Referências Bibliográficas

[Agra00] AGRAWAL, S.; CHAUDHURI, S. ; NARASAYYA, V.. **Automated selection of materialized views and indexes for sql databases**. In: IN PROCEEDINGS OF VLDB, p. 496–505, 2000. 1.1, 2.4, 3.3, 6, 1

[Agra04] AGRAWAL, S.; CHAUDHURI, S.; L., K.; A., M.; NARASAYYA, V. ; SYAMALA, M.. **Database tuning advisor for microsoft sql server 2005**. In: IN PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), 2004. 3.3

[Agra06] AGRAWAL, S.; CHU, E. ; NARASAYYA, V.. **Automated physical design tuning: Workload as a sequence**. In: PROCEEDINGS OF THE ACM SIGMOD, 2006. 3.3, 7.3.3

[AutoAdmin] GROUP, M. D.. **Autoadmin project**. 2007. Disponível em http://research.microsoft.com/dmx/autoadmin/. 3.3

[Bouzeghoub00] BOUZEGHOUB, M.; KEDAD, Z.. **A quality-based framework for physical data warehouse design**. In: IN PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON DESIGN AND MANAGEMENT OF DATA WAREHOUSES (DMDW00), 2000. 2.4

[Brown94] BROWN, K. P.; MEHTA, M. ; CAREY, M. J.. **Towards automated performance tuning for complex workload**. In: IN PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), 1994. 4.5.1

[Bruno05] BRUNO, N.; CHAUDHURI, S.. **Automatic physical database tuning: a relaxation-based approach**. In: PROCEEDINGS OF THE 2005 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, p. 227–238, New York, NY, USA, 2005. ACM. 1.1, 3.3, 8

[Bruno06a] BRUNO, N.; CHAUDHURI, S.. **Physical design refinement: The merge-reduce approach**. In: PROCEEDINGS OF THE EDBT (EDBT06), 2006. 3.3

[Bruno06b] BRUNO, N.; CHAUDHURI, S.. **To tune or not to tune? a lightweight physical design alerter**. In: PROCEEDINGS OF THE 32RD INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB06), 2006. 3.3

[Bruno07a] BRUNO, N.; CHAUDHURI, S.. **An online approach to physical design tuning**. In: PROCEEDINGS OF THE ICDE CONFERENCE (ICDE07), 2007. 1.1, 3.3, 3.3.6, 3.3.6, 7.4, 8

[Bruno07b] BRUNO, N.; CHAUDHURI, S.. **Online autoadmin: (physical design tuning)**. In: PROCEEDINGS OF THE 2007 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA (SIGMOD07), p. 1067–1069. ACM, 2007. 3.3, 3.3.6, 3.3.6

[Burleson04] BURLESON, D.. **Creating a Self-Tuning Oracle Database**. Rampant, 2004. 3.3

[Ceri91] CERI, S.; WIDOM, J.. **Deriving production rules for incremental view maintenance**. In: PROCEEDINGS OF THE 17TH CONFERENCE ON VERY LARGE DATABASES, MORGAN KAUFMAN PUBS. (LOS ALTOS CA), BARCELONA, 1991. 2.4

[Cha97] CHAUDHURI, S.; NARASAYYA, V.. **An efficient, cost-driven index selection tool for microsoft sql server**. In: IN PROCEEDINGS OF VLDB, p. 146–155, 1997. 3.3

[Cha98a] CHAUDHURI, S.; NARASAYYA, V.. **Autoadmin "what-if" index analysis utility**. In: IN PROCEEDINGS OF THE ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, p. 367–377, 1998. 3.3, 3.3.1

[Cha98b] CHAUDHURI, S.; NARASAYYA, V.. **Microsoft index tuning wizard for sql server 7.0**. In: PROCEEDINGS OF THE ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, p. 553–554, 1998. 3.3

[Cha04] CHAUDHURI, S.; DATAR, M. ; NARASAYYA, V.. **Index selection for databases: A hardness study and a principled heuristic solution**. IEEE Transactions on Knowledge and Data Engineering, 16(11):1313–1323, 2004. 1.1, 3.3, 3.3.1, 3.3.2, 4.2.2, 5.1.6

[Cha07a] CHAUDHURI, S.; NARASAYYA, V.. **Self-tuning database systems: A decade of progress**. In: PROCEEDINGS OF THE 33RD INTER-

NATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB07), 2007. 1.1, 3.3

[Cos02] COSTA, R. L. C.; LIFSCHITZ, S.. **Index self-tuning and agent-based databases**. In: IN PROCEEDINGS OF THE LATIN-AMERICAN CONFERENCE ON INFORMATICS (CLEI), 2002. 3.3.1, 4.2.2

[Cos05] COSTA, R. L. C.; LIFSCHITZ, S.; NORONHA, M. ; SALLES, M. V.. **Implementation of an agent architecture for automated index tuning**. In: PROCEEDINGS OF THE ICDE WORKSHOPS, 2005. 1.1, 3.1, 3.3, 3.3.1, 3.3.2, 3.3.5, 4.5.2, 5.1.2, 2

[Dag04] DAGEVILLE, B.; DAS, D.; DIAS, K.; YAGOUB, K.; ZAIT, M. ; ZIAUDDIN, M.. **Automatic sql tuning in oracle 10g**. In: IN PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), p. 1098–1109, 2004. 3.2

[Dageville06] DAGEVILLE, B.; DAS, D.; K., D.; YAGOUB, K.; ZAIT, M. ; ZIAUDDIN, M.. **Oracle's self-tuning architecture and solutions**. IEE Computer Society: Bulletin of Technical Commitee on Data Engineering, 29(3):24–31, 2006. 3.3

[Dias05] DIAS, K.; RAMACHER, M.; SHAFT, U.; VENKATARAMANI, V. ; WOOD, G.. **Automatic performance diagnosis and tuning in oracle**. In: PROCEEDINGS OF THE CIDR CONFERENCE, p. 84–94, 2005. 1.1, 3.3

[Graefe00] GRAEFE, G.. **Dynamic query evaluation plans: Some course corrections?** IEEE Data Eng. Bull., 23(2):3–6, 2000. 2.8, 3.3.5, 5.1.1

[Gupta93] GUPTA, A.; MUMICK, S. ; SUBRAHMANIAN, V.. **Maintaining views incrementally**. In: PROCEEDINGS OF THE 1993 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA (SIGMOD93), p. 157–166, New York, NY, USA, 1993. ACM. 2.4

[Gupta95] GUPTA, A.; MUMICK, S.. **Maintenance of materialized views: Problems, techniques and applications**. IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and Data Warehousing, 18(2):3–18, 1995. 2.4

[Harrison92] HARRISON, J.; DIETRICH, S.. **Maintenance of materialized views in a deductive database: An update propagation approach**. In: DEDUCTIVE DATABASE WORKSHOPS, 1992. 2.4

[Horn01] P., H.. **Autonomic computing: Ibm's perspective on the state of information technology**. In: INTERNATIONAL BUSINESS MACHINES, Armonk, NY, 2001. 3

[Idreos07a] IDREOS, S.; KERSTEN, M. L. ; MANEGOLD, S.. **Updating a cracked database**. In: PROCEEDINGS OF THE 2007 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA (SIGMOD07), p. 413–424, New York, NY, USA, 2007. ACM. 3.3, 3.3.5

[Idreos07b] IDREOS, S.; KERSTEN, M. L. ; MANEGOLD, S.. **Database cracking**. In: PROCEEDINGS OF THE THIRD BIENNIAL CONFERENCE ON INNOVATIVE DATA SYSTEMS RESEARCH (CIDR07), p. 68–78, 2007. 3.3, 3.3.5

[Kai04] SATTLER, K.-U.; SCHALLEHN, E. ; GEIST, I.. **Autonomous query-driven index tuning**. In: IDEAS '04: PROCEEDINGS OF THE INTERNATIONAL DATABASE ENGINEERING AND APPLICATIONS SYMPOSIUM (IDEAS'04), p. 439–448, Washington, DC, USA, 2004. IEEE Computer Society. 1.1, 3.3, 3.3.3, 3.3.3, 3.3.5, 4.5.2, 5.1.5, 7.4, 4

[Kalnis02] KALNIS, P.; MAMOULIS, N. ; PAPADIAS, D.. **View selection using randomized search**. Data Knowledge Engineering, 42(1):89–101, 2002. 2.4

[Konig06] KONIG, A. C.; NABAR, S. U.. **Scalable exploration of physical database design**. In: PROCEEDINGS OF THE 22ND INTERNATIONAL CONFERENCE ON DATA ENGINEERING (ICDE'06), p. 37, Washington, DC, USA, 2006. IEEE Computer Society. 3.3

[Korth06] KORTH, H.; SILBERSCHATZ, A.. **Sistema de Banco de Dados**. Campus, 2006. D

[Lif04a] LIFSCHITZ, S.; MILANES, A. Y. M. ; SALLES, M. V.. **State of the art in self-tuning relational database systems (in portuguese)**. Technical report, Departamento de Informática,PUC-Rio, 2004. 2.6, 2.7, 3.1

[Lif06] LIFSCHITZ, S.; MORELLI, E. T.. **Towards autonomic index maintenance**. In: PROCEEDINGS OF THE BRAZILIAN SYMPOSIUM ON DATABASE, 2006. 3.3, 3.3.5, 4.5.2, 5.1.2, 2

