6
Uma Arquitetura Não-Intrusiva para a Manutenção Automática do Projeto Físico de Bancos de Dados

Neste trabalho, propomos uma arquitetura não intrusiva que permite a manutenção (seleção, criação, remoção e reorganização) automática das estruturas que compõem o projeto físico de bancos de dados relacionais, tais como índices, visões materializadas, etc. De um modo geral, a arquitetura propõe a realização de auto-sintonia durante a operação normal do SGBD, por meio da colaboração entre agentes de *software*, os quais utilizam *drivers* para obter e manipular informações do banco de dados. De posse desses dados, os agentes utilizam heurísticas, que encapsulam o conhecimento de especialistas em sintonia de projeto físico, com a finalidade de selecionar o conjunto de estruturas de acesso que seja mais adequado para a carga de trabalho submetida ao banco, ou seja, que reduza o tempo de execução da carga de trabalho.

Nesta arquitetura (Figura 6.1), um *driver* consiste em uma interface Java que contém um conjunto de métodos abstratos. Assim, para cada SGBD que se deseja utilizar, torna-se necessário instanciar estes *drivers*. Instanciar um determinado *driver* consiste em codificar uma classe Java que implementa a interface correspondente ao *driver*. Contudo, o protótipo implementado para validar as idéias propostas nesta tese já fornece *drivers* instanciados para os SGBDs: PostgreSQL 8.2, Oracle 10g e SQL Server 2005[1].

Nosso trabalho está relacionado com as técnicas baseadas em regras, além de não demandar modificações no código fonte do banco de dados. Portanto, a arquitetura proposta pode, virtualmente, ser usada com qualquer SGBD (desde que existam *drivers* instanciados para o SGBD que se tenciona utilizar). A Figura 6.1 mostra a arquitetura proposta.

Os principais componentes desta arquitetura são:

– **Agent for Workload Obtainment (AWO):** Este agente captura periodicamente a carga de trabalho submetida ao SGBD alvo. Para isso, o agente consulta a metabase do SGBD e recupera as cláusulas SQL que

[1] O apêndice F mostra os componentes da arquitetura proposta que foram implementados no protótipo construído.

