

The logo for Baroudi Bloor features a blue rectangular background with a white triangular shape on the left side. The text "BAROUDI BLOOR" is written in a white, serif font across the top of the blue area.

BAROUDI BLOOR

**L'échec des Bases de Données Relationnelles,
L'essor de la Technologie Objet
et le Recours aux Bases de Données Hybrides.**

"La technologie objet rapproche les langages de programmation du langage naturel et s'imposent rapidement dans la plupart des domaines du développement de logiciels."

Un paradigme révolutionnaire

Lorsqu'elle est présentée pour la première fois à la fin des années 60, la technologie objet est révolutionnaire. A la fin des années 80, elle se répand et ce, pour de bonnes raisons. En effet, non seulement elle simplifie le développement des interfaces, mais elle offre également un moyen plus souple et plus adapté au traitement des données qui change considérablement la manière de créer des applications. Au lieu de représenter les données dans des tables rigides à la manière des bases de données relationnelles, la technologie objet les décrit en termes de classes. Un objet est une instance de classe tout comme un chêne est une instance de la classe des "chênes".

La technologie objet contribue au concept d'héritage en permettant le classement hiérarchique des classes. La classe des "chênes" peut ainsi hériter des structures de données et des comportements de la classe des "arbres", plus générale.

La technologie objet peut tout à fait être comparée à la manière dont l'utilisateur perçoit le monde et les langages OO s'avèrent plus souples dans la plupart des domaines de programmation. Ils permettent de rapprocher les langages de programmation du langage naturel et s'imposent dans la plupart des domaines du développement de logiciels. Le langage OO est considéré comme "le nouveau paradigme" et son influence s'étend de tous côtés.

Des fonctionnalités OO sont rapidement ajoutées à de nombreux langages reconnus, donnant naissance à des langages tels que C++. De nouveaux environnements de développement OO apparaissent et notamment Visual Basic, Visual C++, PowerBuilder, Delphi et Caché. La technologie objet excelle dans les environnements de développement de logiciels, mais il lui faut un certain temps pour qu'elle apparaisse dans les publications officielles. La conception et la création d'un monde véritablement basé sur des objets demande encore plus de temps et nous n'y sommes pas encore parvenu.

Déploiement de la technologie objet à l'aide du Web

Avec la transformation des échanges d'informations de toutes sortes par le Web, le langage de programmation OO Java remporte un grand succès auprès des développeurs Web. Reposant sur le langage C++, Java permet de créer de petites applications (applets Java) pouvant être utilisées à partir d'un navigateur.

Sun fournit gratuitement l'environnement Java pour en promouvoir l'utilisation. En quelques années, des millions de copies sont téléchargées et Java se répand. Par la suite, Java donne naissance à d'autres langages OO tels que JavaScript, C# et JScript. Le développement d'Internet engendre également de nouveaux langages OO, tels que Perl et PHP

Aujourd'hui, les développeurs utilisent tout naturellement les fonctionnalités OO.

L'essor de l'Objet

La technologie objet décrit tous les aspects du développement de logiciels. La modélisation OO domine le marché de la modélisation des applications, la méthodologie UML standard ayant une longueur d'avance.

Au cours des années 90 apparaissent les middleware OO créés pour fournir des services de communication sécurisés entre les applications OO. Leur marché connaît une forte croissance avec l'apparition de JMS (Java Messaging Service) en 1998. JMS définit un ensemble complet d'API de messagerie et instaure l'utilisation d'un serveur JMS dans les implémentations J2EE certifiées. C'est le début de la normalisation ; le coût des middleware diminue considérablement et les développeurs disposent d'une plateforme de création d'applications Objet à l'échelle de l'entreprise.

"Aujourd'hui, les développeurs utilisent tout naturellement les fonctionnalités OO."