[Lightstone02] LIGHTSTONE, S. S.; LOHMAN, G. ; ZILIO, D.. **Toward autonomic computing with db2 universal database**. SIGMOD Rec., 31(3):55–61, 2002. 3.3

[Loh00] LOHMAN, G.; VALENTIN, G.; ZILIO, D.; ZULIANI, M. ; SKELLEY, A.. **Db2 advisor: An optimizer smart enough to recommend its own indexes**. In: IN PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING (ICDE), p. 101–110, 2000. 2.8, 3.3, 3.3.1, 3.3.1, 1, 3.3.5, 5.1.1, 5.1.2, 2

[Loh02] G., L.; LIGHTSTONE, S.. **Smart: Making db2 (more) autonomic**. In: IN PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), 2002. 3.3

[Luh07] LUHRING, M.; SATTLER, K.; SCHMIDT, K. ; SCHALLEHN, E.. **Autonomous management of soft indexes**. In: PROCEEDINGS OF THE 23RD IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING WORKSHOP (ICDE'07), 2007. 1.1, 3.3, 3.3.5, 3.3.5, 3.3.5, 4.2.2, 4.3.2, 4.4, 4.5.2, 5.1.1, 5.1.2, 5.1.3, 5.1.5, 5.2.3, 7.4, 8, 4, 2

[Marques00] MARQUES, F. P.. **O Problema da Mochila Compartimentada**. PhD thesis, Instituto de Ciências Matemáticas e de Computação (ICMC), USP, 2000. 2.8

[Martin02] MARTIN, P.; POWLEY, W.; LI, H. ; ROMANUFA, K.. **Managing database server performance to meet qos requirements in eletronic commerce systems**. In: INTERNATIONAL JOURNAL ON DIGITAL LIBRARIES, 2002. 4.5.1

[Mendes06] MENDES, M. R. N.; MACIEL, P. R. M.. **Análise de desempenho de sistemas oltp utilizando o benchmark tpc-c**. Technical report, Centro de Informática, UFPE, 2006. D, D.2.1

[Milanes04] MILANÉS, A.. **Uma arquitetura para auto-sintonia global de SGBDs usando agentes**. PhD thesis, Department of Informatics, PUC-Rio, 2004. 3.2, 3.2

[Milanes05] MILANÉS, A.; LIFSCHITZ, S.. **Design and implementation of a global self-tuning architecture**. In: PROCEEDINGS OF THE BRAZILIAN SYMPOSIUM ON DATABASE, 2005. 3.2

[Monteiro06a] MONTEIRO, J. M.; LIFSCHITZ, S. ; BRAYNER, A.. **An architecture for automated index tuning**. In: IN PH.D. AND M.S. WORKSHOP OF THE BRAZILIAN SYMPOSIUM ON DATABASE, 2006. 1.1, 2.6

[Monteiro06b] MONTEIRO, J. M.; LIFSCHITZ, S. ; BRAYNER, A.. **Automated selection of materialized views**. Technical report, Departamento de Informática, PUC-Rio, 2006. 2.4, 6, 1

[Monteiro07b] MONTEIRO, J. M.; LIFSCHITZ, S. ; BRAYNER, A.. **Estado da arte em auto-sintonia do projeto físico de banco de dados**. Technical report, Departamento de Informática, PUC-Rio, 2007. 3.3.7

[Monteiro07a] MONTEIRO, J. M.; LIFSCHITZ, S. ; BRAYNER, A.. **Extraindo metadados para a captura da carga de trabalho e planos de execução de sgbds**. Technical report, Departamento de Informática, PUC-Rio, 2007. 4.6

[Monteiro08a] MONTEIRO, J. M.; LIFSCHITZ, S. ; BRAYNER, A.. **Uma ferramenta não-intrusiva para a manutenção automática de Índices**. In: IN V SESSãO DE DEMOS DO XXIII SIMPóSIO BRASILEIRO DE BANCO DE DADOS (SBBD), 2008. 5, 7, 7.3.1, 8.1.6

[Morelli06a] MORELLI, E. M. T.. **Recriação Automática de Índices em um SGBD Relacional (in portuguese)**. PhD thesis, Department of Informatics, PUC-Rio, 2006. 1.1, 2.2, 2.2, 3.2, 3.3, 3.3.1, 3.3.2, 3.3.2, 3.3.5, 4.2.2, 4.5.2, 5.1.2, 5.1.2, 5.1.2, 5.1.2, 5.1.6, 7, 7.1, 7.2, 7.3, 7.3.3, 7.4, 2, 14, 15, 19, E, E

[Morelli06b] MORELLI, E. T.; LIFSCHITZ, S.. **Estudo dos malefícios gerados pela fragmentação de Índices em sistemas de bancos de dados relacionais**. Technical report, Departamento de Informática, PUC-Rio, 2006. 2.2, 2.2

[Navathe05] ELMASRI, R.; NAVATHE, S. B.. **Sistemas de Banco de Dados: Fundamentos e Aplicações**. Addison Wesley, 2005. 2.1, 2.5, 2.8

[OSDL] OSDL. **Osdl**. 2007. Disponível em http://osdldbt.sourceforge.net/. 3.3.1, 7

[Papa06] PAPADOMANOLAKIS, S.; AILAMAKI, A.. **Autopart: automating schema design for large scientific databases using data partitioning**. In: PROCEEDINGS OF THE 16TH INTERNATIONAL CONFERENCE ON SCIENTIFIC AND STATISTICAL DATABASE MANAGEMENT. IEEE, 2004. 1.1, 2.5

[Papa07] PAPADOMANOLAKIS, S.; AILAMAKI, A.. **An integer linear programming approach to database design**. IEEE International Conference on Data Engineering, p. 442–449, 2007. 3.3

[Paton07] PATON, N.. **Automation everywhere: Autonomics and data management**. In: BRITISH NATIONAL CONFERENCE ON DATABASES (BNCOD), p. 3–12, 2007. 7

[PostgreSQL] GROUP, P. G. D.. **Postgresql**. 2007. Disponível em http://www.postgresql.org. 3.3

[Quass96] QUASS, D.; GUPTA, A.; MUMICK, S. ; WIDOM, J.. **Making views selfmaintainable for data warehousing**. In: PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED INFORMATION SYSTEMS, 1996. 2.4, 6, 1

[Ramakrishnan02] RAMAKRISHNAN, R.. **Database Management Systems**. McGraw-Hill, 2002. 2.1, 2.5, A

[Ramalho02] RAMALHO, J. A.. **Oracle 9i**. Berkeley Brasil, 2002. 2.6

[Sal04] SALLES, M. V.. **Autonomic index creation in databases (in portuguese)**. PhD thesis, Department of Informatics, PUC-Rio, 2004. 1.1, 2.1, 2.2, 2.2, 2.2, 2.6, 3.1, 3.3, 3.3.1, 3.3.1, 3.3.2, 3.3.2, 3.3.5, 4.2.2, 4.5.2, 5.1.2, 5.1.6, 7.3, 7.4, 8, 2

[Sal05] SALLES, M. V.; LIFSCHITZ, S.. **Autonomic index management**. In: IN PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING (ICAC), 2005. 1.1, 3.3, 3.3.1, 3.3.2, 3.3.5, 4.5.2, 5.1.2, 5.1.3, 2

[Sattler03] SATTLER, K.; GEIST, I. ; SCHALLEHN, E.. **Quiet: Continuous query-driven index tuning**. In: PROCEEDINGS OF THE 29TH INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES (VLDB03), p. 1129–1132. VLDB Endowment, 2003. 3.3, 3.3.3

[Sattler05] SATTLER, K.; SCHALLEHN, E. ; GEIST, I.. **Towards indexing schemes for self-tuning dbms**. In: PROCEEDINGS OF THE 21ST INTERNATIONAL CONFERENCE ON DATA ENGINEERING WORKSHOPS (ICDEW05), p. 1216, Washington, DC, USA, 2005. IEEE Computer Society. 7

[Sch06] SCHNAITTER, K.; ABITEBOUL, S.; MILO, T. ; POLYZOTIS, N.. **Colt: continuous on-line tuning**. In: PROCEEDINGS OF THE ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA (SIGMOD06), p. 793–795, New York, NY, USA, 2006. ACM. 1.1, 3.3, 3.3.4, 3.3.4, 7.4

[Sch07] SCHNAITTER, K.; ABITEBOUL, S.; MILO, T. ; POLYZOTIS, N.. **On-line index selection for shifting workloads**. In: 2ND INTERNATIONAL WORKSHOP ON SELF-MANAGING DATABASE SYSTEMS (SMDB 2007), p. 459–468, 2007. 3.3.4

[Sha03] SHASHA, D.; BONNET, P.. **Database Tuning: Principles, Experiments and Troubleshooting Techniques**. Morgan Kaufmann, 2003. 1.1, 3.3, 6

[TPC] COUNCIL, T. P. P.. **Tpc**. 2007. Disponível em http://www.tpc.org. 3.3.1, D, E

[TPC-H] COUNCIL, T. P. P.. **Tpc-h**. 2007. Disponível em http://www.tpc.org/tpch. 7

