnie de vente d'items. Celle-ci possède plusieurs entrepôts qui stockent chacun un ensemble d'items à vendre. Les entrepôts sont composés de plusieurs districts. Chaque client est rattaché à un district. Les clients peuvent passer des commandes composées d'items. Les commandes sont d'abord mises en attente avant d'être traitées. Un historique des transactions de paiement effectuées par les clients est conservé. Les entrepôts peuvent suivre l'évolution de leurs stocks. Un lot de transaction typique est ensuite disponible pour interagir avec le système :

- « New_Order » : permet de passer une nouvelle commande qui sera automatiquement mise en attente de traitement ;
- « Delivery » : s'occupe du traitement d'un lot de commande en attente ;
- « Stock_Level » : permet de consulter l'état des stocks ;
- « Payment » : permet à un client d'effectuer un paiement à la compagnie ;
- « Order_Status » : permet de consulter les informations de la dernière commande passée.

La stratégie de la suite de tests est ensuite d'exécuter en concurrence un grand nombre de ces transactions tel que cela serait fait si l'on avait plusieurs clients qui accédaient en même temps à la BD pour y effectuer des opérations. La répartition du type de transaction exécutée est représentative d'une application réelle, soit environ :

- 45 % de « New_Order » ;
- 43 % de « Payment » (2 % des commandes sont annulées) ;
- 4 % de « Order_Status » (pour une commande sur 10, les clients demandent à savoir l'état de leur commande) ;
- 4 % de « Delivery » (une livraison effectuée toutes les 10 commandes, car les livraisons se font par 10) ;
- 4 % de « Stock_Level » (toutes les 10 livraisons on vérifie que le stock des derniers items commandés est au-dessus d'un certain seuil, pour prévoir le réapprovisionnement).

L'efficience ainsi évaluée permet de prendre en compte le traitement transactionnel ainsi qu'un lot de fonctionnalités du langage dans un cas certes un peu particulier, mais suffisamment général pour que l'on puisse en tirer une interprétation. En plus d'une exécution en concurrence des différentes transactions, nous ajouterons une séance d'essai qui

se contente d'effectuer les différentes transactions de manière séquentielle. Ceci nous permettra de tester l'efficience du système sans prendre en compte le mécanisme transactionnel. Bien entendu, ce cas ne reflète plus vraiment le cas d'une application réelle, car il permet en pratique de simuler une application commerciale où il y aurait un seul client, ce qui est peu vraisemblable. Cependant, cela nous aidera à interpréter les résultats obtenus dans le cas d'une application avec plusieurs clients en nous indiquant si le problème vient de la gestion du transactionnel. Dans le cas où le langage permet plusieurs niveaux d'isolation, nous ajouterons un cas de test pour chacun de ces niveaux. En effet, les niveaux d'isolation permettent, en principe, d'améliorer l'efficience du mécanisme transactionnel aux dépens de l'intégrité des données. Il est donc indispensable de tester leurs effets sur les résultats d'efficience.

Les résultats des tests (échec ou réussite) seront uniquement des résultats pour valider le déroulement correct de l'exécution des tests. Si un des tests ne s'est pas exécuté correctement alors on ne peut pas faire confiance aux mesures obtenues. Il faut alors identifier la source du problème d'exécution puis la corriger et relancer les tests.

Les mesures à prendre sont les suivantes :

- Durée d'exécution.
- Temps d'exécution de l'UCT.
- Nombre de lecture sur le disque
- Nombre de lecture sur le disque
- Quantité de mémoire résidente consommée
- Nombre de défauts de page

TPC-C se contente de mesurer la durée d'exécution, mais nous avons choisi de prendre un lot de mesure plus complet. Cela nous donnera plus d'indications pour pouvoir justifier les écarts de temps obtenus.

TPC-C définit un temps de réponse maximum pour chaque type de transaction. 90 % des transactions Stock-Level exécutées doivent avoir un temps de réponse maximum de 20 secondes. Pour les autres transactions, 90 % des transactions doivent avoir un temps de

réponse maximum de 5 secondes. Ces contraintes proviennent du fait que l'on se trouve dans le cas d'une application commerciale qui doit pouvoir satisfaire les clients utilisant le système. Cependant, ces résultats dépendent de la plateforme de test (processeur, mémoire vive...) et pas uniquement du SGBD. Or contrairement à l'objectif de TPC-C, nous ne voulons pas savoir quel est le meilleur couple SGBD-plateforme, mais comparer les SGBD entre eux en isolant le plus possible le facteur de la plateforme.

Nous nous assurerons donc uniquement que les mesures obtenues sont fiables.

Pour cela, nous devons tout d'abord nous assurer de la représentativité de nos échantillons. Le nombre d'échantillons de notre population est potentiellement infini, car l'on pourrait reproduire les tests une infinité de fois. Or, comme il n'est pas envisageable d'effectuer une infinité de répétitions, il faut s'assurer que notre lot d'échantillons est bien représentatif de cette population infinie. Pour cela, nous utiliserons une méthode bien connue des statisticiens. Celle-ci consiste à fixer un niveau de confiance et une marge d'erreur acceptable. Le niveau de confiance indique la probabilité que l'échantillon de résultats obtenus constitue une représentation fidèle de la population et qu'il soit valide. La marge d'erreur représente l'incertitude sur le résultat. On fixe usuellement un niveau de confiance de 95 % et une marge d'erreur de 5 %. Pour obtenir ces conditions, il faut alors évaluer 385 éléments de la population, indépendants les uns des autres. Ces critères pourront être revus à la baisse pour les tests dont ce nombre de répétitions ne peut être atteint en pratique avec le banc d'essai dont nous disposons dans des délais compatibles avec la portée de notre recherche.

Nous devons ensuite nous assurer de la stabilité de nos mesures. Si nos mesures sont instables, cela peut refléter un problème d'isolation du phénomène que l'on mesure. Pour mesurer la stabilité, nous éliminerons tout d'abord les mesures potentiellement aberrantes (le minimum et le maximum). Cela nous permettra de régler le problème d'une éventuelle variation dû à des variables que l'on ne peut pas contrôler comme l'exécution de processus dédiés au système d'exploitation. Nous calculerons ensuite le coefficient de variation qui permet d'évaluer la dispersion des valeurs relativement à leur moyenne. Notre objectif est d'obtenir un coefficient relatif inférieur à 5 %. Nous motivons ce choix sur la base du niveau

de risque acceptable pour des décisions économiques : le choix entre deux SGBD reposera vraisemblablement sur des critères autres que l'efficience si la différence est inférieure à 5 %.

Pour espérer avoir des résultats stables selon ces critères, il faut tout d'abord s'assurer d'avoir un jeu de données suffisamment conséquent pour que les mesures soient significatives. Dans TPC-C, la seule variable laissée au choix de l'utilisateur pour faire varier le jeu de données est le nombre d'entrepôts (« Warehouse »). À partir de la variation de cette valeur, les tables étant pour la plupart liées entre elles par l'identifiant de l'entrepôt, l'ajout d'un seul entrepôt à des répercussions sur la taille du jeu de données de toutes les autres tables. Ce qui permet de faire augmenter assez rapidement le jeu de données. La formule qui permet de calculer le nombre initial de tuples à partir de W (le nombre d'entrepôts de la compagnie) est la suivante :

$$W + W * 10 + W * 30000 + W * 30000 + W * 9000 + W * 300000 + W * 100000 + 100000$$

Ce qui nous donne pour un nombre d'entrepôts allant de 1 à 5 :

Tableau 1 – Nombre de tuples total en fonction du nombre d'entrepôts (W)

W	**1**	**2**	**3**	**4**	**5**
Nb tuples	569 011	1 038 022	1 507 033	1 876 044	2 445 055

Pour des raisons de compromis entre la durée d'exécution de l'essai, la couverture des cas de test possibles et le souci d'avoir des résultats significatifs dans nos comparaisons d'efficience selon les SGBD, nous testerons tout d'abord avec une quantité de trois entrepôts. Ce qui équivaut à un nombre de tuples total dans la BD de 1 507 033. En effet, cela semble être le minimum pour avoir la possibilité que les transactions s'interrompent les unes les autres. Nous testerons en plus pour une quantité de cinq entrepôts pour évaluer l'influence de la taille du jeu de donnée sur la stabilité des résultats.

Une autre chose importante pour garantir l'objectivité des résultats est de s'assurer que le seul paramètre qui varie lors de l'exécution des tests est le SGBD utilisé. Les tests doivent donc

être isolés de toute autre utilisation des ressources du système, le système devant être maintenu à l'identique sous tous les autres aspects matériels et logiciels.

Cette stratégie pourra être combinée avec la stratégie TPCC.Coherence pour tester la fiabilité et la validité transactionnelle.

3.4.2 Stratégie « FonctionnalitesIsolees »

L'objectif de cette stratégie est d'évaluer l'efficience pour chaque fonctionnalité du langage. On pourra ainsi justifier d'une bonne couverture du langage. Nous effectuerons ces tests en parallèle avec les stratégies du critère de validité du langage relationnel qui s'occupe d'évaluer si le comportement des différentes fonctionnalités liées au modèle relationnel est celui attendu.

Seules les mesures des tests qui ont réussi peuvent être interprétées. Les tests sont indépendants, il n'y a donc pas besoin que tous les tests aient réussis pour pouvoir interpréter les mesures.

Les mesures à prendre sont les suivantes :

- Durée d'exécution.
- Temps d'exécution de l'UCT.
- Nombre de lectures sur le disque
- Nombre de lectures sur le disque
- Quantité de mémoire consommée
- Nombre de défauts de page

Elles nous donneront toutes les indications nécessaires pour pouvoir justifier les écarts de durée obtenus.

Comme pour la stratégie TPCC, nous nous assurerons que les mesures effectuées sont fiables en vérifiant la représentativité et la stabilité.

Pour avoir des mesures significatives, nous modifierons le jeu de données initial utilisé pour les tests de validité du langage relationnel sur lequel on exécute les différentes fonctionnalités

par un jeu de données plus conséquent. Nous ajusterons ce volume de données en fonction des résultats obtenus.

De plus, les tests doivent être isolés de toute autre utilisation des ressources du système, le système devant être maintenu à l'identique sous tous les autres aspects matériels et logiciels.

3.4.3 Stratégie « ApplicationExistante »

L'objectif de cette stratégie est d'exécuter des programmes entiers ou des extraits d'applications réelles et en évaluer l'efficience. Cela nous permettra d'étendre notre couverture à des applications non commerciales et de combiner les fonctionnalités du langage plutôt que de les isoler. On aura donc une couverture partielle des chemins possibles fondée sur des cas d'utilisation réels et donc représentatifs.

Les résultats des tests (échec ou réussite) seront uniquement des résultats pour valider le déroulement correct de l'exécution des tests. Si un des tests ne s'est pas exécuté correctement alors on ne peut pas faire confiance aux mesures obtenues. Il faut alors identifier la source du problème d'exécution puis la corriger et relancer les tests.

Les mesures à prendre sont les suivantes :

- Durée d'exécution.
- Temps d'exécution de l'UCT.
- Nombre de lectures sur le disque
- Nombre de lectures sur le disque
- Quantité de mémoire consommée
- Nombre de défauts de page

Elles nous donneront toutes les indications nécessaires pour pouvoir justifier les écarts de durée obtenus.

Comme pour les stratégies précédentes, nous nous assurerons que les mesures effectuées sont fiables en vérifiant la représentativité et la stabilité.

Pour y parvenir, les tests doivent être isolés de toute autre utilisation des ressources du système, le système devant être maintenu à l'identique sous tous les autres aspects matériels et logiciels.

3.5 Expressivité

L'expressivité possède deux aspects. Le premier concerne ce qu'il est possible d'exprimer avec le langage, peu importe la manière de l'exprimer. Ce sera l'objet de notre première stratégie « FonctionnalitesExprimables », car c'est l'aspect le plus important de l'expressivité.

Le deuxième aspect concerne la manière d'exprimer les fonctionnalités. Cet aspect est souvent négligé, probablement parce qu'il est très difficile de déterminer si un vocabulaire ou une syntaxe est plus simple à comprendre que d'autres. De notre côté, nous avons défini trois stratégies différentes pour évaluer au mieux cet aspect : « MesuresComplexite », « NiveauAbstraction » et « ComplexiteSyntaxique ». La stratégie « MesuresComplexite » consiste à utiliser des métriques de complexité éprouvées sur un lot de programme. La stratégie « NiveauAbstraction » consiste à analyser de manière qualitative le niveau d'abstraction du langage. La stratégie « ComplexiteSyntaxique » consiste à analyser les éléments syntaxiques nécessaires pour exprimer des fonctionnalités primordiales du langage. Les trois stratégies évaluent la même chose, mais avec des approches différentes. Cela nous permet d'avoir une meilleure couverture de test. De plus, cela nous permet de valider nos stratégies de test en vérifiant que les résultats obtenus sont cohérents.

3.5.1 Stratégie « FonctionnalitesExprimables »

L'objectif de cette fonctionnalité est d'analyser de manière qualitative les fonctionnalités du langage. Plus le langage permet au programmeur d'exprimer beaucoup de choses, plus il est expressif. La facilité d'expression de la fonctionnalité n'est pas prise en compte ici.

Nous distinguerons les fonctionnalités primordiales qui sont celles qui sont indispensables à un langage de base de données relationnelle et les fonctionnalités optionnelles qui sont celles

qui ajoutent au pouvoir d'expressivité, mais qui sont moins importantes que les fonctionnalités primordiales.

Voir l'Annexe C pour la liste des fonctionnalités que nous avons identifiées ainsi que leurs priorités.

Pour chaque langage, nous passerons en revue chaque fonctionnalité et nous évaluerons la présence de la fonctionnalité. Une grille d'évaluation de quatre catégories nous permettra d'évaluer le degré de présence de la fonctionnalité. Voici ces quatre catégories ;

- La fonctionnalité n'est pas présente
- La fonctionnalité est présente, mais pas au plein de ses capacités. Les éléments manquants semblent primordiaux et donc la fonctionnalité est amoindrie.
- La fonctionnalité est présente, mais pas au plein de ses capacités. Cependant, les éléments manquants ne sont pas primordiaux.
- La fonctionnalité est entièrement présente.

Le classement des fonctionnalités dans l'une ou l'autre des catégories est laissé au jugement de l'évaluateur sous réserve d'une bonne justification.

Les résultats obtenus pour les différents langages seront ensuite comparés entre eux et un classement sera établi. Ce classement sera fait à partir d'une analyse subjective des résultats, car il nous faudra juger à partir de la grille de résultats obtenus du langage le plus cohérent vis-à-vis des fonctionnalités qu'il propose. En effet, le compte du nombre de fonctionnalités présentes ne suffit pas à évaluer l'expressivité, il faut aussi qu'il y ait une certaine cohérence pour pouvoir exprimer aisément un maximum de problèmes.

3.5.2 Stratégie « MesuresComplexite »

Utiliser des mesures de complexité sur plusieurs échantillons de code pour permettre d'évaluer la facilité d'utilisation du langage (du point de vue du programmeur) et la facilité de compréhension du langage (du point de vue du lecteur) qui sont deux éléments clés de l'expressivité d'un langage.

La métrique de Halstead [19] sera tout d'abord calculée sur les différents échantillons. Cette métrique compte le nombre d'opérateurs et d'opérandes pour en déduire la difficulté ou l'effort requis pour écrire ou lire le programme.

Nous calculerons ensuite la métrique de complexité cyclomatique de McCaab [33]. Celle-ci consiste à calculer le nombre de chemins possible lors de l'exécution du programme. Plus il y a de chemins, plus il y a de cas à comprendre dans le programme et donc plus il est difficile à programmer ou à comprendre. Les fonctionnalités présentent dans le langage peuvent réduire ou augmenter cette complexité.

Pour compléter la mesure de complexité de McCaab, nous calculerons aussi la taille fonctionnelle de Wang et Shao [55]. Celle-ci se base aussi sur la complexité des structures conditionnelles, mais prend en compte plus de paramètres. Tout d'abord, selon le type de structure, un poids est affecté. Par exemple, il est plus simple de comprendre un IF que l'appel d'une fonction qui ne comporte qu'un IF, cela demande une étape de plus dans le processus de compréhension du code. Elle prend aussi en compte l'imbrication des structures, et le nombre d'entrées-sorties des différents modules.

Aucune des implémentations existantes que nous avons trouvées ne prend en compte de manière satisfaisante l'adaptation de ces métriques à un langage relationnel. Nous avons donc défini nous même les grands principes à suivre pour calculer ces métriques sur des langages relationnels en essayant de garder au mieux l'esprit de chaque métrique. Ces règles sont présentées en Annexe D.

Pour finir, nous calculerons l'indice de maintenabilité [11] qui est une mesure synthétique qui est fonction du volume de Halstead, de la complexité cyclomatique et du nombre de lignes et du nombre de commentaires. Nous mettrons de côté le terme de l'expressivité des commentaires, car nous voulons isoler l'expressivité liée au langage.

La formule pour le calculer est alors la suivante :

$$Mlwoc = 171 - 5{,}2 \times log(V) - 0{,}23 \times c(V) - 16{,}2 \times log(loc)$$

Avec **V** le volume de Halstead, **c(V)** la complexité cyclomatique et **loc** le nombre de lignes de code.

Cette mesure nous permettra à un moindre coût d'avoir une information supplémentaire plus globale puisqu'elle prend à la fois la longueur du code, les structures de contrôle et les identificateurs et opérandes.

Les mesures prises permettront de comparer les langages entre eux. Pour pouvoir effectuer un classement fiable, il faudra que les différentes mesures donnent des résultats similaires, c'est-à-dire que le classement des langages selon leur complexité ne doit pas dépendre de la mesure choisie. Une fois le classement effectué, nous pourrons avoir un aperçu des langages les plus faciles à utiliser et à comprendre.

3.5.3 Stratégie « NiveauAbstraction »

L'objectif de cette stratégie est d'évaluer le niveau d'abstraction du langage. Le niveau d'abstraction est représentatif de l'expressivité, car il permet de masquer au programmeur et au lecteur la complexité du langage machine en fournissant des mécanismes plus compréhensibles pour un humain.

Nous n'avons trouvé aucun modèle complet existant qui nous permet d'évaluer le niveau d'abstraction d'un langage. Les méthodes que nous allons proposer sont donc expérimentales et nous pourrons les valider par rapport aux résultats obtenus.

Nous évaluerons tout d'abord le niveau d'abstraction du langage de calcul.

Nous avons trois types de niveaux d'abstraction possibles :

- Axiomatique
- Fonctionnel
- Procédural

Un langage procédural demande à ce que soient détaillées toutes les étapes nécessaires pour arriver au résultat attendu, il fait donc peu appel à l'abstraction.

Un langage fonctionnel est construit uniquement à partir de fonctions au sens mathématique du terme. On ne décrit plus les étapes du programme, mais quelles sont les fonctions à appliquer pour qu'à partir des valeurs d'entrées on puisse obtenir l'unique valeur de sortie correspondante. Le niveau d'abstraction est donc un peu plus haut que pour un langage procédural.

Un langage axiomatique a uniquement besoin que le programmeur décrive ce qu'il veut faire sans avoir à définir comment le faire. C'est donc le niveau d'abstraction le plus haut.

Nous nous attendons à ce que les langages analysés comportent des portions de différents niveaux (comme c'est le cas pour le langage SQL ISO/ANSI:2003 qui comprend un langage mi-axiomatique, mi-fonctionnel (SQL) et un langage procédural (SQL/PSM)). Lors de la conception d'un nouveau langage, il sera nécessaire de pousser l'analyse un peu plus loin en justifiant les fonctionnalités dont le niveau d'abstraction pourrait être élevé.

Nous évaluerons ensuite le niveau d'abstraction du système de typage. Pour cela, nous avons utilisé la classification de Cardelli et Wegner [8] pour définir quatre niveaux d'abstraction :

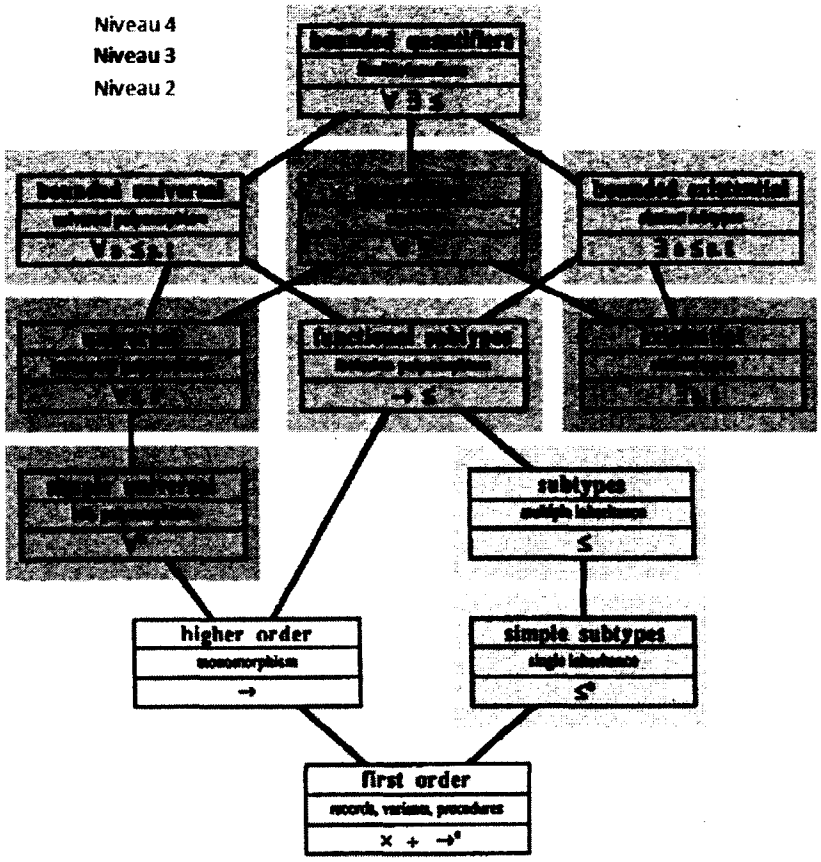

Figure 1 – Niveaux d'abstraction d'un système de typage [2]

Le premier niveau d'abstraction comprend les systèmes de typage « *Higher Order* » et « *First Order* ». Ces deux systèmes de typage permettent de construire des types à partir d'autres types.

Le deuxième niveau comprend les systèmes de typage « *Simple Subtypes* » et « *Subtypes* ». En plus du premier niveau, ils permettent de construire des types à l'aide des propriétés

---

[2] CARDELLI, L. AND WEGNER, P. 1985. On understanding types, data abstraction, and polymorphism. *ACM Comput.Surv.* 17, 471-523.

communes entre les types. On a donc un plus haut niveau d'abstraction, car l'on peut factoriser les propriétés communes.

Le troisième niveau comprend les systèmes de typage « *Simple Universal* », « *Universal* », « *Quantifiers* » et « *Existential* ». Ce niveau introduit tout d'abord le concept de polymorphisme, c'est-à-dire la possibilité qu'un objet (variable, paramètre...) possède plusieurs types selon le contexte. Cela augmente le niveau d'abstraction, car on n'a plus besoin de savoir exactement quel sera le type de l'objet pour pouvoir le manipuler, il suffit de savoir que l'objet a certaines propriétés particulières. Ainsi, les concepts d'abstraction et de généricité permettent de définir ces regroupements de types.

Le quatrième niveau comprend les systèmes de typage « *Functional Subtype* », « *Bounded Existential* », « *Bounded Universal* » et « *Bounded Quantifiers* ». Ce niveau comprend aussi des systèmes de typage polymorphique. Cependant, l'héritage ajoute au niveau d'abstraction, car l'on peut alors hiérarchiser les types et ainsi définir différents niveaux d'abstraction.

On pourrait rajouter un niveau zéro composé d'un système de typage comprenant uniquement des types prédéfinis.

De plus, on pourrait catégoriser les deuxièmes et troisièmes niveaux différemment. En effet, le niveau 2 peut déjà être vu comme une forme de polymorphisme, car un sous-type peut utiliser un opérateur défini pour son sur type de manière transparente, le contexte lui indiquant si cela est possible. On pourrait donc les placer à un même niveau d'abstraction. Cependant, nous avons choisi cette catégorisation pour différencier le polymorphisme par héritage qui est plus explicite, il faut d'abord définir l'objet comme étant le sous-type d'un autre objet pour ensuite pouvoir l'utiliser comme tel tandis que dans le niveau 3, on peut utiliser sur un objet une opération applicable à un type uniquement en vérifiant que cet objet possède les propriétés nécessaires pour être du type en question.

Ces deux approches nous permettent donc de couvrir le niveau d'abstraction du langage en ce qui concerne le langage de typage et le langage algorithmique. Pour assurer une couverture de test complète, il faudrait aussi évaluer le niveau d'abstraction du langage relationnel.

Cependant, nous n'avons pas de pistes pour établir un modèle permettant de l'évaluer. De plus, l'algèbre relationnelle en lui-même se base sur des constructions axiomatiques. Il offre donc déjà un haut niveau d'abstraction. Or, puisque tous les langages que l'on désire tester seront des langages basés sur cette algèbre relationnelle, on peut dire que le niveau d'abstraction du langage relationnel sera de toute manière élevé.

3.5.4 Stratégie « ComplexiteSyntaxique »

L'objectif de cette stratégie est de faire une analyse quantitative du nombre de mot et de règles syntaxiques requises pour exprimer les différents concepts essentiels d'un langage de BD relationnelle (et le rapport entre ces quantités).

Les concepts essentiels que nous avons identifiés sont les suivants :

- intégrité référentielle,
- intégrité axiomatique,
- opérations relationnelles de l'algèbre de base,
- création de schémas,
- création de types.

Les trois premiers sont nécessaires pour avoir un langage relationnel tel que défini par Codd. La création de schémas est nécessaire pour la définition d'un univers clos de représentation des données. Et la création de types est essentielle pour introduire le concept de typage (attributs + méthodes) dans le modèle relationnel.

L'analyse des résultats sera subjective, car nous ne possédons pas de modèle permettant d'être objectif. En effet, il ne suffit pas de dire que le nombre de règles et de mots requis pour exprimer un langage est plus grand pour en déduire que le langage est plus difficile à utiliser. Si le langage est trop minimaliste, il sera encore plus difficile à comprendre et manipuler que dans le cas où il est trop verbeux. Il y a donc un compromis à faire. Pour mieux comprendre les implications de ce compromis et pouvoir établir un modèle, il faudrait approfondir nos recherches dans le domaine des méthodes cognitives et dans l'étude des langues naturelles. Cela ne fera cependant pas l'objet de ce mémoire.

Chapitre 4

Banc d'essai

Pour faciliter la mise en œuvre et l'utilisation des essais, il nous faut mettre au point un outil pour automatiser l'exécution des tests et la collection des résultats. Cette section présente de manière détaillée les objectifs de l'outil et les choix d'implémentation retenus pour répondre aux exigences souhaitées.

4.1 Spécification des exigences

4.1.1 Objectif

L'application à développer a pour but d'automatiser l'exécution des essais. Cela signifie qu'elle doit pouvoir simplifier l'écriture et le lancement de la majorité de nos tests. Les tests devant être automatisés en priorité sont les tests de validité des données, de disponibilité des données et d'efficience. Ils sont prioritaires, car, soit ils demandent l'exécution d'un grand nombre de requêtes (trop grand pour être envoyé une par une par un opérateur humain), soit ils demandent que l'exécution des requêtes et la prise de mesure soient contrôlées de manière très précise (un opérateur humain ne pourra pas assurer cette précision).

4.1.2 Exigences fonctionnelles

F1 : L'outil doit permettre de décrire des tests, des essais et des séances.

F2 : L'outil doit permettre de vérifier la validité de la description des tests, des essais et des séances.

F3 : L'outil doit permettre de garder une trace des différentes versions des tests, des essais et des séances.

F4 : L'outil doit permettre d'exécuter un essai sur un SGBD cible.

F5 : L'outil doit permettre d'exécuter plusieurs essais en concurrences sur un SGBD cible.

F6 : L'outil doit permettre d'exécuter plusieurs essais séquentiellement sur un SGBD cible.

F7 : L'outil doit permettre de contrôler le démarrage de l'exécution des essais. Ils pourront ainsi être différés d'un temps prédéterminé ou aléatoire.

F8 : L'outil doit permettre de prendre et de consigner des mesures au niveau de la séance, au niveau de l'essai et au niveau du test. Les mesures à prendre sont les suivantes : durée d'exécution, temps UCT du processus du SGBD, nombre d'écritures sur le disque effectuées par le processus du SGBD, nombre de lectures sur le disque effectuées par le processus du SGBD, nombre de défauts de page, quantité de mémoire vive consommée par le processus du SGBD.

F9 : L'outil doit permettre d'exclure les requêtes d'initialisation et de terminaison du test de la prise de mesure.

F10 : L'outil doit permettre de consigner le résultat des tests. Si un test ne s'est pas exécuté correctement (erreur retournée par le SGBD) alors le résultat sera faux. De plus, on doit pouvoir définir dans la description d'un test des conditions de validité du test qui entraîneront un résultat faux si les conditions ne sont pas validées.

F11 : L'outil doit permettre de garder une trace des requêtes exécutées sur le SGBD cible ainsi que des erreurs d'exécution survenues.

F12 : L'outil doit permettre de générer automatiquement des requêtes d'insertion ainsi que des requêtes d'appel de procédure. Les types de données qui doivent pouvoir être générées sont les suivantes : entier, flottant, date, chaîne de caractère, tableau d'entier, tableau de flottant, tableau de date, tableau de chaîne de caractère (les tableaux sont nécessaires pour reproduire les tests de la suite TPC-C). La génération peut être aléatoire, séquentielle (pour pouvoir générer des clés) ou bien contrôlée à partir d'un domaine de valeurs. La génération

doit être reproductible, c'est-à-dire que les requêtes générées lors de deux exécutions différentes doivent être exactement les mêmes. Cela permet de garantir la reproductibilité de nos tests.

F13 : L'outil doit permettre de garder une trace des résultats et des mesures obtenus en lien avec les séances, la version des essais et la version des tests.

F14 : L'outil doit permettre de générer des rapports de test.

4.1.3 Exigences non fonctionnelles

NF1 : L'outil doit être facile à utiliser, peu importe le nombre de tests, d'essais, de séances à exécuter.

NF2 : La plateforme où est situé le SGBD testé doit pouvoir être isolée de l'exécution de l'outil.

NF3 : L'outil doit être portable. Dans l'idéal, il doit être possible d'exécuter des séances d'essai sur n'importe quel système d'exploitation et n'importe quel SGBD cible.

NF4 : L'outil doit être bien documenté, pour assurer sa maintenabilité et sa compréhension.

NF5 : L'outil ne doit pas introduire de biais dans les résultats.

4.2 Architecture

L'outil d'automatisation des essais a été séparé en deux applications.

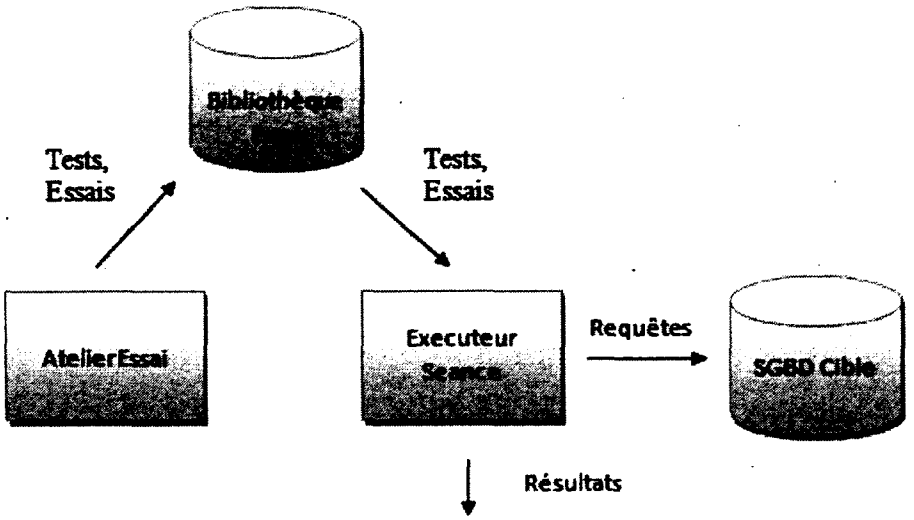

Figure 2 – Architecture globale

La première s'occupe d'établir la bibliothèque d'essai permettant de construire les séances d'essai. Concrètement, elle s'occupe donc de stocker les essais et les tests décrits par l'utilisateur dans une base de données. Nous avons nommé cette application « AtelierEssai ».

La deuxième s'occupe de récupérer les données des essais composant la séance puis d'exécuter les séances d'essai sur le SGBD cible et enfin de retourner les résultats. Nous avons nommé cette application « ExecuteurSeance ».

L'intérêt d'une telle séparation est de bien dissocier la partie définition des tests et la partie lancement des tests. Il est ainsi facile d'avoir une personne qui s'occupe de définir les tests et un opérateur qui s'occupe uniquement de l'exécution.

Le manuel de référence des applications se trouve en Annexe E.

4.2.1 Application « AtelierEssai »

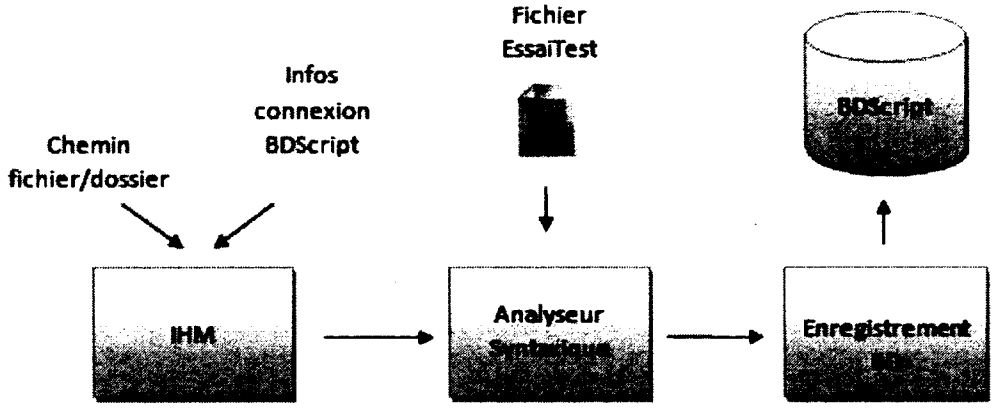

Figure 3 – Architecture de l'application « AtelierEssai »

L'application prend en entrée le chemin des fichiers décrivant les essais et les tests à enregistrer ainsi que les informations de connexion pour se connecter à la base de données (le nom d'utilisateur, le mot de passe et l'URL). Ces données sont récupérées à l'aide d'un des modules « IHM ». Deux IHM différentes sont implémentées. La première permet de passer en argument de la ligne de commande les informations. Tandis que la deuxième est une interface de dialogue avec la console.

Le module Parseur s'occupe d'analyser le contenu d'un fichier pour vérifier que sa syntaxe est conforme à la grammaire que l'on a définie pour décrire les essais et les tests (disponible dans le manuel en Annexe E). Puis s'il n'y a pas d'erreur, il convertit la description des essais et des tests en objets Java.

Pour finir, le module « EnregistrementBD » se connecte à la base de données, crée le schéma si celui-ci n'existe pas déjà et enregistre les tests et les essais.

Si le chemin fourni en entrée de l'application est un dossier, l'application repasse par les modules « AnalyseurSyntaxique » et « EnregistrementBD » pour tous les chemins de fichiers contenus dans ce dossier (à l'exclusion des fichiers cachés). Cela permet d'enregistrer un très grand nombre de fichiers facilement.