XML et services Web

En 1998, HTML, le langage de balises utilisé pour décrire la présentation des pages Web, est étendu et normalisé pour créer le langage XML (eXtended Markup Language). XML offre une syntaxe permettant de créer des formats de données auto descriptifs, similaires aux définitions de données stockées dans les bases de données. Avec XML, les programmes peuvent associer des définitions à des données et échanger à la fois les données et leur signification. XML permet de définir des objets de données normalisés spécifiques, tels que des factures ou des bons de commandes, afin de faciliter les échanges de données au sein d'une entreprise ou entre sociétés. XML a donné naissance aux services Web, des programmes pouvant interagir directement avec d'autres programmes sans aucune personnalisation nécessaire. Actuellement, deux environnements de services Web formels, J2EE et .NET, se sont imposés. Aujourd'hui, l'impact du langage XML sur les données est considérable. Tout comme le langage SQL, XML offre aux développeurs une norme d'accès aux données, mais il fournit en outre un langage standard pour la définition des données au niveau de l'objet. La hausse de popularité du langage XML est pratiquement aussi spectaculaire que celle de la programmation OO. Par conséquent, de nouvelles normes relatives aux objets de données et de nouvelles plateformes de développement basés sur le langage XML continuent d'émerger.

Bases de données objet : le chaînon manquant

L'intégration rapide de la technologie objet dans pratiquement tous les aspects du développement de logiciels est complètement opposée à l'intégration des bases de données objet, lente et jusqu'à récemment limitée et ce, pour de multiples raisons.

Les premiers langages OO n'accordaient aucune importance au stockage des données. Les programmes utilisaient les données en mémoire et stockaient l'intégralité de l'image des données dans un fichier lu ultérieurement lors de leur exécution. Cette approche ne permettait pas aux applications de partager leurs données et rendait impossible les possibilités de récupération, de gestion et d'évolution des données.

Il n'est donc pas étonnant qu'un grand nombre de produits de base de données objet (Versant, Objectivity, ObjectStore, GemStone et bien d'autres) soient apparus sur le marché afin de fournir des entrepôts de données appropriés aux environnements de développement OO. Au départ, ces produits suscitèrent un grand enthousiasme, laissant à penser qu'ils allaient s'accaparer une grande part du marché des bases de données, voire même le dominer.

Malheureusement, les fournisseurs de bases de données relationnelles avaient gagné du terrain et pénétré le marché avant même l'apparition des bases de données objet. Il était facile d'écrire des routines OO pour accéder aux bases de données relationnelles à l'aide de leurs interfaces SQL normalisées. Par contre, la plupart des premières bases de données objet ne proposaient aucune fonctionnalité SQL et ne convenaient pas aux applications d'interrogation. Les bases de données objet ne réussirent donc jamais à s'imposer dans les systèmes d'entreprise. Cependant, elles parvinrent à occuper certaines niches dans des domaines d'application gérant et stockant les objets complexes de certaines applications (CAO/FAO, télécommunications, multimédia, intelligence artificielle, instruments financiers de modélisation, systèmes de suivi médical des patients et applications scientifiques).

Le marché des bases de données n'a jamais prêté une grande attention aux bases de données objet jusqu'à l'émergence de XML comme langage de définition de données. C'est à partir de ce moment qu'elles sont réapparues pour gérer les données définies en XML, pour lesquelles elles convenaient parfaitement. L'utilisation du langage XML, combiné au besoin croissant de stocker des données complexes, semble être la principale cause de la réapparition des bases de données objet.

Une enquête menée auprès des développeurs, publiée par InfoWorld en septembre 2003 révèle des résultats surprenants. Alors que 89,2 % d'entre eux prétendent utiliser des bases de données relationnelles, 52 % déclarent également utiliser des bases de données orientées objet ou XML. En réponse à une question sur les types de données stockées, 40,2 % indiquent qu'ils stockent des objets persistants, 58,9 %, des données XML et 89 %, des données relationnelles. Pour Baroudi Bloor, les bases de données objet sont bien plus répandues qu'on ne le pense, suite à une pénétration discrète du marché en réponse à une demande pressante.

En outre, l'enquête d'InfoWorld indique clairement que les langages OO sont largement plébiscités pour les nouveaux développements. Nous pensons que ces statistiques reflètent le dilemme auquel les développeurs sont actuellement confrontés. Ces derniers requièrent des bases de données compatibles avec les langages OO qu'ils utilisent, mais ils ont toujours besoin des fonctionnalités d'interrogation offertes par les bases de données relationnelles.