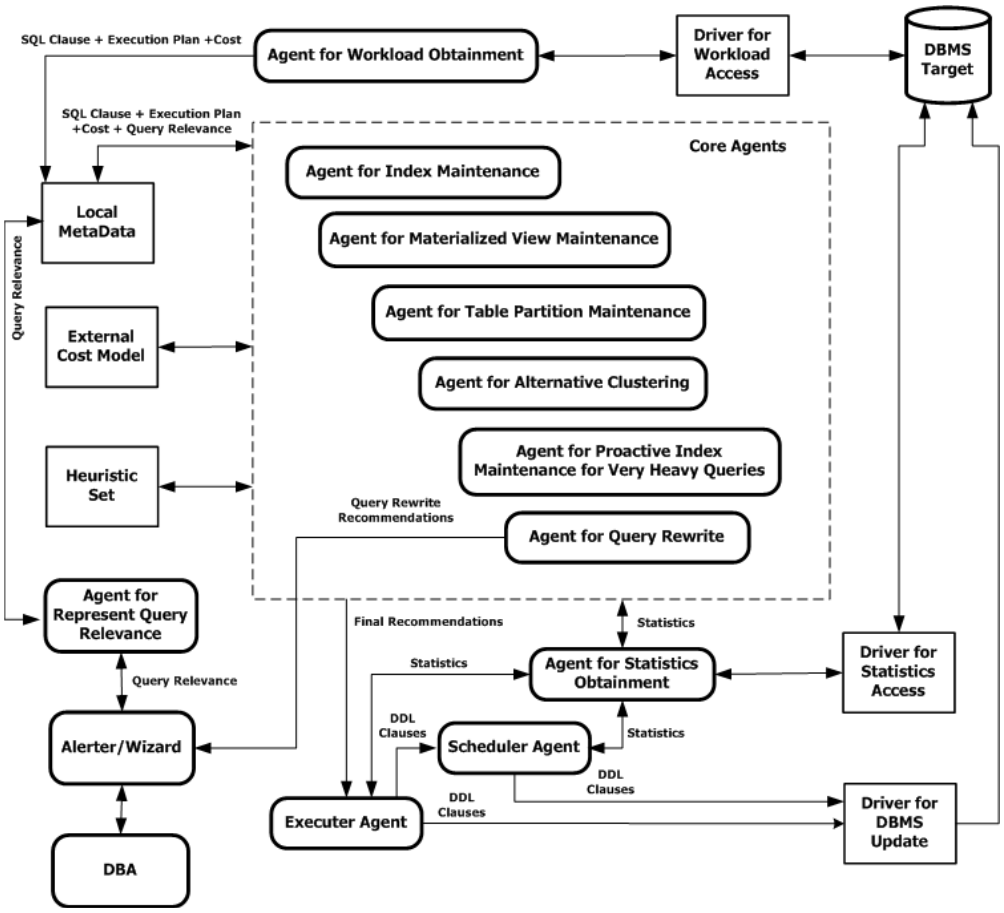

Figura 6.1: Arquitetura Não-Intrusiva para Manutenção Automática de Índices.

foram ou estão sendo executadas, juntamente com seus respectivos planos de execução e estimativas de custo. Em seguida, cada *tripla* <cláusula SQL, plano de execução, custo> capturada é armazenada na metabase local (*LM*).

– **Driver for Workload Access (DWA):** *Driver* que permite recuperar a carga de trabalho submetida ao SGBD alvo.

– **Local MetaData (LM):** A metabase local armazena a carga de trabalho capturada pelo agente AWO e um conjunto de informações acerca das estruturas de acesso gerenciadas pela arquitetura. Para cada estrutura de acesso, são armazenados: um identificador (*id*), o nome da tabela sobre a qual a estrutura é definida, o estado da estrutura (real ou hipotética), o custo estimado de criação da estrutura e o benefício acumulado da estrutura, ou seja, quanto a estrutura contribui (caso real) ou poderia contribuir (caso hipotético) para reduzir o custo da carga de trabalho como um todo. Dependendo do tipo de estrutura gerenciada (índice, visão materializada, etc), outras informações são armazenadas.

Por exemplo, para as estruturas de índice, são armazenados ainda: o tipo do índice (primário ou secundário), as colunas que compõem o índice, etc. Vale ressaltar ainda que uma estrutura hipotética, ao contrário das estruturas reais, existe apenas na metabase local.

- **External Cost Model (ECM):** Modelo de custos externo, ou seja, independente de SGBD. Este modelo de custos é utilizado pelas heurísticas definidas *HS* e possibilita estimar, por exemplo, o custo de uma busca seqüencial, o custo de uma seleção utilizando um índice primário, etc.
- **Agent for Statistics Obtainment (ASO):** Este agente acessa o SGBD alvo e recupera informações estatísticas, tais como o número de páginas ocupadas por uma tabela, a quantidade de *tuplas* de uma tabela, a altura de um índice, etc. Estas informações são requisitadas e utilizadas pelos agentes que compõem o núcleo da arquitetura (Core Agents).
- **Driver for Statistics Access (DSA):** *Driver* que permite recuperar as estatísticas do SGBD alvo.
- **Heuristic Set (HS):** Conjunto de heurísticas utilizadas pelos agentes que compõem o núcleo da arquitetura (*Core Agents*). Este conjunto possui uma heurística genérica (modelo), denominada heurística integrada para seleção e acompanhamento de estruturas de acesso (HISAE). Esta heurística é instanciada (especializada) para cada tipo diferente de estrutura de acesso que se deseja gerenciar. Por exemplo, para realizar a manutenção automática de índices, a heurística HISAE foi instanciada por meio da heurística HISAI (heurística integrada para seleção e acompanhamento de índices).
- **Agent for Index Maintenance (AIM):** Este agente consulta periodicamente a metabase local para verificar se novas *triplas* <cláusula SQL, plano de execução, custo> foram adicionadas. Em caso afirmativo, para cada nova *tripla* adicionada e ainda não analisada, o agente aplica uma heurística integrada para seleção e acompanhamento de índices (*HISAI*). Esta heurística determinará os melhores índices (reais ou hipotéticos) para cada *tripla* analisada e seus respectivos benefícios (ou seja, quanto cada índice contribui ou poderia contribuir para reduzir o custo da cláusula SQL, especificamente). A idéia básica da heurística *HISAI* consiste em alterar o plano de execução original da consulta recebida, usando os índices hipotéticos, na tentativa de obter um plano de execução mais eficiente do que o plano originalmente recebido.

  Em (Sha03), foi demonstrado por meio de testes práticos que a não manutenção periódica das estruturas de índices faz com que o desem-

penho seja degradado com o tempo. Neste sentido, o agente AIM é responsável por solucionar o problema da recriação automática de índices ("*reindex*"). Desta forma, este agente acompanha periodicamente o nível de fragmentação das estruturas de índices e, quando este nível ultrapassa um valor limítrofe, a operação de recriação é enviada ao agente executor (EA). Assim, não é necessário manter um índice fragmentado até o momento em que o agente de seleção final de índices decida eliminá-lo, por exemplo.