[Valluri02] VALLURI, S.; VADAPALLI, S. KARLAPALEM, K.. **View relevance driven materialized view selection**. In: PROCEEDINGS OF THE AUSTRALASIAN DATABASE CONFERENCE (ADC02), 2002. 2.4

[Vieira05] VIEIRA, M.; DURÃES, J. ; MADEIRA, H.. **Especificação e validação de benchmarks de confiabilidade para sistemas transacionais**. In: IEEE LATIN AMERICA TRANSACTIONS, 2005. 2.9

[Weikum94] WEIKUM, G.; HASSE, C. MONKEBERG, A. ; ZABBACK, P.. **The COMFORT automatic tuning project, invited project review**. Information Systems, 19(5):381–432, 1994. 1, 3.3.5, 4.1

[Weikum02] WEIKUM, G.; MÖNKEBERG, A.; HASSE, C. ; ZABBACK, P.. **Self-tuning database technology and information services: from wishful thinking to viable engineering**. In: IN PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), p. 20–31, 2002. 3.1

[Yourdon90] YOURDON, E.; CONSTANTINE, L. L.. **Projeto Estruturado de Sistemas**. Campus, 1990. 3.1

[Zhang01] ZHANG, C.; YAO, X. ; YANG, J.. **An evolutionary approach to materialized view selection in a data warehouse environment**. IEEE Transactions on Systems, Man, and Cybernetics, 31(3):282–294, 2001. 2.4

[Zilio04] ZILIO, D.; RAO, J.; LIGHTSTONE, S.; LOHMAN, G.; STORM, A.; GARCIA-ARELLANO, C. ; FADDEN, S.. **Db2 design advisor: Integrated automatic physical database design**. In: IN PROCEEDINGS

OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), p. 1087–1097, 2004. 1.1, 3.3, 8

[kendall99] KENDALL, E.; KRISHNA, P.; PATHAK, C. ; SURESH, C.. **A Framework for Agent Systems. In Implementing Application Frameworks - Object-Oriented Frameworks at Work, M. Fayad et al. (eds.)**. John Wiley & Sons, 1999. 1

A
O Modelo de Custos Utilizado

Este apêndice apresenta o modelo de custos concebido para estimar o custo (em termos do tempo de execução) das principais operações da álgebra relacional, ou seja, das operações utilizadas em um plano de execução. Logicamente, uma vez que o modelo permite estimar o custo de operações individuais, será possível também estimar o custo completo de um plano de execução. O custo de execução de uma determinada operação da álgebra relacional envolve diversos componentes, como por exemplo: custo de entrada e saída (E/S), custo de CPU, dentre outros. Contudo, é razoável esperar que o custo de entrada e saída domine o custo total. Esta consideração é justificada pela evolução atual dos dispositivos de *hardware*, onde a velocidade das CPU's cresce rapidamente, enquanto a velocidade dos dispositivos de armazenamentos (discos rígidos, por exemplo) não evolui de maneira similar (Ramakrishnan02).

Desta forma, o número de páginas transferidas do disco para a memória principal (custo de E/S) será utilizado como métrica no modelo de custos definido neste apêndice e utilizado no restante da tese.

Vale ressaltar que sistemas comerciais devem considerar outros componentes do custo, tais como o custo de CPU, custo de transmissão (em bancos de dados distribuídos), etc. Entretanto, neste trabalho, o modelo de custos tem como principal objetivo ilustrar e quantificar o benefício das heurísticas e algoritmos propostos, bem como estimar o custo dos planos de execução manipulados por estas heurísticas. Assim, por simplicidade, decidimos concentrar os cálculos somente no custo de E/S. Dado que o custo de E/S é, na grande maioria das vezes, o componente dominante do custo das operações da álgebra, a métrica escolhida (custo de E/S) consiste em uma boa aproximação do custo real (Ramakrishnan02).

Por outro lado, um modelo preciso seria demasiadamente complexo para o propósito assumido neste trabalho: ilustrar as idéias, heurísticas e a abordagem não-intrusiva proposta na tese, além de permitir estimar os custos dos planos de execução manipulados pela abordagem proposta, possibilitando comparar e avaliar estes planos. Este fato justifica a decisão de utilizar um modelo simplificado, o qual simplesmente se baseia na contagem das páginas que são

lidas ou escritas em disco como medida do custo de E/S (Ramakrishnan02). Assim, o modelo de custos definido neste apêndice é utilizado em toda esta tese. A seguir, serão apresentadas as estimativas de custo utilizadas pelas heurísticas descritas anteriormente.

A.1 Estimativas de Custo

Nesta tese, o cálculo dos benefícios das estruturas de acesso, bem como dos custos das operações e planos de consulta, são obtidos a partir de estimativas. Esta seção detalha o cálculo das estimativas utilizadas.

1. T_p**:**

   indica o tamanho (em *bytes*) de uma página (bloco). No PostgreSQL, por exemplo, o valor tipico para T_p é 8192 *bytes* (8K).

2. N_r**:**

   representa cardinalidade da relação r, ou seja, o número de *tuplas* da relação r.

3. P_r**:**

   representa o número de páginas contendo as *tuplas* da relação r.

4. HT_i**:**

   indica o número de níveis da estrutura de índice i. Considerando que o índice utiliza uma estrutura de árvore B^+, HT_i corresponde à altura da árvore.

5. P_K**:**

   tamanho em blocos de um determinado índice K, ou seja, número de páginas contendo as informações do índice K. Estas páginas são denominadas páginas de índices (ou blocos de índices).

6. S_r**:**

   indica o tamanho (em *bytes*) das *tuplas* da relação r, ou seja, o tamanho (em *bytes*) de uma *tupla* da relação r.

7. F_r**:**

   fator de página. Indica o número de *tuplas* da relação r por página, ou seja, o número de *tuplas* da relação r que podem ser armazenados em uma página de dados. Logo, podemos obter o valor de F_r fazendo

$F_r = \lfloor T_p/S_r \rfloor$. Outra forma de se obter o valor de F_r é dividindo-se N_r por P_r. Assim, $F_r = \lfloor N_r/P_r \rfloor$.

8. $FS(A, r)$**:**

   indica a média do número de *tuplas* que satisfazem uma condição de igualdade sobre o atributo A. Logo, se A é uma chave, $FS(A, r) = 1$.

9. EC_{FS}**:**

   indica a estimativa de custo para uma operação de *File Scan*, (ou *Full Scan*, ou ainda *Seq Scan*), ou seja, de uma busca linear. Esta operação é utilizada para localizar e recuperar as *tuplas* que satisfazem uma condição de seleção. Para isso, a tabela é lida seqüencialmente e todas as *tuplas* são testadas. Assim, esta operação não pressupõe ordenação (*sort*) ou existência de índices. Desta forma, podemos assumir que $EC_{FS} = P_r$.

10. $EC_{S_{IP}}$**:**

    indica a estimativa de custo para uma operação de seleção através de um índice primário (*clustering*). Se o atributo definido como chave de busca do índice é a própria chave primária da relação, então: $EC_{S_{IP}} = HT_i + 1$. Caso o atributo definido como chave de busca do índice não seja a chave primária, então: $EC_{S_{IP}} = HT_i + \lceil FS(A, r)/F_r \rceil$.

11. $EC_{S_{IS}}$**:**

    indica a estimativa de custo para uma operação de seleção através de um índice secundário (*not clustering*). Se o atributo definido como chave de busca do índice é a própria chave primária da relação, então: $EC_{S_{IP}} = HT_i + 1$. Caso o atributo definido como chave de busca do índice não seja a chave primária, então: $EC_{S_{IS}} = HT_i + FS(A, r)$.

12. B_{k,q_s}**:**

    o benefício de uma determinada estrutura de acesso (índice, visão materializada, etc) k para um determinado comando SQL q_s. Este benefício é determinado através da aplicação de heurísticas e seu cálculo depende do tipo da estrutura e do seu estado (real ou hipotético).

13. BA_k**:**

    o benefício acumulado de uma determinada estrutura de acesso (índice, visão materializada, etc) k para todos os comandos SQL já processados. O benefício acumulado de uma determinada estrutura corresponde à soma dos seus benefícios para cada um dos comandos SQL anteriormente executados. Assim:

$$BA_k = \sum_{j=1}^{NQ_{Wt}} B_{k,q_j}$$

Onde, NQ_{Wt} é número total de cláusulas SQL da carga de trabalho.

14. EC_{C_K}**:**

    indica a estimativa do custo de criação de uma determinada estrutura de acesso (índice, visão materializada, etc) k. Normalmente, este custo não é disponibilizado na metabase dos SGBDs e deve ser estimado. Por exemplo, em (Morelli06a) encontra-se a seguinte estimativa para se calcular o custo de criação de uma estrutura de índice:

    $CC_I = 2P + cR \log R$

    Em (Morelli06a), supõe-se uma política de criação de índices em que todas as páginas da tabela são lidas, ordenadas e então o índice é criado de uma forma *bottom-up*. O primeiro termo da fórmula leva em conta o custo de E/S de ler todas as páginas da tabela e de escrever todas as páginas do índice (para índices hipotéticos, a mesma quantidade estimada). Já o segundo termo estima o custo de ordenar todas as linhas da tabela em memória.