Bases de données relationnelles : l'autre face

Les données sont apparues avec les programmes. La gestion des données est l'une des premières raisons pour lesquelles les sociétés se sont intéressées à l'informatique. Avec la gestion automatique des données, l'entreprise peut évoluer et rivaliser avec ses concurrents à l'aide de nouveaux moyens. Il n'est donc pas surprenant que les technologues d'entreprise perspicaces aient ciblé très tôt le marché de la gestion des données. Plus d'une dizaine d'années avant même la naissance du concept des bases de données objet, la théorie relationnelle des données proposée par le Docteur E. F. Codd apparaissait dans les applications commerciales de base de données relationnelle. Au milieu des années 80, une conviction quasi fanatique dans certains domaines du secteur informatique soutenait que tous les problèmes théoriques des données étaient désormais résolus et que les problèmes pratiques le seraient bientôt également. Manifestement, ce n'était pas le cas.

Les bases de données relationnelles représentaient les données dans de simples tables à deux dimensions ; un moyen efficace de représenter une quantité importante de données d'une manière facilement compréhensible par les programmeurs. Avec SQL, les bases de données relationnelles instaurèrent un langage d'accès aux données standard. Leur structure logique et physique, indépendante des applications, était compatible avec de nombreuses applications d'entreprise.

Toutefois, la théorie relationnelle reposait en partie sur l'idée que les données et les programmes qui les utilisaient, pouvaient et devaient être indépendants les uns des autres. Cette idée était et reste contradictoire avec le concept même de la technologie objet. La technologie objet encourage le développeur à percevoir les données comme des objets et non comme des tables. Par ailleurs, les objets et les méthodes qui les utilisent ne sont pas indépendants les uns des autres.

"...nous entendons toujours parler de projets qui échouent car les performances de la base de données relationnelle utilisée sont tout simplement insuffisantes."

Prenons le cas d'une voiture en tant qu'objet complexe. Lorsque vous utilisez cette voiture, vous l'utilisez intégralement, comme une même entité ; comme un objet. Maintenant, de nombreuses activités (méthodes en terminologie OO) sont associées à la voiture. Par exemple, vous la manœuvrez, vous changez de vitesse, vous mettez les clignotants, vous allumez les feux, etc. Si la voiture est un objet, ces activités représentent les méthodes de l'objet et lui sont essentielles. Il est absurde de dissocier ces activités de la voiture. Lorsque vous rentrez votre voiture dans un garage, vous la garez comme une même entité, avec toutes ses fonctions. En aucun cas, vous ne garez votre voiture dans le garage et stockez ses fonctions de direction, de transmission, de signalement et d'allumage des feux ailleurs. Les données et les processus associés ne peuvent et ne doivent pas être dissociés ; c'est le cas dans les bases de données objet.

En réalité, ces deux points de vue présentent des avantages et des inconvénients. Certains processus sont réellement indépendants des données. C'est notamment le cas des requêtes qui accèdent à d'importants jeux de données. Une requête se contente de sélectionner les données en fonction d'un ensemble de critères et ne se préoccupe pas de leur contenu ou de la manière dont elles sont organisées, dès lors qu'elles peuvent être extraites rapidement. Les requêtes sont indépendantes des données, mais les méthodes d'un objet ne le sont pas.

Limites des bases de données relationnelles

Les bases de données relationnelles offrent moins de possibilités qu'elles ne le laissent penser. Le stockage et la représentation de structures de données, aussi ordinaires soient-elles, peuvent être assez difficiles. Prenons l'exemple d'un itinéraire de bus, qui correspond en fait à une simple liste ordonnée des arrêts du bus. Les bases de données relationnelles ne conservent les tables que sous la forme de listes non ordonnées et ne peuvent extraire une liste ordonnée que si un index spécialement généré a été ajouté. Une base de données objet, quant à elle, gère parfaitement les listes ordonnées et ne requiert aucun index, ce dernier n'étant créé artificiellement qu'en raison des limites des structures des données relationnelles.