- **Executer Agent (EA):** Este agente é responsável pelas atualizações no esquema do SGBD alvo, isto é, pela criação, remoção ou reorganização das estruturas que compõem o projeto físico do banco de dados. O agente executor recebe uma recomendação dos agentes que compõem o núcleo da arquitetura (Core Agents) e executa esta recomendação no SGBD alvo.
- **Driver for DBMS Update (DDU):** *Driver* responsável por executar os comandos necessários para atualizar o projeto físico do SGBD alvo, como, por exemplo, a criação e remoção de índices, visões materializadas, etc.
- **Scheduler Agent (SA):**

  As ações de criação e reconstrução das estruturas de acesso consomem recursos do SGBD e, caso sejam executadas em momentos em que o nível de atividades do SGBD esteja elevado, podem até mesmo comprometer o desempenho do sistema. O agente escalonador, *Scheduler Agent*, é responsável por monitorar o nível de atividades do SGBD para que as ações de criação e reconstrução das estruturas de acesso tenham um impacto mínimo no desempenho do sistema. Quando o agente executor recebe uma recomendação do agente de seleção de índices (criação ou reconstrução de um índice), por exemplo, este verifica o nível de atividade do SGBD. Caso este nível esteja abaixo de um valor limite, o agente executa o comando recebido no banco de dados alvo; caso contrário, repassa o pedido (recomendação) e a responsabilidade pela execução do comando ao agente escalonador. O agente escalonador mantém uma lista de tarefas pendentes e verifica periodicamente o nível de atividade do SGBD; uma vez detectado que este nível esteja abaixo do valor limite, o agente passa a executar as tarefas armazenadas em sua lista. Contudo, o fato do agente monitorar periodicamente o nível de atividade do SGBD pode ocasionar um "*overhead*" desnecessário. Para evitar este problema, o agente escalonador pode armazenar um histórico do nível de atividades do SGBD e, com base neste histórico, prever qual o próximo momento

no futuro em que o nível de atividades do SGBD estará abaixo do valor limítrofe. Esta funcionalidade pode ser implementada utilizando-se de uma Rede Neural Artificial, por exemplo.

– **Agent for Represent Query Relevance (ARQR):**

Na prática, nem todas as consultas possuem a mesma prioridade. Consultas operacionais, recuperar o valor de um produto, por exemplo, admitem certa espera por parte do usuário (cliente ou vendedor). Por outro lado, consultas utilizadas para suportar o processo de tomada de decisão de alto nível necessitam ser executadas o mais rapidamente possível, uma vez que esperar, neste contexto, pode implicar prejuízos financeiros para a corporação. Neste sentido, o agente denominado agente para representação da importância das consultas ("*Agent for Represent Query Importance*") procura representar o grau de importância das consultas e utilizar este conceito durante o processo de sintonia automática. Uma vez que as consultas capturadas pelo "Agent for Workload Obtainment" são armazenadas em uma metabase local, o agente em questão pode, por intermédio de um "*wizard*", por exemplo, apresentar as consultas capturadas ao DBA e solicitar que este informe o grau de importância de cada uma delas. Este nível de importância pode ser representado por valores entre 1 e 10, por exemplo. Uma vez que a metabase foi enriquecida pelo DBA com o grau de importância de cada consulta, o agente para manutenção de índices (AIM), por exemplo, pode utilizar estes valores no processo de manutenção de índices, priorizando as estruturas de índices utilizadas por consultas com grau de importância mais elevado.

Vale ressaltar ainda que, mesmo que o DBA não informe o grau de importância das consultas capturadas, o restante dos componentes da arquitetura continuam funcionando normalmente (e de forma automática), uma vez que, neste caso, todas as consultas teriam a mesma prioridade. Assim, o agente funcionamento do agente ARQR não interfere no funcionamento dos demais agentes.

– **Agent for Materialized View Maintenance (AMVM):**

Conforme discutimos na Seção 2.4, uma visão materializada pode ser vista como um *cache* (área de armazenamento temporário), ou como uma cópia dos dados que pode ser acessada rapidamente. Desta forma, uma visão materializada oferece acesso rápido aos dados, sendo que esta velocidade pode ser crítica em aplicações nas quais a quantidade de consultas é alta e a complexidade das visões elevada.