15. EC_{A_K}**:**

    indica a estimativa do custo de atualização de uma determinada estrutura de acesso (índice, visão materializada, etc) k durante o processamento de um comando de atualização. Normalmente, este custo não é disponibilizado na metabase dos SGBDs e deve ser estimado. Em (Morelli06a), encontra-se a seguinte estimativa para o custo de atualização de uma estrutura de índice:

    $EC_{A_K} = 2\lceil \frac{r}{R} \rceil P + cr$

    Na fórmula, r é o número de registros atualizados pelo comando, R é o número de registros da tabela, P é o número de páginas da tabela e c é um coeficiente que relaciona, percentualmente, o custo de uma operação de E/S com o custo de CPU para processar um registro que esteja em memória. No PostgreSQL, cada E/S tem custo estimado igual a um e o coeficiente c tem um valor padrão de 1%.

    O primeiro termo da fórmula calcula o custo de E/S da atualização. Estima-se que serão atualizadas uma quantidade de páginas de índice

proporcional à fração de registros da tabela que foram atualizados multiplicada pelo tamanho da tabela em páginas. Lembramos que, para índices hipotéticos, estima-se que o tamanho em páginas do índice é idêntico ao tamanho em páginas da tabela. Multiplica-se o resultado por dois pois, no pior caso, precisa-se ler e escrever cada página atualizada. Além do custo de E/S, ainda somamos o custo estimado de CPU de processar o número de registros que foram atualizados pelo comando.

16. **Razão de Fragmentação (R):**

    R = número de *tuplas* de uma tabela r / quantidade de blocos de índice da tabela r

    ou seja,

    R = N_r / P_k

17. R_i**:**

    Razão de fragmentação obtida logo após a criação da tabela, quando não existe fragmentação.

18. R_a**:**

    Razão de fragmentação obtida num instante de tempo T_a.

19. **Grau de Fragmentação:** A medida do grau de fragmentação de um índice poderia ser obtida comparando-se a razão de fragmentação em dois momentos: logo após a criação R_i e no momento em que deseja-se verificar se vale a pena aplicar uma reconstrução, ou simplesmente seu descarte R_a.

    (Morelli06a) propõe a seguinte fórmula para cálculo do grau de fragmentação de um índice:

    $G_rF = 100 - [(R_a/R_i) \times 100]$

20. **V:**

    número de varreduras nas quais o índice participou (foi utilizado) desde sua criação.

Vale ressaltar que durante a definição das heurísticas propostas nesta tese, primou-se pela simplicidade. Contudo, heurísticas mais elaboradas e estimativas mais precisam podem ser definidas. Neste sentido, um conjunto adicional de estimativas é apresentado a seguir.

1. P_{NF_K}**:**

   número de blocos de um determinado índice K no nível folha da árvore, ou seja, número de nós folha do índice K. P_{NF_K} pode ser estimado da seguinte foma: $P_{NF_K} \geq 2 \times \lceil O/2 \rceil^{HT_i - 2}$. Ou ainda, $P_{NF_K} \geq \lceil N_r/O \rceil$. Onde o é a ordem da árvore B^+.

2. O**:**

   ordem da árvore B^+. A ordem (O) de uma árvore B^+ pode ser calculada da seguinte forma: $(O \times P) + ((O-1) \times C) \leq T_p$. Onde: T_p é o tamanho de uma página em *bytes*, C é o tamanho da chave de busca em *bytes* e P é o tamanho do ponteiro de árvore em *bytes*. Considerando valores típicos do PostgreSQL, por exemplo, temos que: $T_p = 8192$, $P = 4$ e $C = 8$ (para uma chave do tipo *bigint*). Logo, $O \leq 682$, por exemplo.

3. $V(A, r)$**:**

   representa a cardinalidade de $\pi_A(r)$, ou seja, o número de *tuplas* resultantes da projeção do atributo A da relação r. Logo, se A é uma chave, $V(A, r) = N_r$.

4. $EC_{S_{IH}}$**:**

   indica a estimativa de custo para uma operação de seleção através de um índice *hash*. Neste caso, $EC_{S_{IH}} = 2$, considerando a não ocorrência de *bucket overflow*.

5. $EC_{J_{NLJ}}$**:**

   indica a estimativa de custo para uma operação de junção entre duas relações r e s, executada através de um *Nested Loop Join*. Neste caso, $EC_{J_{NLJ}} = P_r + (N_r \times P_s)$, considerando a relação r como a mais externa.

6. $EC_{J_{NLJI}}$**:**

   indica a estimativa de custo para uma operação de junção entre duas relações r e s, executada através de um *Nested Loop Join* utilizando índice. Neste caso, é necessário a existência de um índice sobre o atributo de junção na tabela mais interna (s). Assim, $EC_{J_{NLJI}} = P_r + (N_r \times c)$, onde c representa o custo de uma seleção simples (através de um índice).

7. $EC_{J_{BNLJ}}$**:**

   indica a estimativa de custo para uma operação de junção entre duas relações r e s (onde r é a relação mais externa e s a relação mais interna do laço), executada através de um *Block Nested Loop Join*. Neste caso,

$EC_{J_{BNLJ}} = P_r + (P_r \times P_s)$. Vale ressaltar que a ordem das tabelas pode influir na estimativa de custo. Assim, deve-se usar a menor tabela com a mais externa.

8. $EC_{J_{MJ}}$**:**

   indica a estimativa de custo para uma operação de junção entre duas relações r e s (onde r é a relação mais externa e s a relação mais interna do laço), executada através de um *Merge Join*. Neste caso, é necessário que r e s estejam ordenadas pelo atributo de junção. Assim, $EC_{J_{MJ}} = P_r + P_s$.

9. $EC_{J_{HJ}}$**:**

   indica a estimativa de custo para uma operação de junção entre duas relações r e s (onde r é a relação mais externa e s a relação mais interna do laço), executada através de um *Hash Join*. Neste caso, $EC_{J_{HJ}} = 3(P_r + P_s) + 2 \times max$. Onde $max \geq \lceil P_s/P_r \rceil$ e s é a tabela mais interna (que deve ser a menor tabela).

Este apêndice descreveu o modelo de custos externo, ou seja independente de SGBD, utilizado nesta tese. Este modelo é usado pelas heurísticas concebidas neste trabalho e na abordagem não intrusiva, descrita no Capítulo 4, possibilitando estimar, por exemplo, o custo de uma busca seqüencial, o custo de uma seleção utilizando um índice primário, etc.

B
Heurística Integrada para Seleção e Acompanhamento de Índices

Este apêndice apresenta, em detalhes, a heurística integrada para seleção e acompanhamento de índices (HISAI).

Para cada tripla <Consulta SQL, Plano de Execução, Custo> capturada **faça**
Percorrer o plano de execução (árvore de consulta) procurando por uma operação "*Seq Scan*" ("*Full Scan*" ou "*Table Scan*" ou "*File Scan*")
Se uma operação "*Seq Scan*" for encontrada **então faça**
$TabSS$ = tabela associada à operação de "*Seq Scan*"
EC_{FS} = estimativa do custo do "*Seq Scan*"
Se a operação de "*Seq Scan*" possui filtro (condição de seleção) **então faça**
Se "filtro" é do tipo "atributo=valor" **então faça**
Se "atributo" $\notin$ conjunto dos índices candidatos (hipotéticos) da tabela $TabSS$ **então faça**
Adicione "atributo_primario" como um novo índice primário candidato da tabela $TabSS$
$BA_{atributo\_primario} = 0$
Adicione "atributo_secundario" como um novo índice secundário candidato da tabela $TabSS$
$BA_{atributo\_secundario} = 0$
Fim Se
Estimar o custo de utilização de um *Index Scan*, caso existisse um índice primário ("*clustering*") i definido sobre "atributo"($EC_{S_{IP}}$)
Se "atributo" é chave primária **então faça:**
$EC_{S_{IP}} = H_{T_i} + 1$
Se não:
$EC_{S_{IP}} = H_{T_i} + FS(atributo, TabSS)/F_{TabSS}$
Fim Se
Se $EC_{S_{IP}} < EC_{FS}$ **então faça:**
$BA_{atributo\_primario} = BA_{atributo\_primario} + (EC_{FS} - EC_{S_{IP}})$
Fim Se
Estimar o custo de utilização de um *Index Scan*, caso existisse um índice secundário ("*not clustering*") i definido sobre "atributo" ($EC_{S_{IS}}$)
Se "atributo" é chave primária **então faça:**
$EC_{S_{IS}} = H_{T_i} + 1$
Se não:
$EC_{S_{IS}} = H_{T_i} + FS(atributo, TabSS)$
Fim Se
Se $EC_{S_{IS}} < EC_{FS}$ **então faça:**
$BA_{atributo\_secundario} = BA_{atributo\_secundario} + (EC_{FS} - EC_{S_{IS}})$
Fim Se
Fim Se
Se "filtro" é do tipo "atributo θ valor" (onde θ ="$>$", "$<$", "$\geq$" ou "$\leq$") **então faça**
Se "atributo" $\notin$ conjunto dos índices candidatos (hipotéticos) da tabela $TabSS$ **então faça**
Adicione "atributo_primario" como um novo índice primário candidato da tabela $TabSS$
$BA_{atributo\_primario} = 0$
Adicione "atributo_secundario" como um novo índice secundário candidato da tabela $TabSS$
$BA_{atributo\_secundario} = 0$
Fim Se
Estimar o custo de utilização de um *Index Scan*, caso existisse um índice primário ("*clustering*") i definido sobre "atributo" ($EC_{S_{IP}}$)
Faça $EC_{S_{IP}} = H_{T_i} + FS(atributo, TabSS)/F_{TabSS}$
Se $EC_{S_{IP}} < EC_{FS}$ **então faça:**
$BA_{atributo\_primario} = BA_{atributo\_primario} + (EC_{FS} - EC_{S_{IP}})$
Fim Se
Estimar o custo de utilização de um *Index Scan*, caso existisse um índice secundário ("*not clustering*") i definido sobre "atributo" ($EC_{S_{IS}}$)
$EC_{S_{IS}} = H_{T_i} + FS(atributo, TabSS)$
Se $EC_{S_{IS}} < EC_{FS}$ **então faça:**
$BA_{atributo\_secundario} = BA_{atributo\_secundario} + (EC_{FS} - EC_{S_{IS}})$
Fim Se
Fim Se
Fim Se

Figura B.1: Heurística Integrada para a Seleção e Acompanhamento de Índices: Parte 1.