Prenons maintenant l'exemple d'une nomenclature, soit un produit et ses composants dans un système de fabrication. Les composants peuvent eux-mêmes posséder d'autres composants qui à leur tour peuvent également posséder des composants et ainsi de suite. Une table de base de données relationnelle répertoriant l'intégralité des composants ne représente pas la relation entre les composants et leurs sous-composants, etc. Ces relations représentent des données importantes. La recherche d'un produit et de tous ses composants dans une base de données doit être sans équivoque. La structure d'une base de données relationnelle complique inutilement le travail du développeur, qui consiste à répondre à cette simple requête. Les exemples de ce type abondent : une carte, ses routes, ses

rivières et ses points de repère ou encore un site Web et toutes ses pages, ses liens et ses graphiques. En fait, plus l'ensemble d'informations est complexe, plus il existe de niveaux hiérarchiques et de relations croisées et moins il est possible de représenter ces informations dans les structures de table simples d'une base de données relationnelle. Les bases de données objet ne connaissent pas de telles limites étant donné qu'elles ont été conçues pour résoudre les problèmes de ce genre.

Malgré la maturité des produits de base de données relationnelle et la croissance spectaculaire de la puissance des ordinateurs au cours de la dernière décennie, nous entendons toujours parler de projets qui échouent car les performances de la base de données relationnelle utilisée sont tout simplement insuffisantes. Cela provient généralement de la manière dont les bases de données relationnelles stockent les données. Pour pouvoir assembler les données dont ils ont besoin, les développeurs doivent souvent créer plusieurs jointures (requêtes JOIN) d'une table à une autre, puis à une autre et ainsi de suite. Pour extraire les données, la base de données exécute des routines d'optimisation afin de déterminer le meilleur moyen de collecter les données, puis de les extraire. Ce processus prend souvent un certain temps et peut avoir un impact négatif sur les performances. Les optimiseurs de base de données relationnelle ont eu beau s'améliorer avec le temps, leurs performances restent bien inférieures à celle des bases de données objet.

Bases de données relationnelles et défaut d'impédance

Le problème des bases de données relationnelles réside dans le fait qu'elles utilisent une table à deux dimensions comme structure de données de base. Selon la théorie relationnelle, les données sont censées être organisées en tables normalisées ; les données doivent être organisées de telle sorte qu'il n'existe qu'un seul moyen d'y accéder. Le développeur peut ainsi éliminer tout risque de redondance et garantir la cohérence des modifications apportées aux données. Cette technique de conception a été proposée afin que les tables relationnelles contiennent des jeux de données indépendants associés uniquement par l'intermédiaire d'une clé. Elle provient directement de la théorie des ensembles, mais cette théorie ne permet malheureusement pas de représenter toutes les relations et les structures des données.

Le stockage des données sous une forme normalisée nécessite souvent que le programmeur désassemble un objet pour le stocker dans la base de données, puis l'assemble à nouveau à l'aide de requêtes SQL (plusieurs requêtes JOIN) afin de l'utiliser. Si vous garez une voiture dans un garage, cela revient à démonter les portes, les sièges, les roues, etc. Cela prend du temps et ne présente aucun intérêt.

"Des estimations montrent en effet que les développeurs de programmes OO utilisant des bases de données relationnelles passent entre 25 et 40 % de leur temps à écrire un code mappant les objets aux tables relationnelles." mappant les objets aux tables relationnelles."

Ce problème est apparu lorsque les langages OO ont commencé à s'imposer. Il est généralement décrit comme l'erreur d'impédance objet-relationnel qui représente la différence entre les approches utilisées par les langages OO et les bases de données relationnelles pour accéder aux données et les problèmes résultants que le programmeur doit résoudre. En réalité, la plupart des bases de données relationnelles ne sont pas entièrement normalisées lors de leur implémentation, mais cela n'empêche pas les problèmes d'erreur d'impédance, ce qui complique la tâche du développeur. Des estimations montrent en effet que les développeurs de programmes OO utilisant des bases de données relationnelles passent entre 25 et 40 % de leur temps à écrire un code mappant les objets aux tables relationnelles.

Cette difficulté de base aurait peut-être créé une forte demande de bases de données objet, si ces dernières n'avaient pas rencontré elles aussi un problème majeur : la quasi-inexistence de la prise en charge du langage SQL. De nombreux outils logiciels nécessitent une interface SQL et en particulier les applications d'aide à la décision. Même les bases de données objet comportant des interfaces SQL n'étaient pas conçues pour gérer le type de volume de requêtes généré par les applications d'aide à la décision.