4.2.2 Application ExecuteurSeance

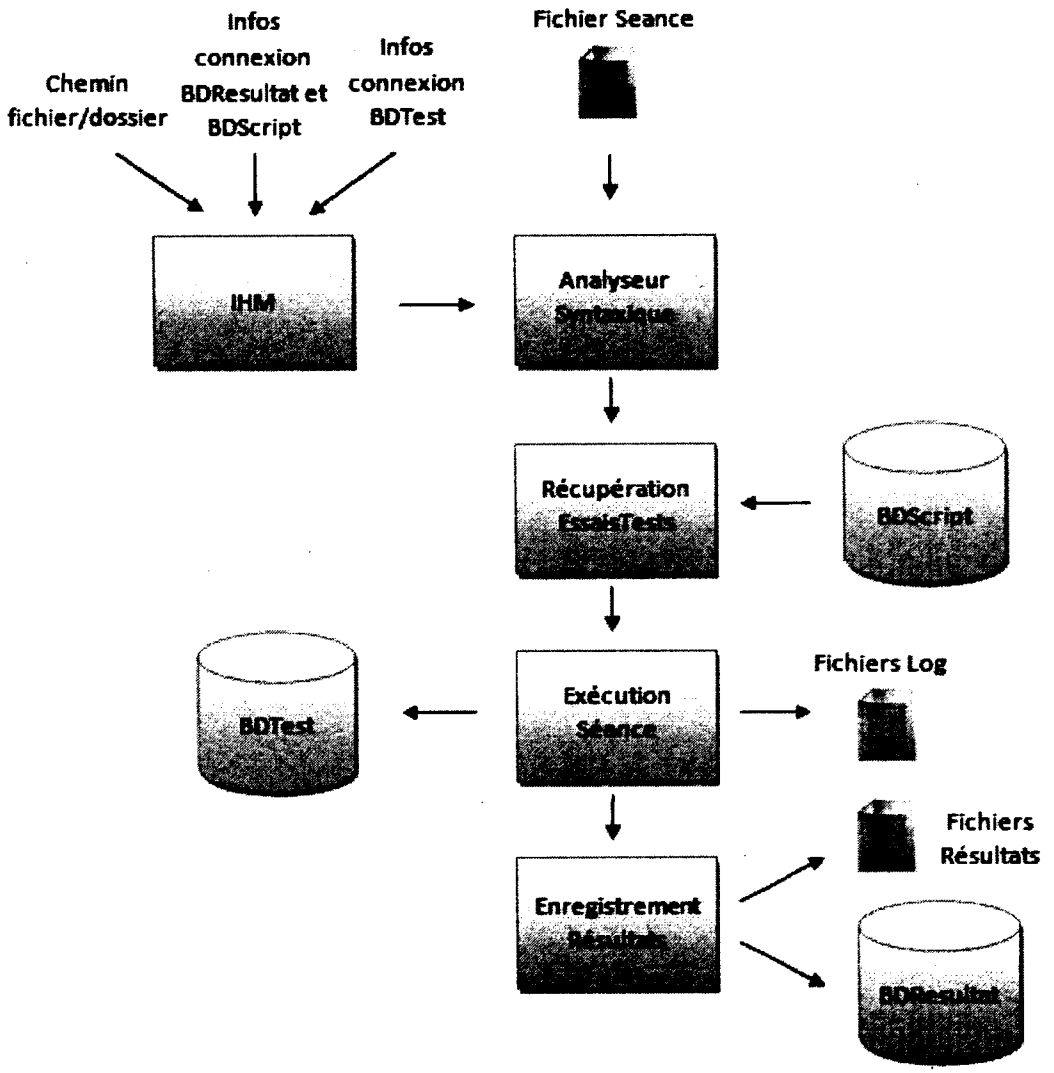

Figure 4 – Architecture de l'application « ExecuteurSeance »

L'application prend en entrée le chemin des fichiers décrivant les séances à exécuter ainsi que les informations de connexion pour se connecter aux bases de données « BDScript » (le nom d'utilisateur, le mot de passe et l'URL), « BDResultat » (le nom d'utilisateur, le mot de passe

et l'URL) et « BDTest » (le nom d'utilisateur et le mot de passe, l'URL faisant partie des scripts de manière à pouvoir contrôler les connexions de chaque instance). Ces données sont récupérées à l'aide d'un des modules IHM. Deux IHM différentes sont implémentées. La première permet de passer en argument de la ligne de commande les informations. Tandis que la deuxième est une interface de dialogue avec la console.

Le module « AnalyseurSyntaxique » s'occupe d'analyser le contenu d'un fichier pour vérifier que sa syntaxe est conforme à la grammaire que l'on a définie pour décrire les séances (disponible dans le manuel en Annexe E). Puis s'il n'y a pas d'erreur, il convertit la description de la séance en objets Java.

Le module « RécuperationEssaisTests » doit se connecter à la base de données « BDScript » où ont été stockés au préalable les tests et les essais (à l'aide de l'application « AtelierEssai »). Il récupère tout d'abord les informations concernant les essais qui composent la séance. Puis il récupère les informations concernant les tests composant ces essais. Si l'un des essais référencés n'est pas présent dans la BD, une erreur est renvoyée et la séance n'est pas exécutée.

Le module « ExecutionSéance » s'occupe de gérer l'exécution des requêtes sur la base de données « BDTest ». Toutes les requêtes exécutées ainsi que les erreurs éventuelles qui se sont produites sont consignées dans des fichiers de logs à la demande de l'utilisateur. Ceux-ci peuvent permettre de vérifier que les requêtes exécutées sont bien les bonnes ou bien d'analyser les erreurs obtenues.

Pour finir, le module « EnregistrementRésultats » s'occupe de consigner les résultats dans la « BDResultat ». Il est ainsi possible d'interroger la base de données par la suite pour extraire des informations permettant l'analyse des résultats. De plus, il est aussi possible de demander à ce qu'un rapport soit généré au format TXT ou au format CSV (en le spécifiant dans le fichier de description de la séance).

Si le chemin fournit en entrée de l'application est un dossier, l'application recommence l'exécution à partir module « AnalyseurSyntaxique » pour tous les chemins de fichiers

contenus dans ce dossier (à l'exclusion des fichiers cachés). Cela permet de lancer un très grand nombre de séances facilement.

4.3 Choix techniques

4.3.1 Langage d'implémentation

Le langage Java a été choisi pour satisfaire l'exigence NF3. En effet, Java est bien connue pour sa portabilité. Pour pouvoir exécuter du code Java, il suffit qu'il y ait une implémentation de la machine virtuelle Java pour ce système d'exploitation. Or, il existe une telle machine virtuelle pour la grande majorité des systèmes d'exploitation existants. En plus de cela, Java permet une portabilité au niveau du SGBD cible. Pour pouvoir établir une connexion à un SGBD, Java utilise des pilotes JDBC qui implémentent une interface commune. Pour pouvoir utiliser un SGBD, il suffit donc d'avoir les pilotes correspondants. Les SGBD les plus populaires fournissent de tels pilotes (Oracle, MySQL, PostgreSQL, SQLServer, DB2, SQLite, Ingres ...). De plus, pour les SGBD qui n'implémentent pas un tel pilote (par exemple TutorialD), il est toujours possible de l'intégrer facilement si celui-ci est implémenté par la suite. Cependant, il faudra veiller à ce que ce choix n'ait pas d'influence sur nos tests d'efficience. En effet, les pilotes JDBC ne sont pas optimisés pour la plateforme ciblée et risquent donc d'être moins efficients que l'utilisation de protocoles natifs (par exemple SQL*Net). Par contre le biais devrait être uniforme à travers différentes mises en œuvre de SQL (ou tout du moins équivalent aux différences d'efficience des différents protocoles natifs des SGBD). Cette situation pourrait s'avérer différente par rapport à des SGBD non-SQL, à ce moment le choix pourra être revu.

4.3.2 Description des tests, essais et séances

Nous avons choisi d'utiliser des fichiers textes pour décrire les tests, essais et séances. Cela nous permet de définir une syntaxe adaptée à nos besoins de manière simple, car il suffit d'implémenter une grammaire. Nous avons choisi de laisser la possibilité de regrouper les essais et les tests dans un même fichier, car il peut être pratique de les organiser de cette façon. Nous avons donc besoin de deux grammaires pour décrire la structure de nos fichiers.

Nous avons choisi l'outil JavaCC pour implémenter cette grammaire. Ce choix a été fait pour plusieurs raisons. La première est que nous désirons une analyse LL [30]. Cela permet d'avoir des grammaires plus simple à écrire et plus compréhensible à utiliser, car l'analyse qui est faite est proche de celle que l'être humain fait pour analyser une phrase, il commence d'abord par analyser les mots en début de phrase puis lit mot par mot pour reconnaître au fur et à mesure la syntaxe. De plus, ce type d'analyseur permet de générer automatiquement l'arbre syntaxique abstrait à partir de la grammaire concrète. La deuxième raison est que JavaCC permet de générer automatiquement du code Java à partir de la description de la grammaire. Ce qui nous permet de l'intégrer facilement au code de notre outil. Pour finir, nous l'avons choisi pour sa facilité de prise en main. La syntaxe utilisée se base sur celle de Java.

4.3.3 Stockage des tests, essais, séances et résultats

Pour implémenter les exigences F3 et F13, nous avons choisi de stocker les informations de description des tests, des essais ainsi que les résultats des séances dans une même base de données. Cela nous permet de lier les résultats des tests à leur description pour savoir précisément à quel test correspond le résultat. Nous avons choisi une base de données pour stocker ces informations pour la facilité d'utilisation qu'elle apporte. Il est très facile de définir une base de données de test puis de les lier aux résultats obtenus pour ces tests lors de l'exécution des séances. La deuxième raison est la facilité avec laquelle on peut par la suite exploiter les résultats. En effet, l'expressivité des langages relationnels permet d'exploiter facilement un grand nombre de résultats (ce qui sera notre cas). Bien entendu, pour respecter l'exigence NF2, cette base de données doit être totalement indépendante de la base de données sur laquelle nous exécuterons les tests. C'est-à-dire que ce ne doit pas être la même base de données, mais aussi qu'elle ne doit pas être située sur le même serveur.

4.3.4 Méthode de prise de mesures

L'implication des exigences non fonctionnelles sur le système de prise de mesure est très contraignante. Il doit :

- être portable vis-à-vis du système d'exploitation (NF3),

- être portable vis-à-vis du SGBD (NF3),
- permettre l'isolation de la plateforme cible (NF2).

Nous avons envisagé trois méthodes différentes pour la prise de mesure.

La première est d'interroger le SGBD pour récupérer les informations de performances stockées par les SGBD eux-mêmes. Cette méthode respecte bien l'isolation de la plateforme cible. Cependant, elle pose quelques problèmes de portabilité. A priori, cette méthode de prise de mesure est indépendante du système d'exploitation. Toutefois, en pratique, elle dépend énormément du SGBD. Ainsi, le SGBD Oracle ne fournit pas les mêmes mesures dépendamment du système d'exploitation sur lequel il est exécuté. Pour Windows, les mesures de défaut de page, du nombre d'écritures sur le disque, du nombre de lectures sur le disque et de la quantité de mémoire consommée ne sont pas accessibles. De plus, en fonction des SGBD, les mesures que l'on désire ne sont pas toujours présentes. On n'est donc pas en mesure de satisfaire l'exigence F8 pour tous les SGBD avec cette méthode. Nous avons donc choisi de l'éliminer.

La deuxième solution est d'utiliser le protocole SNMP pour interroger le serveur sur ces mesures de performances. Cette méthode respecte également l'isolation de la plateforme cible. Cependant, encore une fois la portabilité pose problème. Les catalogues de mesure de base fournie lors de l'installation du protocole dépendent du système d'exploitation, celui de Windows comporte beaucoup moins de mesures que celui de Linux. Nous avons donc cherché à ajouter des mesures au catalogue fourni par Windows. Cependant, nous avons été confrontées à un problème, sous Windows pour pouvoir ajouter les mesures que nous désirons au catalogue il faut développer une librairie Windows (DLL) qui indique comment récupérer ces mesures. Cela demanderait beaucoup de travail, nous avons donc choisi d'éliminer cette méthode. Cependant, cela permettrait d'avoir un système de prise de mesure qui respecte toutes nos exigences. Il serait donc intéressant de prendre la peine de l'implémenter dans des travaux futurs.

La troisième solution qui est celle que nous avons retenue consiste à prendre les mesures en local sur la plateforme de test. Le principal problème de cette méthode est qu'elle ne respecte

pas l'exigence d'isolation de la plateforme de test. Il y a alors un risque que les mesures soient polluées, car la machine ne peut pas allouer toutes ses ressources au SGBD, mais doit aussi s'occuper de l'exécution du programme. Cependant, cette méthode à l'avantage de ne pas influencer les mesures avec la qualité de la connexion réseau. Malgré ses défauts, nous l'avons choisie comme solution temporaire, car c'est la seule qui nous permet d'obtenir les mesures que l'on désire moyennant un temps de développement faible. Pour implémenter cette solution, nous avons utilisé une librairie existante nommée SIGAR [22] qui utilise des librairies natives (écrite en C) pour récupérer les mesures. L'utilisation de librairies natives implique que la portabilité vis-à-vis du système d'exploitation n'est pas totale. Cependant, ces librairies ont été implémentées pour un bon nombre de systèmes d'exploitation (Linux, Windows, Solaris, FreeBSD, MacOS, Aix et HPux). Au niveau de la portabilité vis-à-vis du SGBD, SIGAR fournit la possibilité de récupérer des mesures relatives à un processus identifié auparavant. Il suffit donc d'identifier le nom du processus du SGBD pour obtenir les mesures le concernant.

Pour pouvoir faire évoluer ce système de prise de mesure lors de travaux futurs, nous laisserons la possibilité d'ajouter des modules de prises de mesures différents pour laisser le choix à l'utilisateur de celui qui lui semble le plus convenable pour son utilisation.

4.3.5 Génération de jeu de données

Nous avons choisi d'effectuer la génération des requêtes au fur et à mesure de leur exécution. Ce choix a été fait pour éviter d'utiliser trop de mémoire en générant toutes les données d'un seul coup, car, en pratique, le nombre de requêtes générées par les générateurs doit souvent être de l'ordre de plusieurs millions pour être représentatif d'applications réelles. Nous aurions également pu choisir de générer toutes les requêtes avant l'exécution et de les enregistrer dans un fichier texte, mais nous ne l'avons pas retenu pour minimiser la pollution des mesures de lecture et d'écriture.

Pour garantir la reproductibilité des jeux de données générés (F12), nous avons utilisé le caractère pseudo-aléatoire de la fonction **random** en Java. Cette fonction est basée sur un nombre initial (le germe) à partir duquel est générée une suite de nombres. Il suffit donc de

fixer le germe pour reproduire à chaque exécution la même suite. Nous avons choisi de placer la reproductibilité au niveau des séances et non des essais. C'est-à-dire que deux instances d'un essai exécuté dans une même séance peuvent ne pas avoir les mêmes valeurs, mais deux mêmes séances exécutées à deux moments différents auront les mêmes valeurs pour les données générées. Cela permet de diversifier les données dans le cas de deux exécutions en concurrence d'un même essai.

Ces deux choix sont critiquables, car la reproductibilité de nos séances n'est pas totale. En effet, dans le cas d'une exécution en concurrence, vu que nous ne contrôlons pas la manière dont les **Threads** en Java sont exécutés et que la génération se fait au fur et à mesure de l'exécution, les données générées ne sont pas toujours les mêmes à chaque séance. Cependant, nous avons découvert ce problème que tardivement dans la phase de développement, nous avons donc décidé de ne pas le corriger, mais il faudra le faire dans des travaux futurs.

4.3.6 Traduction automatique des requêtes

Pour satisfaire l'exigence NF1, nous avons décidé de faire en sorte que les requêtes de test puissent être traduites automatiquement dans les différents langages des SGBD cible. En effet, l'objectif est de comparer les résultats des tests entre différents SGBD, la plupart des tests vont donc être exécutés sur différents SGBD. Cependant, étant donné qu'aucun des SGBD n'implémente le même langage (même les SGBD qui se basent sur la norme SQL présentent des variantes syntaxiques), cela signifie que l'utilisateur va avoir à réécrire plusieurs fois les mêmes tests dans les différents langages. Étant donné que le nombre de tests définis dans la spécification des essais est de plus de 500, cela représente beaucoup de travail de devoir faire cette traduction manuellement. Tandis que si on intègre la possibilité d'ajouter un module de traduction pour chaque langage cible, ce travail pourra être fait automatiquement moyennant l'implémentation d'un traducteur. Le langage initial choisi pour implémenter les tests est le langage défini dans la norme ISO/ANSI SQL:2003, car le langage SQL est le langage relationnel le plus utilisé.

Dans le cadre de la maîtrise, nous n'avons pas eu le temps d'implémenter ce traducteur. Nous avons donc fait la traduction manuellement. L'implémentation d'un traducteur est laissée pour des travaux futurs. D'autant plus que cela nous permettra d'envisager l'utilisation d'un autre langage que SQL comme langage source, par exemple Discipulus.

4.3.7 Documentation de l'application

Pour bien documenter l'application (NF4), nous avons tout d'abord choisi d'utiliser Javadoc pour documenter la structure du code. En plus de cela, les portions de code qui méritent d'être expliquées seront annotées avec des commentaires. Un document présentant les choix techniques et l'architecture du programme permettra de comprendre le fonctionnement du programme sans entrer dans le code. Pour finir, un manuel de référence complet sera produit pour que l'utilisateur puisse utiliser et tester le programme simplement.

Chapitre 5

Résultats

Les SGBD que nous avons choisis de tester sont les suivants :

- Oracle 10g Enterprise Edition (Release 10.2.0.1.0)
- PostgreSQL 9.1.0

Ces deux SGBD sont basés sur le langage SQL, par conséquent ils ne reflètent pas la variété des langages que notre suite de tests permet d'évaluer. Cependant, ils ont été choisis, car leur complétude et la possibilité de les interfacer facilement avec notre outil d'automatisation (ils fournissent tous deux un pilote JDBC) nous permettent d'obtenir des résultats pour la majorité des stratégies définies.

Pour chaque critère évalué, nous ferons tout d'abord une analyse des résultats pour chaque langage testé résumant les lacunes identifiées et leurs conséquences. Puis nous comparerons les SGBD entre eux.

5.1 Validité des données

Les tests de validité du langage relationnel ont pour objectif que chaque fonctionnalité relationnelle livre le résultat attendu. Les quatre stratégies développées dans la section 3.1 du Chapitre 3 ont été regroupées pour faciliter l'interprétation des résultats, car elles sont très liées.

Le langage relationnel étant la base des langages que nous testons, les tests de validité du langage relationnel sont les premiers à avoir été effectués, car ils conditionnent les autres tests.

Puisque certains tests ont échoué pour une cause commune, nous les avons regroupés aux fins d'interprétation des résultats. Cela veut aussi dire que lors de la comparaison de la validité des différents SGBD, il ne faut pas uniquement tenir compte du nombre de tests qui ont échoué, car cela peut se ramener à une même erreur.

Pour interpréter les résultats obtenus, nous nous baserons sur les fondements de l'algèbre relationnelle tels qu'établis par Codd [10], sur l'interprétation proposée par Date [13] et sur la norme ISO/ANSI SQL:2003 [24]. Dans certains cas, les résultats attendus diffèrent selon la référence utilisée

5.1.1 Oracle

Le tableau ci-dessous regroupe les tests qui ont échoué lors de l'exécution sur le SGBD Oracle 10 g Enterprise Edition (Release 10.2.0.1.0) :

Tableau 2 – Tests de validité qui ont échoué sur le SGBD Oracle

NomTest
FonctionnalitesRelationnelles.ProduitCartesien.ConflitNom1.Oracle
FonctionnalitesRelationnelles.ProduitCartesien.ConflitNom2.Oracle
FonctionnalitesRelationnelles.JointureTheta.ConflitNom1.Oracle
FonctionnalitesRelationnelles.JointureTheta.ConflitNom2.Oracle
FonctionnalitesRelationnelles.LeftOuterJoin.ConflitNom1.Oracle
FonctionnalitesRelationnelles.LeftOuterJoin.ConflitNom2.Oracle
FonctionnalitesRelationnelles.RightOuterJoin.ConflitNom1.Oracle
FonctionnalitesRelationnelles.RightOuterJoin.ConflitNom2.Oracle
FonctionnalitesRelationnelles.FullOuterJoin.ConflitNom1.Oracle
FonctionnalitesRelationnelles.FullOuterJoin.ConflitNom2.Oracle
FonctionnalitesRelationnelles.Renommage.Colonne.NomExistant1.Oracle
FonctionnalitesRelationnelles.Renommage.Colonne.NomExistant2.Oracle
FonctionnalitesRelationnelles.FullOuterJoin.ResultatL0CN.Oracle
FonctionnalitesRelationnelles.FullOuterJoin.ResultatL1CN.Oracle
FonctionnalitesRelationnelles.FullOuterJoin.ResultatLNCN.Oracle
FonctionnalitesRelationnelles.FullOuterJoin.Doublons1.Oracle

FonctionnalitesRelationnelles.FullOuterJoin.Doublons2.Oracle
FonctionnalitesRelationnelles.FullOuterJoin.Null.Oracle
FonctionnalitesRelationnelles.Comptage.L0.Oracle
FonctionnalitesRelationnelles.Comptage.L1.Oracle
FonctionnalitesRelationnelles.Comptage.LN.Oracle
FonctionnalitesRelationnelles.Comptage.Doublons1.Oracle
FonctionnalitesRelationnelles.Comptage.Doublons2.Oracle
FonctionnalitesRelationnelles.Moyenne.L0.Oracle
FonctionnalitesRelationnelles.Moyenne.L1.Oracle
FonctionnalitesRelationnelles.Moyenne.LN.Oracle
FonctionnalitesRelationnelles.Moyenne.VirguleFixe.Oracle
FonctionnalitesRelationnelles.Moyenne.Doublons.Oracle
FonctionnalitesRelationnelles.Moyenne.UnNull.Oracle
FonctionnalitesRelationnelles.Moyenne.TousNull.Oracle
FonctionnalitesRelationnelles.Somme.L0.Oracle
FonctionnalitesRelationnelles.Somme.L1.Oracle
FonctionnalitesRelationnelles.Somme.LN.Oracle
FonctionnalitesRelationnelles.Somme.VirguleFixe.Oracle
FonctionnalitesRelationnelles.Somme.Doublons.Oracle
FonctionnalitesRelationnelles.Somme.UnNull.Oracle
FonctionnalitesRelationnelles.Somme.TousNull.Oracle
FonctionnalitesRelationnelles.Minimum.L0.Oracle
FonctionnalitesRelationnelles.Minimum.L1.Oracle
FonctionnalitesRelationnelles.Minimum.LN.Oracle
FonctionnalitesRelationnelles.Minimum.VirguleFixe.Oracle
FonctionnalitesRelationnelles.Minimum.VirguleFlottante.Oracle
FonctionnalitesRelationnelles.Minimum.Doublons.Oracle
FonctionnalitesRelationnelles.Minimum.UnNull.Oracle
FonctionnalitesRelationnelles.Minimum.TousNull.Oracle
FonctionnalitesRelationnelles.Maximum.L0.Oracle
FonctionnalitesRelationnelles.Maximum.L1.Oracle
FonctionnalitesRelationnelles.Maximum.LN.Oracle
FonctionnalitesRelationnelles.Maximum.VirguleFixe.Oracle
FonctionnalitesRelationnelles.Maximum.VirguleFlottante.Oracle
FonctionnalitesRelationnelles.Maximum.Doublons.Oracle
FonctionnalitesRelationnelles.Maximum.UnNull.Oracle
FonctionnalitesRelationnelles.Maximum.TousNull.Oracle

Dénomination des colonnes

La première erreur a été révélée par plusieurs de nos tests. Différentes situations permettent d'obtenir comme résultat une relation composée de colonnes ayant le même nom. Les situations où l'on a détecté cette erreur sont les suivantes :

- opération de produit cartésien entre deux relations comportant des colonnes qui ont le même nom ;
- opération de jointure thêta entre deux relations comportant des colonnes qui ont le même nom ;
- opération de jointure externe à droite entre deux relations comportant des colonnes qui ont le même nom ;
- opération de jointure externe à gauche entre deux relations comportant des colonnes qui ont le même nom ;
- opération de jointure externe totale entre deux relations comportant des colonnes qui ont le même nom ;
- opération de renommage de colonne avec un nom déjà porté par une autre colonne de la relation.

La norme n'indique pas quel doit être le comportement du SGBD dans ces cas particuliers. Cependant, il est dit que tous les noms des attributs d'une relation doivent être distincts [24] (p49).

Lors de l'enregistrement d'une relation dans la BD, Oracle vérifie bien que les noms de colonne sont distincts, sinon il lève une exception et empêche l'enregistrement de la relation. Si le conflit de nom ne concerne qu'un résultat temporaire, que l'on ne stocke pas et que l'on n'utilise pas alors il est autorisé. Ce problème de validité est très gênant, car même lors de l'obtention d'un résultat temporaire, l'utilisateur a besoin de savoir à quoi correspond chaque colonne retournée, et ce, sans avoir de connaissances sur les données contenues dans la BD.

Types retournés par les opérateurs d'agrégation

La norme indique certaines contraintes sur les types de donnée à retourner par les opérateurs d'agrégation de comptage, de moyenne, de somme, de maximum et de minimum [24] (p505) :

- Pour une opération de comptage, le type de données retourné doit être un numérique exacte.
- Pour une opération de maximum ou de minimum, le type de donnée du résultat doit être le même que celui de l'attribut sur lequel on fait l'opération.
- Pour une opération de somme, si l'attribut sur lequel on effectue l'opération est un numérique exacte, alors le type retourné doit être un numérique exacte avec la même échelle. Si l'attribut sur lequel on effectue l'opération est un numérique flottant avec une précision p, alors le type retourné doit être un numérique flottant avec une précision supérieure ou égale à p.
- Pour une opération de moyenne, si l'attribut sur lequel on effectue l'opération est un numérique exacte de précision p et d'échelle s, alors le type retourné doit être un numérique exacte avec une précision supérieure ou égale à p et une échelle supérieure ou égale à s. Si l'attribut sur lequel on effectue l'opération est un numérique flottant avec une précision p, alors le type retourné doit être un numérique flottant avec une précision supérieure ou égale à p.

Or, dans tous les cas (peu importe l'opération et le type de l'attribut), Oracle retourne un type numérique flottant. Le problème est alors que si l'utilisateur veut utiliser ce résultat pour faire des calculs ou des comparaisons, les résultats obtenus risquent de ne pas être ceux escomptés. Il est cependant possible de contourner ce problème en convertissant le résultat vers un type exact.

Type retourné par l'opérateur de jointure externe totale

On retrouve aussi un problème concernant le type de donnée retournée pour l'opérateur de jointure externe totale (**FULL OUTTER JOIN**). Les types de données numériques retournés par celui-ci ne sont pas exactement les mêmes que les types de données des tables dont on fait la jointure, certains types sont transformés en flottants. La norme n'est pas très claire à ce sujet. Cependant, nous le considérons comme une erreur, car les opérations de jointure ont pour vocation de combiner les données contenues dans plusieurs tables, mais pas de les modifier. Il est donc important de conserver le même type de donnée lors de cette opération. Cette erreur est d'autant plus étrange que les autres opérations de jointure ont elles un comportement valide. Puisque le type de donnée utilisé pour le résultat (type numérique flottant) est un type qui englobe les autres types numériques d'Oracle, cette erreur ne peut pas engendrer de perte de donnée lors de la conversion de type. Cependant, la conséquence est

qu'il y a une perte de précision en ce qui concerne le domaine de définition du type. Par exemple si le type était un entier, sa transformation en flottant implique que la valeur 1.1 est désormais autorisée. Ce qui peut être dangereux.

Type retourné par l'opérateur de fermeture transitive

Nous avons obtenu un autre cas étrange en ce qui concerne les types de données retournées par l'opérateur de fermeture transitive : le type de donnée du résultat n'est pas le même que le type de données de la table sur laquelle on fait l'opération. Celui-ci n'apparaît pas dans la liste des tests qui ont échoué, car il y a un flou sur le résultat attendu. L'opérateur de fermeture transitive n'est pas spécifié dans la norme SQL-2003. Il existe un opérateur dans la norme qui permet d'implémenter la fermeture transitive (**WITH RECURSIVE**), cependant celui-ci n'existe pas sous Oracle. Oracle fournit toutefois un opérateur qui lui est propre (**CONNECT BY**) qui permet d'implémenter ces tests. Nous ne pouvons donc pas nous baser sur la norme pour justifier le résultat attendu. Cependant, si l'on se réfère aux spécifications du langage Tutorial D, qui lui définit clairement l'opération de fermeture transitive, le résultat devrait être une relation avec exactement le même en-tête [13] (p46). Il nous paraît donc important de mettre en avant ce résultat même si nous ne le considérons pas comme une erreur.

Compatibilité pour les opérateurs ensemblistes (UNION, INTERSECT, MINUS)

Pour finir, certains tests ne figurent pas parmi les tests qui ont échoué alors qu'ils sont pourtant invalides par rapport à l'algèbre relationnelle de Codd. Ces tests permettent de vérifier que lors d'opérations ensemblistes, pour considérer les relations compatibles, il faut que leurs colonnes soient définies dans le même ordre. La raison pour laquelle nous obtenons des résultats valides pour ces tests est que nous les avons définis par rapport à la norme SQL qui va à l'encontre du modèle de Codd qui lui se base sur les noms et les types des colonnes pour déterminer la compatibilité. Les résultats de ces tests dépendent donc de si l'on veut vérifier la validité vis-à-vis de la norme SQL ou de l'algèbre relationnelle de Codd sur laquelle elle se base.

5.1.2 PostgreSQL

Le tableau ci-dessous regroupe les tests qui ont échoué lors de l'exécution sur le SGBD PostgreSQL 9.1.0 :

Tableau 3 – Tests de validité qui ont échoué sur le SGBD PostgreSQL

NomTest
FonctionnalitesAlgorithmiques.ModificationFonction.RelationNonValide.PostgreSQL
FonctionnalitesRelationnelles.Union.ResultatL1C0.PostgreSQL
FonctionnalitesRelationnelles.Intersection.ResultatL0C0.PostgreSQL
FonctionnalitesRelationnelles.Intersection.ResultatL1C0.PostgreSQL
FonctionnalitesRelationnelles.IntersectionAvecDoublons.ResultatL0C0.PostgreSQL
FonctionnalitesRelationnelles.IntersectionAvecDoublons.ResultatL1C0.PostgreSQL
FonctionnalitesRelationnelles.IntersectionAvecDoublons.ResultatLNC0.PostgreSQL
FonctionnalitesRelationnelles.Difference.ResultatL0C0.PostgreSQL
FonctionnalitesRelationnelles.Difference.ResultatL1C0.PostgreSQL
FonctionnalitesRelationnelles.DifferenceAvecDoublons.ResultatL0C0.PostgreSQL
FonctionnalitesRelationnelles.DifferenceAvecDoublons.ResultatL1C0.PostgreSQL
FonctionnalitesRelationnelles.ProduitCartesien.ConflitNom1.PostgreSQL
FonctionnalitesRelationnelles.ProduitCartesien.ConflitNom2.PostgreSQL
FonctionnalitesRelationnelles.JointureTheta.ConflitNom1.PostgreSQL
FonctionnalitesRelationnelles.JointureTheta.ConflitNom2.PostgreSQL
FonctionnalitesRelationnelles.LeftOuterJoin.ConflitNom1.PostgreSQL
FonctionnalitesRelationnelles.LeftOuterJoin.ConflitNom2.PostgreSQL
FonctionnalitesRelationnelles.RightOuterJoin.ConflitNom1.PostgreSQL
FonctionnalitesRelationnelles.RightOuterJoin.ConflitNom2.PostgreSQL
FonctionnalitesRelationnelles.FullOuterJoin.ConflitNom1.PostgreSQL
FonctionnalitesRelationnelles.FullOuterJoin.ConflitNom2.PostgreSQL
FonctionnalitesRelationnelles.Renommage.Colonne.NomExistant1.PostgreSQL
FonctionnalitesRelationnelles.Renommage.Colonne.NomExistant2.PostgreSQL

Modification du corps d'une fonction utilisée dans une contrainte

Même dans le cas où la nouvelle contrainte d'intégrité utilisant la fonction ne valide plus les données déjà présentes dans la relation, PostgreSQL autorise la modification du corps de la fonction.

La norme SQL ne parle pas de ce cas particulier, car elle n'autorise pas la modification du corps d'une fonction. Cependant, il est bien spécifié que si le mode de vérification est immédiat (ce qui est le mode de vérification par défaut), alors la contrainte doit être vérifiée pour chaque requête SQL exécutée [24] (p502). De plus, la norme indique que pour vérifier la validité d'une contrainte **CHECK**, on vérifie que la requête suivante ne retourne pas de résultat : **EXISTS (SELECT * FROM table WHERE NOT (condition))** [24] (p544). Ce qui signifie que les contraintes d'intégrité **CHECK** vérifient bien que les données déjà présentes dans la table doivent être validées et non pas uniquement les nouvelles données insérées.

Bien que ce cas particulier ne puisse pas se produire si on ne considère que les fonctionnalités présentes dans la norme, on peut extrapoler les prescriptions en respectant l'esprit pour en déduire le comportement attendu. Si l'on veut garantir l'intégrité des données en toute situation, il faudrait donc que l'instruction de modification de la fonction échoue. Les conséquences d'un tel comportement pourraient être très gênantes si, par exemple, on modifie la contrainte d'intégrité de manière à s'assurer que la valeur 1 n'est pas présente dans la table (on veut donc empêcher l'insertion de la valeur 1), mais que l'état initial n'est pas vérifié et qu'il enfreint la contrainte. On ne pourra alors plus insérer une valeur, car, la table comprenant déjà une valeur 1, la contrainte sera toujours fausse même si la nouvelle valeur à insérer ne vaut pas 1.

Opérations ensemblistes impliquant des relations sans colonnes

Ni la norme SQL ni l'algèbre de Codd ne mentionnent ce cas particulier, car ils considèrent qu'une relation doit toujours avoir au moins une colonne [24] (p51) [10] (p20). Cependant PostgreSQL autorise ce type de relation. Pour valider le comportement des opérations ensemblistes dans ce cas, nous nous baserons donc sur la spécification du langage Tutorial D qui lui permet d'avoir des relations ne comportant aucune colonne [13] (p158). Tutorial D précise qu'il n'existe qu'un seul tuple possible pour une table ne comportant aucune colonne. S'il y en a d'autres, ils sont alors considérés comme des doublons.

En suivant ce principe, nous présentons dans le tableau ci-dessous les résultats que l'on devrait obtenir pour le langage SQL pour chaque opérateur ensembliste (avec conservation des doublons (**ALL**) ou non). Les relations opérandes et la relation résultat sont exprimées en fonction du nombre de lignes (LN) et du nombre de colonnes (CN) de la relation :

Tableau 4 – Résultats attendus pour les opérateurs d'union, d'intersection et de différence dans le cas où l'une des relations ne comporte aucune colonne

R1	**R2**	**R1 Union R2**
L0C0	L0C0	L0C0
L2C0	L1C0	L1C0
R1	**R2**	**R1 Union All R2**
L0C0	L0C0	L0C0
L0C0	L1C0	L1C0
L1C0	L1C0	L2C0
R1	**R2**	**R1 Intersect R2**
L0C0	L1C0	L0C0
L2C0	L1C0	L1C0
R1	**R2**	**R1 Intersect All R2**
L0C0	L1C0	L0C0
L2C0	L1C0	L1C0
L2C0	L2C0	L2C0
R1	**R2**	**R1 Except R2**
L0C0	L1C0	L0C0
L2C0	L0C0	L1C0
R1	**R2**	**R1 Except All R2**
L0C0	L1C0	L0C0
L2C0	L0C0	L1C0
L2C0	L1C0	L1C0
L2C0	L0C0	L2C0

Or, PostgreSQL retourne des résultats différents, pour chacun de ces opérateurs, il adopte un comportement équivalent de celui de l'opérateur **UNION ALL**. Le comportement de l'opérateur n'est donc pas cohérent par rapport à sa définition.

Dénomination des colonnes

Cette erreur est commune à Oracle et PostgreSQL. Elle concerne la possibilité d'obtenir une relation avec plusieurs colonnes ayant le même nom. Se reporter à l'analyse des résultats pour Oracle pour plus de détails la concernant.

Compatibilité pour les opérateurs ensemblistes (UNION, INTERSECT, EXCEPT)

Comme pour Oracle, les tests qui permettent de vérifier que la compatibilité des opérations ensemblistes est définie par rapport à l'ordre des colonnes dans les relations on réussit, car ils ont été définis par rapport à la norme SQL. Cependant, si l'on se base sur l'algèbre de Codd, ils auraient dû échouer.

5.1.3 Comparaison

Nous avons détecté chez chacun des deux SGBD quatre principaux problèmes de validité, dont deux sont communs. Les problèmes de validité d'Oracle ne sont pas très graves, car ils ne risquent pas d'engendrer des problèmes d'intégrité des données.

Les problèmes de validité de PostgreSQL sont plus graves, le problème sur les opérations ensemblistes peut amener à des résultats invalides le nombre de tuples présent n'étant pas le bon. De même, tel que nous l'avons déjà expliqué, le problème de modification d'une fonction utilisée pour la vérification de contraintes peut amener à avoir des données ne respectant pas les contraintes d'intégrité ce qui pourrait même, dans le pire des cas, bloquer toute possibilité de modification de la table.

Il faut remarquer que ces erreurs de validité de PostgreSQL portent sur des fonctionnalités qui ne sont pas présentes dans Oracle. Cela n'influe en rien notre comparaison, car nous voulons évaluer la capacité du langage à retourner le résultat attendu par l'utilisateur (selon ce que prescrit la norme ou, à défaut, l'esprit du modèle relationnel). La présence ou non de fonctionnalités sera utilisée pour évaluer l'expressivité des langages.

5.2 Disponibilité

Nous rappelons que les tests de disponibilité vérifient que le mécanisme transactionnel se comporte comme prévu en garantissant les propriétés ACID.

Ces tests ont été effectués en second, car ils permettent d'assurer la validité du langage dans les cas où le mécanisme transactionnel rentre en jeu. C'est donc un complément aux tests de validité du langage relationnel.