Se a operação de "*Seq Scan*" não possui filtro (condição de seleção) **então faça**
chave_busca = atributos presentes na cláusula SELECT que pertencem à tabela $TabSS$
Se *chave_busca* $\notin$ conjunto dos índices candidatos (hipotéticos) da tabela $TabSS$ **então faça**
Adicione "*chave_busca_primario*" como um novo índice primário candidato da tabela $TabSS$
$BA_{chave\_busca\_primario} = 0$
Adicione "*chave_busca_secundario*" como um novo índice secundário candidato da tabela $TabSS$
$BA_{chave\_busca\_secundario} = 0$
Fim Se
Estimar o custo de utilização de um *Index Scan*, caso existisse um índice primário ("*clustering*") *i* definido sobre "*chave_busca*" ($EC_{S_{IP}}$)
Faça $EC_{S_{IP}} = H_{T_i} + FS(atributo, TabSS)/F_{TabSS}$
Se $EC_{S_{IP}} < EC_{FS}$ **então faça:**
$BA_{chave\_busca\_primario} = BA_{chave\_busca\_primario} + (EC_{FS} - EC_{S_{IP}})$
Fim Se
Estimar o custo de utilização de um *Index Scan*, caso existisse um índice secundário ("*not clustering*") *i* definido sobre "chave_busca" ($EC_{S_{IS}}$)
$EC_{S_{IS}} = H_{T_i} + FS(atributo, TabSS)$
Se $EC_{S_{IS}} < EC_{FS}$ **então faça:**
$BA_{chave\_busca\_secundario} = BA_{chave\_busca\_secundario} + (EC_{FS} - EC_{S_{IS}})$
Fim Se
Fim Se
Fim Se
Fim Para

Figura B.2: Heurística Integrada para a Seleção e Acompanhamento de Índices: Parte 2.

Para cada tripla <Consulta SQL, Plano de Execução, Custo> capturada **faça**
Percorrer o plano de execução (árvore de consulta) procurando por uma operação "*Index Scan*"
Se uma operação "*Index Scan*" for encontrada **então faça**
$TabSS$ = tabela associada à operação de "*Seq Scan*"
EC_{FS} = estimativa do custo do "*Seq Scan*"
Se a operação de "*Index Scan*" possui filtro (condição de seleção) **então faça**
Se "filtro" é do tipo "atributo=valor" **então faça**
Se "atributo" $\notin$ conjunto dos índices candidatos (hipotéticos) da tabela $TabSS$ **então faça**
Adicione "atributo_primario" como um novo índice primário candidato da tabela $TabSS$
$BA_{atributo\_primario} = 0$
Adicione "atributo_secundario" como um novo índice secundário candidato da tabela $TabSS$
$BA_{atributo\_secundario} = 0$
Fim Se
Se o índice utilizado na operação de *Index Scan* for um índice primário ("*clustering*") **então faça**
Se "atributo" é chave primária **então faça:**
$EC_{S_{IP}} = H_{T_i} + 1$
Se não:
$EC_{S_{IP}} = H_{T_i} + FS(atributo, TabSS)/F_{TabSS}$
Fim Se
Se $EC_{S_{IP}} < EC_{FS}$ **então faça:**
$BA_{atributo\_primario} = BA_{atributo\_primario} + (EC_{FS} - EC_{S_{IP}})$
Fim Se
Fim Se
Se o índice utilizado na operação de *Index Scan* for um índice secundário ("*not clustering*") **então faça**
Se "atributo" é chave primária **então faça:**
$EC_{S_{IS}} = H_{T_i} + 1$
Se não:
$EC_{S_{IS}} = H_{T_i} + FS(atributo, TabSS)$
Fim Se
Se $EC_{S_{IS}} < EC_{FS}$ **então faça:**
$BA_{atributo\_secundario} = BA_{atributo\_secundario} + (EC_{FS} - EC_{S_{IS}})$
Fim Se
Fim Se
Fim Se
Se "filtro" é do tipo "atributo θ valor" (onde θ ="$>$", "$<$", "$\geq$" ou "$\leq$") **então faça**
Se "atributo" $\notin$ conjunto dos índices candidatos (hipotéticos) da tabela $TabSS$ **então faça**
Adicione "atributo_primario" como um novo índice primário candidato da tabela $TabSS$
$BA_{atributo\_primario} = 0$
Adicione "atributo_secundario" como um novo índice secundário candidato da tabela $TabSS$
$BA_{atributo\_secundario} = 0$
Fim Se
Se o índice utilizado na operação *Index Scan* for um índice primário ("*clustering*") **então faça**
$EC_{S_{IP}} = H_{T_i} + FS(atributo, TabSS)/F_{TabSS}$
Se $EC_{S_{IP}} < EC_{FS}$ **então faça:**
$BA_{atributo\_primario} = BA_{atributo\_primario} + (EC_{FS} - EC_{S_{IP}})$
Fim Se
Fim Se
Se o índice utilizado na operação *Index Scan* for um índice secundário ("*not clustering*") *i*
$EC_{S_{IS}} = H_{T_i} + FS(atributo, TabSS)$
Se $EC_{S_{IS}} < EC_{FS}$ **então faça:**
$BA_{atributo\_secundario} = BA_{atributo\_secundario} + (EC_{FS} - EC_{S_{IS}})$
Fim Se
Fim Se
Fim Se
Fim Se
Fim Se
Fim Para

Figura B.3: Heurística Integrada para a Seleção e Acompanhamento de Índices: Parte 3.

C
Possíveis Melhorias na Heurística Integrada para Seleção e Acompanhamento de Índices

As heurísticas definidas nesta tese foram construídas tomando como diretriz a simplicidade, uma vez que o objetivo principal era validar as idéias propostas nesta tese. Contudo, heurísticas mais elaboradas e precisas podem ser definidas. A seguir, discutiremos algumas direções que podem ser seguidas para se obter essas características.

1. **Eliminar o custo de operações intermediárias que existiam no plano real, mas que deixam de existir no plano hipotético.**

   Considere a consulta a seguir, denominada Consulta 1:

   ```
   explain
   select l_orderkey, l_partkey
   from lineitem
   order by l_orderkey
   ```

   A Figura C.1 ilustra o plano de execução produzido para a Consulta 1, na ausência de índices.

	QUERY PLAN text
1	Sort (cost=1087101.79..1102083.91 rows=5992849 width=8)
2	Sort Key: l_orderkey
3	-> Seq Scan on lineitem (cost=0.00..166657.49 rows=5992849 width=8)

Figura C.1: Plano de execução produzido para a Consulta 1, na ausência de índices.

   Considere agora a criação de um índice secundário sobre o atributo l_orderkey.

   A Figura C.2 ilustra o plano de execução produzido para a Consulta 1, na presença do índice criado.

   Observe que, caso o plano de execução mostrado na figura C.2 fosse um plano hipotético, gerado pela abordagem proposta, o custo da operação *sort* deveria ser retirado do custo do plano hipotético.

	QUERY PLAN text
1	Index Scan using ix_lineitem_orderkey on lineitem (cost=0.00..262598.88 rows=6001215 width=8)

Figura C.2: Plano de execução produzido para a Consulta 1, na presença do índice criado.

2. **Trocar** *Bitmap Heap Scan* **por** *Bitmap Index Scan* **.**

   Considere a seguinte consulta, denominada Consulta 2:

```
select l_orderkey
from lineitem
where l_partkey < 3 and l_suppkey < 3
```

   A Figura C.3 ilustra o plano de execução produzido para a Consulta 2, na ausência de índices.

	QUERY PLAN text
1	Bitmap Heap Scan on lineitem (cost=13.38..2269.04 rows=1 width=4)
2	Recheck Cond: (l_partkey < 3)
3	Filter: (l_suppkey < 3)
4	-> Bitmap Index Scan on ix_lineitem_partkey (cost=0.00..13.38 rows=596 width=0)
5	Index Cond: (l_partkey < 3)

Figura C.3: Plano de execução produzido para a Consulta 2, na ausência de índices.

   Considere agora a criação de um índice secundário sobre o atributo l_orderkey.

   A Figura C.4 ilustra o plano de execução produzido para a Consulta 2, na presença do índice criado.