Por outro lado, as alterações realizadas sobre os dados das tabelas base, sobre as quais uma visão materializada é definida, tornam a visão desatualizada. Para que a visão possa estar novamente sincronizada com os dados das tabelas base, será necessário recriar a visão a partir dos dados da origem, ou então atualizá-la de forma incremental (Quass96). Contudo, este processo de atualização (ou manutenção) das visões materializadas consome tempo e recursos computacionais, podendo inclusive degradar o desempenho do sistema. Assim, a escolha do conjunto de visões materializadas a serem utilizadas deve buscar aliar um bom desempenho com um baixo custo de manutenção. Todavia, este também é um problema NP-Difícil (Agra00). Logo, a manutenção de um conjunto de visões materializadas que seja sempre adequado é um dos principais problemas envolvidos no projeto físico de bancos de dados (Agra00, Monteiro06b).

Neste sentido, o agente (*Agent for Materialized View Maintenance (AMVM*) é o componente da arquitetura responsável por efetuar a manutenção automática das visões materializadas. Esta tarefa é realizada utilizando-se a heurística (*Heuristic for Materialized View Maintenance (HMVM)*), que compõe o conjunto das heurísticas da arquitetura *Heuristic Set*.

– **Agent for Table Partition Maintenance (ATPM):**

O particionamento de dados, conforme discutido na Seção 2.5, é um método que consiste em dividir fisicamente as grandes tabelas em diversos segmentos menores de dados, tornando o acesso aos dados mais rápido e seu gerenciamento mais fácil. Assim, o particionamento de grandes tabelas é uma das mais importantes atividades relacionadas ao projeto físico de bancos de dados. Contudo, a manutenção automática do particionamento de grandes tabelas ainda é um problema pouco estudado.

Neste contexto, o agente *ATPM* (*Agent for Table Partition View Maintenance*) é o componente da arquitetura responsável pela manutenção automática do particionamento de grandes tabelas. Para isso, este agente utiliza a heurística *HTPM*(*Heuristic for Table Partition Maintenance*), que compõe o conjunto das heurísticas da arquitetura *Heuristic Set*.

– **Agent for Alternative Clustering (AAC):**

A replicação de dados consiste em manter várias réplicas idênticas de uma relação com o objetivo de:

- **Balanceamento de carga:** no caso em que a maioria do acesso à relação r resulta em uma leitura da relação, vários clientes podem processar consultas envolvendo a relação r paralelamente.
- **Aumentar a disponibilidade dos dados:** no caso de falha em uma das cópias da relação r, o sistema pode continuar a processar consultas envolvendo a relação r mediante a utilização das outras cópias.

Contudo, a replicação também apresenta desvantagens, como o aumento do *overhead* de atualização. O sistema deve assegurar que todas as réplicas de uma relação r estejam consistentes. Assim, toda vez que r é atualizada, a atualização deve ser propagada para todas as demais cópias de r.

Agora suponha que tenhamos as seguintes consultas:

- Consulta 1:

```
SELECT l_orderkey
FROM lineitem
WHERE l_orderkey >= 999 and l_orderkey <= 99999
```

- Consulta 2:

```
SELECT l_partkey
FROM lineitem
WHERE l_partkey >= 999 and l_partkey <= 99999
```

Observe que, para otimizar o desempenho da consulta 1, deveríamos criar um índice primário sobre o atributo l_orderkey da tabela lineitem. Já para otimizar o desempenho da consulta 2, deveríamos criar um índice primário sobre o atributo l_partkey da tabela lineitem. Contudo, uma mesma tabela não pode estar fisicamente ordenada por dois atributos diferentes. Usar uma replicação "convencional" (criar uma cópia exata de lineitem) também não ajudaria a melhorar o desempenho das duas consultas. Neste contexto, o agente de replicação de dados pode optar por criar "réplicas" da tabela lineitem, só que através da utilização de visões materializadas. Neste caso, o agente criará duas visões materializadas, uma ordenada por l_orderkey e outra ordenada por l_partkey. Desta forma, ao receber a consulta 1, o otimizador irá optar por utilizar a primeira visão materializada, já quando receber a consulta 2, o otimizador utilizará

a segunda visão materializada. Desta forma, consegue-se melhorar o desempenho das duas consultas, simultaneamente.

A tarefa de manter um conjunto de réplicas, mediante a utilização de visões materializadas, é responsabilidade do agente *ADR* (*Agent for Alternative Clustering (AAC)*), que, para realizar esta tarefa, utiliza a heurística *HAC* (*Heuristic for Alternative Clustering (HAC)*), a qual compõe o conjunto das heurísticas da arquitetura (*Heuristic Set*).