Comme les langages que nous avons testés se basent sur la norme SQL:2003 qui définit plusieurs niveaux d'isolation possible, nous avons testé pour chaque niveau d'isolation existant que les erreurs qui ne doivent pas se produire ne se produisent pas. Le tableau ci-dessous indique pour chaque type d'erreur si elle peut se produire ou non selon les différents niveaux d'isolation définie dans la norme :

Tableau 5 – Erreurs d'isolations tolérées en fonction des niveaux d'isolation SQL

Erreur **Niveau d'isolation**	**Écriture impropre**	**Lecture impropre**	**Lecture non reproductible**	**Lecture fantôme**	**Agrégation incorrecte**
Serializable	Impossible	Impossible	Impossible	Impossible	Impossible
Repeatable Read	Impossible	Impossible	Impossible	Possible	Non défini
Read Committed	Impossible	Impossible	Possible	Possible	Non défini
Read Uncommitted	Impossible	Possible	Possible	Possible	Non défini

Les agrégations incorrectes ne sont pas mentionnées dans la norme, nous avons donc testé tous les cas.

5.2.1 Oracle

5.2.1.1 Stratégie TPCC.Atomicité

Aucun test d'atomicité n'a échoué ce qui signifie que les mécanismes permettant de construire une transaction puis de la valider ou de l'annuler fonctionnent correctement. L'atomicité est le concept de base du transactionnel, il est donc primordial que ces tests réussissent.

5.2.1.2 Stratégie TPCC.Cohérence

Aucun test de cohérence n'a échoué ce qui signifie que le mécanisme transactionnel n'engendre pas de problème d'intégrité des données.

5.2.1.3 Stratégie TPCC.Isolation

Le tableau ci-dessous indique les résultats obtenus lors de l'exécution des tests d'isolation :

Tableau 6 – Résultat des tests d'isolation pour le SGBD Oracle

Erreur / **Niveau d'isolation**	**Écriture impropre**	**Lecture impropre**	**Lecture non reproductible**	**Lecture fantôme**	**Agrégation incorrecte**
Serializable	Échec[3]	Réussite	Réussite	Réussite	Échec
Read Committed	Réussite	Réussite			Échec

Un échec peut indiquer soit que l'exécution du test a échoué, soit que le résultat obtenu lors de l'exécution des transactions n'est pas celui attendu. Si l'erreur vient d'un problème d'exécution, nous avons indiqué en bas de page l'erreur retournée par le SGBD, car il est possible que cela indique que le SGBD a bien détecté la possibilité d'obtenir une erreur, mais n'a pas trouvé de solution pour sérialiser la transaction et ainsi l'éviter.

[3] Erreur retournée : *ORA-08177 : impossible de sérialiser l'accès pour cette transaction*

Il faut noter qu'Oracle ne propose que deux niveaux d'isolation différents, « Serializable » et « Read Committed ». Les tests ont donc été implémentés pour ces deux niveaux uniquement.

Pour ce qui est des risques d'erreurs de lecture impropre, de lecture non reproductible et de lecture fantôme, le SGBD gère correctement les différents niveaux d'isolation.

Pour les erreurs d'écriture impropre, le SGBD provoque une erreur d'exécution dans le cas du niveau d'isolation « Serializable ». Ceci indique que le comportement du SGBD est bel et bien valide, car il n'est pas possible d'avoir des cas d'écriture impropre (la transaction est annulée et c'est à l'utilisateur de la relancer plus tard). Cependant cela révèle aussi que l'algorithme de sérialisation n'a pas trouvé de solution pour sérialiser la transaction. Ceci est dû au fait que pour sérialiser la transaction, le SGBD utilise un algorithme dit « optimiste ». Cela permet d'améliorer les performances tout en étant capable dans la majorité des cas de sérialiser les transactions. De plus, la transaction que l'on n'a pas pu sérialiser n'est pas relancée automatiquement par le SGBD, car, selon le contexte d'utilisation, il est possible que celle-ci n'ayant pas été exécutée au moment prévu, elle ne soit plus utile ou plus cohérente. Le choix est donc laissé à l'utilisateur de relancer ou non la transaction qui a échoué (manuellement ou automatiquement en récupérant l'erreur). Les conséquences d'un algorithme « conservateur » sur la performance étant trop importantes, le comportement adopté est donc tout à fait légitime, l'essentiel étant que la base de données n'est pas modifiée de manière incohérente et que les résultats retournés par la transaction ne sont pas incohérents.

Pour les erreurs d'agrégation incorrecte, aucun des niveaux d'isolation ne permet de les détecter. La norme ISO/ANSI SQL:2003 ne mentionne pas ce type d'erreur, c'est probablement la raison pour laquelle elle n'est pas détectée. Elles sont mises en avant par Elmasri [16] (p557), car elles peuvent engendrer un problème de cohérence des données. Pour empêcher cette erreur, il faudrait verrouiller non pas le tuple, mais la colonne, voire même toute la BD si les valeurs dont on fait l'agrégation sont issues de plusieurs tables. Or, une telle action aurait des conséquences dramatiques sur l'accessibilité aux données. Il est donc compréhensible de ne pas le faire. Par contre, il faudrait qu'au moins pour un des

niveaux d'isolation (soit le niveau le plus sûr : « *Serializable* » qui par définition devrait les détecter) le SGBD soit en mesure de détecter que le problème est possible et annule la transaction pour éviter de mettre la BD dans un état incohérent.

5.2.1.4 Stratégie TPCC.Durabilité

Ces tests n'ont pas été effectués sur la même plateforme que les autres. Ils ont été effectués sur une machine virtuelle Windows Server 2003 avec le SGBD Oracle 10g Express Edition (Release 10.2.0.1.0) de manière à préserver notre environnement de test principal des erreurs qui peuvent être occasionnées par un arrêt brutal du SGBD et du système d'exploitation.

Les tests de durabilité ont réussi pour tous les types de dysfonctionnements testés :

- Défaillance du processus du SGBD
- Défaillance du processeur
- Défaillance de l'alimentation : débranchement du serveur
- Arrêt programmé du SGBD : ***SHUTDOWN IMMEDIATE***
- Arrêt programmé du SGBD : ***SHUTDOWN ABORT***
- Défaillance du disque dur

Pour le test avec un dysfonctionnement du disque dur, il a fallu faire une récupération des fichiers de log, car Oracle a détecté que l'un d'entre eux était corrompu. Après récupération, nous avons lancé le test pour vérifier l'état de la BD et la validité des données.

Ces tests valident donc que les données modifiées et validées avant la panne sont bien présentes dans la BD, tandis que les données modifiées, mais non validées ont bien été annulées.

Le cas d'un problème de connexion réseau n'a pas été testé, car notre programme qui exécute les tests doit être lancé sur la machine où se trouve la base de données de test pour que la prise de mesure puisse fonctionner.

De manière globale, les tests de validité des propriétés ACID sont plutôt concluants. Seuls les tests d'isolation ont révélé un réel problème dans la gestion transactionnelle, car il est

possible d'obtenir des erreurs d'agrégation incorrecte. Or de telles erreurs pourraient amener à un état incohérent des données. Il est donc très grave que cela soit permis.

5.2.2 PostgreSQL

5.2.2.1 Stratégie TPCC.Atomicité

Comme pour Oracle, aucun test d'atomicité n'a échoué ce qui signifie que les mécanismes permettant de construire une transaction puis de la valider ou de l'annuler fonctionnent correctement. L'atomicité est le concept de base du transactionnel, il est donc primordial que ces tests réussissent.

5.2.2.2 Stratégie TPCC.Cohérence

Aucun test de cohérence n'a échoué ce qui signifie que le mécanisme transactionnel n'engendre pas de problème d'intégrité des données.

5.2.2.3 Stratégie TPCC.Isolation

Le tableau ci-dessous indique les résultats obtenus lors de l'exécution des tests d'isolation :

Tableau 7 – Résultat des tests d'isolation pour le SGBD PostgreSQL

Erreur / **Niveau d'isolation**	**Écriture impropre**	**Lecture impropre**	**Lecture non reproductible**	**Lecture fantôme**	**Agrégation incorrecte**
Serializable	Échec[4]	Réussite	Réussite	Réussite	Échec
Repeatable Read	Échec4	Réussite	Réussite		Échec
Read Committed	Réussite	Réussite			Échec
Read Uncommitted	Réussite				Échec

[4] Erreur retournée : *ERREUR : n'a pas pu sérialiser un accès à cause d'une mise à jour en parallèle*

Un échec peut indiquer soit que l'exécution du test a échoué, soit que le résultat obtenu lors de l'exécution des transactions n'est pas celui attendu. Si l'erreur vient d'un problème d'exécution, nous avons indiqué en bas de page l'erreur retournée par le SGBD, car il est possible que cela indique que le SGBD a bien détecté la possibilité d'obtenir une erreur, mais n'a pas trouvé de solution pour sérialiser la transaction et ainsi l'éviter.

Les résultats sont similaires à ceux d'Oracle. On a donc une bonne gestion des erreurs de type lecture impropre, lecture non reproductible et lecture fantôme. Les erreurs de types agrégation incorrectes ne sont détectées pour aucun des niveaux d'isolation. Et enfin, les erreurs d'écriture impropre ont bien été détectées, mais n'ont pas pu être résolues pour les niveaux d'isolation *Serializable* et *Repeatable Read.*

5.2.2.4 Durabilité

Ces tests ont été effectués sur une machine virtuelle Windows Server 2003 avec le SGBD PostgreSQL 9.1.0 pour éviter de corrompre notre environnement de test principal.

Les tests de durabilité ont réussi dans le cas de la simulation des dysfonctionnements suivants :

- Défaillance du processus du SGBD
- Défaillance du processeur
- Défaillance de l'alimentation : débranchement du serveur
- Arrêt programmé du SGBD : ***STOP FAST***
- Arrêt programmé du SGBD : ***STOP IMMEDIATE***
- Défaillance du disque dur

Ces tests valident donc que les données modifiées et validées avant la panne sont bien présentes dans la BD, tandis que les données modifiées, mais non validées ont bien été annulées.

Le cas d'un problème de connexion réseau n'a pas été testé, car notre programme qui exécute les tests doit être lancé sur la machine ou se trouve la base de données de test pour que la prise de mesure puisse fonctionner.

De manière globale, les tests de validité des propriétés ACID sont plutôt concluants. Seuls les tests d'isolation ont révélé un réel problème dans la gestion du transactionnel, car il est possible d'obtenir des erreurs d'agrégation incorrecte. Or de telles erreurs pourraient amener à un état incohérent des données. Il est donc très grave que cela soit permis.

5.2.3 Comparaison

Les résultats de validité du transactionnel des deux SGBD sont équivalents. Tous deux ont un seul problème important : la non-détection des erreurs d'agrégation incorrecte.

5.3 Sécurité

Les tests de sécurité consistent à évaluer les fonctionnalités permettant de mettre en place une politique de sécurité. C'est la troisième et dernière partie de l'évaluation du langage.

La stratégie qui consiste à mettre en place différents scénarios d'intrusion n'a pas été effectuée, car elle ne s'applique qu'aux langages qui fournissent un mécanisme de détection d'intrusion. Or les deux langages que nous évaluons n'en possèdent pas.

Pour évaluer la présence de ces fonctionnalités, nous nous baserons sur la norme ISO/ANSI SQL:2003 ainsi que sur les sites officiels des différents SGBD.

5.3.1 Oracle

5.3.1.1 Authentification

Le tableau ci-dessous présente les résultats de l'évaluation des fonctionnalités liées au mécanisme d'authentification :

Tableau 8 – Résultat des tests du mécanisme d'authentification pour le SGBD Oracle

Fonctionnalité	NP	TL	PL	TP	Commentaires
Mécanisme interne				X	Présent (par mot de passe)
Mécanisme externe				X	Possible
Ajout Utilisateur				X	CREATE USER nom IDENTIFIED BY motDePasse

Modification Utilisateur			X	ALTER USER nom IDENTIFIED BY motDePasse
Suppression Utilisateur			X	DROP USER nom Auparavant, il faut supprimer les objets de son schéma
Validation des données d'authentification			X	0 % faux positifs 0 % faux négatifs
Renforcement des mots de passe			X	Complexité du mot de passe paramétrable. Renouvellement des mots de passe imposable.
Cryptage des données d'authentification		X		Mots de passe stockés de manière cryptée (algorithme Triple-DES par défaut). Communication à travers le réseau cryptée (algorithme AES modifié).
Redondance	X			Non

NP=Non Présente TL=Très Limitée PL=Peu Limitée TP=Totalement Présente

Oracle permet également d'utiliser un système d'authentification externe (par exemple celui du système d'exploitation). Nous n'avons toutefois évalué que le système d'authentification interne puisque c'est le seul dont la disponibilité est assurée et uniforme à travers toutes les plates-formes.

Le mécanisme d'authentification interne à Oracle repose sur une authentification par mot de passe. On a donc une authentification fiable, on peut être sur à 100 % que si le bon nom d'utilisateur et le bon mot de passe sont entrés alors l'utilisateur sera authentifié. Par contre ce type de mécanisme demande à ce que les mots de passe soient bien protégés du vol. Pour lutter contre le vol, Oracle stocke et communique les mots de passe de manière cryptée. Si un utilisateur interne ou externe au SGBD met la main sur le mot de passe, il ne peut pas l'utiliser sans casser l'algorithme de cryptage. Pour le stockage, l'algorithme utilisé est le Triple-DES. Cet algorithme est basé sur l'ancien standard DES [37] qui est maintenant dépassé à cause de ces faiblesses [46]. Pour combler les lacunes de cet algorithme, le Triple-DES applique trois fois l'algorithme de cryptage DES pour le rendre plus difficile à déchiffrer. Cela permet d'atteindre un niveau de sécurité suffisant dans la plupart des cas.

Cependant, il reste moins efficace que l'algorithme AES [38] qui est le nouveau standard recommandé par le NIST pour les applications qui requiert un haut niveau de sécurité. Pour les communications réseau, l'algorithme utilisé est AES qui est donc très fiable.

En plus de cela, Oracle permet de renforcer la politique des mots de passe en obligeant par exemple à n'avoir que des mots de passe comportant caractères spéciaux, lettre et chiffres ou bien en permettant de paramétrer la fréquence de renouvellement des mots de passe.

Pour finir, une seule authentification à l'aide d'un mot de passe permet de se connecter. Pour renforcer la sécurité, il faudrait pouvoir utiliser plusieurs mécanismes pour se connecter. Par exemple, un mot de passe, plus une série de questions personnelles, plus des données biométriques. Bien entendu, c'est une question de compromis, bien que cela renforce la sécurité, il ne faut pas non plus que ce soit un fardeau de se connecter au système. Pour certaines applications manipulant des données extrêmement sensibles, il faudrait offrir de telles possibilités, éventuellement à l'aide d'un mécanisme externe.

5.3.1.2 Gestion des autorisations

Le tableau ci-dessous présente les résultats de l'évaluation des fonctionnalités liées au mécanisme de gestion des autorisations :

Tableau 9 – Résultat des tests du mécanisme de gestion des autorisations pour le SGBD Oracle

Fonctionnalité	NP	TL	PL	TP	Commentaires
Mécanisme de Résolution de Conflits				X	Pas de conflit possible, car pas de droits négatifs. Il suffit qu'un droit soit accordé pour pouvoir accéder à l'objet
Vérificateur de Politique	X				Non présent. Il faut tester la politique de manière empirique en faisant attention de n'oublier aucun cas de test.
Héritage des Droits				X	Hiérarchies présentent : Rôle -> Sous Rôle System -> Table -> colonne/ligne
Regroupement de Droits				X	CREATE ROLE
Regroupement	X				Non présent

d'Utilisateurs				
Droits Prédéfinis			X	166 privilèges système. 24 privilèges au niveau des objets. Tous les privilèges primordiaux sont présents à part les droits relatifs aux lignes le droit de lecture relatif à une colonne.
Groupe de Droits Prédéfinis			X	DBA, Owner et autres Le rôle OWNER est donné à l'utilisateur qui possède le schéma et donc pas forcement au créateur de l'objet.
Politiques Prédéfinies	X			Aucunes
Politiques Personnalisables			X	Présent
Attribution de droits			X	Présent
Granularité de l'Attribution des Droits		X		Granularité au niveau des lignes : *Virtual Private Database*. Granularité au niveau de la sélection des colonnes : utiliser des vues
Droits positifs et négatifs	X			Uniquement des droits positifs que l'on ajoute et supprime.

NP=Non Présente TL=Très Limité PL=Peu Limité TP=Totalement Présente

Le mécanisme permettant de gérer les autorisations manque de fiabilité. En effet, il n'y a pas de moyen qui permette de vérifier que la politique de sécurité mise en place agit bien comme prévu. C'est à l'administrateur de faire tous les tests nécessaires. Si l'on a affaire à une grosse BD qui interagit avec de nombreux utilisateurs qui ont des droits très variés, il sera très difficile d'arriver à une couverture de test totale. Or, il est important de s'assurer qu'il n'y a aucune faille dans la politique mise en place, la moindre faille pourrait avoir des conséquences dramatiques sur la confidentialité et l'intégrité des données.

Au niveau de la flexibilité, le mécanisme est très complet. On peut avoir pour chacun des utilisateurs de la BD des droits différents sur tous les niveaux possibles des objets. Seul le droit SELECT ne peut être défini au niveau de la colonne, mais il est possible d'utiliser les vues pour combler cette lacune. Pour la granularité au niveau des lignes, il faut utiliser le mécanisme de *Virtual Private Database* (non disponible dans les versions *Express*) qui

fonctionne en ajoutant une clause WHERE implicite à chaque requête déterminant si la ligne sera accessible ou non.

En ce qui concerne la simplicité, quelques mécanismes qui permettraient de l'améliorer manquent. Tout d'abord, aucune politique prédéfinie ne peut être appliquée (nous ne prenons pas en compte l'extension *Oracle Labeled Security* qui le permet, car elle ne fait pas partie du produit de base). Ceci permettrait grandement de faciliter la définition de politiques génériques tout en permettant de définir une politique personnalisée par ailleurs pour les cas où les politiques génériques ne s'appliqueraient pas.

Ensuite en ce qui concerne les droits prédéfinis, aucun droit au niveau des lignes n'est présent. Il faut passer par la création d'une procédure retournant la condition WHERE implicite pour pouvoir définir une règle d'accès à ce niveau. Cela rend la tâche un peu plus compliquée.

Au niveau des rôles prédéfinis, les rôles DBA et OWNER (que nous avons considérés comme primordiaux, car utiles peu importe l'utilisation de la BD) sont bien présents. Cependant, le rôle OWNER ne peut être donné à quelqu'un, il est forcement attribué au propriétaire du schéma dans lequel se trouve l'objet. Donc si l'on désire donner un objet à un autre utilisateur, il faut le copier dans son schéma. De plus, puisqu'il faut supprimer le schéma pour pouvoir supprimer l'utilisateur, il n'est pas possible d'avoir un objet sans propriétaire.

Pour finir, il n'y a pas de mécanisme spécifique permettant de faire des groupes d'utilisateur. Mais les rôles permettent de lier un groupe d'utilisateur à un groupe de droits et il est possible d'avoir une hiérarchie des rôles. Il suffit donc de définir le groupe d'utilisateur en associant les utilisateurs à un rôle puis d'attribuer à ce rôle les rôles que l'on veut donner au groupe d'utilisateur. Un rôle peut donc se comporter exactement comme un groupe d'utilisateur.

De manière globale, il est assez complexe de définir une politique qui demande un haut niveau de granularité. Ceci est dû au compromis entre flexibilité et simplicité qui fait que plus l'on veut pouvoir définir des règles précises, plus ce sera complexe de le faire. Oracle a choisi

d'avoir une grande flexibilité aux dépens de la simplicité (encore une fois on ne considère pas l'extension Oracle Labeled Security qui apporte un plus haut niveau d'abstraction).

5.3.1.3 Audit

Le tableau ci-dessous présente les résultats de l'évaluation des fonctionnalités liées au mécanisme d'audit :

Tableau 10 – Résultat des tests du mécanisme d'audit pour le SGBD Oracle

Fonctionnalité	NP	TL	PL	TP	Commentaires
Présence mécanisme d'audit				X	Présent.
Expressivité du mécanisme d'audit		X			Peu expressif.
Flexibilité du mécanisme d'audit				X	Granularité fine.
Fiabilité du déclenchement de l'algorithme de détection	X				Pas de mécanisme de détection
Fiabilité de l'algorithme de détection	X				Pas de mécanisme de détection
Fiabilité de la réaction	X				Pas de mécanisme de réaction

NP=Non Présente TL=Très Limité PL=Peu Limité TP=Totalement Présent

Le mécanisme d'audit d'Oracle permet d'automatiser la création de logs des opérations effectuées sur la BD. Il est possible de configurer très finement les opérations que l'on veut surveiller. Ainsi, il est possible de limiter la consignation des opérations en fonction de l'action effectuée, de l'utilisateur qui l'effectue, de l'objet sur lequel elle est effectuée. Cependant, le pendant de cette fine granularité est le faible niveau d'expressivité pour effectuer ces configurations.

Par ailleurs, Oracle ne permet pas de détecter automatiquement les comportements anormaux. C'est à l'administrateur de s'occuper d'analyser les données d'audit en définissant des procédures et **TRIGGER** ou à l'aide d'un outil externe. Or, définir soit même les règles d'analyse est assez compliqué et ne permet pas une détection très performante. De plus, il faut configurer toutes les informations à noter dans les logs ce qui est complexe si l'on a

besoin d'une grande granularité. Le mécanisme de détection d'intrusion est donc très peu fiable, car il repose sur la configuration et l'analyse faite par l'utilisateur.

5.3.1.4 Étanchéité

Dans le cadre de ce mémoire, nous n'avons pas eu le temps d'évaluer cet aspect.

5.3.2 PostgreSQL

Le tableau ci-dessous présente les résultats de l'évaluation des fonctionnalités liées au mécanisme d'authentification :

5.3.2.1 Authentification

Tableau 11 – Résultat des tests du mécanisme d'authentification pour le SGBD PostgreSQL

Fonctionnalité	NP	TL	PL	TP	Commentaires
Mécanisme interne				X	Présent (par mot de passe)
Mécanisme externe				X	Possible
Ajout Utilisateur				X	CREATE USER nom WITH PASSWORD motDePasse
Modification Utilisateur				X	ALTER USER nom WITH PASSWORD motDePasse
Suppression Utilisateur				X	DROP USER nom Auparavant, il faut supprimer les objets lui appartenant.
Validation des données d'authentification				X	0 % faux positifs 0 % faux négatifs
Renforcement des mots de passe	X				Complexité du mot de passe non paramétrable. Renouvellement des mots de passe non imposable.
Cryptage des données d'authentification		X			Mots de passe stockés de manière cryptée (MD5). Communication à travers le réseau cryptée (double cryptage à l'aide de l'algorithme md5 par défaut, SSL possible).
Redondance	X				Non

NP=Non Présente TL=Très Limitée PL=Peu Limitée TP=Totalement Présente

Comme pour Oracle, PostgreSQL permet d'utiliser un système d'authentification externe que nous n'avons pas évalué.

Comme pour Oracle, le mécanisme d'authentification interne de PostgreSQL utilise une authentification par mot de passe. L'authentification est donc fiable. De plus, PostgreSQL crypte les mots de passe lors de leur stockage et lors de leur envoi à travers le réseau. Cependant, les algorithmes de cryptage ne sont pas très fiables. Dans les deux cas, l'algorithme MD5 est utilisé. Or, il a été montrer que cet algorithme pouvait être facilement cassé [54]. PostgreSQL ne propose pas non plus de définir des règles de renforcement de mot de passe. Le risque de vol de mot de passe est donc assez élevé, ce qui est un gros problème de sécurité.

En plus de cela, il n'est pas possible d'avoir des mécanismes d'authentification redondants qui lui permettraient d'être plus sûr.

5.3.2.2 Gestion des autorisations

- Le tableau ci-dessous présente les résultats de l'évaluation des fonctionnalités liées au mécanisme de gestion des autorisations :

Tableau 12 – Résultat des tests du mécanisme de gestion des autorisations pour le SGBD PostgreSQL

Fonctionnalité	NP	TL	PL	TP	Commentaires
Mécanisme de Résolution de Conflits				X	Inutile. Pas de conflit possible, car pas de droits négatifs. Il suffit qu'un droit soit accordé pour pouvoir accéder à l'objet
Vérificateur de Politique	X				Non présent. Il faut tester la politique de manière empirique en faisant attention de n'oublier aucun cas de test.
Héritage des Droits			X		Hiérarchies présentent : Rôle -> Sous Rôle Table -> colonne
Regroupement de Droits				X	CREATE ROLE
Regroupement d'Utilisateurs	X				Non présent
Droits Prédéfinis			X		Il n'y a pas de privilège permettant d'attribuer

				uniquement un droit de création de table, de création de type ou de création de routine. Il n'y a pas de privilège de lecture, modification et suppression de données au niveau des tuples.
Groupe de Droits Prédéfinis			X	DBA(postgres), Owner Le rôle OWNER est donné lors de la création d'un objet à son créateur. Il peut être donné par celui-ci à un autre utilisateur.
Politiques Prédéfinies	X			Aucunes
Politiques Personnalisable			X	Présent
Attribution de droits			X	Présent
Granularité de l'Attribution des Droits		X		Le privilège concernant la création d'objets (table, type, routine) ne peut être attribué uniquement pour un type d'objet et non pour tous les objets de la BD. Les privilèges de lecture, modification et suppression de données ne sont pas possibles au niveau des lignes.
Droits positifs et négatifs	X			Uniquement des droits positifs que l'on ajoute et supprime.

NP=Non Présente TL=Très Limité PL=Peu Limité TP=Totalement Présente

Comme pour Oracle, le mécanisme permettant de gérer les autorisations manque de fiabilité. Il n'y a pas de moyen qui permette de vérifier que la politique de sécurité mise en place agit bien comme prévu. Il est donc difficile pour un administrateur de s'assurer qu'il n'y a pas d'erreurs dans les règles de sécurité qu'il a définies.

Au niveau de la flexibilité, le mécanisme manque un peu de granularité. Ainsi, il n'est pas possible d'attribuer un droit uniquement pour la création de table ou la création de routines ou la création de type. La plupart du temps, c'est une même personne qui crée tous les objets du schéma, mais on pourrait très bien vouloir autoriser uniquement l'utilisateur à créer des fonctions qui agissent sur des tables et des types définis et fixés auparavant.

Il n'est pas non plus possible d'avoir une granularité au niveau des lignes alors que c'est une fonctionnalité indispensable à certaines politiques de sécurité. On pourrait penser à combler

cette lacune en utilisant des vues qui sélectionnent les lignes accessibles puis en donnant à l'utilisateur les droits sur la vue et non sur la table. Cependant, il faut savoir qu'il y a une faille de sécurité à cette méthode.

Considérons par exemple la vue suivante qui permet de filtrer les personnes et leur numéro de téléphone :

```
CREATE VIEW phone_number AS SELECT person, phone FROM
phone_data WHERE phone NOT LIKE '6%';
```

Dans PostgreSQL, il est possible dans un SELECT d'utiliser une fonction dans la clause WHERE. On peut donc faire :

```
SELECT * FROM phone_number WHERE expose_person(person,
phone);
```

Or, pour des raisons d'optimisation, le filtrage de la vue est effectué après avoir évalué la fonction. Ce qui signifie que si la fonction « expose_person » affiche sur la sortie du SGBD les couples (« *person* », « *phone* ») passés en paramètre, alors l'utilisateur aura accès à l'ensemble des données de la table « phone_data » sans aucun filtrage. Il est donc dangereux d'utiliser ce moyen, surtout si l'utilisateur dont on veut limiter l'accès a la possibilité de créer ses propres fonctions.

En ce qui concerne la simplicité, les fonctionnalités sont similaires à celle d'Oracle, ce qui signifie qu'il y a quelques lacunes. Tout d'abord, aucune politique prédéfinie ne peut être appliquée, ce qui permettrait grandement de faciliter la définition de politique pour certains cas. Ensuite en ce qui concerne les droits prédéfinis, il manque une certaine granularité. Cependant, il n'y a pas d'autre moyen de définir un tel droit, c'est donc plutôt un problème de flexibilité.

Pour finir, il n'y a pas de mécanisme spécifique permettant de faire des groupes d'utilisateur. Mais les rôles permettent de remplacer cette fonctionnalité.

De manière globale, il est assez complexe de définir une politique qui demande un haut niveau de granularité. Ceci est dû au compromis entre flexibilité et simplicité qui fait que plus l'on veut pouvoir définir des règles précises, plus ce sera complexe de le faire. La flexibilité a été choisie aux dépens de la simplicité.

5.3.2.3 Audit

Le tableau ci-dessous présente les résultats de l'évaluation des fonctionnalités liées au mécanisme d'audit :

Tableau 13 – Résultat des tests du mécanisme d'audit pour le SGBD PostgreSQL

Fonctionnalité	NP	TL	PL	TP	Commentaires
Présence mécanisme d'audit	X				Pas de mécanisme d'audit
Expressivité du mécanisme d'audit	X				Pas de mécanisme d'audit
Flexibilité du mécanisme d'audit	X				Pas de mécanisme d'audit
Fiabilité du déclenchement de l'algorithme de détection	X				Pas de mécanisme de détection
Fiabilité de l'algorithme de détection	X				Pas de mécanisme de détection
Fiabilité de la réaction	X				Pas de mécanisme de réaction

NP=Non Présente TL=Très Limité PL=Peu Limité TP=Totalement Présente

Il n'y a pas véritablement de mécanisme d'audit dans PostgreSQL. Le seul moyen de tracer les opérations effectuées sur la BD est de définir des **TRIGGER** ce qui implique beaucoup de travail si l'on veut tout tracer. De même pour le mécanisme de détection. L'usage est donc très limité.

5.3.2.4 Étanchéité

Dans le cadre de ce mémoire, nous n'avons pas évalué cet aspect.

5.3.3 Comparaison

Au niveau du mécanisme d'authentification, les deux SGBD proposent à peu près les mêmes mécanismes. La seule différence réside dans les algorithmes de stockage utilisés lors du cryptage pour le stockage des mots de passe et la communication à travers le réseau. Oracle utilise des algorithmes de cryptage plus robustes aux attaques.

Au niveau de la gestion des autorisations, Oracle est plus complet et plus flexible. Il apporte tout d'abord une plus grande portée de définition des droits. Il permet de définir des droits au niveau des lignes tandis que PostgreSQL ne le permet pas. De plus, Oracle permet une plus grande granularité de droits. Il est par exemple possible de définir un droit de création de tables et non de routines. PostgreSQL ne permet que de définir le droit de création sans granularité au niveau des types, fonctions, tables.

Pour finir, au niveau du mécanisme d'audit, il est inexistant pour PostgreSQL. Bien qu'il soit possible de le faire soi-même à l'aide du langage SQL, cela demande beaucoup plus de travail à mettre en place. Oracle l'emporte donc sur PostgreSQL bien qu'aucun mécanisme de détection d'intrusion ne soit présent.

De manière globale, Oracle apporte quelques fonctionnalités supplémentaires qui peuvent être utiles, voire indispensables, à certaines utilisations.

5.4 Efficience

Les tests d'efficience consistent à évaluer les ressources consommées par le SGBD lors de l'utilisation de la BD. Ce critère doit être évalué après les trois critères concernant la validité du langage, car il faut d'abord s'assurer que le langage utilisé se comporte correctement avant d'en mesurer l'efficience.

Pour des raisons de temps, seuls les tests concernant la stratégie TPCC ont été exécutés dans le cadre de ce mémoire.

Les résultats d'efficience peuvent être influencés par de nombreux paramètres. C'est pourquoi nous avons défini des mesures de fiabilité dans la section 3.4 du Chapitre 3.

La première est une mesure de représentativité. Nous rappelons que pour avoir un échantillon représentatif de la population totale, il nous faudrait 385 échantillons. Cependant, étant donné que nous voulons une durée d'exécution suffisamment courte pour encourager la reproduction des résultats, nous avons revu cette quantité à la baisse et nous avons décidé

d'effectuer uniquement 20 répétitions. Un programme d'essai complémentaire devra donc être entrepris afin de confirmer les résultats préliminaires que nous avons obtenus.

La deuxième est une mesure de stabilité. Pour cela nous rappelons que nous utiliserons le coefficient de variation qui mesure la proportion de l'écart entre l'écart type et la moyenne des résultats.

De plus, nous avons choisi de faire varier certains paramètres pour pouvoir identifier la configuration qui nous apporte les résultats les plus stables.

Nous avons tout d'abord fait varier la taille du jeu de données (nombres de tuples dans les relations sur lesquelles on exécute les transactions) :

- W3 : le jeu de données représente un système composé de trois entrepôts (« *Warehouse* »). Ce qui correspond au total à un jeu de donnée de 1 507 033 tuples.
- W5 : le jeu de données représente un système composé de cinq entrepôts (« *Warehouse* »). Ce qui correspond au total à un jeu de donnée de 2 445 055 tuples.

Nous avons ensuite fait varier la plateforme d'exécution :

- P1 : Machine virtuelle Windows Serveur 2003, version 32 bits qui tourne en parallèle avec d'autres machines virtuelles dont on ne contrôle pas la charge de travail.

  Ressources allouées à la VM : 1,5 Go de RAM, processeur[5] avec 1 cœur à 2,67 GHz.
- P2 : Machine virtuelle Windows Serveur 2003, version 32 bits dont on a minimisé la charge d'exécution du serveur qui la fait tourner (tous les processus inutiles du serveur ont été stoppés et le processus de la machine virtuelle a été mis en priorité

[5] Intel® Xeon® Processor X5550

maximale). Ressources allouées à la VM : 1,5 Go de RAM. Processeur[6] avec 1 cœur à 2,83 GHz.

- P3 : Windows 7, version 32 bits en configuration directe (sans machine virtuelle). Ressources disponibles : 4 Go de RAM. Processeur[7] avec 2 cœurs à 1,83 GHz.

Voici un récapitulatif des configurations que nous avons testées pour chaque SGBD :

Tableau 14 – Différentes configurations pour l'exécution des tests d'efficience

Plateforme / **Warehouse**	**P1**	**P2**	**P3**
W3	P1.W3	P2.W3	P3.W3
W5	P1.W5	P2.W5	P3.W5

5.4.1 Oracle

Les résultats obtenus sur Oracle dans chacune des configurations sont présentés en Annexe F.

Les résultats concernant la séance avec un niveau d'isolation « *Serializable* » ne seront pas pris en compte, car le SGBD n'a pas pu sérialiser l'accès aux données et renvoie donc une erreur.

Pertinences des mesures

Certaines mesures sont en majorité instable, peu importe la configuration que l'on considère. On peut donc présumer que le problème vient de notre stratégie de prise de mesure. Ces mesures sont les suivantes : nombre de lecture sur le disque, quantité de mémoire résidente consommée, nombre de défauts de page.

---

[6] Intel® Core™2 Quad Processor Q9550

[7] Intel® Core™ Duo Processor T2400

Nous ne spéculerons pas sur les causes de cette instabilité, car aucune ne nous paraît évidente. Pour justifier l'instabilité de ces mesures, il nous faudrait faire une analyse approfondie des éléments suivants :

- fiabilité de l'API utilisée pour la prise de mesure,
- méthode de gestion des ressources faites par les SGBD analysés,
- variables influentes que nous n'avons pas prises en compte.

Dans le cadre de ce mémoire, nous ne ferons pas ces analyses.

Il nous reste donc trois mesures interprétables, la durée d'exécution, le temps d'exécution de l'UCT et le nombre d'écritures sur le disque.

Influence des configurations

Nous ne considérerons que les mesures qui ont été validées dans notre analyse du paragraphe précédent (soit les mesures de durée d'exécution, de temps d'exécution de l'UCT et du nombre d'écritures sur le disque).

Nous pouvons observer une assez grande différence de stabilité entre les mesures prises sur la plateforme P1 et les mesures prises sur les plateformes P2 et P3. Les différences matérielles entre la plateforme P1 et la plateforme P2 nous semblent négligeables. Le fait que nous ne contrôlons pas l'environnement sur lequel tourne notre machine virtuelle paraît plus vraisemblable pour expliquer cette différence. En effet, le contrôle de l'environnement permet d'éliminer des paramètres fluctuants tels que l'occupation du processeur à d'autres tâches et la contention au niveau des disques.

En ce qui concerne les plateformes P2 et P3, les différences sont faibles. La plateforme P3 est légèrement plus stable. Cependant, il nous est difficile d'identifier la cause de cette différence, cela pourrait être dû à la différence de la configuration matérielle ou bien à l'élimination de la machine virtuelle. Il nous faudra mieux isoler les variables de chaque configuration pour pouvoir tirer des conclusions.

Pour finir, la stabilité pour les configurations W1 et W3 est similaire (pour une même plateforme). Cela peut soit signifier que le jeu de données n'a pas d'influence au-delà d'une certaine envergure que dépasseraient W1 et W3, soit que les deux configurations sont trop proches pour obtenir une différence significative. Il nous faudrait tester des configurations qui comportent encore plus de tuples pour pouvoir confirmer nos résultats.

5.4.2 PostgreSQL

Les résultats obtenus sur PostgreSQL dans chacune des configurations sont présentés en Annexe G.

Les résultats concernant la séance avec un niveau d'isolation *Serializable* et *Repeatable Read* ne seront pas pris en compte, car le SGBD n'a pas pu sérialiser l'accès aux données et renvoie donc une erreur.

Pertinences des mesures

Comme pour Oracle, nous avons rencontré un problème de stabilité concernant certaines mesures. Ces mesures sont les suivantes : temps d'exécution de l'UCT, nombre de lectures sur le disque, quantité de mémoire résidente consommée, nombre de défauts de page.