	QUERY PLAN text
1	Bitmap Heap Scan on lineitem (cost=26.99..31.01 rows=1 width=4)
2	Recheck Cond: ((l_suppkey < 3) AND (l_partkey < 3))
3	-> BitmapAnd (cost=26.99..26.99 rows=1 width=0)
4	-> Bitmap Index Scan on ix_lineitem_suppkey (cost=0.00..13.36 rows=594 width=0)
5	Index Cond: (l_suppkey < 3)
6	-> Bitmap Index Scan on ix_lineitem_partkey (cost=0.00..13.38 rows=596 width=0)
7	Index Cond: (l_partkey < 3)

Figura C.4: Plano de execução produzido para a Consulta 1, na presença do índice criado.

   Observe que, o plano de execução mostrado na figura C.4 poderia ser um plano hipotético, gerado pela abordagem proposta, a partir do plano original (Figura C.3).

3. **Considerar as diferenças entre os SGBDs.**

   Por exemplo, considere a consulta a seguir, denominada consulta 3:

```
select l_orderkey, l_partkey
from lineitem
```

   Assuma também a criação de um índice composto sobre os atributos l_orderkey, l_partkey.

   A Figura C.5 ilustra o plano de execução produzido para a Consulta 3, na presença do índice criado. Esse plano foi capturado a partir da execução da Consulta 3 no PostgreSQL 8.3.

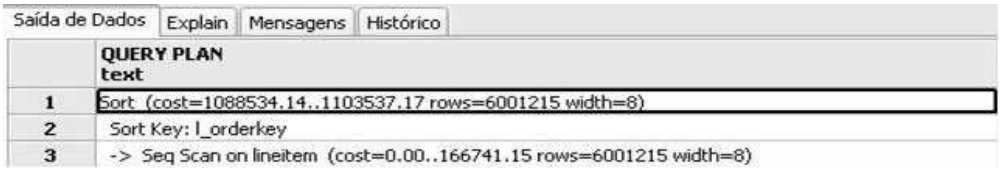

Figura C.5: Plano de execução produzido para a Consulta 3, na presença do índice criado, no PostgreSQL 8.3.

   Observe que o índice criado não foi utilizado. Contudo, se esta mesma consulta fosse executada no SQL Server, o índice seria utilizado, como mostrado na Figura C.6.

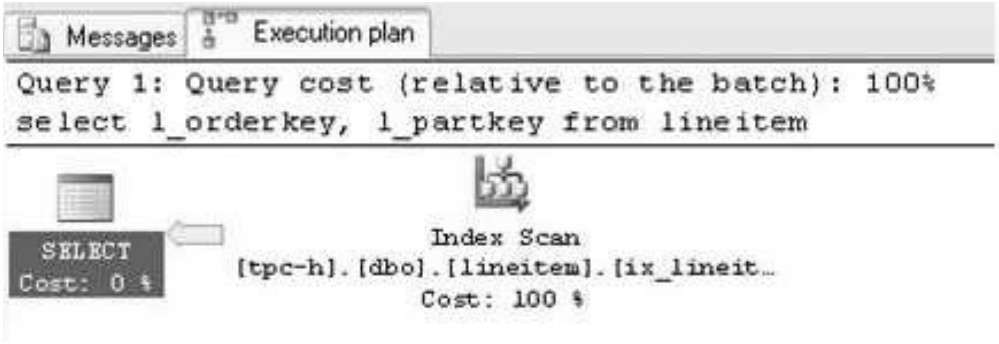

Figura C.6: Plano de execução produzido para a Consulta 3, na presença do índice criado, no PostgreSQL 8.3.

4. **Considerar a influência da seletividade.**

   Considere a consulta a seguir, denominada Consulta 4:

```
explain
select l_orderkey
from lineitem
where l_orderkey > 3000
```

Assuma também a criação de um índice secundário definido sobre o atributo l_orderkey.

A Figura C.7 ilustra o plano de execução produzido para a Consulta 4, na presença do índice criado.

	QUERY PLAN text
1	Seq Scan on lineitem (cost=0.00..181744.19 rows=6000619 width=4)
2	Filter: (l_orderkey > 3000)

Figura C.7: Plano de execução produzido para a Consulta 4, na presença do índice criado.

Observe que, o índice criado não foi utilizado. Isto deve-se ao fato da seletividade ser muito baixa, ou seja, muitas *tuplas* possuem $l\_orderkey > 3000$.

Considere agora a consulta a seguir, denominada Consulta 5:

```
explain
select l_orderkey
from lineitem
where l_orderkey < 3000
```

A Figura C.8 ilustra o plano de execução produzido para a Consulta 5, na presença do índice criado.

Saída de Dados | Explain | Mensagens | Histórico

	QUERY PLAN text
1	Index Scan using ix_lineitem_orderkey on lineitem (cost=0.00..33.34 rows=596 width=4)
2	Index Cond: (l_orderkey < 3000)

Figura C.8: Plano de execução produzido para a Consulta 5, na presença do índice criado.

Observe que, agora, o índice criado foi utilizado. Isto deve-se ao fato da seletividade ser alta, ou seja, poucas *tuplas* possuem $l\_orderkey < 3000$.

5. **Propor índices sobre os atributos usados nas cláusulas** *order by* **e** *group by* **para buscas seqüenciais que não possuem filtro.**

   Considere a seguinte consulta, denominada Consulta 6:

```
select l_orderkey, l_partkey
from lineitem
order by l_orderkey
```

Assuma também que, foi criado um índice secundário sobre o atributo l_orderkey.

A Figura C.9 ilustra o plano de execução produzido para a Consulta 6, na presença do índice criado.

	QUERY PLAN text
1	Index Scan using ix_lineitem_orderkey on lineitem (cost=0.00..262598.88 rows=6001215 width=8)

Figura C.9: Plano de execução produzido para a Consulta 6, na presença do índice criado.

Note que, o índice criado sobre o atributo utilizado na cláusula order by contribuiu para diminuir o custo da consulta 6.

D
Benchmark TPC-C

O *benchmark* TPC-C é uma carga de trabalho, composta por um conjunto de transações de leitura e escrita, que simula as atividades encontradas em um ambiente OLTP (*On-Line Transaction Processing*) complexo. O propósito do TPC-C é prover dados relevantes de desempenho que auxiliem os usuários a comparar de maneira objetiva dois ou mais sistemas quanto a sua capacidade de processar transações (Mendes06, Korth06, TPC).

O ambiente simulado pelo TPC-C consiste em um sistema de processamento de transações usado para registrar as atividades de uma empresa atacadista que produz, vende e distribui seus produtos. Essa companhia encontra-se geograficamente distribuída, de maneira que suas vendas estão espalhadas por um número de armazéns (*Warehouses*) e distritos (*Districts*). O tamanho da organização é definido pelo número de armazéns que ela possui; conforme ela cresce mais armazéns e distritos precisam ser criados. Cada armazém mantém estoque para todos os cem mil produtos vendidos pela companhia e cada um dos seus distritos atende a três mil clientes (*Customers*) (Mendes06).

A Figura D.1 ilustra a hierarquia entre armazéns, distritos e clientes. Já a Figura D.2 mostra o diagrama ER do banco de dados TPC-C.

O sistema da companhia é usado para:

- Registrar solicitações de compras de clientes;
- Consultar a situação de pedidos feitos previamente;
- Registrar pagamentos dos clientes;
- Processar entregas de pedidos;
- Consultar o nível de estoque de produtos;

Como será mostrado na Seção D.2, cada uma dessas ações corresponde a uma das transações que compõem a carga de trabalho do *benchmark*.

O banco de dados especificado pelo TPC-C é composto por nove tabelas. A Figura D.2 mostra essas tabelas e os seus respectivos relacionamentos. Vale ainda ressaltar que:

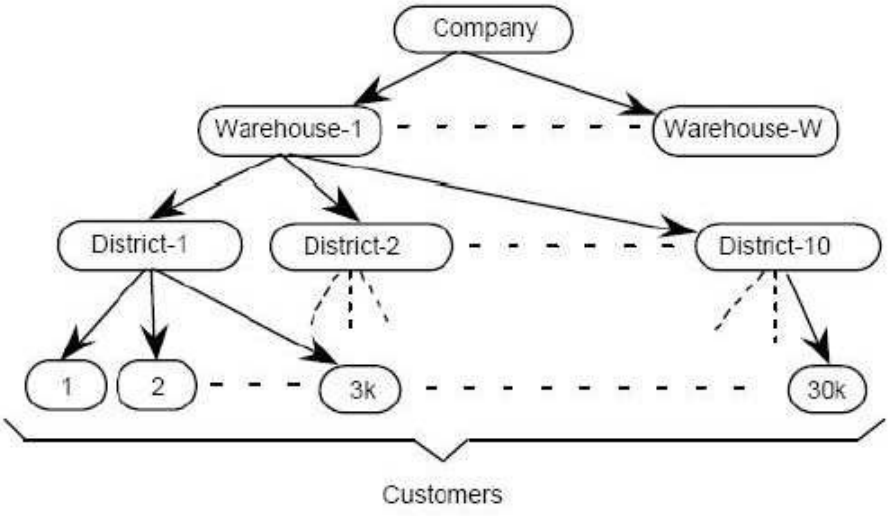

Figura D.1: Relação entre armazéns, distritos e clientes.