Neste sentido, o agente *AAC* consulta periodicamente[2] a metabase local para verificar se novas tarefas (*triplas* <cláusula SQL, plano de execução, custo>) foram adicionadas. Em caso afirmativo, para cada nova *tripla* adicionada e ainda não analisada, o agente aplica uma heurística *HAC*. Esta heurística determinará as melhores alternativas de *clusterização* para cada *tripla* analisada e seus respectivos benefícios (ou seja, quanto cada alternativa de *clusterização* contribui ou poderia contribuir para reduzir o custo da cláusula SQL, especificamente).

– **Agent for Proactive Index Maintenance for Very Heavy Queries (APIM):**

Determinadas consultas, como consultas em aplicações OLAP, por exemplo, apresentam tempo de execução bastante grande, ou seja, são consultas muito demoradas ("pesadas" ou de custo elevado). Tais consultas são de tal modo demoradas que a criação de estruturas de índices adequadas trariam ganhos de desempenho mesmo que a consulta fosse executada uma única vez, e que o índice fosse excluído logo após sua execução. Em outras palavras, o benefício de se materializar a estrutura de índices é maior do que seu custo de criação em apenas uma execução da consulta. Neste contexto, seria interessante ter um histórico das execuções das consultas que possuem esse perfil e, com base nesse histórico, prever o momento em que uma uma dessas consultas seria executada novamente, com o objetivo de se antecipar a este evento e criar, de forma automática e pró-ativa, as estruturas de índices adequadas para acelerar a execução da referida consulta. O agente denominado "*Agent for Proactive Index Maintenance for Very Heavy Queries (APIM)*" é responsável por esta tarefa. Após a execução da consulta, o índice seria automaticamente removido. Dentre as técnicas que poderiam ser utilizadas para se prever o momento em que uma consulta "pesada" seria executada, poderíamos destacar as Redes Neurais Artificiais. A técnica escolhida irá compor o

[2]Este periodicidade pode ser parametrizada pelo DBA.

núcleo da heurística utilizada pelo agente (*Heuristic for Proactive Index Maintenance for Very Heavy Queries (HPIM)*).

– **Agent for Query Rewrite (AQR):**

Uma vez que as consultas capturadas pelo agente AWO ("Agent for Workload Obtainment") são armazenadas em uma metabase local. O agente para reescrita de consultas ("*Agent for Query Rewrite (AQR)*") pode analisar as consultas armazenadas e sugerir ao DBA (por meio de alertas, *wizards* ou relatórios) oportunidades de reescrita que possam trazer ganhos de desempenho. Neste sentido, suponha que a consulta a seguir foi capturada e armazenada na metabase local:

```
SELECT l_orderkey
FROM lineitem
WHERE l_orderkey != 999
```

O agente AQR poderia, por meio de uma base de regras, sugerir a reescrita da consulta para:

```
SELECT l_orderkey
FROM lineitem
WHERE l_orderkey NOT IN (999)
```


6.1 Resumo do Capítulo

O processo de sintonia automática utilizado neste trabalho é baseado num ciclo de controle de retorno (*feedback*) que refina, gradualmente, as decisões de sintonia com o conhecimento local de métricas fornecidas pelos componentes do banco de dados. Este trabalho defende que, para evitar problemas de desempenho, existem duas opções: a primeira consiste em executar as heurísticas em momentos de inatividade ou de atividade baixa. A segunda opção é limitar o espaço de buscas cobertos pelas heurísticas, para manter sob controle seu tempo de execução e uso dos recursos disponíveis.

Este capitulo apresentou uma arquitetura não-intrusiva para a manutenção automática do projeto físico de bancos de dados. A automação de diversas atividades relacionadas ao projeto físico foram discutidas, como, por exemplo, a manutenção automática de índices, visões materializadas, particionamentos de tabelas, etc. Além disso, discutiu-se como a arquitetura proposta pode tratar alguns problemas relevantes e ainda não abordados na

literatura, como a manutenção pró-ativa de índices para consultas pesadas, por exemplo. A utilização da arquitetura como base para a construção de um *wizard* que indique ao DBA a possibilidade de obter ganhos de desempenho por meio da reescrita de consultas também foi comentada.

No próximo capítulo serão apresentados os resultados experimentais obtidos a partir dos testes de desempenho realizados. Estes testes comprovam a eficiência da abordagem proposta e de sua instanciação para os problemas da manutenção automática de índices e da manutenção automática de *clusters* de dados obtidos a partir da utilização de visões materializadas.