Pour les mesures suivantes : temps d'exécution de l'UCT, quantité de mémoire résidente consommée et nombre de défauts de page, nous avons identifié la principale cause de l'instabilité. Notre outil ne prend les mesures qu'au début et à la fin de chaque séance. Or PostgeSQL, contrairement à Oracle, utilise plusieurs processus. Le problème est que nous ne contrôlons pas le démarrage ni l'arrêt de ces processus. Il s'ensuit qu'un processus peut démarrer et s'arrêter en pleine exécution de notre séance, il peut même démarrer et s'arrêter au sein de la séance et passer ainsi totalement inaperçu. Les ressources consommées par ces processus seront alors mal comptabilisées, voire pas du tout.

Cela nous amène à remettre en cause le choix du système de prise de mesure. Il semble difficile de prendre des mesures fiables si ce n'est pas le SGBD qui fournit lui-même ces

mesures, car les mesures peuvent être dépendantes des choix d'implémentation du SGBD, ce que nous ne pouvons pas contrôler.

Il est probable que d'autres problèmes ajoutent à l'instabilité des mesures comme c'est le cas pour Oracle.

Il ne nous reste donc que deux mesures interprétables, la durée d'exécution et le nombre d'écritures sur le disque.

Influence des configurations

Nous ne considérerons que les mesures qui ont été validées dans notre analyse du paragraphe précédent (soient les mesures de durée d'exécution et du nombre d'écritures sur le disque).

Les résultats de stabilité obtenus sur les trois plateformes sont similaires, ce qui est différent des résultats obtenus avec Oracle pour lequel les résultats de la plateforme P1 étaient beaucoup plus instables que pour les deux autres plateformes. Ceci ne contredit cependant pas l'hypothèse émise en regard au contrôle de l'environnement sous P1. En effet, cela peut être dû à une coïncidence, car, puisque nous ne contrôlons pas l'environnement, il est possible que l'activité parallèle sur le serveur ait été peu fluctuante lors de l'exécution sur PostgreSQL et forte lors de l'exécution sur Oracle. Pour confirmer notre interprétation, il nous faudrait exécuter les tests sur une plateforme dont on contrôle l'environnement et faire varier l'activité des machines virtuelles concurrentes.

Pour finir, la stabilité pour les configurations W1 et W3 est similaire (pour une même plateforme). Cela confirme les résultats obtenus pour Oracle.

5.4.3 Comparaison

Seules les mesures stables peuvent être comparées. La seule mesure stable que nous avons obtenue aussi bien pour Oracle et pour PostgreSQL est la mesure de durée d'exécution.

Nous ne prendrons pas en compte les résultats de la séance Suppression, car elles ne sont pas suffisamment stables. Ceci peut probablement s'expliquer du fait que la durée d'exécution de cette séance est très courte et donc les résultats ne sont pas significatifs.

Pour les séances « ReadCommitted » et « Serializable », la durée d'exécution est globalement deux fois plus courte pour Oracle. Pour la séance « Initialisation », les résultats pour PostgreSQL sont globalement deux fois plus courts. Cela révèle une différence de gestion de l'optimisation des requêtes entre les deux SGBD, chacun privilégie des fonctionnalités différentes. La stratégie qui prend en compte chaque fonctionnalité de manière isolée pourrait nous permettre d'approfondir notre interprétation.

5.5 Expressivité

Les tests d'expressivité consistent à évaluer la facilité à exprimer des solutions dans le langage et à les comprendre.

Ce critère a été évalué après les trois critères concernant la validité du langage, car il faut d'abord identifier les fonctionnalités qui sont valides pour pouvoir ensuite analyser leur expressivité.

5.5.1 Oracle

5.5.1.1 Stratégie Fonctionnalité

Le tableau ci-dessous regroupe les fonctionnalités qui ne sont pas totalement présentes dans le langage :

Tableau 15 – Fonctionnalités d'expressivité limitées pour le SGBD Oracle

I	Types prédéfinis	NP	TL	PL	Commentaires
O	Virgule Flottante Binaire 128 b	X			Non présent dans la norme SQL
P	Booléen		X		BOOLEAN Présent uniquement dans la portion PL/SQL du langage
P	Intervalle Temps			X	INTERVAL YEAR TO MONTH

				.	INTERVAL DAY TO SECOND On ne peut pas avoir un intervalle étendu allant de l'année à la seconde
I	**Création de schéma**	**NP**	**TL**	**PL**	**Commentaires**
P	Creation Type Derive			X	Héritage Simple : CREATE TYPE fille UNDER mere (i_name VARCHAR(24))
P	Création Type Contrainte		X		Au moment de la création de la relation CREATE TABLE contrainte (i INTEGER CHECK [VALUE IN (1,2,3)))
P	Création Relation			X	Impossible de créer une table sans colonnes CREATE TABLE table (i INTEGER, j VARCHAR(24))
P	Modification Tuples Cascade	X			FOREIGN KEY (colonne) REFERENCES table(colonne) ON UPDATE CASCADE UPDATE table SET colonne=2 WHERE colonne=1
P	Modification Fonction			X	CREATE OR REPLACE FUNCTION fonction (echec INTEGER) RETURN INTEGER IS BEGIN RETURN echec ; END ; ALTER FUNCTION ne permet pas de modifier le corps de la fonction. La fonctionnalité est donc combinée à celle de création.
I	**Opérations sur les Relations**	**NP**	**TL**	**PL**	**Commentaires**
O	Division			X	SELECT colonne_non_commune FROM table_divise) EXCEPT ([SELECT colonne_non_commune FROM (SELECT * FROM table_diviseur, (SELECT colonne_non_commune FROM table_divise) T1 EXCEPT SELECT * FROM table_divise)T2]) Impossibilité d'obtenir un résultat sans colonne
I	**Opération sur les Collections**	**NP**	**TL**	**PL**	**Commentaires**
P	Intersection	X			SELECT * FROM table1 INTERSECT ALL SELECT * FROM table2
P	Différence	X			SELECT * FROM table1 MINUS ALL SELECT * FROM table2
O	Division	X			Non présent dans la norme SQL
I	**Représentation**	**NP**	**TL**	**PL**	**Commentaires**

	des données manquantes				
P	Traitement Cas Particuliers	X			Impossible pour l'utilisateur de gérer le comportement des opérateurs dans le cas où une information est manquante
I	**Opérateurs booléens**	**NP**	**TL**	**PL**	**Commentaires**
P	Et			X	AND Disponible uniquement dans la clause WHERE ou dans le langage procédural
P	Ou			X	OR Disponible uniquement dans la clause WHERE ou dans le langage procédural
O	Ou Exclusif	X			Non présent dans la norme SQL
P	Non			X	NOT Disponible uniquement dans la clause WHERE ou dans le langage procédural
I	**Opérateurs de comparaison**	**NP**	**TL**	**PL**	**Commentaires**
P	Égal			X	valeur1=valeur2 Disponibles uniquement dans la clause WHERE ou dans le langage procédural
O	Différent			X	valeur1!=valeur2 Disponibles uniquement dans la clause WHERE ou dans le langage procédural
P	Supérieur			X	valeur1>valeur2 Disponibles uniquement dans la clause WHERE ou dans le langage procédural
P	Inférieur			X	valeur1<valeur2 Disponibles uniquement dans la clause WHERE ou dans le langage procédural
O	Supérieur ou Égal			X	valeur1>=valeur2 Disponibles uniquement dans la clause WHERE ou dans le langage procédural
O	Inférieur ou Égal			X	valeur1<=valeur2 Disponibles uniquement dans la clause WHERE ou dans le langage procédural

I=Importance NP=Non Présente TL=Très Limitée PL=Peu Limitée
P=Primordiale O=Optionnelle

Si on analyse ce tableau de manière quantitative, on obtient les résultats suivants :

Tableau 16 – Quantité de fonctionnalités d'expressivité limitées pour le SGBD Oracle (partitionné selon le niveau de priorité et la gravité de la limitation)

	NP	TL	PL
P	4	2	10
O	3	0	4

NP=Non Présente TL=Très Limitée PL=Peu Limitée O=Optionnelle P=Primordiale

Cela nous permet d'avoir une première idée de l'étendue des lacunes du langage. Cependant, ce n'est pas suffisant, nous devons analyser chacune de ces lacunes pour savoir quel est son impact sur le langage. À partir de là, on pourra alors conclure quant à la capacité du langage à pouvoir exprimer aisément un maximum de problèmes.

Tout d'abord, on remarque que les types booléens sont absents de la portion SQL du langage. Or le type booléen permet d'utiliser les opérateurs de l'algèbre de Boole ce qui est utile pour obtenir facilement une réponse à certains problèmes. Pour pallier l'absence de cette fonctionnalité, on peut pour certains problèmes utiliser les valeurs 0 et 1 (ou 'true' et 'false') respectivement à la place des valeurs faux et vrai mais cela ne nous permet pas d'utiliser les opérateurs booléens. L'autre solution est de passer par une fonction pour évaluer si l'expression est vrai ou fausse (le langage PL/SQL supportant lui les booléens) puis de renvoyer 0 ou 1 (ou 'true' et 'false') selon le résultat obtenu. Une autre restriction de l'expressivité qui est liée à ce problème est l'impossibilité d'utiliser les opérateurs booléens et de comparaison dans la clause **SELECT**. En effet, ces opérateurs retournent en principe des types booléens. Pour qu'ils fonctionnent dans la clause **SELECT**, il faudrait donc pouvoir retourner une relation avec une colonne de type booléen (ou bien modifier la nature de ces opérateurs en retournant 0 ou 1).

On note aussi l'absence du type à virgule flottante binaire sur 128 bits. Ce n'est pas une grande restriction, car le type à virgule flottante binaire sur 64 bits devrait être suffisant pour la plupart des problèmes. De plus, si un problème requiert absolument une taille de 128 bits, le type à virgule flottante décimale sur 128 bits peut être utilisé. La différence est qu'il apporte une plus grande précision lors des calculs ce qui ne peut être que bénéfique.

Cependant cela peut poser quelques problèmes. Tout d'abord, cet ajout de précision peut influer sur l'efficience. L'autre problème est que les conversions de types entre les applications extérieures utilisant une base binaire et le modèle de la BD utilisant une base décimale peuvent entraîner une perte de précision.

Le type prédéfini permettant de représenter des intervalles de temps est quelque peu restreint. On ne peut pas avoir un intervalle de temps précis de l'année à la seconde, il faut choisir entre une précision de l'année au mois et du jour à la seconde. Certains problèmes pourraient demander une plus grande précision et ce serait alors plus complexe à implémenter.

En ce qui concerne les fonctionnalités de définitions de type, plusieurs mécanismes sont limités.

Tout d'abord, en ce qui concerne la dérivation de type, il est impossible d'avoir de l'héritage multiple. Cela limite le mécanisme d'héritage, car un type ne peut hériter des propriétés que d'un seul autre type. Au lieu du mécanisme d'héritage multiple on pourrait aussi avoir un mécanisme d'interface associé au mécanisme d'héritage simple (qui est présent dans le langage d'Oracle), car il est possible pour un type d'implémenter plusieurs interfaces et donc le type peut être utilisé dans plusieurs contextes sans savoir quel est le type effectivement manipulé, juste en connaissant certaines propriétés du type (méthodes fournies, attributs possédés). Cela permettrait donc d'exprimer les mêmes problèmes.

En plus de cela, la création de type à partir d'un type prédéfini dont on restreint le domaine est limitée. Oracle ne permet pas de faire cela au niveau de la création de type, mais uniquement au niveau de la création de relation. Certes cela ne restreint pas le nombre de solutions que l'on peut implémenter, mais cela implique que la contrainte doit être ajoutée à chaque fois qu'une table utilisant ce type est créée. De plus, si le type est créé par un utilisateur puis la table qui l'utilise est ensuite créée par un autre utilisateur, il ne peut pas lui être imposé la contrainte sur le type, ce qui est très gênant, car cela peut causer des problèmes d'intégrité.

Au niveau des fonctionnalités de définition de schéma, Oracle manque quelque peu de flexibilité. La fonctionnalité de création de relation ne permet pas de définir des relations sans colonnes. Or, cela peut être utile lorsque le schéma n'est pas entièrement connu à l'avance. On peut alors créer une table sans colonnes puis ajouter une colonne par la suite (ou bien un autre utilisateur s'occupe d'ajouter les colonnes).

Pour finir, le fait que la modification du corps d'une fonction soit combinée à la fonctionnalité de création de fonction est critiquable. En effet, cela signifie que selon ce qui est présent dans la BD, on ne fait pas la même chose. L'utilisateur doit donc lui-même s'assurer de savoir si la fonction existe déjà ou non et s'il se trompe ou ne fait pas la vérification, aucun mécanisme ne pourra détecter son erreur.

Au niveau de la manipulation des relations, la fonctionnalité de modification de lignes en cascade est manquante. Elle permet de faciliter grandement la tâche de l'utilisateur s'il veut modifier la valeur d'une clé référencée par une ou même plusieurs tables. Sans cette fonctionnalité, il faudrait faire des copies de tables pour y arriver, ce qui est très contraignant.

Au niveau des opérateurs relationnels, quelques petites lacunes sont présentes.

Tout d'abord, il n'existe pas d'opérateur spécifique permettant d'exprimer la division. Il est possible de l'exprimer à l'aide du produit cartésien et de la différence. Cependant, cet équivalent demande à ce que l'on précise les colonnes présentent dans la table que l'on divise et non présente dans la table diviseur. La première conséquence est que cela demande plus de connaissances sur le schéma des tables. La deuxième conséquence est que l'on ne peut pas diviser les tables si elles comportent uniquement des colonnes communes. L'utilisation d'un tel opérateur de manière générique sur des relations dont on ne connaît pas la nature est donc impossible.

Pour ce qui est de la division dans le cas de la manipulation de collection, aucun moyen ne permet de l'implémenter. Ce qui réduit le nombre de problèmes que l'on peut exprimer.

De même pour l'intersection et la différence sur les collections, ces fonctionnalités n'existent pas ce qui limite l'expression de certains problèmes.

Pour la définition des conditions de recherche (par exemple dans la clause **WHERE** d'un **SELECT**), il n'est pas possible d'utiliser l'opérateur **XOR**. Cet opérateur n'est pas primordial, car il peut être exprimé à l'aide d'une combinaison des autres opérateurs booléens. Ainsi, A XOR B correspond à A OR B AND (NOT(A AND B)). Cependant cela faciliterait l'expression et la lisibilité d'une telle condition.

En ce qui concerne la gestion des valeurs manquantes (les **NULL**), le traitement des cas particuliers lors de la manipulation des relations est géré de manière fixe par rapport aux comportements définis par la norme SQL. Le problème est que cela entraîne des comportements pas toujours logiques.

Par exemple, si l'on considère l'opération suivante :

```
SELECT SUM(i) FROM table
```

Si la colonne **i** comporte des informations manquantes qui signifient que la donnée n'est pas applicable, il serait logique que l'opérateur d'agrégation de somme ignore la valeur. Mais lorsque la valeur signifie que la valeur n'est pas encore connue alors on pourrait vouloir que l'opération renvoie un message d'erreur indiquant que la somme ne peut être calculée, car des informations sont manquantes. Selon la signification du **NULL** et le contexte dans lequel on l'utilise, l'utilisateur peut vouloir un comportement différent. Il nous paraît donc important à partir du moment où le langage comporte des **NULL** de donner la possibilité de gérer au cas par cas le comportement des opérateurs en présence de **NULL**.

5.5.1.2 Stratégie MesureComplexite

Un groupe d'étudiants nous a aidées à implémenter une version préliminaire du programme de calculs de métriques. Ce programme nommé M-SQL se concentre sur le calcul des métriques pour le langage SQL spécifique à Oracle.

En ce qui concerne la métrique de Halstead, nous avons rencontré des problèmes d'implémentation. Pour pouvoir distinguer l'unicité des opérandes, on ne peut pas se fier uniquement au nom de la variable. Il faut aussi connaître la portée de la variable. Ainsi, on peut avoir un même nom de colonne dans deux relations différentes.

Ce problème peut tout d'abord se manifester dans le cas de l'utilisation de requêtes imbriquées. Par exemple, si l'on considère la requête suivante :

```
SELECT x.nom, x.prenom FROM etudiant AS x WHERE
x.idUniversite IN (SELECT x.idUniversite FROM universite
AS x) ;
```

Il faut que les deux opérandes **x.idUniversite** soient différenciés par rapport à la portée dans laquelle ils se trouvent, car ils ne représentent pas la même variable.

Le deuxième cas est plus spécifique aux langages de BD. Il est possible que la requête en elle-même ne fournisse pas d'indication quant à la portée de l'opérande. Par exemple, si l'on considère les requêtes suivantes :

```
SELECT nom, prenom FROM etudiant, universite WHERE
nomUniversite='UdS' ;

SELECT nom, prenom FROM professeur, universite WHERE
nomUniversite='UdS' ;
```

Une analyse syntaxique ne permet pas de se rendre compte que les opérandes **nom** et **prenom** de la première requête sont distincts des opérandes **nom** et **prenom** de la deuxième requête. Ils peuvent être des attributs de la relation **universite** dans lequel cas ils représentent la même chose ou bien ils peuvent être des attributs distincts des relations **etudiant** et **professeur.** Le seul moyen de connaître la portée réelle des opérandes est de connaître le schéma des relations que l'on manipule. Il faudrait donc soit avoir la définition des relations dans le script analysé, soit se connecter à la base de données sur laquelle le script sera exécuté.

Halstead est donc complexe à implémenter pour un langage interrogeant des BD. Si l'on ajoute à cela le flou entourant les spécifications de la métrique que nous avons soulevée dans l'Annexe D, la mesure ne semble pas convenir à nos besoins. Par conséquent, la mesure de l'indice de maintenabilité est aussi à remettre en question, car elle réutilise la métrique de Halstead.

Pour les autres métriques, l'implémentation n'a pas posé de problème. Cependant, nous ne sommes pas convaincues de l'exactitude des choix pris pour l'adaptation des mesures à SQL. Elles sont le reflet de notre interprétation des métriques, mais une tout autre interprétation aurait pu être adoptée et justifiée de manière tout à fait légitime.

Cela nous amène à conclure qu'il faudrait définir de nouveaux modèles de complexité plus adaptés aux langages de BD.

5.5.1.3 Stratégie NiveauAbstraction

Pour l'évaluation du niveau d'abstraction du système de typage, nous avons réutilisé les résultats de la stratégie de fonctionnalité pour identifier les mécanismes de création de type existants dans le langage.

Dans Oracle, on retrouve un mécanisme qui permet la création d'un type par agrégation de plusieurs autres types. Ce mécanisme fait partie de ceux nécessaires pour avoir un système de typage « *First Order* » (niveau 1). Un autre mécanisme de création de type important est la possibilité d'avoir de l'héritage simple. Ce mécanisme fait partie de ceux nécessaires pour avoir un système de typage « *Simple Subtype* » (niveau 2). Pour finir, Oracle permet à l'utilisateur de modifier les types dynamiquement lors de l'exécution du programme (**ALTER TYPE**) ce qui est nécessaire pour avoir un système de typage « *Functional Subtype* » (niveau 4). Le langage d'Oracle se trouve donc à cheval entre le plusieurs niveaux. Les mécanismes d'héritage multiple (ou d'interface) et de généricité manquent pour atteindre complètement le niveau 4.

Pour ce qui est du niveau d'abstraction du langage de calcul, le langage est composé d'une portion mi-axiomatique, mi-fonctionnelle (SQL) et d'une portion procédurale (PL/SQL). En effet, pour définir des fonctions ou des procédures il faut détailler les opérations à faire pour arriver au résultat attendu. Tandis que pour exprimer les requêtes relationnelles, il suffit de spécifier soit les propriétés du résultat (clauses **WHERE**) ou les fonctions (sélection, projection...) que l'on veut obtenir et l'algorithme est généré automatiquement.

Bien que le niveau d'abstraction choisi pour le langage PL/SQL soit le plus bas, il nous paraît intuitivement le plus adapté. En effet, le risque avec un niveau d'abstraction plus élevé est de réduire l'étendue des solutions que l'on peut exprimer. C'est d'ailleurs pourquoi la plupart des langages de programmation existants sont procéduraux.

De plus, comme SQL est basé sur l'algèbre relationnelle de Codd et que son langage utilise autant l'abstraction axiomatique que fonctionnelle, il aurait été superflu d'avoir en plus un langage algorithmique axiomatique puisque le niveau d'abstraction du langage était déjà suffisamment élevé. L'intérêt d'un langage algorithmique procédural est alors de suppléer à des lacunes d'expressivité opératoire (aux fins d'optimisation, par exemple) du langage relationnel.

5.5.2 PostgreSQL

5.5.2.1 Stratégie Fonctionnalité

Le tableau ci-dessous regroupe les fonctionnalités qui ne sont pas totalement présentes dans le langage :

Tableau 17 – Fonctionnalités d'expressivité limitées pour le SGBD PostgreSQL

I	Types prédéfinis	NP	TL	PL	Commentaires
O	Char	X			CHAR
P	Var Char	X			VARCHAR
O	Grand VarChar	X			CLOB
O	Virgule Flottante Binaire 128 b	X			Non présent dans la norme SQL

O	Virgule Flottante Decimal 64b	X			La norme indique que l'implémentation binaire ou décimale est au choix.
O	Virgule Flottante Decimal 128b	X			La norme indique que l'implémentation binaire ou décimale est au choix.
O	Multi Ensemble	X			La norme définit le type MULTISET
I	**Création de schéma**	**NP**	**TL**	**PL**	**Commentaires**
P	Creation Type Derive		X		Héritage simple limité, car défini au niveau des relations : CREATE TABLE fille (k INTEGER) INHERITS (mere); Type equivalent : CREATE DOMAIN equivalent AS INTEGER

P	Modification Fonction			X	CREATE OR REPLACE FUNCTION fonction (echec INTEGER)
					RETURN INTEGER IS
					BEGIN
					RETURN echec ;
					END ;
					
					ALTER FUNCTION ne permet pas de modifier le corps de la fonction. La fonctionnalité est donc combinée à celle de création.

I	**Opérations sur les Relations**	**NP**	**TL**	**PL**	**Commentaires**
O	Division			X	(SELECT colonne_non_commune FROM table_divise) EXCEPT ([SELECT colonne_non_commune FROM (SELECT * FROM table_diviseur, (SELECT colonne_non_commune FROM table_divise) T1 EXCEPT SELECT * FROM table_divise)T2]) Impossibilité d'obtenir un résultat sans colonne
I	**Opération sur les Collections**	**NP**	**TL**	**PL**	**Commentaires**
O	Division	X			Non présent dans la norme
I	**Représentation des données manquantes**	**NP**	**TL**	**PL**	**Commentaires**

P	Traitement Cas Particuliers	X			Impossible pour l'utilisateur de gérer le comportement des opérateurs dans le cas ou une information est manquant
I	**Opérateurs booléens**	**NP**	**TL**	**PL**	**Commentaires**
O	Ou Exclusif			X	# : Uniquement dans la clause SELECT
I	**Structures Conditionnelles**	**NP**	**TL**	**PL**	**Commentaires**
O	SWITCH			X	CASE variable WHEN condition_1 THEN resultat_1 WHEN condition_2 THEN resultat_2 ELSE resultat END Non disponible dans le langage procédural

I=Importance NP=Non Présente TL=Très Limitée PL=Peu Limitée
P=Primordiale O=Optionnelle

Si on analyse ce tableau de manière quantitative, on obtient les résultats suivants :

Tableau 18 – Quantité de fonctionnalités d'expressivité limitées pour le SGBD PostgreSQL (partitionné selon le niveau de priorité et la gravité de la limitation)

	NP	**TL**	**PL**
P	2	1	1
O	7	0	3

NP=Non Présente TL=Très Limitée PL=Peu Limitée O=Optionnelle
P=Primordiale

Cela nous permet d'avoir une première idée de l'étendue des lacunes du langage. Cependant, ce n'est pas suffisant, nous devons analyser chacune de ces lacunes pour savoir quel est son impact sur le langage. À partir de là, on pourra alors conclure quant à la capacité du langage à pouvoir exprimer aisément un maximum de problèmes.

Tout d'abord, au niveau des types de données prédéfinis, on remarque l'absence de certains types jugés comme non primordiaux. Ainsi, les types **CHAR**, **VARCHAR** et **CLOB** tels qu'ils sont définis dans la norme SQL n'existent pas. Ce n'est pas une lacune très grave, car ils peuvent être remplacés par les types **NCHAR**, **NVARCHR** et **NCLOB** Encodage qui permettent de stocker des chaînes de caractère avec encodage Unicode. On peut y stocker n'importe quelle

autre chaîne de caractère sans perdre de données. L'inconvénient est que ces types de données demandent davantage d'espace de stockage, car ils stockent un caractère sur deux octets au lieu d'un. Il manque aussi de choix parmi les types de données numériques. Le type de donnée à virgule flottante binaire sur 128 bits qui fait partit des types de bases à implémenter selon la norme IEEE-754 n'est pas présent. De même pour les types de données à virgule flottante binaire sur 64 bits et 128 bits. La première conséquence est que les flottants sont limités à une taille maximale de stockage de 128 bits. L'autre conséquence est que la précision décimale qui permet d'avoir des calculs plus précis lors d'opération entre flottants ne peut être utilisée. La lacune n'est cependant pas si grave, car on peut à la place utiliser les types numériques exacts. Cependant cela risque de diminuer les performances. Dernière lacune au niveau des types de données, le type « Multi Ensemble » qui permet de construire des collections non ordonnées n'est pas disponible. Or ce type de données est utile dans le cas ou l'on veut manipuler des ensembles de données et effectuer des opérations d'intersection, d'union, de différence ou de sous ensemble.

Au niveau des mécanismes de création de type, PostgreSQL comporte de nombreuses lacunes.

Comme pour Oracle, il est n'y a aucun mécanisme d'héritage multiple et d'interface. Comme nous l'avons dit pour Oracle, cela limite le mécanisme d'héritage, car un type ne peut hériter des propriétés que d'un seul autre type. De plus, le mécanisme de création de type par héritage simple est limité à l'héritage des propriétés d'une table. Or les tables ne sont pas considérées comme des types en SQL, elles ne peuvent être utilisées comme colonne d'une table. Le mécanisme de création de type équivalent existant dans PostgreSQL peut être considéré comme une forme d'héritage simple limitée qui s'applique aux types et non aux tables. Le nouveau type hérite d'un type de base, mais il est limité, car on ne peut pas y ajouter d'autres propriétés.

Au niveau de la création de schéma, on retrouve comme pour Oracle la lacune concernant la modification de fonction qui est combinée à la création de fonction. Se référer à l'analyse des résultats d'Oracle pour les explications sur ses conséquences.

Au niveau des opérateurs relationnels, on retrouve comme pour Oracle le problème de la division qui demande à ce que l'on connaisse les attributs de la relation que l'on divise. Une autre lacune commune avec Oracle est l'absence de cette fonctionnalité de division sur les collections (avec conservations des doublons). Se référer à l'analyse des résultats d'Oracle pour plus d'explications sur les conséquences de ces lacunes sur l'expressivité.

En ce qui concerne la gestion des valeurs manquantes (les **NULL**), on retrouve comme pour Oracle l'impossibilité de gérer des cas particuliers lors de la manipulation des relations. Se référer à l'analyse des résultats d'Oracle pour plus d'explications sur les conséquences de cette lacune sur l'expressivité.

Pour les opérateurs booléens, la fonctionnalité de « ou exclusif » est présente uniquement dans la clause **SELECT**, mais pas dans la clause **WHERE** pour exprimer les conditions de recherche. Ce n'est pas une lacune très grave, car comme nous l'avons déjà expliqué pour Oracle, le ou exclusif peut s'exprimer à l'aide des autres opérateurs : A XOR B correspond à A OR B AND (NOT(A AND B)). Cela donne cependant une expression plus complexe à exprimer et à comprendre.

Pour finir, la structure conditionnelle **SWITCH** n'est présente que dans la portion SQL du langage et non dans la portion PL/PGSQL. On ne peut donc pas faire un **SWITCH** dans une fonction sans passer par une instruction **SELECT**. C'est d'autant plus étrange que le langage PL/PGSQL permet de définir le corps des routines et donc il serait plus logique d'avoir cette fonctionnalité dans le langage PL/PGSQL que dans le langage SQL. Cependant, le langage PL/PGSQL permet d'utiliser la structure conditionnelle **IF THEN ELSE** qui permet d'exprimer les mêmes choses qu'un **SWITCH** mais la lisibilité est alors moins bonne.

5.5.2.2 Stratégie MesureComplexite

Nous n'avons pas implémenté le calcul des mesures de complexité pour PostgreSQL, car nous nous attendons à très peu de différences avec ceux d'Oracle. D'autant plus que l'implémentation que nous avons pour Oracle ne prend pas en compte les fonctionnalités les

plus exotiques qui sont pour la plupart spécifiques à Oracle. Il pourrait être intéressant dans des travaux futurs d'implémenter la totalité des deux langages pour pouvoir vérifier notre hypothèse. Mais ce n'est pas notre but premier.

5.5.2.3 Stratégie NiveauAbstraction

Pour l'évaluation du niveau d'abstraction du système de typage, nous avons réutilisé les résultats de la stratégie de fonctionnalité pour identifier les mécanismes de création de type existants dans le langage.

Dans PostgreSQL, on retrouve un mécanisme qui permet la création d'un type par agrégation de plusieurs autres types. Ce mécanisme fait partie de ceux nécessaires pour avoir un système de typage « *First Order* » (niveau 1). PostgreSQL propose aussi un mécanisme d'héritage simple, mais au niveau des relations et non des types. Nous ne pouvons donc pas le considérer dans cette évaluation du système de typage. Deux autres mécanismes de construction de type sont présents dans le langage de PostgreSQL : la création de types équivalents et la création de type par restriction du domaine avec des contraintes. Ceux-ci peuvent être considérés comme une forme simplifiée d'héritage simple, car le nouveau type hérite des propriétés de l'ancien (avec la possibilité de restreindre ces propriétés), mais il n'est pas possible par exemple d'ajouter des attributs. Pour finir, PostgreSQL permet à l'utilisateur de modifier les types dynamiquement (**ALTER TYPE**) lors de l'exécution du programme ce qui est nécessaire pour avoir un système de typage « *Functional Subtype* » (niveau 4). Le langage se trouve donc à cheval entre plusieurs niveaux. Les mécanismes d'héritage multiple (ou une combinaison des mécanismes d'héritage simple et d'interface) et de généricité manquent pour atteindre complètement le niveau 4.

Pour ce qui est du niveau d'abstraction du langage de calcul, comme pour Oracle, le langage est décomposé en une portion axiomatique (SQL) et une portion procédurale (PL/PGSQL), ce qui nous semble approprié (voir section 5.5.1.3 pour une analyse plus complète).

5.5.3 Comparaison

La majorité des lacunes d'expressivité que nous avons pu identifier en termes de fonctionnalités que l'on peut exprimer sont communes pour les deux SGBD. Ceci est tout à fait normal, car les deux langages se basent sur la même spécification qui est la norme SQL ISO/ANSI:2003. Cependant, quelques petites différences existent qui sont dues soit à la mesure dans laquelle la norme est respectée, soit au contraire à l'ajout de fonctionnalités manquantes dans la norme. Les lacunes que l'on retrouve dans les deux langages et qui sont dues à une lacune de la norme sont les suivantes :

- Les types numériques de base définis par la norme IEEE 754 ne sont pas tous implémentés.
- La création d'un type par dérivation de plusieurs autres types n'existe pas (pas de mécanisme d'héritage multiple ni d'interface).
- Le seul mécanisme existant qui permette de modifier l'implémentation d'une fonction est combiné avec celui de création de fonction.
- L'opération de division demande à ce que l'on connaisse le schéma des tables.
- L'opération de division ne peut être exprimée pour des collections.
- Il n'est pas possible de gérer les opérations sur des informations manquantes au cas par cas de manière personnalisée.

Par ailleurs, PostgreSQL présente peu de lacunes graves (qui ne peuvent être comblées simplement). Les seules lacunes qui méritent véritablement d'être soulignées sont les suivants :

- impossibilité d'utiliser le mécanisme d'héritage simple au niveau de la construction de type ;
- type prédéfini de « multi ensemble » inexistant.

Oracle lui présente plusieurs lacunes qui diminuent fortement l'expressivité, car elles empêchent ou compliquent grandement la tâche d'exprimer certains problèmes. Ces fonctionnalités sont les suivantes :

- type booléen absent du langage SQL ;
- impossibilité de modifier des lignes en cascade ;

- absence d'opérateurs d'intersection et de différence sur les collections ;
- impossibilité d'avoir une table sans colonne.

À cela on peut ajouter certaines lacunes moins gênantes, car elles peuvent être contournées sans trop de difficultés ou bien que les problèmes qu'elles permettent d'exprimer sont peu nombreux :

- La création de type avec restriction du domaine n'est possible qu'au niveau de la création de table.
- La restriction du type prédéfini permettant de représenter des intervalles de temps a une précision restreinte.

Conclusion

Contributions

Nous avons proposé un banc d'essai permettant d'évaluer d'importants critères caractérisant les principaux groupes d'exigences applicables à un SGBD. Or, à notre connaissance, les bancs d'essai existant se contentaient d'évaluer un ou deux groupes d'exigences, mais jamais l'ensemble.

En particulier, nous avons intégré dans notre banc d'essai le critère d'expressivité qui a souvent été négligé, probablement parce que les SGBD testés étaient toujours basés sur le langage SQL et donc, en théorie, devaient présenter une expressivité similaire. Cela ne tenait pas compte de la grande variété des dialectes.

L'évaluation de l'expressivité d'Oracle et de PostgeSQL a cependant permis de détecter quelques différences qui pourraient être suffisamment pénalisantes dans le cas de certaines applications pour motiver une décision quant au choix d'un SGBD. Il est donc intéressant de l'évaluer même pour des langages basés sur la norme SQL, car la norme n'est pas entièrement respectée en pratique.

Nous avons aussi pris en compte dans les spécifications de notre banc d'essai la capacité à comparer des SGBD basés sur des modèles relationnels différents. Nous n'avons pas eu le temps de faire une preuve de concept de cette flexibilité. Cependant, notre travail pourra servir de base pour une implémentation future. Notamment, l'outil d'automatisation des essais que nous avons implémenté a été conçu de manière à être suffisamment flexible pour s'adapter à n'importe quel SGBD qui implémente un pilote JDBC.

Le travail effectué nous a aussi permis d'identifier des caractéristiques importantes qui devront être intégrées à la définition du nouveau langage Discipulus.

Tout d'abord, il nous a permis de confirmer l'importance d'avoir un langage implémenté par tous les SGBD en respectant strictement la norme établie. En effet, la présence de dialectes tels que c'est le cas pour SQL rend la tâche beaucoup plus difficile pour l'utilisateur de maîtriser le langage, car il ne peut pas se fier à la norme pour savoir si une fonctionnalité est supportée.

Cela nous a aussi permis de mettre en avant la nécessité d'intégrer aux spécifications du langage un mécanisme standardisé pour la prise de mesures. En effet, nous avons mis en avant la difficulté d'obtenir des mesures fiables avec un mécanisme de prise de mesures externe, car elles sont trop dépendantes de l'implémentation du SGBD. Un mécanisme interne permettrait de comparer simplement les SGBD implémentés sur la base de ce nouveau langage. La standardisation permettra d'avoir des mesures fiables en évitant l'introduction de biais qui mettrait en avant un SGBD plutôt qu'un autre.

Pour finir, les problèmes rencontrés pour calculer les mesures de complexité nous ont permis de déduire que le langage implémenté devra faire en sorte de pouvoir être mesuré facilement (en termes de complexité). Il nous faudra auparavant définir un modèle de complexité adapté à un langage de BD.

On peut ajouter que ce travail nous a permis d'illustrer l'importance de concevoir les tests avant la conception. En effet, sans cela nous n'aurions pas pu identifier ces éléments et notre langage aurait été moins bon.

Critique du travail

Notre objectif initial était de pouvoir comparer des SGBD relationnels basés sur des langages différents. Or en l'état actuel des choses, nous n'avons pu mettre en œuvre le banc d'essai développé dans cette optique que sur des SGBD relationnels basés sur SQL. Nous n'avons donc pas pu démontrer toute la puissance de notre banc d'essai.

De plus, malgré notre désir de spécifier les essais de manière générique pour des langages basés sur différents modèles relationnels, nous n'avons pu l'appliquer pour tous les critères. Ainsi, le critère de validité du langage demande à ce que toutes les fonctionnalités du langage soient testées. Les cas de tests à effectuer sont donc trop liés au langage pour pouvoir les spécifier à un haut niveau. Nous avons procédé en nous basant sur les langages SQL, Tutorial D et Discipulus, le nouveau langage développé en parallèle par notre laboratoire de recherche. Cependant, pour tester un autre langage relationnel que nous n'avons pas pris en compte, il faudra probablement reprendre cette stratégie et ajouter des cas de tests spécifiques.

En ce qui concerne les résultats obtenus, les résultats d'efficience sont peu satisfaisants. Cela est dû au système de prise de mesure que nous avons choisi. Notre choix comportait trop de compromis pour être le bon, les résultats que nous avons obtenus sont en conséquence difficiles à interpréter, car trop instables. Nous avons aussi rencontré quelques problèmes avec les mesures de complexité. Les principes utilisés pour le calcul des métriques ont été difficiles à définir de manières incontestables. Cela remet en cause notre choix d'adapter des métriques habituellement utilisées pour des langages « ordinaires » de programmation à un langage de base de données qui présente quelques particularités (portée des opérandes pas toujours présente dans le code, modules non clairement définis, flou concernant les entrées-sorties).