Base de données objet-relationnel

Les fournisseurs de bases de données relationnelles n'ignoraient pas que les objets allaient s'imposer. Il était clair que la normalisation de données complexes n'avait aucun sens. Prenons l'exemple extrême suivant : si vous normalisez une image bitmap (une liste ordonnée de pixels), vous obtenez une table dont les lignes correspondent aux pixels et leurs attributs et une clé principale indiquant leur ordre. Il est évidemment préférable de stocker les données comme objet.

Ces mêmes fournisseurs proposèrent l'idée d'une base de données "Objet-Relationnel", de manière à conserver la structure de la base de données relationnelle prépondérante, tout en permettant aux colonnes d'une table relationnelle de contenir des objets complexes. Ces objets pouvaient être accompagnés de processus (procédures stockées de même type) permettant de décomposer les données complexes. SQL fut alors amélioré pour permettre les appels à l'équivalent relationnel des "méthodes objet".

Cette approche ignore complètement la théorie relationnelle des données, mais elle permet de définir des objets complexes (cartes, graphiques vectoriels, photographies et même des tables complètes) dans une structure relationnelle et de les conserver comme entités. Ces fonctionnalités furent donc implémentées et devinrent même des marques. Chez Informix, les processus imbriqués étaient des DataBlades, tandis que chez Oracle, il s'agissait de Cartridges.

Il était donc possible de stocker des objets sans avoir recours à des bases de données objet, mais le principal problème restait à résoudre. En effet les bases de données Objet-Relationnel connaissent toujours le problème d'erreur d'impédance.

Bases de données objets versus bases de données relationnelles

En pratique, les bases de données objet présentent des avantages considérables par rapport aux bases de données relationnelles :

- L'exécution est beaucoup plus rapide pour les applications transactionnelles.
- Elles gèrent de manière bien plus efficace les objets complexes.
- Elles permettent aux développeurs d'obtenir une meilleure productivité.
- Elles sont faciles à administrer.

Dans certains cas, les bases de données objet ont remplacé les bases de données relationnelles pour des raisons de performances. Cette tendance se retrouve même dans certaines applications d'entreprise à grande échelle qui ne nécessitent pas le stockage d'objets complexes (opération qui semblerait pourtant relever du domaine des bases de données relationnelles).

Le principal avantage des bases de données objet en matière de performances est le suivant : contrairement aux bases de données relationnelles, les bases de données objet n'ont généralement pas besoin d'assembler les données pour pouvoir les utiliser. Elles ont tendance à stocker les données sous leur forme la plus couramment utilisée, ce qui permet généralement d'améliorer les performances. Les bases de données objet peuvent implémenter des stratégies de mise en mémoire cache ; ainsi, les données sont plus susceptibles de se trouver en mémoire lorsqu'elles sont demandées. Ces bases ne requièrent qu'une optimisation limitée pour l'extraction des données.

A mesure que de nouveaux systèmes sont développés, le besoin de manipuler des données complexes, telles que des documents, des graphiques sophistiqués, des pages Web et des fichiers multimédia, continue d'augmenter et ces types de demande sont mieux traités par les bases de données objet.

Le paysage OO d'aujourd'hui

L'intégration de la technologie objet a connu une croissance soutenue dans tous les domaines du développement de logiciels. Même dans le dernier domaine réfractaire, les bases de données (bien que les bases de données objet, telles qu'elles existent, n'ont pas détrôné les bases de données relationnelles), InfoWorld signale que 52 % des développeurs utilisent des bases de données orientées objet ou des bases de données XML qui généralement sont elles-mêmes des bases de données objet. Certains développeurs utilisent même une forme hybride permettant une utilisation facile des constructions OO. L'interface Web étant fortement conseillée dans le développement de nouvelles applications et les services Web étant le mécanisme recommandé pour les interactions entre applications, la création pour et dans un environnement orienté objet semble correspondre à la réalité d'aujourd'hui.