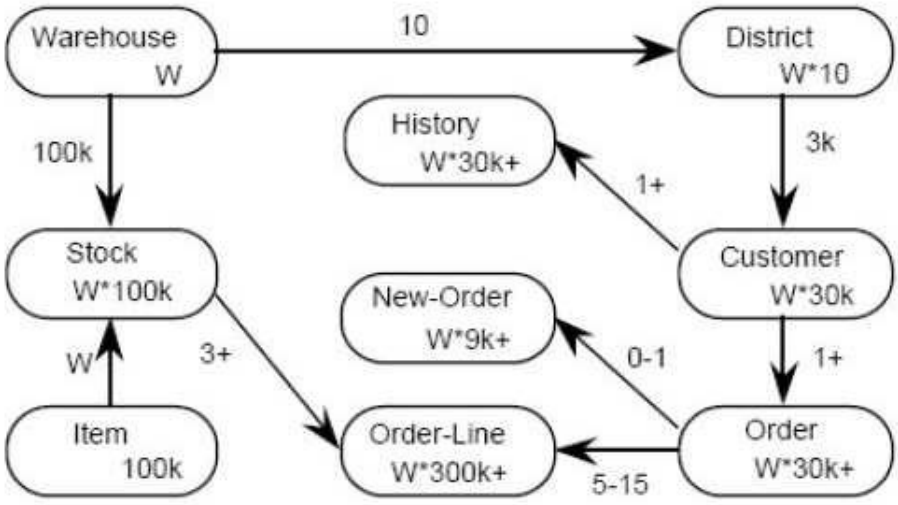

Figura D.2: Diagrama ER do Banco de Dados TPC-C.

– Os números dentro das entidades indicam a cardinalidade das tabelas (número de linhas). Por exemplo, W indica que a tabela warehouse é populada com W itens de dados. A tabela district, por sua vez, tem $10 \times W$ linhas. Interessante notar que, excetuando-se a tabela Item, cuja cardinalidade é fixada em 100 mil linhas, todas as outras tabelas têm suas cardinalidades proporcionais ao número de linhas da tabela warehouse;

– Os números próximos às linhas de relacionamento representam a cardinalidade dos relacionamentos;

D.1 O Ambiente OLTP

Sistemas de processamento de transação formam provavelmente o tipo mais comum de sistemas de banco de dados em uso nos dias de hoje. Também conhecidos como sistemas OLTP, eles são empregados para registrar as atividades de uma organização.

Exemplos de aplicações OLTP incluem: sistemas de reserva de passagem aérea, sistemas bancários, sistemas de acompanhamento de mercado de ações, etc. Embora, sejam empregados nas mais diversas áreas, tais sistemas compartilham algumas características comuns:

– **Grande quantidade de usuários**: O sistema deve suportar o acesso simultâneo de muitos usuários;

– **Restrições sobre o tempo de resposta**: O sistema deve responder às requisições dos usuários em um tempo inferior a um valor especificado;

– **Disponibilidade**: Devido à importância que têm para as organizações, sistemas OLTP normalmente devem estar em operação 24 horas por dia, sete dias por semana;

– **Padrão de acesso a dados**: Em um sistema OLTP típico, as transações são compostas por um misto de leituras e escritas. Os dados são lidos por meio de consultas e escritos por meio de inserções, atualizações e exclusões;

Cada um desses fatores implica em necessidades específicas para que o sistema opere de maneira normal. Em particular, o número de usuários (que determina a intensidade da carga submetida ao sistema) e o tempo gasto para responder às suas requisições (um critério para a avaliação do desempenho do sistema) têm uma maior relevância para esse trabalho.

D.2
Cargas de Trabalho TPC-C

A carga de trabalho TPC-C é formada por cinco transações. A seguir, discutiremos cada uma destas transações, destacando as operações executadas e os padrões de acesso a dados de cada uma delas.

D.2.1
Novo Pedido (*New-Order*)

A transação *New Order* consiste no registro de uma operação de compra de um cliente. As operações executadas incluem:

- Inserção de registros nas tabelas Order, New Order e Order Line;
- Para cada um dos itens que compõem a compra, atualização do nível de estoque (tabela Stock);
- Consulta de informações das tabelas Warehouse, District, Customer e Item;

Esta transação (Figura D.3) é executada com uma freqüência alta, que impõe uma carga média ao sistema por meio de operações de leitura e de escrita, sendo a principal transação do *benchmark* TPC-C (Mendes06).

```
1. SELECT w_tax FROM warehouse
   WHERE w_id = 1

2. SELECT d_tax, d_next_o_id FROM district
   WHERE d_w_id = 1 AND d_id = 1

3. UPDATE district SET d_next_o_id = d_next_o_id + 1
   WHERE d_w_id = 1 AND d_id = 1

4. SELECT c_discount, c_last, c_credit FROM customer
   WHERE c_w_id = 1 AND c_d_id = 1 AND c_id = 1

5. INSERT INTO new_order (no_o_id, no_d_id, no_w_id)
   VALUES (-1, 1, 1)

6. INSERT INTO orders(o_id, o_d_id, o_w_id, o_c_id, o_entry_d, o_carrier_id,
                      o_ol_cnt, o_all_local)
   VALUES (-1, 1, 1, 1, current_timestamp, NULL, 1, 1)

7. SELECT i_price, i_name, i_data FROM item
   WHERE i_id = 1

8. SELECT s_quantity, s_dist_01, s_data FROM stock
   WHERE s_i_id = 1 AND s_w_id = 1

9. UPDATE stock SET s_quantity = s_quantity - 10
   WHERE s_i_id = 1 AND s_w_id = 1

10. INSERT INTO order_line (ol_o_id, ol_d_id, ol_w_id, ol_number, ol_i_id,
                            ol_supply_w_id, ol_delivery_d, ol_quantity,
                            ol_amount, ol_dist_info)
    VALUES (-1, 1, 1, 1, 1, 1, NULL, 1, 1.0, 'hello kitty')
```

Figura D.3: Transação *New Order*.

D.2.2
Pagamento (*Payment*)

A transação *Payment* (Figura D.4)consiste no registro do pagamento de uma compra. As operações executadas incluem:

- Atualização de informações de saldo do cliente (tabela Customer), do distrito (tabela District) e do armazém (tabela Warehouse);
- Insere um novo registro na tabela History;

Trata-se de uma transação de peso leve, com alta freqüência de execução, composta por operações de leitura e escrita.

```
1. SELECT w_name, w_street_1, w_street_2, w_city, w_state, w_zip
   FROM warehouse WHERE w_id = 1

2. UPDATE warehouse SET w_ytd = w_ytd + 1.0 WHERE w_id = 1

3. SELECT d_name, d_street_1, d_street_2, d_city, d_state, d_zip
   FROM district
   WHERE d_id = 1 AND d_w_id = 1

4. UPDATE district SET d_ytd = d_ytd + 1.0 WHERE d_id = 1 AND d_w_id = 1

5. SELECT c_id FROM customer
   WHERE c_w_id = 1 AND c_d_id = 1 AND c_last = 'BARBARBAR'
   ORDER BY c_first ASC

6. SELECT c_first, c_middle, c_last, c_street_1, c_street_2, c_city,
   c_state, c_zip, c_phone, c_since, c_credit, c_credit_lim, c_discount,
   c_balance, c_data, c_ytd_payment
   FROM customer
   WHERE c_w_id = 1 AND c_d_id = 1 AND c_id = 1

7. UPDATE customer SET c_balance = c_balance - 1.0,
   c_ytd_payment = c_ytd_payment + 1
   WHERE c_id = 1 AND c_w_id = 1 AND c_d_id = 1

8. UPDATE customer SET c_balance = c_balance - 1.0,
   c_ytd_payment = c_ytd_payment + 1, c_data = 'hello dogger'
   WHERE c_id = 1 AND c_w_id = 1 AND c_d_id = 1

9. INSERT INTO history (h_c_id, h_c_d_id, h_c_w_id, h_d_id, h_w_id,
                        h_date, h_amount, h_data)
   VALUES (1, 1, 1, 1, 1, current_timestamp, 1.0, 'ab    cd')
```

Figura D.4: Transação *Payment*.

D.2.3
Estado do Pedido (*Order-Status*)

A transação *Order Status* (Figura D.5) representa uma consulta da situação da última compra de um cliente. As operações executadas são as seguintes:

- Seleção de informações do cliente a partir da tabela Customer;

– Seleção de informações de compra a partir das tabelas Order e OrderLine;

É uma transação somente-leitura, executada com baixa freqüência, que impõe uma carga moderada ao sistema.

```
1. SELECT c_id FROM customer
   WHERE c_w_id = 1 AND c_d_id = 1 AND c_last = 'BARBARBAR'
   ORDER BY c_first ASC

2. SELECT c_first, c_middle, c_last, c_balance FROM customer
   WHERE c_w_id = 1 AND c_d_id = 1 AND c_id = 1

3. SELECT o_id, o_carrier_id, o_entry_d, o_ol_cnt FROM orders
   WHERE o_w_id = 1 AND o_d_id = 1 AND o_c_id = 1
   ORDER BY o_id DESC

4. SELECT ol_i_id, ol_supply_w_id, ol_quantity, ol_amount, ol_delivery_d
   FROM order_line
   WHERE ol_w_id = 1 AND ol_d_id = 1 AND ol_o_id = 1
```

Figura D.5: Transação *Order Status*.

