Le langage PL/SQL (http://sql.bdpedia.fr/plsql.html)

Le langage SQL n’est pas un langage de programmation au sens courant du terme. Il ne permet pas, par exemple, de définir des fonctions ou des variables, d’effectuer des itérations ou des instructions conditionnelles. Il ne s’agit pas d’un défaut dans la conception du langage, mais d’une orientation délibérée de SQL vers les opérations de recherche de données dans une base volumineuse, la priorité étant donnée à la *simplicité* et à l’efficacité\*. Ces deux termes ont une connotation forte dans le contexte d’un langage d’interrogation, et correspondent à des critères (et à des contraintes) précisément définis. La simplicité d’un langage est essentiellement relative à son caractère *déclaratif*, autrement dit à la capacité d’exprimer des recherches en laissant au système le soin de déterminer le meilleur moyen de les exécuter. L’efficacité est, elle, définie par des caractéristiques liées à la complexité d’évaluation sur lesquelles nous ne nous étendrons pas ici. Signalons cependant que la terminaison d’une requête SQL est *toujours* garantie, ce qui n’est pas le cas d’un programme écrit dans un langage plus puissant.

Il est donc clair que SQL ne suffit pas pour le développement d’applications, et tous les SGBD relationnels ont, dès l’origine, proposé des interfaces permettant de l’associer à des langages plus classiques comme le C ou Java. Ces interfaces de programmation permettent d’utiliser SQL comme outil pour récupérer des données dans des programmes réalisant des tâches très diverses : interfaces graphiques, traitements “batch”, production de rapports ou de sites web, etc. D’une certaine manière, on peut alors considérer SQL comme une interface d’accès à la base de données, intégrée dans un langage de programmation généraliste. Il s’agit d’ailleurs certainement de son utilisation la plus courante.

Pour certaines fonctionnalités, le recours à un langage de programmation “externe” s’avère cependant inadapté ou insatisfaisant. Une évolution des SGBD consiste donc à proposer, au sein même du système, des primitives de programmation qui viennent pallier le manque relatif d’expressivité des langages relationnnels. Le présent chapitre décrit ces évolutions et leur application à la création de *procédures stockées* et de *triggers*. Les premières permettent d’enrichir un schéma de base de données par des calculs ou des fonctions qui ne peuvent pas - parfois même dans des cas très simples - être obtenus avec SQL ; les seconds étendent la possibilité de définir des contraintes.

Parallèlement à ces applications pratiques, les procédures stockées illustrent simplement les techniques d’intégration de SQL à un langage de programmation classique, et soulignent les limites d’utilisation d’un langage d’interrogation, et plus particulièrement du modèle relationnel.

Procédures stockées

Comme mentionné ci-dessus, les procédures stockées constituent une alternative à l’écriture de programmes avec une langage de programmation généraliste. Commençons par étudier plus en détail les avantages et inconvénients respectifs des deux solutions avant d’entrer dans les détails techniques.

Rôle et fonctionnement des procédures stockées

Une procédure stockée s’exécute au sein du SGBD, ce qui évite les échanges réseaux qui sont nécessaires quand les mêmes fonctionnalités sont implantées dans un programme externe communiquant en mode client/serveur avec la base de données. La figure [Comparaison programmes externes/procédures stockées](http://sql.bdpedia.fr/plsql.html#plsql) illustre la différence entre les deux mécanismes. À gauche un programme externe, écrit par exemple en C, doit tout d’abord se connecter au serveur du SGBD. Le programme s’exécute alors en communiquant avec le serveur pour exécuter les requêtes et récupérer les résultats. Dans cette architecture, chaque demande d’exécution d’un ordre SQL implique une transmission sur le réseau, du programme vers le client, suivie d’une analyse de la requête par le serveur, de sa compilation et de son exécution (Dans certains cas les requêtes du programme client peuvent être précompilées, ou “préparées”. Ensuite, chaque fois que le programme client souhaite récupérer un n-uplet du résultat, il doit effectuer un appel externe, via le réseau. Tous ces échanges interviennent de manière non négligeable dans la performance de l’ensemble, et cet impact est d’autant plus élevé que les communications réseaux sont lentes et/ou que le nombre d’appels nécessaires à l’exécution du programme est important.