L'enquête InfoWorld de septembre 2003 montre également que les programmeurs utilisent quasi-systématiquement les langages OO. En effet, même si certains déclarent ne travailler qu'en langage C, les langages OO semblent être les langages de prédilection de 90 % des programmeurs d'aujourd'hui. Selon cette même enquête, la majorité des programmeurs préféreraient les applications Web, la programmation objet conviviale et les langages de script. Avec de plus en plus d'ingénieurs logiciels formés en programmation OO, l'utilisation de la technologie orientée objet devrait être le seul critère retenu pour le développement de nouvelles applications.

Conclusion

Il se peut que les bases de données relationnelles continuent de dominer le marché des bases de données et que les bases de données objet ne quittent pas la niche qu'elles se sont créées, mais il est également possible que la part de marché de ces dernières augmente à mesure qu'elles parviennent davantage à gérer les objets complexes utilisés de nos jours. Toutefois, il existe une autre possibilité : la technologie des bases de données pourrait évoluer et générer de véritables produits hybrides combinant les avantages de l'interface relationnelle et de l'interface objet. Nous savons que cela est possible. En effet, il existe au moins un produit de ce type, le produit Caché d'InterSystems. (D'ailleurs, la base de données Caché se définit non comme une base de données relationnelle ou une base de données objet, mais comme un produit post-relationnel). Les fournisseurs de bases de données, que leurs produits appartiennent à la catégorie des bases de données relationnelles ou objet, pourraient adopter ce type de produit.

Cette approche consiste à fournir la base de données avec une couche de mapping, par l'intermédiaire de laquelle les développeurs peuvent accéder à la base de données. Cette couche de mapping doit reposer sur des normes ouvertes afin de résoudre le problème d'erreur d'impédance.

"Ce produit hybride ouvre la possibilité de n'avoir qu'un seul moteur de base de données, avec un seul jeu de définitions de données convenant à toutes les applications."

Les appels à la base de données peuvent alors être effectués en SQL ou sous forme de requêtes directes à une classe d'objets ou un ensemble de classes. La couche de mapping convertit alors ces appels en demandes de données physiques transmises à la base de données pour extraire les données, obviant ainsi au défaut d'impédance.

La modification de l'un ou l'autre des deux types de base de données actuels pour parvenir à cette fin représente un véritable défi. Les bases de données objet nécessiteraient des fonctionnalités d'indexation rapides pour extraire les requêtes sélectionnées dans de vastes ensembles de données. Les bases de données relationnelles qui y parviennent le mieux utilisent des index bitmap, mais ces derniers demandent souvent des ressources importantes lors de la mise à jour des données. C'est pour cette raison que peu de bases de données objet disposent de cette fonctionnalité.

Les bases de données relationnelles, quant à elles, devraient offrir des structures de données physiques bien plus souples. Au cours de leur évolution, elles ont eu tendance à implémenter des tables au niveau physique. Elles devraient abandonner cette contrainte rigide et permettre le stockage de structures de données variables. Pour les utilisateurs de base de données, les avantages seraient considérables. Imaginez que les avantages des bases de données objet et des bases de données relationnelles soient combinés :

- Bonnes performances transactionnelles
- Gestion des données complexes
- Facilité d'administration
- Développement rapide
- Fonctionnalités d'interrogation souples
- Interface d'accès aux données standard
- Utilisation indiquée pour les applications d'aide à la décision

Ce produit hybride offre la possibilité de n'avoir qu'un seul moteur de base de données, avec une seule définition de données convenant à toutes les applications. Pour la société Baroudi Bloor, les bases de données hybrides sont indispensables au secteur informatique. Les fournisseurs doivent se libérer de leurs idées reçues et adopter une approche hybride ou disparaître au même titre que le langage COBOL et les cartes perforées.

Copyright 2003, 2004 Baroudi Bloor International, Inc.

Ce document a été rédigé par Robin Bloor de la société Baroudi Bloor, une société spécialisée en recherche, analyse et conseil stratégique dans le domaine de l'informatique.

Robin Bloor est Directeur de la Recherche chez Baroudi Bloor International Inc et Président de Bloor Research., l'un des leaders mondiaux en conseil et analyse informatique offrant des services de recherche et d'analyse aux utilisateurs et fournisseurs informatiques, partout dans le monde. Vous pouvez le contacter à l'adresse suivante : robin@baroudi.com.



BAROUDI BLOOR

175 Pleasant Street — Arlington, MA 02476 — 617-747-4045 — www.baroudi.com