Travaux futurs de recherche

Notre travail n'est qu'une première version du banc d'essai qui sera effectivement utilisé pour valider la pertinence Discipulus. Encore beaucoup de travail reste à faire.

Tout d'abord, nous avons mis de côté certains éléments tout au long de notre cheminement. Il faudra reprendre ces éléments pour compléter notre banc d'essai. Ces éléments sont recensés dans le tableau de l'Annexe H.

Il faudra ensuite définir des modèles de complexité mieux adaptés aux langages de SGBDR.

Il faudra aussi améliorer l'outil d'automatisation des essais. Pour cela, nous formulons les quatre recommandations suivantes :

- La première recommandation est d'implémenter un nouveau système de prise de mesure. Plusieurs approches ont été documentées dans ce mémoire. Celle qui a été choisie s'est avérée, à l'expérience, partiellement satisfaisante, mais elle avait l'avantage d'être facile à mettre en œuvre. En outre, elle nous a permis de nous rendre compte qu'une bonne évaluation de l'efficience repose sur une infrastructure fournie à même le SGBD. Cependant, comme SQL ne comporte aucune prescription à ce sujet, les implémentations existantes fournissent ou non les informations dont on a besoin. Il faudra donc réfléchir à la solution la plus adaptée à nos besoins permettant de faire un compromis entre la fiabilité des mesures et la possibilité de l'implémenter pour la majorité des SGBD existants.

- La deuxième recommandation est de modifier la manière dont sont implémentés les générateurs pour les rendre reproductibles. Cela nous permettra d'avoir des résultats plus fiables, car les comparaisons entre les différents SGBD se feront avec exactement les mêmes données de test.

- La troisième recommandation est d'implémenter un traducteur qui nous permettra un énorme gain de temps quant à l'implémentation des essais dans les différents langages. Bien sûr, certains cas de tests spécifiques aux langages ne pourront bénéficier de cette traduction. Cependant la suite de tests ayant été définie le plus génériquement possible, ces cas de test seront peu nombreux.

- La quatrième recommandation est d'améliorer la validation expérimentale du banc d'essai ainsi établi. Pour cela, il faudra tout d'abord tester des langages relationnels non basés sur SQL. Il faudra avant de pouvoir faire ces expérimentations implémenter les pilotes JDBC qui permettront à l'outil d'automatisation de fonctionner avec ces SGBD.

Finalement, il faudra aussi exécuter les tests d'efficience sur des plateformes plus variées que celles que nous avons utilisées. En effet, la plateforme (système d'exploitation + matériel) est un paramètre qui a une grande influence sur les mesures d'efficience. Cela donnera donc des résultats plus objectifs.

Perspectives

Nous espérons tout d'abord que ce banc d'essai aidera les chercheurs qui, comme nous, ont pour objectif de mettre au point un nouveau modèle relationnel qui résoudrait les problèmes existants dans le langage SQL.

Nous espérons aussi encourager les chercheurs qui pensent que le modèle relationnel fait toujours partie du futur des SGBD. Un modèle fermé tel que le propose le modèle relationnel reste indispensable pour de nombreuses applications nécessitant exactitude et fiabilité. De plus, pour ce qui est des critiques quant à l'efficience du modèle face au traitement de large volume de données, il a été montré que les implémentations verticales et l'adaptation concurrente des méthodes horizontales peuvent permettre de résoudre ce problème [43] tout en préservant la grande qualité du modèle relationnel : l'universalité.

Annexe A

Présentation de quelques modèles de sécurité

A.1 Gestion de l'authentification

L'authentification est le moyen qui permet de s'assurer de l'identité de l'utilisateur. Le mécanisme doit donc être très fiable.

Trois moyens permettent de s'assurer de l'identité d'une personne [15] :

- **Utiliser un objet unique qu'il possède** : par exemple, une carte magnétique ou une clé. Le risque est alors de se faire voler l'objet ou de le voir être falsifié.
- **Vérifier ce qu'il sait** : par exemple, à l'aide d'un mot de passe ou d'une question personnelle. La fiabilité de cette méthode est discutable. Tout d'abord, dans le cas d'une question personnelle ou d'un mot de passe simple, il est possible de contourner la sécurité en générant un grand nombre de mots de passe à l'aide d'un dictionnaire. Dans le cas où l'on utilise un mot de passe suffisamment long, complexe (composé de lettres, chiffres et caractères spéciaux), le problème de fiabilité vient du fait que l'utilisateur aura du mal à retenir son mot de passe et il devra donc le noter quelque part, il y a donc un grand risque qu'il soit volé.
- **Utiliser des données biométriques** : il existe deux types de données biométriques selon qu'elles sont fondées sur les caractéristiques physiologiques ou sur les caractéristiques comportementales. Pour les premières, cela peut être une reconnaissance du visage, de l'iris ou des empreintes digitales. Pour les deuxièmes, cela peut être la dynamique d'une signature, la démarche de l'utilisateur, ou bien la reconnaissance vocale. Ces techniques sont les plus fiables, car très difficiles à voler ou à falsifier. Ce n'est cependant pas impossible à voler, et cela peut d'ailleurs entraîner une incitation à la violence. Cela pose aussi des problèmes éthiques et légaux [26], car une donnée biométrique est quelque chose de très personnel, si elle est volée, elle peut permettre d'usurper l'identité de la personne et non plus de s'identifier à un seul système, mais à tous les systèmes utilisant le même type d'authentification. Le problème vient du fait que la donnée

qui permet de s'authentifier ne peut pas être modifiée. De plus, cela pose un problème quant à la vie privée, car on peut alors tracer les activités d'une personne.

Les techniques d'identification biométriques actuellement existantes ne sont pas fiables à 100 % [15]. En effet, il y a une faible probabilité d'obtenir de faux positifs ou de faux négatifs. Le problème vient du fait que contrairement à un mot de passe ou un objet, les données biométriques sont quelque peu variables. Par exemple, une empreinte digitale peut varier en fonction de l'humidité, la saleté, de changements morphologiques ou physiologiques (coupure, brûlure, vieillissement, etc.). Il est alors difficile de faire un système fiable dans ces conditions.

A.2 Gestion des autorisations

La gestion des autorisations repose sur la définition d'une politique de sécurité. Cette politique définit les règles qui vont permettre de créer des utilisateurs et des groupes d'utilisateurs, de leur attribuer un certain nombre de droits en fonction de leur statut (aux groupes ou aux utilisateurs directement), mais aussi la manière dont ces droits sont définis par défaut et propagés.

Plusieurs modèles prédéfinis de sécurité existent. L'utilisation d'un modèle de sécurité permet d'assurer que l'on n'a pas de moyen de contourner la politique de sécurité implémentée.

Nous avons analysé plusieurs modèles existants de manière à en tirer les critères d'une bonne politique de sécurité.

On peut par exemple se baser sur le modèle de Bell-Lapadula. L'utilisateur définit des niveaux d'autorisation de plus en plus élevés. La règle est alors que « la lecture n'est pas autorisée sur des données d'un niveau supérieur et l'écriture n'est pas autorisée sur des données d'un niveau inférieur » (voir figure). Cela permet de garantir la confidentialité des données. Les données de haut niveau ne sont pas accessibles aux utilisateurs de bas niveau.

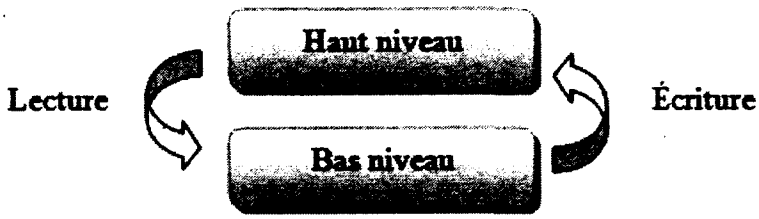

Figure 5 – Modèle de sécurité de Bell-Lapadula

Le modèle de Biba [4] (ou intégrité stricte) plutôt que de garantir la confidentialité des données permet de s'assurer de l'intégrité des données. La règle est alors que « l'écriture n'est pas autorisée sur des données d'un niveau supérieur et la lecture n'est pas autorisée sur des données d'un niveau inférieur » (voir figure ci-dessous). Cela permet de garantir qu'une personne d'un niveau inférieur ne pourra pas corrompre les données d'un niveau supérieur et qu'une personne d'un niveau supérieur ne pourra pas être corrompue par les données fournies par une personne d'un niveau inférieur. Des dérivés de ce modèle existent tels que la variante « *low-water mark* » ou la variante « *ring* », mais nous ne les aborderons pas ici.

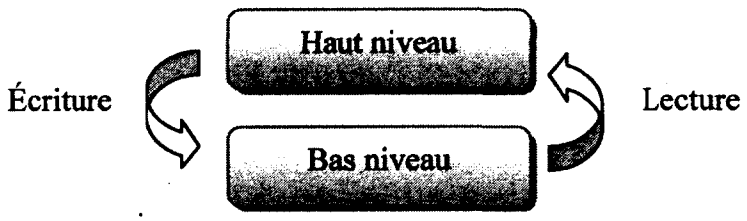

Figure 6 – Modèle de sécurité de Biba

Le modèle de Brewer et Nash [6] a pour but d'éviter les conflits d'intérêts. Pour que le modèle fonctionne, les données doivent être organisées de manière hiérarchique selon un modèle prédéfini. Le plus bas niveau comporte des données individuelles (représentant par exemple les employés d'une compagnie). Le niveau intermédiaire correspond à un regroupement de données individuelles (par exemple l'ensemble des employés appartenant à une même compagnie). Le niveau le plus haut regroupe les données de groupes intermédiaires qui ont un conflit d'intérêts (par exemple, des compagnies en concurrence, c'est-à-dire celles qui ont le même domaine d'activité). Dans notre exemple, chaque donnée

est donc associée à un employé, à une compagnie et à un « groupe de conflits d'intérêts » (voir figure ci-dessous).

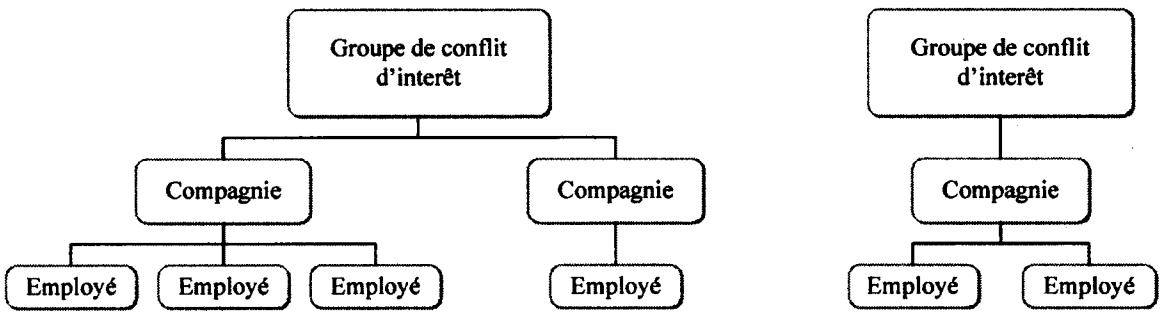

Figure 7 – Hiérarchie des données pour le modèle de Brewer et Nash

À partir de là, la règle pour qu'un utilisateur puisse lire une donnée est que :

- L'utilisateur a déjà accédé à des données de cette compagnie auparavant
  OU
- L'objet appartient à un groupe de conflits d'intérêts avec lequel l'utilisateur n'a jamais eu aucun lien.

Cela signifie que si un utilisateur accède aux données d'une compagnie faisant partie du groupe de conflits d'intérêts Banque, alors il ne sera par la suite plus autorisé à accéder aux autres compagnies de type Banque.

À cela s'ajoute une règle en écriture. Un utilisateur peut écrire une donnée :

- S'il a le droit de la lire (selon la règle définie précédemment)
  ET
- Si l'utilisateur n'a jamais lu une donnée d'une compagnie que celle dans laquelle il désire écrire

De cette manière, on évite de propager des informations qui pourraient ensuite être utilisées par des utilisateurs qui n'en ont normalement pas le droit.

La particularité de ce modèle est que l'accès aux données n'est pas défini selon les propriétés des données, mais par rapport aux droits déjà définis sur les autres données. Le modèle

évolue donc constamment prenant en compte les différents accès aux données qui ont été effectués par le passé.

Par contre, ce modèle est très restrictif, il faut tout d'abord avoir un système qui repose sur la hiérarchie définie auparavant. Les problèmes que l'on peut modéliser sont donc très particuliers. Il faut que l'on ait affaire à des compagnies qui font affaire avec d'autres compagnies, mais qui veulent éviter les conflits d'Intérêts entre les compagnies concurrentes. De plus, les opérations que l'utilisateur peut effectuer sont très limitées. Un utilisateur qui a lu des données dans plus d'une compagnie ne pourra plus écrire nulle part, on risque même de se retrouver avec un système ou plus personne n'a le droit d'écrire. De même, un utilisateur peut à la fois lire et écrire les données d'une seule compagnie.

Au final, deux critères ressortent pour évaluer une politique de sécurité. Elle doit tout d'abord être applicable à de nombreux problèmes pour satisfaire l'utilisateur dans tout type de situation. Or la variété des problèmes que l'on peut modéliser dans une base de données est tellement vaste que l'on ne peut se cantonner à une seule politique prédéfinie. Il faut laisser de la flexibilité à l'utilisateur en lui laissant le choix entre plusieurs politiques prédéfinies, voire d'adapter celle-ci à sa manière au cas où elles ne seraient pas suffisantes.

En parallèle, une politique de sécurité doit être fiable. C'est-à-dire que l'on doit pouvoir prévoir son comportement dans n'importe quel cas de figure.

Les deux critères sont très liés. Si on ne laisse le choix à l'utilisateur que d'utiliser des modèles prédéfinis, la fiabilité de la politique sera déjà éprouvée. Tandis que si on lui laisse le choix de modifier ces modèles où même de définir un modèle entièrement, il faut alors pouvoir prouver que la politique est fiable.

Pour définir une politique, il faut que le système que l'on veut sécuriser fournisse un certain nombre de mécanismes. Nous avons analysé différents systèmes largement utilisés ou issus de la recherche pour en retirer les mécanismes nécessaires à la gestion des autorisations.

La gestion d'accès du système de fichier d'UNIX [56]

Elle se base sur plusieurs rôles et droits d'accès prédéfinis. Les trois droits d'accès sont les droits en lecture, en écriture et en exécution. Dans le cas où l'on manipule un dossier, le droit en lecture correspond au fait de voir l'existence du dossier tandis que le droit en exécution correspond au fait de pouvoir parcourir le dossier et donc d'accéder à ses sous-composants. Ces droits sont ensuite attribués aux utilisateurs selon leur rôle.

Le rôle qui donne tous les droits est le rôle de superutilisateur (ou « *root* »). Un superutilisateur a donc les droits en lecture, écriture et exécution sur tous les fichiers du système et personne ne peut lui enlever ces droits.

Le deuxième rôle est celui de propriétaire. Le propriétaire d'un fichier est par défaut celui qui a créé le fichier. Cependant, il est possible que le superutilisateur donne ce rôle à un autre utilisateur. Par défaut, un propriétaire a les droits en lecture, écriture pour un fichier et les droits en lecture, écriture et exécution pour un dossier, mais ces droits peuvent être modifiés par lui-même ou par le superutilisateur.

Le troisième rôle est celui de groupe. Il comprend tous les utilisateurs qui appartiennent au même groupe que celui auquel le fichier est associé. Par défaut, le groupe auquel est associé le fichier est celui du propriétaire, mais le superutilisateur peut le changer. Ce rôle est plus flexible, c'est l'administrateur qui définit les groupes ainsi que l'appartenance des utilisateurs a un ou plusieurs groupes. Par défaut, le groupe a les droits en lecture, écriture pour un fichier et les droits en lecture, écriture et exécution pour un dossier, mais ces droits peuvent être modifiés par le propriétaire ou par le superutilisateur.

Pour finir, le dernier rôle « Autre » correspond à tous les utilisateurs qui n'appartiennent pas aux trois rôles définis précédemment. Par défaut, ils ont les droits en lecture, écriture pour un fichier et les droits en lecture, écriture et exécution pour un dossier, mais ces droits peuvent être modifiés par le propriétaire ou par le superutilisateur.

Les mécanismes fournis par UNIX sont trop limités, ils ne permettent pas de répondre à tous les besoins que pourrait avoir l'utilisateur. Il est en effet handicapant de ne pas pouvoir

définir tous les droits de manière spécifique pour chaque utilisateur et groupe. Ainsi, le modèle de Brewer-Nash ne peut pas être implémenté par les mécanismes d'UNIX, car il faudrait un groupe supplémentaire pour pouvoir modéliser la hiérarchie des données telle que définie dans ce modèle.

La gestion d'accès du système de fichier VMS [20]

C'est l'ancêtre du système de fichier de Windows, il propose deux solutions pour gérer les accès aux fichiers. La première, la protection RMS, est très proche du modèle UNIX défini précédemment. Les rôles sont « Système », « Propriétaire », « Groupe » et « World » et correspondent respectivement aux rôles « Root », « Propriétaire », « Groupe » et « Autre ». Une différence, le rôle « System » peut-être accordé à plusieurs utilisateurs par l'administrateur. Une autre différence est qu'un utilisateur peut appartenir à plusieurs groupes. Les droits d'accès sont les mêmes que pour UNIX avec en plus le droit de suppression. Par défaut, les droits pour chacun des rôles sont les suivants :

Directory	S: RWED, O: RWED, G: RWED, W: RE
File	S: RWD, O: RWD, G: RWD, W:R

Figure 8 – Droits d'accès VMS[8]

L'administrateur peut ensuite modifier ces droits à sa guise.

À noter que contrairement à UNIX, le propriétaire peut transférer son droit de propriétaire à qui il veut tandis que l'administrateur peut s'approprier le fichier, mais pas le « donner » à une tierce personne.

La deuxième solution fournie pour gérer les droits est l'utilisation d'une liste de contrôle d'accès (ACL). Ces dernières sont basées sur le principe de matrice de contrôle d'accès dont

---

[8] http://h71000.www7.hp.com/doc/73final/6556/6556pro_009.html
Consulté le 2011-11-23.

Lampson [29] fut le premier à mentionner. Une ACL est associée à chaque élément (fichier ou dossier) du système. Chaque ACL contient les droits d'accès pour chaque utilisateur ou groupe d'utilisateur (définis par l'administrateur) à l'élément auquel elle est associée. Cela permet plus de flexibilité, car on peut alors définir des cas particuliers de droits d'accès pour chaque groupe ou utilisateur et de ne pas considérer uniquement le groupe auquel appartient le propriétaire.

Pour déterminer quels sont les droits accordés à un utilisateur sur un objet, le système examine d'abord l'ACL associée à l'objet si elle existe. Sinon, il regarde les droits définis par la protection RMS.

Lorsqu'un fichier/dossier est créé, il hérite par défaut des droits d'accès de son dossier parent. Si le nouveau dossier ne spécifie pas de droits particuliers, seul le propriétaire du fichier aura la permission de l'utiliser. De même, lorsqu'un fichier/dossier est copié à un autre emplacement, il perd tout les droits qui lui étaient associés et hérite des droits du dossier parent.

La gestion des autorisations du système de fichier NTFS [40]

NTFS est une évolution de VMS, il a été conçu pour le système d'exploitation Windows NT. Il se base lui uniquement sur les ACL, celle-ci étant assez flexible pour faire ce que l'on veut. Par rapport aux ACL de VMS, NTFS propose quelques fonctionnalités supplémentaires.

Tout d'abord, l'héritage par défaut des droits d'accès d'un fichier de son dossier parent peut être modifié. On peut ainsi spécifier à partir de quel niveau se fera l'héritage sur les sous dossier ou bien carrément supprimer tout héritage.

On peut ensuite définir des autorisations négatives, c'est-à-dire que plutôt que d'autoriser l'utilisateur à accéder à un objet, on peut explicitement lui interdire un accès.

En cas de conflit, si plusieurs droits, négatifs ou positifs, implicites (hérités) ou explicites, sont définis sur un même objet, un ordre de prévalence permet de choisir le droit à appliquer :

1. Droit négatif explicite
2. Droit positif explicite
3. Droit négatif implicite
4. Droit positif implicite

Autre particularité, des droits spécifiques existent selon le type d'objet. Par exemple, les objets de type dossier ont un droit spécifique qui permet de lister le contenu du dossier, ce qui n'aurait aucun sens pour un objet de type imprimante par exemple.

Il existe aussi des groupes de droits qui permettent d'associer les droits aux objets sans rentrer dans les détails. Par exemple, le droit Lecture regroupe tous les droits de lectures qui peuvent être effectués sur les différents objets du système.

Par rapport à VMS, le système de fichier NTFS permet encore plus de flexibilité, mais sans perdre au niveau du contrôle. Notamment, le principe de droits négatifs permet de s'assurer que les objets que l'on veut sécuriser le sont bien, même si l'on ne contrôle pas tous les droits positifs que l'on a définis par ailleurs.

Les modèles de VMS et Windows NT permettent d'implémenter davantage de modèles de sécurité que celui d'UNIX. Cependant, ils sont du coup plus difficile à maîtriser, car plus de choses doivent être définies manuellement.

Système de gestion des autorisations [14]

Tous les SGBD se basant sur SQL gèrent de façon similaire le contrôle d'accès aux données.

Chaque utilisateur est associé à au moins un rôle. Dans certains SGBD tels que SQL Server, des rôles prédéfinis existent. On peut ensuite créer ses propres rôles en plus. Le « rôle » public est particulier, car il est associé à tous les utilisateurs et tous les rôles. Il est possible d'associer un rôle à un utilisateur ou a un autre rôle. De cette manière, l'utilisateur ou les utilisateurs de cet autre rôle pourront s'il le souhaite activer le rôle ainsi associé.

Des privilèges sont ensuite associés aux différents rôles. Les privilèges que l'on peut associer à un rôle dépendent de l'objet sur lequel ils portent. Par exemple, dans MySQL, on peut donner les privilèges suivants :

Privilege	Column	Context
CREATE	Create_priv	databases, tables, or indexes
DROP	Drop_priv	databases, tables, or views
GRANT OPTION	Grant_priv	databases, tables, or stored routines
REFERENCES	References_priv	databases or tables
EVENT	Event_priv	databases
ALTER	Alter_priv	tables
DELETE	Delete_priv	tables
INDEX	Index_priv	tables
INSERT	Insert_priv	tables or columns
SELECT	Select_priv	tables or columns
UPDATE	Update_priv	tables or columns
CREATE TEMPORARY TABLES	Create_tmp_table_priv	tables
LOCK TABLES	Lock_tables_priv	tables
TRIGGER	Trigger_priv	tables
CREATE VIEW	Create_view_priv	views
SHOW VIEW	Show_view_priv	views
ALTER ROUTINE	Alter_routine_priv	stored routines
CREATE ROUTINE	Create_routine_priv	stored routines
EXECUTE	Execute_priv	stored routines
FILE	File_priv	file access on server host
CREATE USER	Create_user_priv	server administration
PROCESS	Process_priv	server administration
RELOAD	Reload_priv	server administration
REPLICATION CLIENT	Repl_client_priv	server administration
REPLICATION SLAVE	Repl_slave_priv	server administration
SHOW DATABASES	Show_db_priv	server administration
SHUTDOWN	Shutdown_priv	server administration
SUPER	Super_priv	server administration
ALL [PRIVILEGES]		server administration
USAGE		server administration

Figure 9 – Liste des privilèges MySQL[9]

La colonne « *Context* » décrit les objets sur lesquels peuvent porter les privilèges.

[9] http://dev.mysql.com/doc/refman/5.1/en/privileges-provided.html Consulté le 2011-11-23.

À tout moment, on peut retirer l'association entre un rôle et un privilège (**REVOKE**).

Le propriétaire d'un objet (créateur de l'objet) a tous les droits sur celui-ci. Il peut ensuite transférer ces droits à l'exception des droits **DROP** et **ALTER** qui permettent respectivement de supprimer et modifier l'objet et qui sont réservés au propriétaire de l'objet. Il peut aussi autoriser les autres utilisateurs à pouvoir donner des droits sur son propre objet.

Le modèle de SQL se distingue par son adéquation aux bases de données. En effet, les privilèges sont beaucoup plus nombreux et correspondent aux actions que l'on peut effectuer sur une base de données. Par exemple, le droit en lecture est raffiné en **SELECT** qui permet de lire dans les tables et **SHOW VIEW** qui permet de lire des vues de table. Or cela n'est pas vraiment utile, car selon l'objet sur lequel porte le droit de lecture, on sait très bien si l'on accorde le droit d'utiliser la commande **SELECT** ou **SHOW VIEW**. Au niveau des droits en écriture, on retrouve le même problème par exemple avec les privilèges **DROP** et **DELETE**. La commande **DROP** permet de supprimer une base de données, une table ou une vue tandis que la colonne **DELETE** permet de supprimer une ligne dans une base de données.

Par contre, il est intéressant de raffiner le droit d'écriture en créer, modifier et supprimer pour permettre une meilleure séparation des devoirs.

On peut aussi intégrer des droits en exécution. Cela peut être utile pour l'exécution de transactions qui permettraient d'être plus précis que les droits en lecture, écriture, car l'on serait autorisé à effectuer un enchaînement particulier de lecture et d'écriture.

De plus, la séparation des privilèges en fonction des objets sur lesquels ils portent permet d'être encore plus précis dans leur attribution. On peut, par exemple, définir le privilège **DROP** sur toute la base de données et donc permettre de supprimer toutes les tables de la base ou bien uniquement autoriser le privilège **DROP** sur une table en particulier. Ceci se base sur une hiérarchie logique de la base de données. Cependant, MySQL (pas plus que SQL) ne

permet pas de raffiner encore plus ce privilège de **DROP** en autorisant son utilisation sur une colonne en particulier. On pourrait donc encore améliorer la précision des droits.

Pour ce qui est de la possibilité de propager des droits, étant donné le caractère autodescriptif d'une base de données, les utilisateurs ainsi que les droits auxquels ils sont associés sont stockés dans la base au même titre que n'importe quelles données. Nous n'avons donc pas besoin de définir un nouveau droit. En effet, le droit qui permet de modifier, supprimer ou ajouter une ligne dans une table est suffisant.

Gestion des autorisations par Clark et Wilson [9]

Ces chercheurs proposent un autre mécanisme très adapté aux bases de données. Il garantit l'intégrité grâce à un certain nombre de règles d'accès très précises. Tout d'abord, les données du système sont divisées en deux catégories, les CDI (Constrained Data Items) qui sont les éléments qui nécessitent que l'on vérifie l'intégrité et les UDI (Unconstrained Data Items) pour lesquels on n'applique pas le modèle d'intégrité. On peut ensuite appliquer sur ces éléments deux types de procédure. L'IVP (Integrity Verification Procedure) est une procédure qui permet de vérifier que les données sont dans un état valide par rapport à des propriétés définies sur les données (par un administrateur par exemple). La TP (Transformation Procedure) est une procédure qui modifie les données sans altérer la validité de l'état de chacune des données du système.

À partir de ces différents éléments, Clark et Wilson définissent ensuite des règles de certification (c) et des règles d'application (e). Les neuf règles assurent l'intégrité externe et interne des données élémentaires.

Règles de certification :

- C1 – Le système doit fournir une IVP pour valider l'intégrité de chaque CDI.
- C2 – Les TP doivent être certifiées comme étant valides, c'est-à-dire qu'elles doivent transformer un CDI vers un état final valide (qui vérifie toutes les IVP). Les TP sont associées aux CDI qu'elles sont autorisées à manipuler à l'aide d'une relation de la forme : (TPi, (CDIa, CDIb, ...))

- C3 – La liste des autorisations d'exécution des TP doit respecter le principe de séparation des devoirs, c'est-à-dire que les TP ne doivent pas toutes être accessibles par un même utilisateur, car cela signifierait qu'il a le contrôle total du système, il y a donc un risque au niveau de la sécurité.
- C4 – Toute action effectuée par une TP doit être enregistrée dans un fichier de log.
- C5 – Si une TP agit sur un UDI, elle doit alors produire un CDI valide. Une UDI sert à manipuler les données qui viennent de l'extérieur du système et donc qui ne sont pas encore certifiées comme valides, on peut ensuite les enregistrer dans le système à condition de leur faire passer une certification avant en appliquant une TP dessus qui permet de les transformer en CDI.

Règles d'application :

- E1 – Seules les TP certifiées peuvent manipuler les CDI.
- E2 – Les TP ne doivent pouvoir être exécutées que par les utilisateurs qui y sont autorisés. Le système fournit la liste des autorisations à l'aide de relations de la forme : (UserID, TPi, (CDIa, CDIb, ...)).
- E3 – Le système doit s'assurer de l'identité d'un utilisateur qui tente d'exécuter une TP.
- E4 – Seul l'administrateur peut définir et modifier des autorisations sur les TP.

Le modèle de Clark et Wilson permet de gérer l'intégrité vis-à-vis des utilisateurs qui n'ont pas l'autorisation d'accéder aux données grâce au triplet qui établit une relation entre utilisateur, TP et CDI. Mais il permet aussi d'empêcher des utilisateurs autorisés de transformer les données vers un état non valide en leur permettant uniquement d'utiliser des opérations qui ont été certifiées (les TP). Ainsi, la validation des données ne se fait plus uniquement par rapport à la confiance que l'on accorde à l'utilisateur en question, mais aussi à l'aide de propriétés définies par l'administrateur et qui permettent de valider que les données son encore valide après que l'utilisateur les ait transformées.

Avec ce modèle, on peut aussi permettre à l'utilisateur d'accéder à des données statistiques globales, mais pas aux données nominales en donnant le droit d'exécuter une procédure qui comporte une agrégation et en s'assurant à l'aide d'une IVP que la quantité de données est suffisamment importante pour ne pas pouvoir en déduire les données nominales.

De plus, la répartition des autorisations d'accès est très flexible, pour chaque utilisateur, on peut définir un droit différent sur chaque CDI. Les différents droits d'accès sont aussi flexibles, ils correspondent aux TP que l'utilisateur est autorisé à exécuter, on peut donc définir de nouveaux droits à volonté. Cependant, cette grande flexibilité le rend aussi très difficile à manipuler, car tout doit être fait au cas par cas.

Gestion de l'héritage des droits par Jajodia [25]

Jajodia définit un Framework qui permet de définir plusieurs politiques pour un même système, selon le type d'objet concerné par exemple.

Pour ce faire, l'utilisateur doit spécifier au système comment l'héritage des autorisations d'accès doit se faire par rapport à la hiérarchie des objets, des utilisateurs ou des rôles.

Les différentes politiques d'héritage qu'il peut définir sont les suivantes :

- pas d'héritage,
- héritage sans surcharges,
- héritage avec surcharge spécifique : priorité aux nœuds les plus bas dans la hiérarchie.

Une fois les autorisations propagées, il est possible que l'on se retrouve avec plusieurs droits pour un même nœud de la hiérarchie.

L'utilisateur doit alors spécifier la politique de résolution de conflits parmi les suivantes :

- pas de conflits : un conflit est considéré comme une erreur,
- une autorisation négative est prédominante,
- une autorisation positive est prédominante,
- on se réfère à une politique de décision personnalisée.

Il est aussi possible de définir de nouvelles politiques grâce à un langage appelé *Autorisation Specification Langage* (ASL) proche de DataLog. Il suffit alors de définir des règles logiques pour créer une nouvelle politique pour l'un des trois objectifs précédents.

Pour assurer la fiabilité des politiques ainsi définies, il est aussi possible de définir des règles d'intégrité formelles sur les spécifications d'autorisations pour vérifier qu'aucune erreur n'a été faite au niveau des différentes politiques.

Récapitulatif des mécanismes présents dans les différents systèmes de gestion des autorisations étudiés :

Tableau 19 – Comparaison de différents systèmes de gestion d'autorisation

	UNIX	**VMS**	**NTFS**	**SQL**	**Clark et Wilson**	**Jajodia**
Nombre de droits	3 prédéfinis	4 prédéfinis	Beaucoup prédéfinis	Beaucoup prédéfinis (adaptés aux BD)	Illimités, mais non prédéfinis	Illimités, mais non prédéfinis
Groupe de droits	Non	Non	Oui	Oui – avec des rôles	Non	Oui – avec des rôles
Héritage des droits	Non	Oui	Oui — modifiable	Oui	Non	Oui – modifiable et variable selon les objets
Groupe d'utilisateurs	Oui	Oui	Oui	Oui – avec des rôles	Non	Oui
Rôles prédéfinis	4	4	0	Ça dépend des versions	0	0
Propagation des droits	Oui – par le « root »	Oui – par le possesseur de l'objet	Oui – par le possesseur de l'objet	Oui – par le possesseur de l'objet	Non	Oui – par le « root »
Haut niveau d'abstraction	Non	Non	Non	Non	Non	Oui
Attribution de droits positifs et négatifs	Positifs	Positifs	Positifs et négatifs	Positifs	Positifs	Positifs et négatifs
Pour chaque objet, on peut définir un droit	Non	Oui	Oui	Oui	Oui	Non

différent pour chaque utilisateur						
Résolution des conflits	Oui – fixée	Oui – fixée	Oui – fixée	Oui – fixée	Oui – fixée	Oui – modifiable
Système de vérification de politique	Non	Non	Non	Non	Non	Oui

A.3 Gestion de l'audit

Pour pouvoir gérer l'audit des accès à la base, il faut tout d'abord entretenir un fichier de log. Le contenu de ce fichier de log doit au moins permettre de savoir qui (ou quoi) a effectué quelle action sur quel objet et de retracer le déroulement des actions composant une attaque de sécurité. Le fichier de log peut aussi contenir une série de mesure sur les utilisateurs pour pouvoir surveiller leur comportement. Le contenu d'un fichier de log varie donc selon les systèmes.

L'audit comprend aussi l'audit des fichiers de log ainsi créés. L'audit consiste à analyser les actions effectuées par les utilisateurs sur la base (en analysant le fichier de log) pour permettre de détecter une intrusion ou une tentative d'intrusion dans le système. Selon les systèmes, le déclenchement des audits peut se faire :

- en continu, en tout temps on surveille si une attaque est détectée. Cela demande beaucoup de mémoire, mais offre une plus grande sécurité ;
- lorsqu'une faille est suspectée, par exemple si plus de trois mots de passe erronés ont été rentrés. Seules les données suspectées d'être corrompues sont examinées, cela demande donc moins de mémoire. Cette technique est moins sécuritaire, car pour qu'une attaque soit suspectée, il faut anticiper les signes qui apparaissent lors d'une attaque, ce qui n'est pas possible à chaque fois ;
- de manière occasionnelle pour prévenir d'une faille qui n'aurait pas été détectée. Cette technique seule n'est pas du tout efficace, car il faudrait que l'attaque coïncide avec le choix arbitraire de déclencher l'audit pour que l'on puisse effectivement détecter celle-ci. C'est pourquoi cette technique est en général

utilisée en plus de la technique précédente, car on ne peut pas anticiper toutes les attaques.

Ces trois possibilités pour le déclenchement d'un audit peuvent déterminer un critère d'évaluation du système d'audit, car chacun offre un niveau de sécurité plus ou moins élevé. Le niveau de sécurité le plus bas est celui de l'audit qui se déclenche aléatoirement, puis vient l'audit qui se déclenche lorsqu'une faille est suspectée, puis une combinaison de ces deux méthodes et pour finir l'audit en continu est celui qui offre le plus haut niveau de sécurité.

Pour ensuite détecter une intrusion, plusieurs techniques existent. L'article [3] propose une taxonomie de ces différentes techniques :

anomaly	self-learning	non time series	rule modelling	W&S A.4[a]
			descriptive statistics	IDES A.3, NIDES A.14, EMERALD A.19, JiNao A.18, Haystack A.1
		time series	ANN	Hyperview(1)[b] A.8
	programmed	descriptive stat	simple stat	MIDAS(1) A.2, NADIR(1) A.7, Haystack(1)
			simple rule-based	NSM A.6
			threshold	ComputerWatch A.5
		default deny	state series modelling	DPEM A.12, JANUS A.17, Bro A.20
signature	programmed	state-modelling	state-transition	USTAT A.11
			petri-net	IDIOT A.13
		expert-system	NIDES A.14, EMERALD A.19, MIDAS-direct A.2, DIDS A.9, MIDAS(2) A.2	
		string-matching	NSM A.6	
		simple rule-based	NADIR A.7, NADIR(2) A.7, ASAX A.10, Bro A.20, JiNao A.18, Haystack(2) A.1	
signature inspired	self-learning	automatic feature sel	Ripper A.21	

[a] Letter and number provide reference to section where it is described [b] Number in brackets indicates level of two tiered detectors

Figure 10 – Classification des principes de détection

À partir de cette catégorisation, on peut classer les différentes techniques par ordre d'efficacité.