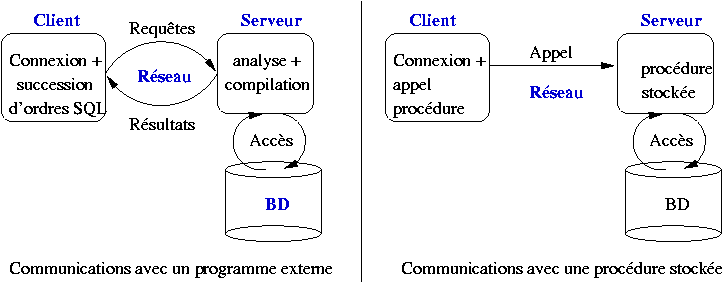
[](http://sql.bdpedia.fr/_images/plsql.png)

Fig. 15 Comparaison programmes externes/procédures stockées

Le recours à une procédure stockée permet de regrouper du côté serveur l’ensemble des requêtes SQL et le traitement des données récupérées. La procédure est compilée une fois par le SGBD, au moment de sa création, ce qui permet de l’exécuter rapidement au moment de l’appel. De plus les échanges réseaux ne sont plus nécessaires puisque la logique de l’application est étroitement intégrée aux requêtes SQL. Le rôle du programme externe se limite alors à se connecter au serveur et à demander l’exécution de la procédure, en lui passant au besoin les paramètres nécessaires.

Bien entendu, en pratique, les situations ne sont pas aussi tranchées et le programme externe est en général amené à appeler plusieurs procédures, jouant en quelque sorte le rôle de coordinateur. Si les performances du système sont en cause, un recours judicieux aux procédures stockées reste cependant un bon moyen de réduire le trafic client-serveur.

L’utilisation de procédures stockées est par ailleurs justifiée, même en l’absence de problèmes de performance, pour des fonctions très “sensibles”, terme qui recouvre (non exclusivement) les cas suivants :

1. la fonction est basée sur des règles complexes qui doivent être implantées très soigneusement ;
2. la fonction met à jour des données dont la correction et la cohérence sont indispensable au bon fonctionnement de l’application ;
3. la fonction évolue souvent.

On est souvent amené, quand on développe une application, à utiliser plusieurs langages en fonction du contexte : le langage C ou Java pour les traitements *batch*, PHP ou Python pour l’interface web, un générateur d’application propriétaire pour la saisie et la consultation à l’écran, un langage de script pour la production de rapports, etc. Il est important alors de pouvoir factoriser les opérations de base de données partagées par ces différents contextes, de manière à les rendre disponibles pour les différents langages utilisés. Par exemple la réservation d’un billet d’avion, ou l’exécution d’un virement bancaire, sont des opérations dont le fonctionnement correct (pas deux billets pour le même siège ; pas de débit sans faire le crédit correspondant) et cohérent (les mêmes règles doivent être appliquées, quel que soit le contexte d’utilisation) doit toujours être assuré.

C’est facile avec une procédure stockée, et cela permet d’une part d’implanter une seule fois des fonctions “sensibles”, d’autre part de garantir la correction, la cohérence et l’évolutivité en imposant l’utilisation systématique de ces fonctions au lieu d’un accès direct aux données.

Enfin, le dernier avantage des procédures stockées est la relative facilité de programmation des opérations de bases de données, en grande partie à cause de la très bonne intégration avec SQL. Cet aspect est favorable à la qualité et à la rapidité du développement, et aide également à la diffusion et à l’installation du logiciel puisque les procédures sont compilées par les SGBD et fonctionnent donc de manière identique sur toute les plateformes.

Il existe malheureusement une contrepartie à tous ces avantages : chaque éditeur de SGBD propose sa propre extension procédurale pour créer des procédures stockées, ce qui rend ces procédures incompatibles d’un système à un autre. Cela peut être dissuasif si on souhaite produire un logiciel qui fonctionne avec tous les SGBD relationnels.

La description qui suit se base sur le langage PL/SQL d’Oracle (“PL” signifie *Procedural Language*) qui est sans doute le plus riche du genre. Le même langage, simplifié, avec quelques variantes syntaxiques mineures, est proposé par PostgreSQL, et les exemples que nous donnons peuvent donc y être transposés sans trop de problème. Les syntaxes des langages utilisés par d’autres systèmes sont un peu différentes, mais tous partagent cependant un ensemble de concepts et une proximité avec SQL qui font de PL/SQL un exemple tout à fait représentatif de l’intérêt et de l’utilisation des procédures stockées.

Introduction à PL/SQL