Les techniques de détection d'anomalie sont les moins efficaces. Pour détecter le comportement anormal d'une entité (utilisateur, processus, programme...), le système enregistre un certain nombre de mesures statistiques relatives au comportement de l'utilisateur par exemple la durée de connexion ou les horaires de connexion. Lorsque l'utilisateur dévie de son comportement normal tel qu'il a été identifié auparavant, on peut alors supposer que l'utilisateur est suspect. Cette technique à l'avantage de pouvoir détecter

des attaques nouvelles et imprévisibles, car elle ne repose pas sur des connaissances que l'on pourra avoir. Cependant, elle a aussi plusieurs inconvénients. Le premier est que pour que le système fonctionne, il faut que les utilisateurs autorisés aient des comportements peu variables comme le fait de se connecter à peu près toujours aux mêmes heures, sinon il est beaucoup plus incertain d'en déduire un comportement anormal. La plupart du temps, un comportement anormal ne signifie pas une attaque, mais juste un changement de comportement de l'utilisateur. Cette technique laisse donc passer beaucoup de fausses alarmes (faux positifs et faux négatifs), c'est ensuite à un opérateur de faire le tri. Le deuxième problème est que cette technique peut être contournée si l'attaquant sait comment fonctionne le système. En effet, il peut très bien dès sa première connexion ou par la suite modifier son comportement de manière régulière pour que le système ne s'en aperçoive pas.

Les techniques de détection par signature sont beaucoup plus efficaces. Le principe est de définir un certain nombre de règles en se basant sur des schémas bien connus d'intrusion. Avec cette technique, il est possible de détecter beaucoup plus de menaces, car beaucoup d'attaques suivent un scénario prédéfini. De plus, il y a beaucoup moins de chance de tomber sur une fausse alerte, car les schémas sont très précis. Le seul inconvénient de ces techniques est que l'on ne peut détecter les attaques qui ne suivent pas un schéma connu. Cependant puisque ce ne sont pas la majorité des attaques, cette technique est bien plus efficace que la détection d'anomalie.

Pour finir, les techniques inspirées par la détection de signatures sont les plus efficaces. Elles sont l'association des deux techniques vues précédemment, détection d'anomalie et détection par signature. Elles permettent donc de détecter plus d'attaques, mais aussi de déterminer de manière plus sûre que l'on a affaire à une attaque. En effet, si l'on détecte un modèle bien défini d'une attaque et qu'en plus on est capable de déterminer que cette attaque se produit en pleine nuit, heure à laquelle l'utilisateur n'a jamais accédé au système, on peut davantage affirmer qu'une attaque est en cours.

On peut évaluer ces différentes techniques selon le critère de leur efficience en fonction du type d'attaque à détecter. On rappelle que les différents types d'attaque sont les suivants :

nouvelles (qui n'agissent pas selon un modèle connu par le système) ou connues (qui agissent selon un modèle connu). On peut alors mesurer le taux de fausse alerte et de non-détection en mettant en place différents scénarios des deux types. À noter qu'un système qui détecte uniquement des attaques de type connues sera considéré comme plus efficace qu'un système qui détecte uniquement des attaques de type nouvelles, car il se trouve que statistiquement, les attaques nouvelles sont les moins courantes.

La dernière étape de la gestion de l'audit est la manière dont réagit le système lorsqu'il détecte une menace.

Il peut tout d'abord réagir de manière passive en déclenchant une alarme pour prévenir l'opérateur d'examiner le problème. Le principal problème de cette technique est le manque de réactivité. Si l'opérateur n'est pas disponible au moment de l'alerte ou bien s'il tarde à analyser le problème, l'attaque ne pourra être empêchée, il faudra alors prendre des mesures correctives si cela est possible. De plus, cela demande un employé pour l'analyse des menaces.

La deuxième manière de réagir est d'effectuer une action corrective. C'est-à-dire que le système de manière autonome va réparer les dégâts occasionnés au fur et à mesure qu'il les détecte. Cette technique est plus réactive que la précédente, car c'est le système qui agit et non un opérateur. Cependant, si le but de l'attaque est de voler des données, cela ne pourra être empêché.

La troisième manière est que le système effectue de manière autonome une action défensive. Pour cela, il faut prévoir ce que l'attaquant va faire et mettre en place des actions pour l'en empêcher. La confidentialité ainsi que l'intégrité des données sont alors assurées.

Pour finir, on peut combiner les actions défensives et les actions correctives en cas d'échec de l'action défensive.

Annexe B

Types de conflits transactionnels

L'accès simultané de plusieurs utilisateurs aux mêmes données peut introduire des problèmes d'intégrité. Ainsi, deux transactions sont en conflit si l'exécution de T1 en parallèle à T2 donne un résultat différent de T1 suivie de T2, ou de T2 suivie de T1. Nous présentons dans cette annexe une liste des différentes catégories d'erreur répertoriées par ElMasri [16].

Perte de mise à jour (ou écriture impropre) : Une transaction T2 modifie une donnée qui a au préalable été modifiée par une transaction T1 dont l'exécution n'est pas finie. Si l'une des deux transactions est ensuite annulée, on ne sait alors pas quelle devrait être la valeur de la donnée. Voici un exemple :

Begin(T1)	
Read(X)	Begin(T2)
	Read(X)
Write(X)	
Commit(T1)	Write(X)
	Abort(T2)

Figure 11 – Exemple de perte de mise à jour

Lecture impropre : Une transaction T2 lit une donnée qui a au préalable été modifiée par une transaction T1 dont l'exécution n'est pas finie. Si la transaction T1 est ensuite annulée ou modifie de nouveau la donnée lue, la transaction T2 utilisera alors une donnée erronée. Voici deux exemples :

Begin(T1)	
Write(X)	Begin(T2)
	Read(X)

	Commit(T2)
Write(X)	
Commit(T1)	

Figure 12 – Exemple de lecture impropre 1

Begin(T1)	
	Begin(T2)
	Write(X)
Read(X)	
Commit(T1)	
	Abort(T2)

Figure 13 – Exemple de lecture impropre 2

Lecture non reproductible : Une transaction T2 modifie une donnée qui a au préalable été lue par une transaction T1 dont l'exécution n'est pas finie. Si la transaction T1 relie ensuite la donnée alors que la modification a été faite, les deux valeurs lues ne seront alors pas les mêmes. Voici un exemple :

Begin(T1)	
Read(X)	Begin(T2)
	Write(X)
	Commit(T2)
Read(X)	

Figure 14 – Exemple de lecture non reproductible

Références fantômes : Une transaction T2 insère un tuple qui vérifie la condition d'une sélection qui a au préalable été modifiée par une transaction T1. Si la transaction T1 effectue une deuxième fois cette sélection, le résultat ne sera donc plus le même. Voici un exemple :

Begin(T1)	
Select *	Begin(T2)
	delete x; insert y;
	Commit(T2)
Select *	

Figure 15 – Exemple de références fantômes

Agrégation incorrecte : Une transaction T1 modifie plusieurs données. Alors que T1 n'a pas fini de modifier les données, la transaction T2 calcule la somme de ces données. Le résultat sera alors erroné, car l'agrégation prendra en compte qu'une partie des mises à jour faites par la transaction T1. Voici un exemple :

Begin(T1)	
Read(X)	Begin(T2)
X:=X-1	
Write(X)	
	sum:=0
	Read(X)
	sum:=sum+X
	Read(Y)
	sum:=sum+Y
	Commit(T2)
Read(Y)	
Y:=Y+1	
Write(Y)	
Commit(T1)	

Figure 16 – Exemple d'agrégation incorrecte

Ce dernier type de problème n'est souvent pas mentionné, car assimilé au problème de lecture non reproductible. Cependant, nous le considérons comme un problème de nature différente, car il nécessite de verrouiller l'accès non pas à une valeur, mais à toutes les valeurs de la colonne sur laquelle porte l'agrégation. Or, pour ne pas bloquer l'accès à un trop grand nombre de données et ainsi améliorer l'efficience, le choix de ne pas le considérer est souvent adopté.

Ce dernier cas n'est pas mentionné par la norme ISO/ANSI:2003. Cependant, la définition même du niveau d'isolation « *Serializable* » de la norme indique que l'exécution de transactions en concurrence doit donner le même résultat que l'exécution en série de ces mêmes transactions, des erreurs d'agrégation incorrecte ne doivent donc pas pouvoir avoir lieu. Pour les autres niveaux d'isolation, le résultat est incertain.

Annexe C

Fonctionnalités Exprimables

I	Types prédéfinis	Exemple SQL ANSI/ISO : 2003
O	Char	CHAR
P	Var Char	VARCHAR
O	Grand Var Char	CLOB
O	Char Encodage	NCHAR
O	Var Char Encodage	NVARCHAR
O	Grand VarChar Encodage	NCLOB
O	Grande Chaine Bit	BLOB
P	Numérique Exacte	NUMERIC
P	Virgule Flottante Binaire 32 b	REAL
O	Virgule Flottante Binaire 64 b	DOUBLE PRECISION
O	Virgule Flottante Binaire 128 b	Non présent
O	Virgule Flottante Décimale 64 b	Non présent
O	Virgule Flottante Décimale 128 b	Non présent
O	Entier 16 b	SMALLINT
P	Entier 32 b	INTEGER
O	Entier 64 b	BIGINT
P	Booléen	BOOLEAN
O	Date	DATE
P	Estampille	TIMESTAMP
P	Intervalle Temps	INTERVAL
O	Tableau	ARRAY
O	Multi Ensemble	MULTISET
I	**Création de schéma**	**Exemple SQL ANSI/ISO : 2003**
P	Création Type Dérivé	Deux mécanismes possibles dans SQL 2003 : Héritage simple : CREATE TYPE subtype UNDER supertype AS (entier INTEGER) Type equivalent : CREATE DOMAIN monEntier AS

		INTEGER Mécanismes non présents : héritage multiple, interface, clonage à partir de l'en-tête d'un tuple ou d'une relation
P	Création Type Contrainte	CREATE DOMAIN monEntier AS INTEGER CHECK (1=2)
P	Création Type Composite (agrégat, classe ou tuple)	CREATE TYPE composite AS (entier INTEGER, chaîne VARCHAR[5]);
P	Modification Type	ALTER TYPE monType ADD ATTRIBUTE (entier INTEGER)
P	Suppression Type	DROP TYPE monType
P	Création Relation	CREATE TABLE MaTable (entier INTEGER)
P	Ajout Colonne Relation	ALTER TABLE MaTable ADD entier INTEGER
P	Modification Colonne Relation	ALTER TABLE MaTable ALTER entier SET DEFAULT 0
P	Suppression Colonne Relation	ALTER TABLE MaTable DROP entier
P	Suppression Relation	DROP TABLE MaTable
P	Insertion Tuple	INSERT INTO MaTable (entier) VALUES 1
P	Modification Tuple	UPDATE MaTable SET entier=0 WHERE entier=1
P	Modification Tuples Cascade	CREATE TABLE Table1 (entier INTEGER, CONSTRAINT fk FOREIGN KEY[entier] REFERENCES Table2[entier] ON UPDATE CASCADE); UPDATE Table1 SET entier=0 WHERE entier=1
P	Suppression Tuple	DELETE FROM MaTable WHERE entier=0;
O	Suppression Tuples Cascade	CREATE TABLETable1 (entier INTEGER, CONSTRAINT fk FOREIGN KEY(entier) REFERENCES Table2(entier) ON DELETE CASCADE); DELETE FROM Table1 WHERE entier=0;
P	Création Fonction	CREATE FUNCTION increment (IN entier INTEGER) RETURNS INTEGER RETURN(i+1);
P	Définition Operateur Agrégation	Non présent
P	Modification Fonction	Non présent
P	Suppression Fonction	DROP FUNCTION increment
P	Integrite Axiomatique Schemas	Deux façons : CREATE ASSERTION monAssertion CHECK (SELECT

		MAX[i] FROM table1 < SELECT MAX[j] FROM table2)
		
		CREATE TRIGGER monTrigger AFTER INSERT ON table1
		WHEN (SELECT MAX[i] FROM table1 < SELECT MAX[j]
		FROM table2)
		BEGIN
		DELETE FROM table1 WHERE id=new.id;
		END;

P	Integrite Referentielle	CREATE TABLE Cle (entier INTEGER, PRIMARY KEY [entier]); CREATE TABLE Reference (entier INTEGER, CONSTRAINT fk FOREIGN KEY (entier) REFERENCES Cle (entier));
I	**Opérations sur les Relations**	**Exemple SQL ANSI/ISO : 2003**
P	Selection	SELECT DISTINCT * FROM MaTable WHERE monAttribut=0
P	Projection	SELECT DISTINCT monAttribut FROM MaTable
P	Union	SELECT Table1 UNION Table2
P	Intersection	SELECT Table1 INTERSECT Table2
P	Différence	SELECT Table1 EXCEPT Table2
P	Produit Cartesien	SELECT DISTINCT attribut1, attribut2 FROM maTable1, maTable2
P	Theta Join	SELECT DISTINCT Table1 t1 JOIN Table2 t2 ON t1.attribut1>t2.attribut1
O	Natural Join	SELECT DISTINCT Table1 NATURAL JOIN Table2
O	Division	(SELECT colonne_non_commune FROM table_divise) MINUS ([SELECT colonne_non_commune FROM (SELECT * FROM table_diviseur, (SELECT colonne_non_commune FROM table_divise) T1 MINUS SELECT * FROM table_divise) T2])
P	Renommage Relation	SELECT DISTINCT * FROM (SELECT * FROM MaTable) AS alias
P	Renommage Colonne	SELECT DISTINCT attribut AS alias FROM MaTable
O	Comptage	SELECT DISTINCT COUNT(*) FROM MaTable
O	Moyenne	SELECT DISTINCT AVG(attribut) FROM MaTable

O	Somme	SELECT DISTINCT SUM(attribut) FROM MaTable
O	Minimum	SELECT DISTINCT MIN(attribut) FROM MaTable
O	Maximum	SELECT DISTINCT MAX(attribut) FROM MaTable
O	Groupement	SELECT DISTINCT * FROM MaTable GROUP BY attribut
O	Tri	SELECT DISTINCT * FROM MaTable ORDER BY attribut DESC
O	Fenêtrage	SELECT DISTINCT * FROM MaTable ORDER BY attribut DESC
P	Fermeture Transitive	WITH RECURSIVE closure (head, tail, path) AS (SELECT head, tail, head || '.' || tail || '.' AS path FROM table UNION ALL SELECT tc.head, t.tail, tc.path || t.tail || '.' AS path FROM table AS t JOIN closure AS tc ON t.head = tc.tail WHERE tc.path NOT LIKE '%' || t.tail || '.%') SELECT DISTINCT head, tail FROM closure ORDER BY head, tail
P	Imbrication des opérateurs	Possible dans la clause FROM : SELECT * FROM (SELECT * FROM table) t1 Possible dans la clause WHERE : SELECT * FROM table WHERE colonne IN (SELECT * FROM table) Pas de limitation spécifiée
I	**Opération sur les Collections**	**Exemple SQL ANSI/ISO : 2003**
P	Sélection	SELECT * FROM table WHERE colonne=1 AND colonne=2
P	Projection	SELECT colonne1, colonne2 FROM table
P	Union	SELECT * FROM table1 UNION ALL SELECT * FROM table2
P	Intersection	SELECT * FROM table1 INTERSECT ALL SELECT * FROM table2
P	Différence	SELECT * FROM table1 MINUS ALL SELECT * FROM table2
P	Produit Cartésien	SELECT * FROM table1, table2
P	Jointure Theta	SELECT * FROM table1 JOIN table2 ON (table1.colonne >

		table2.colonne)
O	Jointure Naturelle	SELECT * FROM table1 NATURAL JOIN table2
O	Division	Non présent
P	Renommage Collection	SELECT colonne FROM table t1 WHERE t1.colonne=1
P	Renommage Colonne	SELECT colonne AS alias FROM table
O	Comptage	SELECT COUNT(*) FROM table
O	Moyenne	SELECT AVG(colonne) FROM table
O	Somme	SELECT SUM(colonne) FROM table
O	Minimum	SELECT MIN(colonne) FROM table
O	Maximum	SELECT MAX(colonne) FROM table
O	Groupement	SELECT colonne FROM table GROUP BY (colonne)
O	Tri	SELECT colonne FROM table ORDER BY colonne DESC
O	Fenêtrage	SELECT colonne FROM table WHERE ROWNUM<2
O	Fermeture Transitive	WITH RECURSIVE closure (head, tail, path) AS (SELECT head, tail, head || '.' || tail || '.' AS path FROM table UNION ALL SELECT tc.head, t.tail, tc.path || t.tail || '.' AS path FROM table AS t JOIN closure AS tc ON t.head = tc.tail WHERE tc.path NOT LIKE '%' || t.tail || '.%') SELECT head, tail FROM closure ORDER BY head, tail
P	Imbrication des opérateurs	Possible dans la clause FROM : SELECT * FROM (SELECT * FROM table) t1 Possible dans la clause WHERE : SELECT * FROM table WHERE colonne IN (SELECT * FROM table) Pas de limitation spécifiée
I	**Représentation des données manquantes**	**Exemple SQL ANSI/ISO : 2003**
P	Insertion Tuple Incomplet	INSERT INTO table VALUES NULL
P	Selection Valeur Manquante	SELECT * FROM table WHERE colonne IS NULL
P	Modification Valeur Manquante	UPDATE table SET colonne=1 WHERE colonne IS NULL
P	Traitement Cas Particuliers	Non Présent
P	Left Outer Join	SELECT * FROM Table1 LEFT OUTER JOIN Table2 ON Table1.colonne=Table2.colonne
P	Right Outer Join	SELECT * FROM Table1 RIGHT OUTER JOIN Table2 ON

		Table1.colonne=Table2.colonne
P	Full Outer Join	SELECT * FROM Table1 FULL OUTER JOIN Table2 ON Table1.colonne=Table2.colonne
I	**Opérateurs numériques**	**Exemple SQL ANSI/ISO : 2003**
P	Addition	1 + 1
P	Soustraction	1 - 1
P	Multiplication	1 * 1
P	Division	1 / 1
P	Racine Carre	SQRT(1)
O	Arrondis	Non Présent
O	Troncature	Non Présent
P	Plafond	CEIL(1.4)
P	Plancher	FLOOR(1.4)
P	Valeur absolue	ABS(-1)
P	Exponentielle	EXP(2)
O	Logarithme	Non Présent
P	Logarithme néperien	LN(2)
P	Modulo	MOD(15, 3)
P	Puissance	POWER(2, 2)
O	Signe	Non Présent
I	**Opérateurs sur les chaînes**	**Exemple SQL ANSI/ISO : 2003**
P	Concaténation	'chaine1' || 'chaine2'
P	Sous Chaine	SUBSTRING('chaîne', FROM 1 FOR 10)
P	Majuscules	UPPER('chaîne')
P	Minuscules	LOWER('chaîne')
P	Longueur	CHAR_LENGTH('chaîne')
P	Contient	LIKE '%souschaine%'
P	Position Sous Chaine	POSITION ('souschaine' IN 'chaîne')
P	Remplacement Sous chaîne	OVERLAY ('chaine' PLACING 'souschaine' FROM 1 FOR 10)
I	**Opérateurs booléens**	**Exemple SQL ANSI/ISO : 2003**
P	Et	AND
P	Ou	OR
O	Ou Exclusif	Non présent
P	Non	NOT
P	Imbrication des opérateurs	SELECT * FROM table WHERE colonne=1 AND colonne=2 AND colonne=3 ... Pas de limite spécifiée
I	**Opérateurs de comparaison**	**Exemple SQL ANSI/ISO : 2003**
P	Égal	valeur1=valeur2

O	Different	valeur1<>valeur2
P	Superieur	valeur1>valeur2
P	Inférieur	valeur1<valeur2
O	Supérieur Ou Egal	valeur1>=valeur2
O	Inférieur Ou Égal	valeur1<=valeur2
I	**Structures Conditionnelles**	**Exemple SQL ANSI/ISO : 2003**
P	While	WHILE condition DO statements END WHILE
O	Do While	REPEAT statements UNTIL condition END REPEAT

P	If	IF condition THEN
		statements
		ELSE
		statements
		END IF

O	For	FOR compteur IN min..max DO statements END FOR
O	Switch	Dans la portion procédurale du langage : CASE variable WHEN condition_1 THEN resultat_1; WHEN condition_2 THEN resultat_2 ; ELSE resultat END CASE Dans la portion SQL du langage : SELECT variable, CASE variable WHEN 1 THEN resultat_1; WHEN 2 THEN resultat_2 ; ELSE resultat END FROM table
P	Retour de valeur	RETURN valeur
P	Saut	GOTO label
P	Récupération d'exception	WHENEVER SQLSTATE(0) CONTINUE

P	Itération de Curseur	FOR variable AS curseur CURSOR FOR DO END FOR

P	Imbrication des opérateurs	DECLARE
		v_i INTEGER:=0;
		BEGIN
		WHILE v_i=0 LOOP
		WHILE v_i=0 LOOP
		v_i:=1;
		END LOOP;
		END LOOP;
		END;
		
		Pas de limite définie

I	**Autres opérateurs algorithmiques**	**Exemple SQL ANSI/ISO : 2003**
P	Affectation	SET variable = valeur
P	Déclaration de variables	DECLARE variable TYPE

I : Importance P : Primordiale O : Optionnelle

Annexe D

Calcul des métriques de complexité

Halstead

Nous rappelons que la métrique consiste à compter le nombre d'opérateurs et d'opérandes distincts et totaux.

Il nous a été très difficile d'extraire les grands principes sous-jacents du calcul de la métrique de Halstead, car ils ne sont pas très bien définis par Halstead lui-même. Par exemple, il ne donne aucune information sur comment identifier les opérateurs et les opérandes. Pour les définir, nous nous referons donc aux définitions données par le grand dictionnaire terminologique [41] :

- Un opérande est une donnée, une valeur ou un nombre entrant dans une opération arithmétique ou logique, ou une instruction.
- Un opérateur est une instruction mathématique servant à préciser l'opération à effectuer sur un ou plusieurs opérandes.

De plus, Halstead définit un principe important qui est qu'un algorithme est constitué d'opérateurs et d'opérandes et de rien d'autre [19] (p8). Il suffit donc de déterminer qu'un élément n'est pas un opérande pour en déduire que c'est un opérateur (et inversement).

Un autre principe important que Halstead ne définit pas clairement est qu'un opérateur peut correspondre à plusieurs unités lexicales. Halstead définit ce principe uniquement en référent à quelques exemples. Ainsi, si l'on considère les deux éléments lexicaux **BEGIN END** qui permettent de délimiter le début et la fin d'un groupe d'instruction, ils doivent être considérés comme un unique opérateur, car ils contribuent à effectuer la même fonctionnalité.

Ce principe est très clair dans certains cas. Par exemple, si l'on considère les éléments lexicaux **CREATE TABLE** qui permettent d'exprimer la création d'une nouvelle relation, il est logique de le considérer comme un seul opérateur. On aurait très bien pu utiliser un seul élément lexical pour le définir (par ex. : **CreateTable**), cela n'aurait rien changé à la complexité.

Par contre, il y a des cas où ce principe est ambigu, car on peut identifier des fonctionnalités qui sont composées de sous fonctionnalités. Par exemple, si l'on considère le morceau de code suivant : **maFonction(a,b,c).**

On peut compter les parenthèses et les virgules comme des éléments lexicaux de l'appel de la fonction **maFonction**. On peut aussi compter les parenthèses comme un opérateur distinct qui permet de regrouper les paramètres de la fonction et chaque virgule comme un opérateur qui permet de séparer les paramètres de la fonction.

Nous avons choisi de compter les parenthèses et les virgules comme étant des opérateurs distincts. Cela reflète mieux l'esprit de la métrique tel que défini par Halstead qui est de comptabiliser tous les éléments. De plus, cela nous évitera d'éliminer des éléments qui ne nous semblent pas être révélateurs de complexité alors qu'en réalité ils pourraient l'être. Cependant ce choix est discutable.

McCabe

Nous rappelons que la métrique consiste à évaluer le nombre de chemins possibles dans le programme.

Pour calculer cela, McCabe propose de compter les structures de contrôle qui modifie l'ordre séquentiel d'exécution du programme. Une structure de contrôle est un mode d'organisation permettant de combiner des opérations et de définir le déroulement des traitements dans un programme [41]. Par exemple, on peut citer les structures de contrôle suivantes : boucle, récursivité, test si, saut, appel de fonction...

McCabe indique aussi qu'il faut compter le nombre de prédicats présents dans la condition de la structure de contrôle, car cela masque l'utilisation de plusieurs structures en une seule. Par exemple, la structure de contrôle suivante : **IF (a AND b) THEN**. La structure de contrôle comporte deux prédicats (**a** et **b**). On pourrait la réécrire de la manière suivante : **IF (a) THEN IF (b) THEN**. Le nombre de prédicats a donc une influence sur le nombre de chemins possible.

L'adaptation de la métrique à un langage relationnel demande une réflexion supplémentaire pour identifier les structures de contrôles présentent. Le modèle relationnel est un langage qui tente de masquer au maximum les structures de contrôles à l'utilisateur pour simplifier son utilisation. Cependant, certaines structures conditionnelles n'ont pas pu être éliminées. Ainsi lorsque l'on fait une opération de sélection, il faut préciser les conditions de la sélection des tuples. Bien que prenant une forme non usuelle, ces conditions sont celles d'une structure de contrôle qui s'apparenterait à un **IF**, il faudra donc les compter comme telles.

Wang et Shao

Nous rappelons que la métrique consiste à évaluer la complexité d'un programme en prenant en compte la décomposition en modules, le nombre d'entrées-sorties des modules et la complexité des structures de contrôle qui compose chaque module.

Pour la décomposition en modules, nous définissons un module comme étant une suite de requêtes formant une unité transactionnelle. La définition d'une routine est donc considérée comme un module de même qu'une suite de requêtes en dehors de toute routine.

Pour les entrées-sorties, il nous faudra non seulement prendre en compte les entrées-sorties explicitement définies pour le module (comme les paramètres d'une routine), mais il nous faudra aussi identifier les effets de bord. Pour cela, nous comptabiliserons le nombre de relations distinctes utilisées dans les différentes instructions du module, car on peut les voir comme des variables lues et modifiées.

Pour l'identification des structures de contrôles, nous suivrons les mêmes règles que pour le calcul de la métrique de McCabe. Si aucune structure de contrôle n'est détectée, il s'agit d'une séquence.

L'article original de Wang et Shao [55] présente la façon de mettre en relation ces différents paramètres pour obtenir la valeur de la métrique.

Annexe E

Manuel de référence

E.1 Application « AtelierEssai »

E.1.1 Présentation

L'application permet d'enregistrer les tests et les essais dans une base de données de manière à constituer une base de données d'essais constitués de tests prêts à être exécutés.

Figure 17 – Interfaces externes de l'application « AtelierEssai »

La base de données « BDResultat » conservera aussi par la suite les résultats d'exécution des tests.

E.1.2 Grammaire « EssaiTest »

Les fichiers décrivant les tests et les essais doivent respecter la grammaire décrite ci-dessous.

Un fichier permet de décrire des domaines, des essais et des tests.

Il est aussi possible d'ajouter des commentaires en commençant une ligne par '--'. Ceux-ci seront totalement ignorés lors de l'analyse du fichier.

```
Fichier ::= Domaines Tests Essais
```

Un essai est défini par un nom, une version, des paramètres ainsi que la liste des tests qui le compose.

```
Essais ::= (Essai)*

Essai ::= '#' 'Essai' 'Nom' '=' Nom 'Version' '='
Version ParametresEssai ReferencesTests
```

L'élément lexical <ID> correspond à un identifiant tel que défini dans Java auquel on a ajouté la possibilité d'utiliser le caractère '.'.

```
Nom ::= <ID>
```

L'élément lexical <NUM> correspond à un nombre entier.

```
Version ::= <NUM>
```

Pour l'instant aucun paramètre n'est implémenté.

```
ParametresEssai ::= ['Parametres' '=' (ParametreEssai)+]

ParametreEssai ::= Nom Valeur

Valeur ::= <NUM>
```

Une référence vers un test est composée de son nom et accessoirement de sa version.

Si la version du test n'est pas renseignée, la dernière version sera utilisée.

```
ReferencesTests ::= (ReferenceTest)+

ReferenceTest ::= '#' 'ReferenceTest' 'Nom' '=' Nom
['Version' '=' Version]
```

Un test est compose de son nom, de sa version, d'une courte description de son objectif, des plateformes spécifiques sur lesquelles il peut être exécutés, d'une section de requêtes à exécuter en prélude, d'une section de requêtes à exécuter après le prélude, d'une section de requêtes à exécuter en postlude, d'une liste de mesure à prendre, de la requête permettant de

vérifier la validité du résultat et, éventuellement, si le test doit échoué ou non (par défaut on considère que le test doit réussir).

```
Tests ::= (Test)*

Test ::= '#' 'Test' 'Nom' '=' Nom 'Version' '=' Version
'Objectif' '=' Objectif Specificites Prelude Corps
Postlude Mesures Resultat [Echec]
```

L'élément lexical <CHAINE> correspond à tout type de caractère à part le retour chariot.

```
Objectif ::= <CHAINE>
```

Les spécificités permettent de savoir sur quelles plateformes les tests peuvent être exécutés. Les valeurs possibles sont amenées à évoluer.

```
Specificites ::= (Couple)*

Couple ::= 'Specificite' '=' Sgbd ',' Os

Sgbd ::= ('ORACLE' | 'POSTGRESQL' | 'MYSQL')

Os ::= ('WINDOWS' | 'LINUX')
```

Les domaines permettent de définir un ensemble de valeur parmi lesquels les générateurs choisissent arbitrairement des valeurs. Les types de données possibles sont STRING, DATE, INT ou FLOAT.

```
Domaines ::= (Domaine)*

Domaine ::= '#' 'Domaine' Nom (DomaineString |
DomaineDate | DomaineInt | DomaineFlottant)

DomaineString ::= 'STRING' Chaine (',' Chaine)*

Chaine ::= <CHAINE>

DomaineDate ::= 'DATE' Date (',' Date)*

Date ::= <DATE>

DomaineInt ::= 'INT' EntierSigne (',' EntierSigne)*

EntierSigne ::= ['-'] <NUM>

DomaineFlottant ::= 'FLOAT' FlottantSigne (','
FlottantSigne)*

FlottantSigne ::= ['-'] <FLOAT>
```

Le prélude est la section du test qui est exécutée avant le début de la prise de mesure.

```
Prelude ::= 'Prelude' '=' Sequence
```

Le corps est la section du test qui sur lequel porte la prise de mesure.

```
Corps ::= 'Corps' '=' Sequence
```

Le postlude est la section du test qui est exécutée après la fin de la prise de mesure et après le calcul du résultat.

```
Postlude ::= 'Postlude' '=' Sequence
```

Une séquence est un ensemble de requête ou de générateurs.

```
Sequence ::= [LangageRequete] (Requete |
GenerateurTuples | GenerateursProcedure)*

LangageRequete ::= ('TD' | 'AQL')
```

Une requête peut comporter n'importe quel caractère. Si la requête est définie sur plusieurs lignes, chaque nouvelle ligne doit commencer par le caractère '+'.

```
Requete ::= 'Requete' '=' <STRING> (<EOL> '+'
<STRING>])*
```

Un générateur de tuples permet de générer automatiquement un grand nombre de requêtes d'insertion dans une table. Il faut pour le définir fournir le nom de la table ainsi que la liste de ses attributs.

```
GenerateurTuples ::= 'GenTuple' '=' Nbr 'Table' '=' '('
ListeChamps ')'

ListeChamps ::= Champ (',' Champ)*

Champ ::= NonCle | Cle
```

Les champs clé sont automatiquement des entiers dont on générera les valeurs de manière séquentielle pour s'assurer d'avoir des valeurs uniques. Il suffit donc de spécifier les valeurs minimum et maximum désirées pour cet entier. Ces valeurs peuvent être négatives.

```
Cle ::= Nom MinEntier MaxEntier

MinEntier ::= 'MIN' '=' [-] <NUM>

MaxEntier ::= 'MAX' '=' [-] <NUM>
```

Les champs qui ne sont pas des clés peuvent être des entiers, des flottants, des dates ou des chaînes de caractères.

```
NonCle ::= (ChampEntier | ChampFlottant | ChampDate |
ChampString)
```

Pour les entiers, il faut spécifier le minimum et le maximum ou le domaine à partir duquel on veut générer les valeurs.

```
ChampEntier ::= 'INT' Nom ([MinEntier MaxEntier] |
'FROM' NomDomaine)
```

Pour les flottants, il faut spécifier le minimum et le maximum ou le domaine à partir duquel on veut générer les valeurs.

```
ChampFlottant ::= 'FLOAT' Nom ((MinFlottant MaxFlottant)
| 'FROM' NomDomaine)
```

L'élément lexical <FLOAT> correspond à un nombre suivi d'un point suivi d'un autre nombre.

```
MinFlottant ::= 'MIN' '=' [-] <FLOAT>

MaxFlottant ::= 'MAX' '=' [-] <FLOAT>
```

Pour les dates, il faut spécifier le domaine à partir duquel on veut générer les valeurs. Si aucun domaine n'est spécifié, le générateur utilisera la date courante du système.

```
ChampDate ::= 'DATE' Nom ['FROM' NomDomaine]
```

Pour les chaînes de caractères, il faut spécifier le minimum et le maximum de la taille de la chaîne ou le domaine à partir duquel on veut générer les valeurs.

```
ChampString ::= 'STRING' Nom ([MinString MaxString] |
'FROM' NomDomaine)

MinString ::= 'MIN' '=' <NUM>

MaxString ::= 'MAX' '=' <NUM>

NomDomaine ::= <ID>
```

Un générateur de procédures permet de générer automatiquement un grand nombre de requêtes d'appel de procédures. Il faut pour le définir fournir le nom des procédures ainsi que la liste de ses paramètres pour chaque procédure.

```
GenerateurProcedure ::= 'GenProcedure' Nbr (Procedure)+

Procedure ::= 'Procedure' '=' Nom '(' [listeParametre]
')'

ListeParametre ::= Parametre (',' Parametre)*
```

Les paramètres peuvent être des entiers, des flottants, des dates, des chaînes de caractères, des tableaux d'entiers, des tableaux de flottants, des tableaux de dates ou des tableaux de chaînes de caractères.

```
Parametre ::= (ParametreEntier | ParametreFlottant |

ParametreDate | ParametreString | ParametreTableauEntier
| ParametreFlottant | ParametreDate | ParametreString |
ParametreTableauEntier | ParametreTableauFlottant |
ParametreTableauDate | ParametreTableauString)
```

Pour les tableaux d'entiers, il faut spécifier le minimum et le maximum ou le domaine à partir duquel on veut générer les valeurs, ainsi que le nombre d'éléments à générer.

```
ParametreTableauEntier ::= 'TABLEINT' nomTypeTableau '('
NombreElements ')' ([MinEntier MaxEntier] | 'FROM'
NomDomaine)
```

Pour les tableaux de flottant, il faut spécifier le minimum et le maximum ou le domaine à partir duquel on veut générer les valeurs, ainsi que le nombre d'éléments à générer.

```
ParametreTableauFlottant ::= 'TABLEFLOAT' NomTypeTableau
'(' NombreElements ')' ([MinFlottant MaxFlottant] |
'FROM' NomDomaine)
```

Pour les tableaux de date, il faut spécifier le domaine à partir duquel on veut générer les valeurs, ainsi que le nombre d'éléments à générer. Si aucun domaine n'est spécifié, le générateur utilisera la date courante du système.

```
ParametreTableauDate ::= 'TABLEDATE' NomTypeTableau '('
NombreElements ')' ['FROM' NomDomaine]
```

Pour les chaînes de caractères, il faut spécifier le minimum et le maximum de la taille de la chaîne ou le domaine à partir duquel on veut générer les valeurs ainsi que le nombre d'éléments à générer.

```
ParametreTableauString ::= 'TABLESTRING' NomTypeTableau
'(' NombreElements ')' ((MinString MaxString) | 'FROM'
NomDomaine)
```

Cet élément correspond au nom donné au constructeur de tableau dans le langage utilisé.

```
NomTypeTableau ::= <ID>
```

Les paramètres de type entier, flottant, date, chaîne de caractères sont à définir de la même manière que les champs qui ne sont pas des clés.

```
ParametreEntier ::= 'INT' Nom ([MinEntier MaxEntier] |
'FROM' NomDomaine)

ParametreFlottant ::= 'FLOAT' Nom ((MinFlottant
MaxFlottant) | 'FROM' NomDomaine)

ParametreDate ::= 'DATE' Nom ['FROM' NomDomaine]

ParametreString ::= 'STRING' Nom ([MinString MaxString]
| 'FROM' NomDomaine)
```

Une mesure est définie à partir de la méthode de prise de mesure à utiliser et du nom de la mesure à prendre.

```
Mesures ::= (Mesure)*

Mesure ::= 'Mesure' '=' TypeMesure NomMesure
```

Pour l'instant seul le type de prise de mesure LOCALE est implémenté

```
TypeMesure ::= ('LOCALE' | 'CIBLE' | 'SNMP')
```

TPS représente la durée d'exécution.

CPU représente le temps d'exécution du processeur.

NBECRITURE_ME représente le nombre d'écritures sur le disque.

NBLECTURE_ME représente le nombre de lectures sur le disque.

NBDEFAUT_MI représente le nombre défaut de page.

RESIDENTE_MI représente la mémoire résidente consommée.

```
NomMesure ::= ('TPS' | 'CPU' | 'NBECRITURE_ME' |
'NBLECTURE_ME' | 'NBDEFAUT_MI' | 'RESIDENTE_MI')
```

Pour spécifier la requête de résultat, la macro #R#(requête retournant 1 ou 0) doit être utilisée, car elle permet d'utiliser le nom de l'instance qui exécute le test pour stocker de manière unique les résultats dans une table temporaire.

```
Resultat ::= 'Resultat' '=' [LangageRequete] Requete
```

Si un échec du test est considéré comme un comportement normal, utiliser la valeur TRUE. Par défaut, la valeur est FALSE.

```
Echec ::= 'Echec' '=' ('TRUE' | 'FALSE')
```

E.1.3 Schéma « BDResultat »

Voici ci-dessous le schéma de la base de données où sont stockées les différentes versions des tests, des essais et des séances.

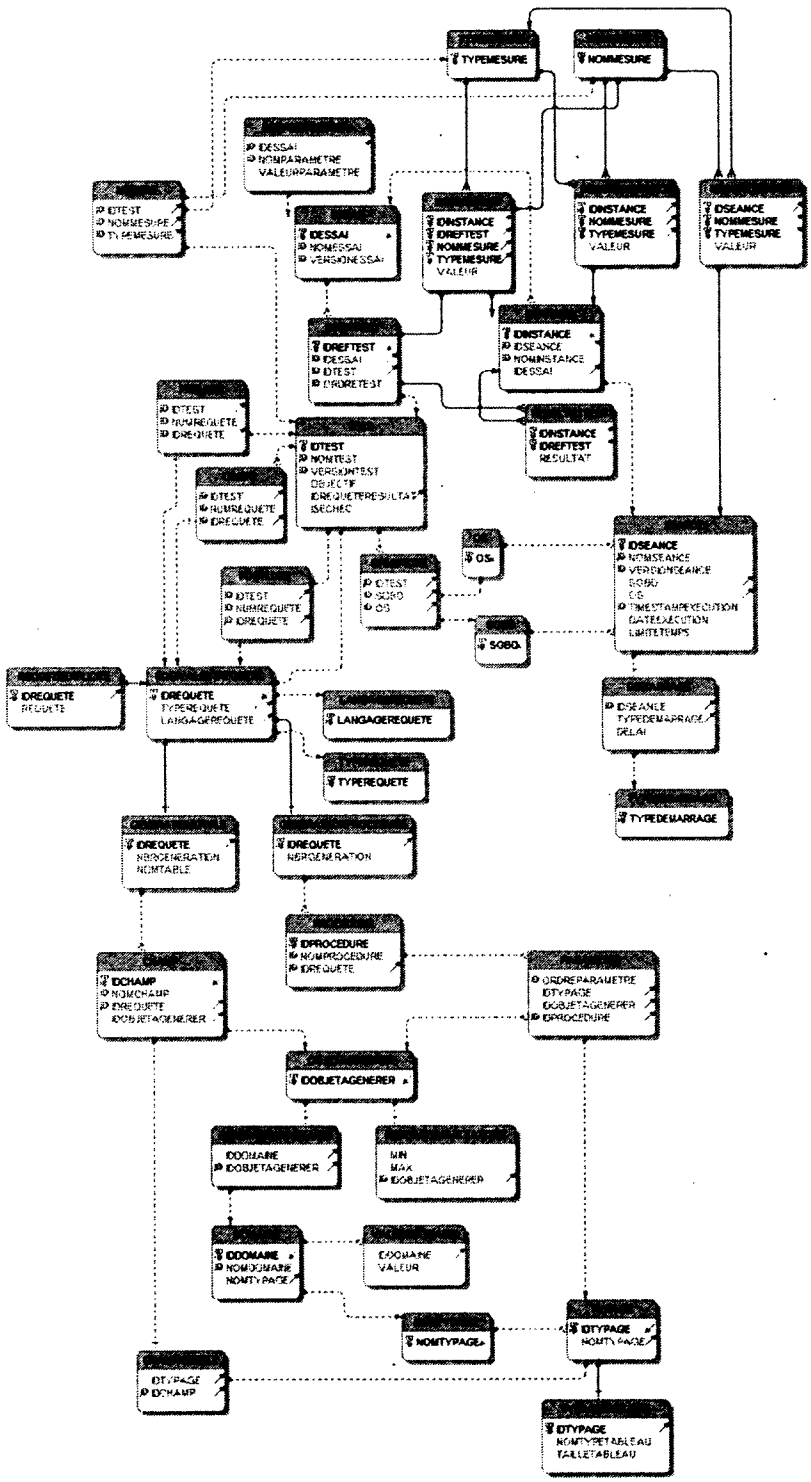

Figure 18 – Schéma de la BD où sont stockés la description et les résultats des tests

E.1.4 Lancement du programme

Le programme peut être lancé avec ou sans paramètres. Si aucun paramètre n'est renseigné en arguments, le renseignement des informations nécessaire au déroulement du programme (informations de connexion à la base de données et chemin du fichier contenant la description des tests et des essais) se fera via une interface console. « BDResultat » doit être une base de données Oracle, car c'est le langage SQL d'Oracle qui a été choisi pour implémenter les requêtes permettant de créer le schéma. Cependant d'autre SGBD utilisant la même syntaxe peuvent fonctionner.

Pour lancer le programme sans arguments, il faut exécuter la ligne de commande suivante :

```
java -jar AtelierEssai.jar
```

Le programme demande alors les informations nécessaires à son exécution. Pour arrêter en cours d'exécution, il suffit d'actionner les touches envoyant le signal SIGQUIT (Ctrl+D sous Linux ou Ctrl+Z sous Windows) à la place de rentrer les informations demandées.

Au lancement du programme, il est possible de spécifier les paramètres suivants :

–URL [urlConnexionBD] L'URL de connexion à la BD où l'on veut enregistrer la description des tests et des essais.

–user [utilisateurBD] L'utilisateur avec lequel on veut se connecter à la BD.

–pass [motDePasse] Le mot de passe associé à l'utilisateur.

–file [cheminFichier] Le chemin du fichier où sont décrits les tests et les essais à enregistrer.

La ligne de commande à exécuter ressemble alors à cela :

```
java -Djava.library.path=sigarlib -jar
AtelierEssai.jar - jdbc:oracle:thin:@//127.0.0.1:1521/xe
-user MonNom -pass MonMdp -file Tests/Test1.txt
```

E.2 Application « ExecuteurSeance »

E.2.1 Présentation

Cette application permet d'exécuter une séance de test sur un SGBD cible et d'enregistrer les résultats dans la base de données où sont stockés les tests.

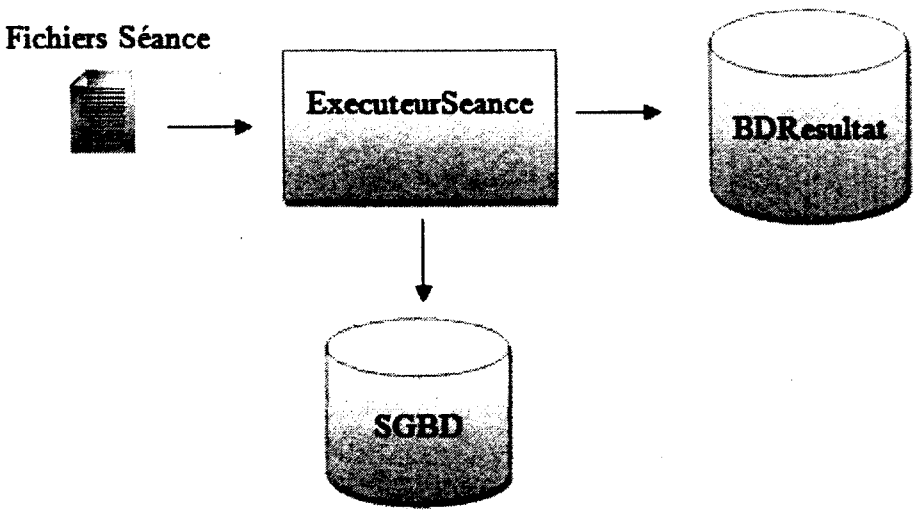

Figure 19 – Interfaces externes de l'application « ExecuteurSeance »

E.2.2 Grammaire « Seance »

Les fichiers décrivant les séances doivent respecter la grammaire décrite ci-dessous.

Il est possible d'ajouter des commentaires en commençant une ligne par '--'. Ceux-ci seront totalement ignorés lors de l'analyse du fichier.

Une séance est composée de son nom, de sa version, du nom du SGBD et du système d'exploitation (OS) sur lesquelles la séance va être exécutée, d'une limite de durée d'exécution à ne pas dépasser, du démarrage utilisé pour contrôler l'exécution essais, du

format de génération des rapports, de la liste de mesures à prendre sur la séance et de la liste des instances d'essais qui la compose.

```
Seance ::= Nom Version SgbdCible OsCible LimiteTemps
Demarrage FormatsRapports Mesures Instances

Version ::= 'Version' '=' <NUM>

Nom ::= 'Nom' '=' <ID>
```

La plateforme ciblée par la séance est un couple SGBD, système d'exploitation (OS). Cette spécificité permet de savoir dans quels langages le test doit être traduit. Mais, pour l'instant, il n'y a pas de traducteur implémenté. Ces données sont aussi utilisées pour les résultats pour indiquer sur quelles plateformes les tests ont été exécutés pour donner ces résultats. Les valeurs possibles sont amenées à évoluer.

```
SdbdCible ::= 'SGBD' '=' ('ORACLE' | 'POSTGRESQL' |
'MYSQL')

OsCible ::= 'OS '=' ('WINDOWS' | 'LINUX')
```

La limite de durée d'exécution est un entier qui détermine le nombre de secondes maximum que doit prendre l'exécution d'une séance, au-delà, la séance est interrompue.

```
LimiteTemps ::= 'LimiteTemps' '=' <NUM>
```

Le démarrage permet de spécifier si les instances d'essai vont être exécutées de manière séquentielle, de manière concurrente avec un démarrage simultané de tous les essais ou bien de manière concurrente avec un démarrage différé constant ou aléatoire.

```
Demarrage ::= 'Demarrage' '=' (Simultane | Differe |
Sequentiel)

Simultane ::= 'SIMULTANE'

Sequentiel ::= 'SEQUENTIEL'

Differe ::= TypeDemarrage Delai

TypeDemarrage ::= ('CONSTANT' | 'ALEATOIRE')
```

La valeur du délai est à définir en millisecondes.

```
Delai ::= <NUM>
```

Trois types de rapport sont possibles. Soit un rapport au format d'un fichier texte, soit un rapport affiché sur la console, soit un rapport au format d'un fichier CSV. Il est possible de générer les trois fichiers à chaque fois.

```
FormatsRapports ::= (FormatRapport)*

TypeRapport ::= ('CSV' | 'TEXTE' | 'CONSOLE')
```

Une mesure est définie à partir de la méthode de prise de mesure à utiliser et du nom de la mesure à prendre.

```
Mesures ::= (Mesure)*

Mesure ::= 'Mesure' '=' TypeMesure NomMesure
```

Pour l'instant seul le type de prise de mesure LOCALE est implémenté.

```
TypeMesure ::= ('LOCALE'|'CIBLE'|'SNMP')
```

TPS représente la durée d'exécution.

CPU représente le temps d'exécution du processeur.

NBECRITURE_ME représente le nombre d'écritures sur le disque.

NBLECTURE_ME représente le nombre de lectures sur le disque.

NBDEFAUT_MI représente le nombre défaut de page.

RESIDENTE_MI représente la mémoire résidente consommée.

```
NomMesure ::= ('TPS' | 'CPU' | 'NBECRITURE_ME' |
'NBLECTURE_ME' | 'NBDEFAUT_MI' | 'RESIDENTE_MI')
```

Une instance représente une exécution d'un essai. Cela permet d'exécuter plusieurs instances d'un même essai en concurrence ou en série dans une même séance.

```
Instances ::= (Instance)*
```

Pour décrire une instance, il faut spécifier un nom permettant d'identifier l'instance de manière unique, la référence de l'essai référencé, l'URL de connexion à utiliser pour cet essai, la liste des mesures à prendre au niveau de l'essai. On peut aussi spécifier un facteur de duplication dans le cas où l'on veut plusieurs instances identiques d'un même essai.

```
Instance ::= '#' 'Instance' [' :' FacteurDuplication] Id
ReferenceEssai Connexion Mesure

FacteurDuplication ::= <NUM>

Id ::= 'Id' '=' <ID>
```

L'essai référencé est identifié par son nom et éventuellement sa version. Si aucune version n'est spécifiée, la dernière version existante de l'essai est utilisée.

```
ReferenceEssai ::= Nom [Version]

Connexion ::= 'URL' '=' <STRING>
```

E.2.3 Lancement du programme

Le programme peut être lancé avec ou sans paramètres. Si aucun paramètre n'est renseigné en argument, le renseignement des informations nécessaire au déroulement du programme (informations de connexion aux bases de données et chemin du fichier contenant la description des tests et des essais) se fera via une interface console. BDResultat doit avoir été initialisé au préalable par le programme AtelierEssai. De plus, les informations utilisées pour

se connecter au SGBD cible doivent être celle d'un utilisateur qui a les droits nécessaires pour exécuter les tests.

Pour lancer le programme sans arguments, il faut exécuter la ligne de commande suivante :

```
java -Djava.library.path=sigarlib -jar
ExecuteurSeance.jar
```

L'argument –Djava.library.path permet de renseigner le chemin du dossier contenant les librairies natives pour la prise de mesure. Ces librairies sont présentes dans le dossier sigarlib fourni avec l'archive jar du programme.

Le programme demande alors les informations nécessaires à son exécution. Pour arrêter en cours d'exécution, il suffit d'actionner les touches envoyant le signal SIGQUIT (Ctrl+D sous Linux ou Ctrl+Z sous Windows) à la place de rentrer les informations demandées.

Au lancement du programme, il est possible de spécifier les paramètres suivants :

–urlR [urlConnexionBD] L'URL de connexion à la BD où sont stockés les tests et les essais et où seront stockés les résultats de la séance.

–userR [utilisateurBD] L'utilisateur avec lequel on veut se connecter à la BD où seront stockés les tests et les résultats.

–passR [motDePasse] Le mot de passe associé à l'utilisateur.

–userT [utilisateurBD] L'utilisateur avec lequel on veut se connecter à la BD où seront exécutés les tests.

–passT [motDePasse] Le mot de passe associé à l'utilisateur.

–file [cheminFichier] Le chemin du fichier où est décrite la séance.

La ligne de commande à exécuter ressemble alors à cela :

```
java -Djava.library.path=sigarlib -jar
ExecuteurSeance.jar -urlR
jdbc:oracle:thin:@//127.0.0.1:1521/xe -userR MonNom -
passR MonMdp -file Seances/Seance1.txt -userT MonNom -
passT MonMdp
```

E.2.4 Résultats

Les fichiers rapports de résultat et les fichiers de log générés par l'application se trouvent respectivement dans les dossiers rapport et log à la racine du dossier dans lequel se trouve l'exécutable.

Annexe F

Résultats d'efficience pour le SGBD Oracle

Tableau 20 – Résultats d'efficience pour le SGBD Oracle

CFG : Configuration σ : écart-type $\bar{X}$: Moyenne MIN : Valeur minimum
MAX : Valeur maximum STAB : Stabilité

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[10] (%)
P1.W3	ReadCommitted	CPU	1339,67	11771,05	9875	14734	9,64
P1.W3	ReadCommitted	NBDEFAUT_MI	623,99	951,6	461	2922	49,54
P1.W3	ReadCommitted	NBECRITURE_ME	205,96	1257,8	949	1766	13,05
P1.W3	ReadCommitted	NBLECTURE_ME	56,72	53,05	0	183	95,06
P1.W3	ReadCommitted	RESIDENTE_MI	1067850,14	-3402137,6	-4730880	565248	13,28
P1.W3	ReadCommitted	TPS	1432	12608,6	10500	15781	9,54
P1.W5	ReadCommitted	CPU	1053,77	11812,65	9891	13782	7,55
P1.W5	ReadCommitted	NBDEFAUT_MI	561,96	797,85	462	2372	60,65
P1.W5	ReadCommitted	NBECRITURE_ME	250;65	2088,45	1702	2527	10,73
P1.W5	ReadCommitted	NBLECTURE_ME	761,02	971,5	257	3878	38,07
P1.W5	ReadCommitted	RESIDENTE_MI	688104,23	-3725516,8	-5160960	-2863104	16,32
P1.W5	ReadCommitted	TPS	1278,9	12946,15	10672	15391	8,37
P2.W3	ReadCommitted	CPU	250,37	11712,4	11172	12125	1,77
P2.W3	ReadCommitted	NBDEFAUT_MI	520,44	883,45	501	2641	40,39
P2.W3	ReadCommitted	NBECRITURE_ME	148,33	1158,95	1034	1580	9,96
P2.W3	ReadCommitted	NBLECTURE_ME	53,64	41,35	0	164	109,28
P2.W3	ReadCommitted	RESIDENTE_MI	1486228,44	-3049267,2	-4018176	2514944	21,89
P2.W3	ReadCommitted	TPS	430,67	13278,05	12704	14500	2,37
P2.W5	ReadCommitted	CPU	430,68	12118,6	11235	12750	3,06
P2.W5	ReadCommitted	NBDEFAUT_MI	1464,79	1106,9	510	7281	22,83
P2.W5	ReadCommitted	NBECRITURE_ME	53,38	1692,8	1595	1806	2,55
P2.W5	ReadCommitted	NBLECTURE_ME	293,78	2007,45	1575	2993	8,36

[10] $|\sigma/\bar{X}|*100$

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[10] (%)
P2.W5	ReadCommitted	RESIDENTE_MI	839165,37	-3977830,4	-6242304	-2195456	13,81
P2.W5	ReadCommitted	TPS	1532,47	24928,1	22109	27484	5,34
P3.W3	ReadCommitted	CPU	592,05	22268,3	21200	23354	2,27
P3.W3	ReadCommitted	NBDEFAUT_MI	290,24	674,5	485	1485	35,68
P3.W3	ReadCommitted	NBECRITURE_ME	172,89	1705,3	1469	2047	8,97
P3.W3	ReadCommitted	NBLECTURE_ME	61,14	77,15	19	185	77,07
P3.W3	ReadCommitted	RESIDENTE_MI	1190188,16	2723430,4	1970176	6082560	36,04
P3.W3	ReadCommitted	TPS	1275,93	19095,25	14867	20487	4,11
P3.W5	ReadCommitted	CPU	1248,85	26987,35	25053	30357	3,42
P3.W5	ReadCommitted	NBDEFAUT_MI	5186,03	1788,95	484	23787	47,94
P3.W5	ReadCommitted	NBECRITURE_ME	79,59	1693,05	1544	1810	4,16
P3.W5	ReadCommitted	NBLECTURE_ME	366,86	1565,25	48	1893	3,91
P3.W5	ReadCommitted	RESIDENTE_MI	1228321,52	2354585,6	987136	7344128	12,20
P3.W5	ReadCommitted	TPS	6184,58	43328	22557	51044	8,28
P1.W3	Serializable	CPU	2598,96	9962,45	4875	14655	21,78
P1.W3	Serializable	NBDEFAUT_MI	418,71	921,7	447	2236	32,07
P1.W3	Serializable	NBECRITURE_ME	206,93	1165,45	935	1753	12,97
P1.W3	Serializable	NBLECTURE_ME	40,91	44,4	0	115	62,18
P1.W3	Serializable	RESIDENTE_MI	868988,61	-3556761,6	-4829184	-458752	11,38
P1.W3	Serializable	TPS	2720,78	10685,2	5516	15687	21,38
P1.W5	Serializable	CPU	1031,85	6316,45	5078	8078	15,22
P1.W5	Serializable	NBDEFAUT_MI	880,34	1134,45	509	3870	61,27
P1.W5	Serializable	NBECRITURE_ME	346,66	1721,15	1065	2343	17,04
P1.W5	Serializable	NBLECTURE_ME	318,31	675,35	118	1296	40,02
P1.W5	Serializable	RESIDENTE_MI	964183,82	-3283763,2	-4202496	524288	9,59
P1.W5	Serializable	TPS	1712,06	7567,25	5594	11312	19,91
P2.W3	Serializable	CPU	2202,08	10257,85	5313	12124	18,56
P2.W3	Serializable	NBDEFAUT_MI	412,87	812,3	499	2385	24,86
P2.W3	Serializable	NBECRITURE_ME	111,21	1148,9	1015	1498	6,49
P2.W3	Serializable	NBLECTURE_ME	54,65	40,85	0	157	101,65
P2.W3	Serializable	RESIDENTE_MI	897570,07	-3593830,4	-5021696	-851968	15,73
P2.W3	Serializable	TPS	2189,97	11673,25	6735	13625	16,19
P2.W5	Serializable	CPU	452,73	5507	4844	6922	5,29
P2.W5	Serializable	NBDEFAUT_MI	440,63	861,9	524	2104	42,12
P2.W5	Serializable	NBECRITURE_ME	42,68	1139,55	1080	1263	2,43
P2.W5	Serializable	NBLECTURE_ME	166,92	2157,85	1828	2359	6,90
P2.W5	Serializable	RESIDENTE_MI	516209,94	-3781427,2	-4894720	-2772992	10,76

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[10] (%)
P2.W5	Serializable	TPS	1666,43	23413,2	20203	26766	5,79
P3.W3	Serializable	CPU	3405,23	13253,2	2137	16022	16,32
P3.W3	Serializable	NBDEFAUT_MI	216,46	625,1	416	1253	26,81
P3.W3	Serializable	NBECRITURE_ME	229,14	1041,95	128	1279	6,38
P3.W3	Serializable	NBLECTURE_ME	86,77	53,05	11	378	114,87
P3.W3	Serializable	RESIDENTE_MI	883168,95	2552422,4	1703936	5132288	26,64
P3.W3	Serializable	TPS	3717,94	15148,5	1623	19500	11,60
P3.W5	Serializable	CPU	2855,12	13759,5	2621	16537	7,47
P3.W5	Serializable	NBDEFAUT_MI	5161,73	1702,7	455	23625	25,36
P3.W5	Serializable	NBECRITURE_ME	214,86	970	111	1156	6,79
P3.W5	Serializable	NBLECTURE_ME	794,89	664,75	51	1814	122,53
P3.W5	Serializable	RESIDENTE_MI	286656,17	2183782,4	1851392	3022848	9,63
P3.W5	Serializable	TPS	11284,9	23671,25	2200	39016	41,90
P1.W3	Sequentielle	CPU	1667,76	8626,3	5953	12235	16,16
P1.W3	Sequentielle	NBDEFAUT_MI	2253,47	1450,45	455	10158	96,01
P1.W3	Sequentielle	NBECRITURE_ME	149,39	1398	1092	1658	8,88
P1.W3	Sequentielle	NBLECTURE_ME	288,1	243,2	0	1071	103,82
P1.W3	Sequentielle	RESIDENTE_MI	1066542,17	-593100,8	-2490368	3178496	65,94
P1.W3	Sequentielle	TPS	1827,61	9649,25	6672	13437	16,01
P1.W5	Sequentielle	CPU	1698,71	11745,15	8047	13609	12,53
P1.W5	Sequentielle	NBDEFAUT_MI	577,03	740,15	349	2337	67,81
P1.W5	Sequentielle	NBECRITURE_ME	411,16	2249,5	1556	3025	15,76
P1.W5	Sequentielle	NBLECTURE_ME	427,14	933,2	200	2119	32,84
P1.W5	Sequentielle	RESIDENTE_MI	488023,45	-634060,8	-2150400	499712	38,21
P1.W5	Sequentielle	TPS	2784,9	14274,2	9016	19359	16,45
P2.W3	Sequentielle	CPU	438,13	7479,05	6640	8173	5,08
P2.W3	Sequentielle	NBDEFAUT_MI	640,34	900,55	236	2828	55,00
P2.W3	Sequentielle	NBECRITURE_ME	106,07	1511,95	1275	1708	5,53
P2.W3	Sequentielle	NBLECTURE_ME	66,33	51,5	0	267	92,14
P2.W3	Sequentielle	RESIDENTE_MI	1351365,01	-285696	-2031616	4730880	125,92
P2.W3	Sequentielle	TPS	629,03	9099	8016	10406	5,75
P2.W5	Sequentielle	CPU	391,09	8696,15	8015	9767	3,17
P2.W5	Sequentielle	NBDEFAUT_MI	1432,43	1521,9	317	4687	89,69
P2.W5	Sequentielle	NBECRITURE_ME	109,79	1593,5	1390	1780	5,95
P2.W5	Sequentielle	NBLECTURE_ME	301,42	2997,25	2469	3459	8,98
P2.W5	Sequentielle	RESIDENTE_MI	526079,3	-915046,4	-2109440	-73728	47,82
P2.W5	Sequentielle	TPS	3497,36	29538,15	24719	35593	10,81

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[10] (%)
P3.W3	Sequentielle	CPU	722,36	14878,55	13805	16536	4,00
P3.W3	Sequentielle	NBDEFAUT_MI	306,52	669,15	347	1409	39,81
P3.W3	Sequentielle	NBECRITURE_ME	89,76	1347,95	1173	1537	5,30
P3.W3	Sequentielle	NBLECTURE_ME	12,74	27,7	6	47	41,25
P3.W3	Sequentielle	RESIDENTE_MI	1218860,41	1679360	950272	4939776	64,43
P3.W3	Sequentielle	TPS	1264,32	19266,2	15350	21310	4,09
P3.W5	Sequentielle	CPU	4671,56	22846,9	20575	42416	3,49
P3.W5	Sequentielle	NBDEFAUT_MI	16438,43	4558,3	479	74373	49,52
P3.W5	Sequentielle	NBECRITURE_ME	71,87	1488,5	1363	1673	3,58
P3.W5	Sequentielle	NBLECTURE_ME	1366,86	8438,45	6529	13740	5,18
P3.W5	Sequentielle	RESIDENTE_MI	1493036,42	1656627,2	450560	5890048	77,24
P3.W5	Sequentielle	TPS	1742,98	61032,05	57564	64413	2,33
P1.W3	Initialisation	CPU	207772,42	839979,1	578313	1078624	24,03
P1.W3	Initialisation	NBDEFAUT_MI	25970,11	54598	27569	142063	30,79
P1.W3	Initialisation	NBECRITURE_ME	10908,09	36747,6	21507	57305	26,65
P1.W3	Initialisation	NBLECTURE_ME	8303,5	3416,4	639	38468	59,42
P1.W3	Initialisation	RESIDENTE_MI	113443218	27617075,2	-2514944	506490880	502,78
P1.W3	Initialisation	TPS	213461,22	1061350	775625	1351203	19,12
P1.W5	Initialisation	CPU	298485,2	1362146,75	983937	1723469	21,20
P1.W5	Initialisation	NBDEFAUT_MI	17140,46	72827,5	47978	109376	20,22
P1.W5	Initialisation	NBECRITURE_ME	17507,65	60516,05	38614	86024	27,55
P1.W5	Initialisation	NBLECTURE_ME	1605,6	1985,35	1151	8541	27,17
P1.W5	Initialisation	RESIDENTE_MI	52280602,52	12972032	-1646592	233734144	397,89
P1.W5	Initialisation	TPS	297679,28	1722467,9	1307328	2091531	16,49
P2.W3	Initialisation	CPU	46375,18	1000374	921609	1076405	4,12
P2.W3	Initialisation	NBDEFAUT_MI	32836,79	61395,45	12626	170412	32,42
P2.W3	Initialisation	NBECRITURE_ME	5207,37	26238,85	19432	40936	14,85
P2.W3	Initialisation	NBLECTURE_ME	2237,82	1747,8	732	10772	56,21
P2.W3	Initialisation	RESIDENTE_MI	141738249,7	28175769,6	-53411840	628142080	492,54
P2.W3	Initialisation	TPS	51636,06	1286077,05	1208672	1354592	3,77
P2.W5	Initialisation	CPU	72855,19	1611572,6	1498516	1793750	3,53
P2.W5	Initialisation	NBDEFAUT_MI	29927,16	92380,35	27130	137325	26,85
P2.W5	Initialisation	NBECRITURE_ME	4722,86	38596,85	32795	54883	6,95
P2.W5	Initialisation	NBLECTURE_ME	1106,06	2456,7	1233	4741	41,05
P2.W5	Initialisation	RESIDENTE_MI	23938924	5465088	-2220032	104951808	1430,58
P2.W5	Initialisation	TPS	77397,44	2064566,65	1945984	2247603	3,02
P3.W3	Initialisation	CPU	51190,82	1709437,9	1602458	1813464	2,35

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[10] (%)
P3.W3	Initialisation	NBDEFAUT_MI	61934,18	45167,8	6743	192799	137,23
P3.W3	Initialisation	NBECRITURE_ME	714,41	15042,9	14022	16817	3,78
P3.W3	Initialisation	NBLECTURE_ME	2962,45	2229	486	13483	80,71
P3.W3	Initialisation	RESIDENTE_MI	244243171,6	118669107,2	335872	688939008	229,36
P3.W3	Initialisation	TPS	62517,17	1999754,4	1869683	2122690	2,49
P3.W5	Initialisation	CPU	54166,81	2793548,65	2705759	2912398	1,60
P3.W5	Initialisation	NBDEFAUT_MI	20723,59	32589,9	11762	115947	19,72
P3.W5	Initialisation	NBECRITURE_ME	1046,82	25803,45	24703	28637	3,19
P3.W5	Initialisation	NBLECTURE_ME	4409,87	4010,45	797	16971	95,33
P3.W5	Initialisation	RESIDENTE_MI	80719838,06	27591475,2	389120	363532288	162,74
P3.W5	Initialisation	TPS	64990,55	3272174,15	3165996	3408163	1,67
P1.W3	Supression	CPU	159,02	1162,55	813	1515	10,09
P1.W3	Supression	NBDEFAUT_MI	1942,75	5434,85	2767	9849	30,46
P1.W3	Supression	NBECRITURE_ME	473,84	19107,75	18130	19739	2,16
P1.W3	Supression	NBLECTURE_ME	23,56	14,95	1	101	114,26
P1.W3	Supression	RESIDENTE_MI	583668,99	-654131,2	-2433024	827392	40,05
P1.W3	Supression	TPS	1301,42	4692,05	1782	6156	23,54
P1.W5	Supression	CPU	296,09	1275	875	1969	19,45
P1.W5	Supression	NBDEFAUT_MI	2689,68	5510,55	2162	12603	39,65
P1.W5	Supression	NBECRITURE_ME	1319,76	18484,35	16762	22981	4,07
P1.W5	Supression	NBLECTURE_ME	14835,92	3340,45	0	66371	199,55
P1.W5	Supression	RESIDENTE_MI	8911508,36	1088716,8	-2588672	38834176	74,89
P1.W5	Supression	TPS	10224,2	6703,85	2734	50016	15,61
P2.W3	Supression	CPU	7693,05	7276,6	1485	24594	102,31
P2.W3	Supression	NBDEFAUT_MI	3463,81	7619,35	3025	14280	41,17
P2.W3	Supression	NBECRITURE_ME	263	21291,95	20524	21652	0,87
P2.W3	Supression	NBLECTURE_ME	53,92	39,2	1	220	109,56
P2.W3	Supression	RESIDENTE_MI	710224,74	-452198,4	-2031616	1236992	108,91
P2.W3	Supression	TPS	7876,88	25394,3	16421	42094	27,75
P2.W5	Supression	CPU	8446,55	8491,35	1406	26375	96,17
P2.W5	Supression	NBDEFAUT_MI	3938	8878,45	2945	16361	39,33
P2.W5	Supression	NBECRITURE_ME	2647,38	19340,5	8166	20383	1,49
P2.W5	Supression	NBLECTURE_ME	25,12	20,15	2	80	117,06
P2.W5	Supression	RESIDENTE_MI	665300,28	-714752	-2035712	806912	69,66
P2.W5	Supression	TPS	9240,36	27978,75	19328	48328	29,78
P3.W3	Supression	CPU	178,72	2561,65	2106	2808	5,46
P3.W3	Supression	NBDEFAUT_MI	503,09	2636,6	2013	3862	15,86

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[10] (%)
P3.W3	Supression	NBECRITURE_ME	538,25	19678,05	18959	20955	2,26
P3.W3	Supression	NBLECTURE_ME	10,75	17,25	6	52	37,06
P3.W3	Supression	RESIDENTE_MI	851740,06	1239859,2	688128	3907584	53,43
P3.W3	Supression	TPS	754,6	14051,85	12776	15990	4,03
P3.W5	Supression	CPU	175,04	2621	2294	2917	5,76
P3.W5	Supression	NBDEFAUT_MI	516,16	2501,05	1910	3708	17,68
P3.W5	Supression	NBECRITURE_ME	1523,66	20698,6	19133	25105	5,51
P3.W5	Supression	NBLECTURE_ME	10,65	18,55	13	63	11,09
P3.W5	Supression	RESIDENTE_MI	303790,45	844185,6	-90112	1622016	14,89
P3.W5	Supression	TPS	687,27	14758,4	13588	15710	4,25

Annexe G

Résultats d'efficience pour le SGBD PostgreSQL

Tableau 21 – Résultats d'efficience pour le SGBD PostgreSQL

CFG : Configuration σ : écart-type $\bar{X}$: Moyenne MIN : Valeur minimum
MAX : Valeur maximum STAB : Stabilité

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[11] (%)
P1.W3	ReadCommitted	CPU	1465,89	835,56	-90	3985	175,44
P1.W3	ReadCommitted	NBDEFAUT_MI	10473,54	-8380,89	-16208	24695	124,97
P1.W3	ReadCommitted	NBECRITURE_ME	1927,67	10215,33	7921	13559	18,87
P1.W3	ReadCommitted	NBLECTURE_ME	0	0	0	0	0,00
P1.W3	ReadCommitted	RESIDENTE_MI	17193453,42	-46597916,44	-68567040	-11182080	36,90
P1.W3	ReadCommitted	TPS	2123,91	21936,61	18453	24719	9,68
P1.W5	ReadCommitted	CPU	808,67	319,44	-137	2707	253,15
P1.W5	ReadCommitted	NBDEFAUT_MI	16079,19	-8531,61	-15364	37006	188,47
P1.W5	ReadCommitted	NBECRITURE_ME	1736,68	6942,22	4125	9852	25,02
P1.W5	ReadCommitted	NBLECTURE_ME	0	0	0	0	0,00
P1.W5	ReadCommitted	RESIDENTE_MI	14318488,82	-51356558,22	-61448192	-12722176	27,88
P1.W5	ReadCommitted	TPS	2148,52	22152,83	19218	26391	9,70
P2.W3	ReadCommitted	CPU	4860,66	1471,94	-4043	10204	330,22
P2.W3	ReadCommitted	NBDEFAUT_MI	25917,61	-10272,67	-41443	46465	252,30
P2.W3	ReadCommitted	NBECRITURE_ME	1764,85	11133,44	7871	13461	15,85
P2.W3	ReadCommitted	NBLECTURE_ME	0,5	0,39	0	1	128,21
P2.W3	ReadCommitted	RESIDENTE_MI	30648568,53	-58532067,56	-110792704	-3461120	52,36
P2.W3	ReadCommitted	TPS	5731,28	41341,11	32094	52657	13,86
P2.W5	ReadCommitted	CPU	2244,04	817,33	-75	9470	274,56
P2.W5	ReadCommitted	NBDEFAUT_MI	21700,33	-1432,06	-14741	47991	1515,32
P2.W5	ReadCommitted	NBECRITURE_ME	1396,92	7342	4933	9738	19,03

[11] $|\sigma/\bar{X}|*100$

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[II] (%)
P2.W5	ReadCommitted	NBLECTURE_ME	0,46	0,28	0	1	164,29
P2.W5	ReadCommitted	RESIDENTE_MI	19032533,28	-41810375,11	-59420672	-5738496	45,52
P2.W5	ReadCommitted	TPS	2576,77	29683,17	24391	34141	8,68
P3.W3	ReadCommitted	CPU	657,35	34067,56	33133	35302	1,93
P3.W3	ReadCommitted	NBDEFAUT_MI	5320,14	65131,17	54707	71113	8,17
P3.W3	ReadCommitted	NBECRITURE_ME	506,12	8116,33	7455	9100	6,24
P3.W3	ReadCommitted	NBLECTURE_ME	0,38	0,17	0	1	223,53
P3.W3	ReadCommitted	RESIDENTE_MI	3212354,16	164278727,1	158314496	170627072	1,96
P3.W3	ReadCommitted	TPS	1343,1	31523,39	29203	33665	4,26
P3.W5	ReadCommitted	CPU	3476,77	38794,28	36254	46832	8,96
P3.W5	ReadCommitted	NBDEFAUT_MI	5136,26	64924,61	53938	70926	7,91
P3.W5	ReadCommitted	NBECRITURE_ME	453,84	8125,39	7340	8820	5,59
P3.W5	ReadCommitted	NBLECTURE_ME	0,24	0,06	0	1	400,00
P3.W5	ReadCommitted	RESIDENTE_MI	2606739,82	164365880,9	159948800	168349696	1,59
P3.W5	ReadCommitted	TPS	2224,62	31655,94	28485	36254	7,03
P1.W3	ReadUncommitted	CPU	1218,75	552,33	-122	3518	220,66
P1.W3	ReadUncommitted	NBDEFAUT_MI	28988,5	-5044,39	-29994	75897	574,67
P1.W3	ReadUncommitted	NBECRITURE_ME	1197,58	10679,33	9048	12951	11,21
P1.W3	ReadUncommitted	NBLECTURE_ME	0,24	0,06	0	1	400,00
P1.W3	ReadUncommitted	RESIDENTE_MI	27786649,68	-49252352	-112271360	262144	56,42
P1.W3	ReadUncommitted	TPS	2628,56	21937,5	18344	26953	11,98
P1.W5	ReadUncommitted	CPU	3342,15	1733,94	-44	10050	192,75
P1.W5	ReadUncommitted	NBDEFAUT_MI	18471,9	-5717,94	-15224	43113	323,05
P1.W5	ReadUncommitted	NBECRITURE_ME	1629,01	7801,67	4606	10452	20,88
P1.W5	ReadUncommitted	NBLECTURE_ME	0,24	0,06	0	1	400,00
P1.W5	ReadUncommitted	RESIDENTE_MI	18661668,35	-46556956,44	-60858368	-3756032	40,08
P1.W5	ReadUncommitted	TPS	2637,32	23111,17	19437	29047	11,41
P2.W3	ReadUncommitted	CPU	2908,54	1247,06	-59	9658	233,23
P2.W3	ReadUncommitted	NBDEFAUT_MI	27255,07	3168,72	-15978	70068	860,13
P2.W3	ReadUncommitted	NBECRITURE_ME	1233,94	11813,5	9777	13707	10,45
P2.W3	ReadUncommitted	NBLECTURE_ME	0,38	0,17	0	1	223,53
P2.W3	ReadUncommitted	RESIDENTE_MI	21111023,08	-40383146,67	-64966656	-1474560	52,28
P2.W3	ReadUncommitted	TPS	4016,69	43408,11	38015	53329	9,25
P2.W5	ReadUncommitted	CPU	3943,56	1716,56	-76	10549	229,74
P2.W5	ReadUncommitted	NBDEFAUT_MI	15794,37	-7737,89	-14502	41951	204,12
P2.W5	ReadUncommitted	NBECRITURE_ME	1197,3	9102,22	7000	11088	13,15
P2.W5	ReadUncommitted	NBLECTURE_ME	0,32	0,11	0	1	290,91

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[II] (%)
P2.W5	ReadUncommitted	RESIDENTE_MI	16142292,6	-46570609,78	-58540032	-6135808	34,66
P2.W5	ReadUncommitted	TPS	4126,34	34072,11	28250	41156	12,11
P3.W3	ReadUncommitted	CPU	659,74	34926,67	33698	36085	1,89
P3.W3	ReadUncommitted	NBDEFAUT_MI	6375,81	67099,94	53800	77310	9,50
P3.W3	ReadUncommitted	NBECRITURE_ME	544,56	9190,83	8473	10570	5,93
P3.W3	ReadUncommitted	NBLECTURE_ME	0,32	0,11	0	1	290,91
P3.W3	ReadUncommitted	RESIDENTE_MI	14769455,91	170808888,9	158920704	212680704	8,65
P3.W3	ReadUncommitted	TPS	2505,42	34054,83	30389	40248	7,36
P3.W5	ReadUncommitted	CPU	1495,36	38434,44	37189	43619	3,89
P3.W5	ReadUncommitted	NBDEFAUT_MI	2050,71	68172,17	65332	71882	3,01
P3.W5	ReadUncommitted	NBECRITURE_ME	327,91	9092,56	8311	9542	3,61
P3.W5	ReadUncommitted	NBLECTURE_ME	0	0	0	0	0,00
P3.W5	ReadUncommitted	RESIDENTE_MI	3296750,1	163923740,4	158744576	169885696	2,01
P3.W5	ReadUncommitted	TPS	1351,01	34037,61	31357	36598	3,97
P1.W3	RepeatableRead	CPU	1816,06	1405,83	-75	4659	129,18
P1.W3	RepeatableRead	NBDEFAUT_MI	40434,1	14160,5	-14653	93003	285,54
P1.W3	RepeatableRead	NBECRITURE_ME	1307,49	5967,78	2785	7829	21,91
P1.W3	RepeatableRead	NBLECTURE_ME	0	0	0	0	0,00
P1.W3	RepeatableRead	RESIDENTE_MI	21752273,47	-4901105,78	-58761216	311296	62,33
P1.W3	RepeatableRead	TPS	2352,1	14684,17	10781	19079	16,02
P1.W5	RepeatableRead	CPU	71,23	29,22	-90	160	243,77
P1.W5	RepeatableRead	NBDEFAUT_MI	1460,83	-11866,5	-14343	-9135	12,31
P1.W5	RepeatableRead	NBECRITURE_ME	1621,52	4984,11	2184	6946	32,53
P1.W5	RepeatableRead	NBLECTURE_ME	0,38	0,17	0	1	223,53
P1.W5	RepeatableRead	RESIDENTE_MI	5905418,34	-7337927,11	-57249792	-36167680	12,48
P1.W5	RepeatableRead	TPS	2329,09	16672	12906	19890	13,97
P2.W3	RepeatableRead	CPU	3224,39	1850,33	-75	8738	174,26
P2.W3	RepeatableRead	NBDEFAUT_MI	35384,15	15567,94	-13073	107854	227,29
P2.W3	RepeatableRead	NBECRITURE_ME	1932,37	6980,39	2942	10888	27,68
P2.W3	RepeatableRead	NBLECTURE_ME	0,32	0,11	0	1	290,91
P2.W3	RepeatableRead	RESIDENTE_MI	22357284,27	-2089656,89	-54730752	3313664	69,67
P2.W3	RepeatableRead	TPS	5189,7	25979,06	16563	34907	19,98
P2.W5	RepeatableRead	CPU	3329,75	1750,61	-76	8939	190,21
P2.W5	RepeatableRead	NBDEFAUT_MI	31670,5	3945,78	-14103	91870	802,64
P2.W5	RepeatableRead	NBECRITURE_ME	1381,02	5005,06	2171	6951	27,59
P2.W5	RepeatableRead	NBLECTURE_ME	0	0	0	0	0,00
P2.W5	RepeatableRead	RESIDENTE_MI	19447810,43	-8263694,22	-56926208	-1503232	50,83

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[II] (%)
P2.W5	RepeatableRead	TPS	3171,68	19617,17	12969	24594	16,17
P3.W3	RepeatableRead	CPU	4568,58	25744,61	12528	28079	17,75
P3.W3	RepeatableRead	NBDEFAUT_MI	9438,28	57639,72	36509	66604	16,37
P3.W3	RepeatableRead	NBECRITURE_ME	862,3	2908,06	2184	5135	29,65
P3.W3	RepeatableRead	NBLECTURE_ME	0	0	0	0	0,00
P3.W3	RepeatableRead	RESIDENTE_MI	6523358,65	144859591,1	136171520	159019008	4,50
P3.W3	RepeatableRead	TPS	1596,29	20996,78	16645	22620	7,60
P3.W5	RepeatableRead	CPU	5345,62	27659,28	16099	37988	19,33
P3.W5	RepeatableRead	NBDEFAUT_MI	8334,93	54458,67	36830	64111	15,31
P3.W5	RepeatableRead	NBECRITURE_ME	632,42	2761,11	2044	3881	22,90
P3.W5	RepeatableRead	NBLECTURE_ME	0	0	0	0	0,00
P3.W5	RepeatableRead	RESIDENTE_MI	8017261,32	146472960	135643136	163893248	5,47
P3.W5	RepeatableRead	TPS	3139,98	19329,28	12309	25163	16,24
P1.W3	Serializable	CPU	1836,99	1574,67	-119	4518	116,66
P1.W3	Serializable	NBDEFAUT_MI	42248,27	16321,22	-13859	114118	258,85
P1.W3	Serializable	NBECRITURE_ME	2427,97	5092,06	1185	9622	47,68
P1.W3	Serializable	NBLECTURE_ME	0,43	0,22	0	1	195,45
P1.W3	Serializable	RESIDENTE_MI	20224017,82	-0652188,44	-55287808	-3072000	65,98
P1.W3	Serializable	TPS	2887,22	11881,06	6109	16578	24,30
P1.W5	Serializable	CPU	2444,33	789,11	-199	9329	309,76
P1.W5	Serializable	NBDEFAUT_MI	14495,61	-8603,39	-14428	48156	168,49
P1.W5	Serializable	NBECRITURE_ME	1568,04	3229,5	1026	6216	48,55
P1.W5	Serializable	NBLECTURE_ME	0	0	0	0	0,00
P1.W5	Serializable	RESIDENTE_MI	15568481,45	-5470833,78	-57540608	-1126400	34,24
P1.W5	Serializable	TPS	3614,55	14266,39	6687	19953	25,34
P2.W3	Serializable	CPU	580,82	199,39	-139	1861	291,30
P2.W3	Serializable	NBDEFAUT_MI	23177,81	-3978	-14705	77177	582,65
P2.W3	Serializable	NBECRITURE_ME	2455	4125,17	760	7786	59,51
P2.W3	Serializable	NBLECTURE_ME	0,32	0,11	0	1	290,91
P2.W3	Serializable	RESIDENTE_MI	16547968,54	-2578602,67	-59166720	-4239360	38,86
P2.W3	Serializable	TPS	7430,4	16364,78	5610	27063	45,40
P2.W5	Serializable	CPU	2394,17	1615,28	-310	5908	148,22
P2.W5	Serializable	NBDEFAUT_MI	33006,78	3654	-31030	98381	903,31
P2.W5	Serializable	NBECRITURE_ME	1742,89	3194,78	436	6075	54,55
P2.W5	Serializable	NBLECTURE_ME	0,7	0,44	0	2	159,09
P2.W5	Serializable	RESIDENTE_MI	22145176,06	-42309632	-87179264	-4927488	52,34
P2.W5	Serializable	TPS	5763,69	14441,94	7172	24250	39,91

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[II] (%)
P3.W3	Serializable	CPU	1836,21	9244,06	7690	12494	19,86
P3.W3	Serializable	NBDEFAUT_MI	2261,49	32964,22	28589	37027	6,86
P3.W3	Serializable	NBECRITURE_ME	1120,44	1479,11	483	4266	75,75
P3.W3	Serializable	NBLECTURE_ME	0,51	0,17	0	2	300,00
P3.W3	Serializable	RESIDENTE_MI	8166613,28	134225464,9	117002240	145444864	6,08
P3.W3	Serializable	TPS	2319,57	8456,22	6880	14352	27,43
P3.W5	Serializable	CPU	2458,21	12311,89	10964	20440	19,97
P3.W5	Serializable	NBDEFAUT_MI	1918,91	32902,89	29031	36284	5,83
P3.W5	Serializable	NBECRITURE_ME	852,8	1332,5	564	3390	64,00
P3.W5	Serializable	NBLECTURE_ME	0	0	0	0	0,00
P3.W5	Serializable	RESIDENTE_MI	6715921,64	133130467,6	118665216	145879040	5,04
P3.W5	Serializable	TPS	971	10778,78	10187	14493	9,01
P1.W3	Sequentielle	CPU	10960,15	9750,72	329	25563	112,40
P1.W3	Sequentielle	NBDEFAUT_MI	96387,71	58109,22	3407	247729	165,87
P1.W3	Sequentielle	NBECRITURE_ME	1614,54	17835,33	15858	21406	9,05
P1.W3	Sequentielle	NBLECTURE_ME	0,24	0,06	0	1	400,00
P1.W3	Sequentielle	RESIDENTE_MI	23784622,49	32591644,44	6656000	70516736	72,98
P1.W3	Sequentielle	TPS	2124,54	28820,33	26172	33141	7,37
P1.W5	Sequentielle	CPU	21267,4	2196,39	-70735	27188	968,29
P1.W5	Sequentielle	NBDEFAUT_MI	97703,39	47456,67	-5009	295533	205,88
P1.W5	Sequentielle	NBECRITURE_ME	1360,74	14172,83	12388	17131	9,60
P1.W5	Sequentielle	NBLECTURE_ME	0,51	0,5	0	1	102,00
P1.W5	Sequentielle	RESIDENTE_MI	29351163,47	31559680	-18333696	95203328	93,00
P1.W5	Sequentielle	TPS	2306,21	32239,72	27984	36688	7,15
P2.W3	Sequentielle	CPU	9124,96	13919,39	3890	22967	65,56
P2.W3	Sequentielle	NBDEFAUT_MI	113827,95	143017,39	18496	292607	79,59
P2.W3	Sequentielle	NBECRITURE_ME	1342,41	17171,22	15186	20697	7,82
P2.W3	Sequentielle	NBLECTURE_ME	0,49	0,33	0	1	148,48
P2.W3	Sequentielle	RESIDENTE_MI	21764401,89	47358407,11	18464768	80932864	45,96
P2.W3	Sequentielle	TPS	4037,3	56364,67	48688	62891	7,16
P2.W5	Sequentielle	CPU	783,19	560,94	204	3657	139,62
P2.W5	Sequentielle	NBDEFAUT_MI	3476,57	11235,67	6228	19719	30,94
P2.W5	Sequentielle	NBECRITURE_ME	1026,32	13652,39	12051	15862	7,52
P2.W5	Sequentielle	NBLECTURE_ME	0,32	0,11	0	1	290,91
P2.W5	Sequentielle	RESIDENTE_MI	9618776,18	37874346,67	21573632	55033856	25,40
P2.W5	Sequentielle	TPS	3109,88	47468,67	43000	52110	6,55
P3.W3	Sequentielle	CPU	2104,67	44051,17	40715	47581	4,78

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[II] (%)
P3.W3	Sequentielle	NBDEFAUT_MI	2813,65	5724,72	2209	14329	49,15
P3.W3	Sequentielle	NBECRITURE_ME	1268,9	9846,83	8072	12003	12,89
P3.W3	Sequentielle	NBLECTURE_ME	2,12	0,56	0	9	378,57
P3.W3	Sequentielle	RESIDENTE_MI	6897992,39	-2418688	-13213696	9695232	285,20
P3.W3	Sequentielle	TPS	4614,76	57213	51231	65754	8,07
P3.W5	Sequentielle	CPU	2464,68	55139,11	50888	59422	4,47
P3.W5	Sequentielle	NBDEFAUT_MI	4179,32	4731,11	-315	15233	88,34
P3.W5	Sequentielle	NBECRITURE_ME	616,81	10052,89	8956	11187	6,14
P3.W5	Sequentielle	NBLECTURE_ME	0,24	0,06	0	1	400,00
P3.W5	Sequentielle	RESIDENTE_MI	8474757,85	3236295,11	-2920448	35999744	261,87
P3.W5	Sequentielle	TPS	2728,1	67174,67	61573	71745	4,06
P1.W3	Initialisation	CPU	34029,05	12674,94	3561	148985	268,48
P1.W3	Initialisation	NBDEFAUT_MI	139146,83	31911,28	-1816	589371	436,04
P1.W3	Initialisation	NBECRITURE_ME	8996,46	244622,61	225937	258078	3,68
P1.W3	Initialisation	NBLECTURE_ME	7,97	2,17	0	34	367,28
P1.W3	Initialisation	RESIDENTE_MI	13690138,6	-6312618,67	-11853824	34463744	216,87
P1.W3	Initialisation	TPS	34233,23	673202,33	636172	738234	5,09
P1.W5	Initialisation	CPU	50677,94	17386,61	4642	220438	291,48
P1.W5	Initialisation	NBDEFAUT_MI	191969,65	44596,22	-1284	813801	430,46
P1.W5	Initialisation	NBECRITURE_ME	9777,95	298400,17	280978	313774	3,28
P1.W5	Initialisation	NBLECTURE_ME	0,98	0,61	0	4	160,66
P1.W5	Initialisation	RESIDENTE_MI	11315910,1	-6638250,67	-12197888	36532224	170,47
P1.W5	Initialisation	TPS	57821,15	1018345,44	958407	1121016	5,68
P2.W3	Initialisation	CPU	45984,24	26063,5	1704	112234	176,43
P2.W3	Initialisation	NBDEFAUT_MI	207601,09	85771,67	-2015	690336	242,04
P2.W3	Initialisation	NBECRITURE_ME	5167,87	246778,67	238103	253400	2,09
P2.W3	Initialisation	NBLECTURE_ME	0,38	0,17	0	1	223,53
P2.W3	Initialisation	RESIDENTE_MI	19686349,68	-1173276,44	-12853248	35561472	1677,90
P2.W3	Initialisation	TPS	10880,03	741279,5	725359	766266	1,47
P2.W5	Initialisation	CPU	50390,73	19990,11	2187	168250	252,08
P2.W5	Initialisation	NBDEFAUT_MI	278968,91	87988,94	-1440	1101994	317,05
P2.W5	Initialisation	NBECRITURE_ME	11364,53	290862,28	270389	307147	3,91
P2.W5	Initialisation	NBLECTURE_ME	13,57	5,39	0	46	251,76
P2.W5	Initialisation	RESIDENTE_MI	13133863,15	-6736327,11	-12668928	34537472	194,97
P2.W5	Initialisation	TPS	21587,26	1114717	1091672	1169812	1,94
P3.W3	Initialisation	CPU	27327,75	590992,89	551074	635079	4,62
P3.W3	Initialisation	NBDEFAUT_MI	1526,42	10078,61	9062	14173	15,15

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[II] (%)
P3.W3	Initialisation	NBECRITURE_ME	6437,68	161423,33	143087	167185	3,99
P3.W3	Initialisation	NBLECTURE_ME	33,36	18,11	0	109	184,21
P3.W3	Initialisation	RESIDENTE_MI	5994584,1	37177344	34250752	54464512	16,12
P3.W3	Initialisation	TPS	41302,25	1069861,83	999837	1151609	3,86
P3.W5	Initialisation	CPU	29204,97	943846,72	892109	985366	3,09
P3.W5	Initialisation	NBDEFAUT_MI	448,94	10070,11	9430	10908	4,46
P3.W5	Initialisation	NBECRITURE_ME	4518,28	200507,28	190143	207334	2,25
P3.W5	Initialisation	NBLECTURE_ME	16,37	10,22	0	45	160,18
P3.W5	Initialisation	RESIDENTE_MI	854568,05	35532800	34304000	36798464	2,41
P3.W5	Initialisation	TPS	42489,69	1600021,11	1532671	1675412	2,66
P1.W3	Suppression	CPU	1683,33	-1768,22	-4108	1	95,20
P1.W3	Suppression	NBDEFAUT_MI	9711,23	-24908,44	-38470	-2961	38,99
P1.W3	Suppression	NBECRITURE_ME	130,95	110,11	28	559	118,93
P1.W3	Suppression	NBLECTURE_ME	0,43	0,22	0	1	195,45
P1.W3	Suppression	RESIDENTE_MI	23340568,72	-1345564,44	-97357824	-11182080	38,05
P1.W3	Suppression	TPS	564,76	2064,28	1172	3188	27,36
P1.W5	Suppression	CPU	1244,59	-762,83	-3780	0	163,15
P1.W5	Suppression	NBDEFAUT_MI	9224,96	-15431,11	-36258	-2560	59,78
P1.W5	Suppression	NBECRITURE_ME	18,13	43,61	22	96	41,57
P1.W5	Suppression	NBLECTURE_ME	0,32	0,11	0	1	290,91
P1.W5	Suppression	RESIDENTE_MI	22305105,58	-8073614,22	-86458368	-10276864	46,40
P1.W5	Suppression	TPS	502,19	1593,67	1187	2219	31,51
P2.W3	Suppression	CPU	1350,45	-1091,83	-4171	47	123,69
P2.W3	Suppression	NBDEFAUT_MI	6498,88	-18448,33	-31713	-3126	35,23
P2.W3	Suppression	NBECRITURE_ME	74,56	65,44	12	345	113,94
P2.W3	Suppression	NBLECTURE_ME	0,43	0,22	0	1	195,45
P2.W3	Suppression	RESIDENTE_MI	16628789,68	-5510357,33	-68431872	2961408	29,96
P2.W3	Suppression	TPS	239,6	1127,5	203	1219	21,25
P2.W5	Suppression	CPU	1307,89	-1269,17	-3904	-342	103,05
P2.W5	Suppression	NBDEFAUT_MI	6633,17	-22668,39	-37914	-14431	29,26
P2.W5	Suppression	NBECRITURE_ME	31,73	44,44	20	143	71,40
P2.W5	Suppression	NBLECTURE_ME	13,78	4,78	0	46	288,28
P2.W5	Suppression	RESIDENTE_MI	17744916,15	-8533134,22	-96026624	-48451584	25,89
P2.W5	Suppression	TPS	508,88	1605,83	1187	2235	31,69
P3.W3	Suppression	CPU	336,69	-85,44	-873	125	394,07
P3.W3	Suppression	NBDEFAUT_MI	7182,79	-4163,17	-16401	926	172,53
P3.W3	Suppression	NBECRITURE_ME	25,75	21,94	4	68	117,37

CFG	SÉANCE	MESURE	σ	$\bar{X}$	MIN	MAX	STAB[II] (%)
P3.W3	Suppression	NBLECTURE_ME	1,25	0,5	0	5	250,00
P3.W3	Suppression	RESIDENTE_MI	27922545,66	-6204458,67	-60989440	3694592	172,31
P3.W3	Suppression	TPS	563,14	749,78	343	1638	75,11
P3.W5	Suppression	CPU	15,96	150,83	125	187	10,58
P3.W5	Suppression	NBDEFAUT_MI	52,6	638,39	564	743	8,24
P3.W5	Suppression	NBECRITURE_ME	1,91	5	3	9	38,20
P3.W5	Suppression	NBLECTURE_ME	0	0	0	0	0,00
P3.W5	Suppression	RESIDENTE_MI	210247,77	2493553,78	2211840	2945024	8,43
P3.W5	Suppression	TPS	33,49	394,33	374	484	8,49

Annexe H

Éléments non traités dans le cadre du mémoire

Tableau 22 – Éléments non traités dans le cadre du mémoire

Niveau Critère /Stratégie	Problématique	Stratégie	Implémentation Oracle et PostgreSQL
Validité des données	Pas de modèle pour évaluer l'adéquation au monde réel		
Fonctionnalites Typage		Uniquement les fonctionnalités liées au relationnel	OK
Fonctionnalites Algorithmiques		Uniquement les fonctionnalités liées au relationnel	OK
Fonctionnalites Relationnelles		OK	OK
Fonctionnalites Combinées		Seulement deux exemples d'illustration	OK
Disponibilité	OK		
Atomicité		OK	OK
Cohérence		OK	Seules les 4 premières propriétés définies par TPC-C
Isolation		OK	OK
Durabilité		OK	Problème de connexion réseau non testé
Temps Récupération		Mis de côté	
Sécurité	OK	OK	OK
Fonctionnalites Securite	OK	OK	Étanchéité mise de côté
Scenario Intrusion	OK	OK	Ne s'applique pas aux SGBD testés

Efficience	OK		
TPCC		OK	OK
Fonctionnalites Langage		Mêmes limitations que pour l'ensemble des stratégies du critère d'adéquation du langage	Implémenté, mais non exécuté
Applications Existantes		OK	Non implémenté
Expressivité	Pas de modèle pour évaluer la pertinence des éléments lexicaux du langage		
Fonctionnalites Exprimables		OK	OK
Metriques Complexite		OK	Implémentation partielle pour Oracle et aucune implémentation pour PostgreSQL.
Niveau Abstraction		OK	Pas d'analyse des éléments du langage de calcul dont le niveau d'abstraction pourrait être élevé.
Complexite Syntaxique		OK	Nom Implémenté

Annexe I

Définitions [23, 41]

Anomalie (« ***anomaly*** ») : écart constaté entre le comportement effectif et le comportement attendu.

Banc d'essai (« ***test bed*** ») : réunion d'un essai (suite de tests) et d'un environnement d'essai (matériel et logiciel).

Caractéristique d'un logiciel (« ***software characteristic*** ») : Un trait, une qualité ou une propriété inhérente, possiblement accidentelle, du logiciel.

Cas de test (« ***test case*** ») : un ensemble de données, de conditions d'exécution et de critères (de passage ou d'échec).

Cohérence (« ***coherence*** ») : Propriété d'un système logique caractérisé par une absence, apparente ou réelle, de contradictions.

Cohérence des données (« ***data concistency*** ») : Propriété selon laquelle les données contenues dans la base de données reflètent bien la réalité qu'elles représentent.

Complexité (« ***complexity*** ») : La mesure dans laquelle un composant a une conception ou une implémentation difficile à comprendre et à vérifier.

Critère (« ***condition*** ») : Mesure qualitative ou quantitative d'une caractéristique ou d'un attribut qui spécifie une capacité requise. Par exemple, le système doit être redevable d'un débit de 400 transactions à la minute.

Défaillance (« ***failure*** ») : incapacité constatée de réaliser une fonction à un moment et dans des conditions documentées.

Défaut (« ***fault*** ») : cause de l'erreur.

Disponibilité (« ***availability*** ») : La capacité d'un composant à être accessible lorsque l'on en a besoin.

Efficience (« ***efficiency*** ») : La capacité d'un composant à accomplir sa fonction en consommant un minimum de ressources.

Erreur (« ***error*** ») : écart constaté entre la spécification et la mise en œuvre (conception ou mise en exploitation).

Essai (« ***test suite*** ») : un ensemble (généralement ordonné) de cas de tests.

Expressivité (« ***expressive power*** ») : caractérise la capacité à pouvoir s'exprimer aisément dans un langage.

Fiabilité (« ***reliability*** ») : Le logiciel livre toujours un résultat dans un temps raisonnable (ne plante pas, ne boucle pas).

Performance (« ***performance*** ») : La capacité d'un composant à accomplir sa fonction selon des contraintes établies sur la consommation des ressources.

Propriété (« ***attribute*** ») : La caractéristique d'un objet, par exemple sa couleur, sa taille, son type.

Robustesse (« ***robustness*** ») : Capacité d'un composant à continuer de fonctionner même quand il reçoit de mauvaises données en entrée ou dans des conditions environnementales anormales.

Séance d'essai (« ***test execution*** ») : l'activité qui consiste à exécuter un essai et en consigner les résultats; pour certains, la séance comprend également l'interprétation des résultats.

Sécurité (« ***security*** ») : Fonction qui permet de s'assurer que les services sont employés de façon adéquate par les clients appropriés.

Seuil (« ***bound*** ») : Limite au-dessus ou au-dessous de laquelle une caractéristique est applicable.

Tolérance aux pannes (« ***fault tolerance*** ») : Capacité d'un composant à continuer de fonctionner malgré la défaillance de sous-composants (matériels ou logiciels). Très complexe non seulement parce qu'il est impossible de prévoir toutes les fautes, mais aussi parce qu'elles peuvent avoir été introduites très tôt dans le cycle de vie du logiciel.

Tolérance aux erreurs (« ***error tolerance*** ») : Capacité d'un composant à continuer de fonctionner malgré la présence de données erronées.

Traçabilité (« ***traceability*** ») : La mesure dans laquelle la raison d'être de chaque composant du système est motivée. C'est aussi la capacité à pouvoir identifier facilement leur évolution et les liens entre les composants.

Utilisabilité (« ***usability*** ») : Caractérise l'aisance avec laquelle un utilisateur peut se servir d'un composant (exécution, préparation des entrées, interprétation des sorties).

Validation (« ***validation*** ») : S'assurer de l'adéquation par rapport au domaine d'application compte tenu d'objectifs (de critères) connus.

Vérification (« ***verification*** ») : S'assurer de la conformité par rapport à la spécification.

Validité (« ***validity*** ») : Lorsque le logiciel livre un résultat, celui-ci est juste.

Bibliographie

[1] Angles, R. et Gutierrez, C. (2008). The Expressive Power of SPARQL. Dans ISWC '08: *Proceedings of the 7th International Conference on The Semantic Web*. Springer-Verlag, Berlin, Heidelberg, p. 114-129.

[2] Anthes, G. (2010). Happy Birthday, RDBMS! *Commun.ACM*, volume 53, numéro 5, p. 16-17.

[3] Axelsson, S. (2000). Intrusion Detection Systems: A Survey and Taxonomy. Chalmers Univ.

[4] Biba (1977). Integrity Considerations for Secure Computer Systems. *MITRE Co., technical report ESD-TR 76-372,*

[5] Bitton, D., DeWitt, D. J. et Turbyfill, C. (1983). Benchmarking Database Systems A Systematic Approach. Dans *VLDB '83: Proceedings of the 9th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, p. 8-19.

[6] Brewer, D. F. C. et Nash, M. J. (1989). The Chinese Wall security policy. Dans *IEEE Symposium on Security and Privacy Proceedings, 1989*, p. 206-214.

[7] Burleson Enterprises, I. (2006). Database Benchmark Wars : What you need to know. http://www.dba-oracle.com/art_db_benchmark.htm (page consultée le 2 décembre 2010).

[8] Cardelli, L. et Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *ACM Comput. Surv.*, volume 17, numéro 4, p. 471-523.

[9] Clark, D. D. et Wilson, D. R. (1987). A Comparison of Commercial and Military Computer Security Policies. Dans *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, p. 184-194.

[10] Codd, E. F. (1990). *The Relational Model for Database Management, Version 2.* Addison-Wesley,

[11] Coleman, D., Lowther, B. et Oman, P. (1995). The application of software maintainability models in industrial software systems. *Journal of Systems and Software,* volume 29, numéro 1, p. 3.

[12] Date, C. J. (2008). A critique of Claude Rubinson's paper nulls, three - valued logic, and ambiguity in SQL: critiquing Date's critique. *SIGMOD Rec.*, volume 37, numéro 3, p. 20-22.

[13] Date, C. J. et Darwen, H. (2006). *Databases, Types and the Relational Model*, 3rd édition. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA,

[14] De Capitani di Vimercati, S., Paraboschi, S. et Samarati, P. (2003). Access control: principles and solutions. *Softw. Pract. Exper.*, volume 33, numéro 5, p. 397-421.

[15] Debnath, B., Rahul, R., Farkhod, A. A. et Minkyu, C. (2009). Biometric Authentication: A Review. *International Journal of u- and e- Service*, volume 2, numéro 3,

[16] Elmasri, R. et Navathe, S. B. (2003). *Fundamentals of Database Systems, Fourth Edition.* Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA,

[17] Fry, J. P. et Sibley, E. H. (1976). Evolution of Data-Base Management Systems. *ACM Comput. Surv.*, volume 8, numéro 1, p. 7-42.

[18] Gray, J. (1981). The Transaction Concept: Virtues and Limitations (Invited Paper). Dans *VLDB '81: Proceedings of the Very Large Database Conference*, p. 144-154.

[19] Halstead, M. H. (1977). Elements of Software Science. *Elsevier Science Inc*, New York, NY, USA,

[20] Hewlett-Packard Development Company (2009). HP OpenVMS Systems Documentation. http://h71000.www7.hp.com/doc/73final/6556/6556pro_009.html (page consultée le 25 Août 2010).

[21] Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Commun.ACM*, volume 12, numéro 10, p. 576-580.

[22] Hyperic (2009). *SIGAR API.* http://www.hyperic.com/products/sigar (page consultée le 13 novembre 2011)

[23] IEEE (1991). *IEEE-STD-610 ANSI/IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology.* Dans IEEE.

[24] ISO/IEC 9075-2:2003 *Information technology -- Database languages -- SQL -- Part 2 : Foundation.* ISO/IEC.

[25] Jajodia, S., Samarati, P., Sapino, M. L. et Subrahmanian, V. S. (2001). Flexible support for multiple access control policies. *ACM Trans.Database Syst.*, volume 26, numéro 2, p. 214-260.

[26] Kim, H. (1995). Biometrics, is it a viable proposition for identity authentication and access control? *Computers & Security*, volume 14, numéro 3, p. 205.

[27] Klemola, T. et Rilling, J. (2003). A cognitive complexity metric based on category learning. Dans *Proceedings of The Second IEEE International Conference on Cognitive Informatics, 2003*, p. 106.

[28] Kushwaha, D. S. et Misra, A. K. (2006). A complexity measure based on information contained in the software. Dans *Proceedings of the 5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*. Stevens Point, Wisconsin, USA, p. 187-195.

[29] Lampson, B. W. (1974). Protection. *SIGOPS Oper.Syst.Rev.*, volume 8, numéro 1, p. 18-24.

[30] Lewis, I.,P.M. et Stearns, R. E. (1968). Syntax-Directed Transduction. *ACM*, volume 15, numéro 3, p. 465-488.

[31] Libkin, L. (2001). *Expressive Power of SQL*. Database Theory - ICDT 2001, p. 1-21.

[32] Little, J., Emami-Naeini, A. et Bangert, S. (1984). CTRL-C and matrix environments for the computer-aided design of control systems. *Proc. 6th International Conference on Analysis and Optimisation (INRIA)*, volume 63, p. 191-205.

[33] McCabe, T. J. (1976). A Complexity Measure. *IEEE Trans. Software Eng*, volume SE-2, numéro 4, p. 308.

[34] Meyer, B. (1992). Applying 'design by contract'. *Computer*, volume 25, numéro 10, p. 40.

[35] Mitchell, J. C. (1993). On abstraction and the expressive power of programming languages. *Science of Computer Programming*, volume 21, numéro 2, p. 141.

[36] MySQL, (2005). MySQL Performance Benchmarks. http://www.mysql.com/why-mysql/white-papers/performance.php (page consultée le 8 juillet 2010).

[37] National Institute of Standards and Technology (1999). *FIPS PUB 46-3 : Data Encryption Standard (DES)*. http://www.itl.nist.gov/fipspubs/fip186-2.pdf (page consultée le 13 novembre 2011).

[38] National Institute of Standards and Technology (2001). *FIPS PUB 197 : Advanced Encryption Standard (AES)*. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (page consultée le 13 novembre 2011).

[39] National Institute of Standards and Technology (1996). User's Guide for the SQL Test Suite, Version 6.0. http://www.itl.nist.gov/div897/ctg/sql-testing/sqlman60.htm (page consultée le 3 décembre 2010).

[40] NTFS Permissions. http://www.itl.nist.gov/div897/ctg/sql-testing/sqlman60.htm (page consultée le 12 janvier 2011).

[41] Office québécois de la langue française. Grand Dictionnaire Terminologique. Dans http://www.granddictionnaire.com (page consultée le 3 décembre 2011).

[42] The Open Source Database Benchmark (2009). http://osdb.sourceforge.net/ (page consultée le 3 décembre 2010).

[43] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S. et Stonebraker, M. (2009). A comparison of approaches to large-scale data analysis. *Dans SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*. ACM, New York, NY, USA, p. 165-178.

[44] PostgreSQL. Pgbench. http://www.postgresql.org/docs/9.0/interactive/pgbench.html (page consultée le 12 janvier 2011).

[45] Rubenstein, W. B., Kubicar, M. S. et Cattell, R. G. G. (1987). Benchmarking simple database operations. Dans *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*. ACM, New York, NY, USA, p. 387-394.

[46] Sanchez-Avila, C. et Sanchez-Reillol, R. (2001). The Rijndael block cipher (AES proposal): a comparison with DES. Dans *IEEE 35th International Carnahan Conference on Security Technology, 2001*, p. 229.

[47] The Center for Internet Security (2005). *Center for Internet Security Benchmark for Oracle 9i/10g*. *http://www.cisecurity.org* (page consultée le 12 janvier 2011).

[48] Transaction Processing Performance Council (TPC) (2010). *TPC BENCHMARK ™ C*. *http://tpc.org/tpcc/default.asp* (page consultée le 12 janvier 2011).

[49] Transaction Processing Performance Council (TPC) (2010). *TPC BENCHMARK ™ E*. *http://tpc.org/tpce/default.asp* (page consultée le 12 janvier 2011).

[50] Transaction Processing Performance Council (TPC) (2010). *TPC BENCHMARK ™ H*. *http://tpc.org/tpch/default.asp* (page consultée le 12 janvier 2011).

[51] Turbyfill, C., Orji, C. et Bitton, D. (1989). AS3AP - A Comparative Relational Database Benchmark. Dans *COMPCON Spring '89 : Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers*. p. 560.

[52] Verifysoft (2010). Measurement of Halstead Metrics with Testwell CMT++ and CMTJava (Complexity Measures Tool). http://www.verifysoft.com/en_halstead_metrics.html (page consultée le 14 janvier 2011).

[53] Vieira, M. et Madeira, H. (2005). Towards a security benchmark for database management systems. Dans *DSN 2005. Proceedings of the International Conference on Dependable Systems and Networks*, p. 592.

[54] Wang, X. et Yu, H. (2005). How to break MD5 and other hash functions. Dans *Cryptology-Eurocrypt'05, p.19-35* Springer-Verlag,

[55] Wang, Y. et Shao, J. (2003). Measurement of the Cognitive Functional Complexity of Software. Dans *Proceedings of the 2nd IEEE International Conference on Cognitive Informatics*. ICCI '03. IEEE Computer Society, Washington, DC, USA, p. 67.

[56] Wikipedia. Permissions Unix. http://fr.wikipedia.org/wiki/Permissions_Unix (page consultée le 25 août 2010).

[57] Young, M., Pezze, M. (2005). Software Testing and Analysis: Process, Principles and Techniques. John Wiley & Sons, p. 32-33.

[58] Zaitsev, P. (2003). *The MySQL Benchmark and Testing Project.* http://www.mysqlperformanceblog.com/files/presentations/UC2003_MySQLBenchmarkandTesting.pdf (page consultée le 12 janvier 2011).