D.2.4 Entrega (*Delivery*)

A transação *Delivery* (Figura D.6) registra a entrega de um pedido. Ela consiste no processamento em *batch* de dez novos pedidos. Para cada distrito de um armazém, as seguintes operações são executadas:

– Busca e exclusão do registro mais antigo da tabela New Order que esteja associado ao distrito em questão;
– Atualização de registros nas tabelas Order, OrderLine e Customer;

Trata-se de uma transação de leitura e escrita, com baixa freqüência de execução, que impõe uma carga moderada ao sistema.

D.2.5 Nível de Estoque (*Stock-Level*)

A transação *Stock Level* (Figura D.7) é utilizada para determinar, dentre os itens vendidos recentemente, quais estão com o estoque abaixo de um determinado limite. Ela é composta das seguintes operações:

– Seleção das últimas vinte compras de um distrito (tabela District);
– Contagem dos itens cujo estoque encontra-se abaixo do limite estabelecido (tabelas Stock e Order Line);

Trata-se de uma transação somente-leitura com baixa freqüência de execução, mas que impõe uma carga pesada ao sistema.

```
1. SELECT no_o_id
   FROM new_order
   WHERE no_w_id = 1 AND no_d_id = 1

2. DELETE FROM new_order
   WHERE no_o_id = 1 AND no_w_id = 1 AND no_d_id = 1

3. SELECT o_c_id
   FROM orders
   WHERE o_id = 1 AND o_w_id = 1 AND o_d_id = 1

4. UPDATE orders SET o_carrier_id = 1
   WHERE o_id = 1 AND o_w_id = 1 AND o_d_id = 1

5. UPDATE order_line SET ol_delivery_d = current_timestamp
   WHERE ol_o_id = 1 AND ol_w_id = 1 AND ol_d_id = 1

6. SELECT SUM(ol_amount * ol_quantity)
   FROM order_line
   WHERE ol_o_id = 1 AND ol_w_id = 1 AND ol_d_id = 1

7. UPDATE customer SET c_delivery_cnt = c_delivery_cnt + 1,
   c_balance = c_balance + 1
   WHERE c_id = 1 AND c_w_id = 1 AND c_d_id = 1
```

Figura D.6: Transação *Delivery*.


```
1. SELECT d_next_o_id FROM district
   WHERE d_w_id = 1 AND d_id = 1

2. SELECT count(*) FROM order_line, stock, district
   WHERE d_id = 1 AND d_w_id = 1 AND d_id = ol_d_id AND d_w_id = ol_w_id
   AND ol_i_id = s_i_id AND ol_w_id = s_w_id AND s_quantity < 15
   AND ol_o_id BETWEEN (1) AND (20)
```

Figura D.7: Transação *Stock-Level*.

E
Benchmark TPC-H

O TPC-H é um *benchmark* de suporte a decisões que consiste em um conjunto de consultas *ad-hoc* voltadas para os negócios e modificações de dados simultâneas. Tem por finalidade simular e avaliar o desempenho de um ambiente de *Data Warehouse*. *Data Warehouse* é um grande repositório de dados coletados de diversas fontes que se destina a gerar informações para o nível gerencial sendo fonte para tomadas de decisão.

Os testes do método TPC-H são realizados sob uma estrutura padrão composta por oito tabelas. Destas, seis são tabelas dimensionais (*Region*, *Nation*, *Supplier*, *Part*, *Customer* e *Partsupp*) e duas de fatos (*Orders* e *Lineitem*).

As tabelas de fato estão no centro do modelo dimensional (modelo para suporte à decisão), seus valores são usados para medir o desempenho do negócio. As tabelas de dimensão são áreas ou focos do negócio e fornecem um método geral de organizar a informação corporativa provendo múltiplas perspectivas dos dados. As tabelas de fatos armazenam os fatos ocorridos e as chaves para as características correspondentes nas tabelas dimensionais. As consultas ocorrem inicialmente nas tabelas de dimensão e depois nas tabelas de fatos, assegurando a precisão dos dados.

A Figura E.1 mostra o modelo dimensional do *benchmark* TPC-H. Já a Figura E.2 ilustra o modelo relacional do *benchmark* TPC-H.

Algumas observações devem ser ressaltadas, uma vez analisada a Figura E.2 (Morelli06a):

- O número que aparece logo abaixo do nome da tabela representa sua cardinalidade. Esta pode ser fixa (tabelas region e nation), ou variável, onde multiplica-se uma constante por um *scale factor* determinado no momento da criação da base. Por exemplo, caso SF valha 3, haverá 450.000 *tuplas* na tabela de clientes (customer).
- Estão representadas cinco regiões (continentes), que congregam vinte e cinco nações (tabelas region e nation, respectivamente). Clientes e Fornecedores (tabelas supplier e customer) estão associados às nações. Enquanto os primeiros realizam pedidos de compras (tabela orders),

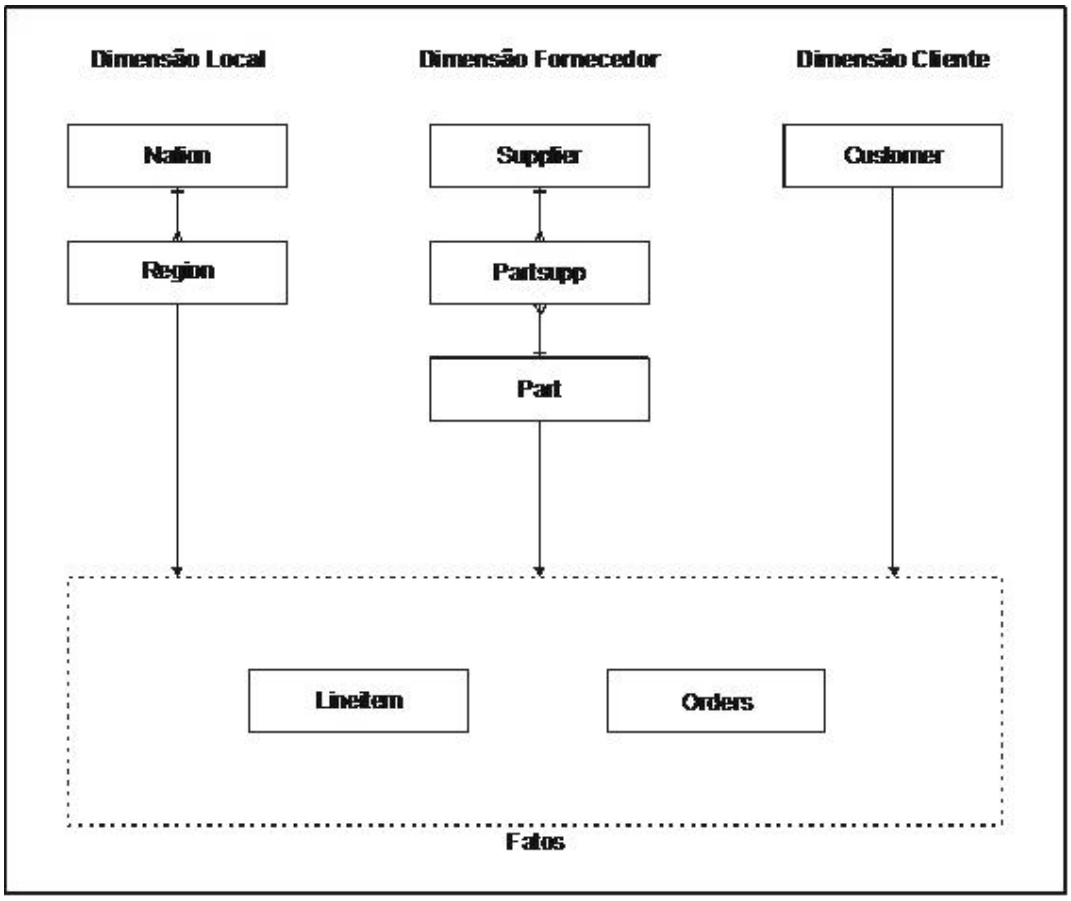

Figura E.1: Modelo Dimensional do TPC-H.

os segundos fornecem componentes (tabela part) de itens de compra. Como um fornecedor pode oferecer vários itens e um item pode ser disponibilizado por vários fornecedores, existe uma tabela para registrar esta relação $N \times N$ (partsupp). Finalmente, a tabela mais volumosa do modelo (lineitem) associa itens de compra a pedidos.

– Os parênteses ao lado dos nomes das tabelas indicam o prefixo utilizado para denominar os campos da tabela em questão. Desta forma, a chave primária da tabela de fornecedores chama-se S_SUPPKEY;

– As flechas indicam as associações entre chaves primárias e estrangeiras. Assim, vemos que a chave primária da tabela nation está vinculada ao campo nationkey na tabela de fornecedores (tabela supplier);

O tamanho total da base de dados depende do fator de escalabilidade (SF - *scale factor*). Para SF=1, a base completa ocupa aproximadamente 1 GB e os volumes de cada tabela são os que aparecem na Figura E.2. Já para SF=30, a base ocupará 30 GB e a tabela de itens de pedidos de compra possuirá cento e oitenta milhões de tuplas (30 × 6 milhões) (Morelli06a).

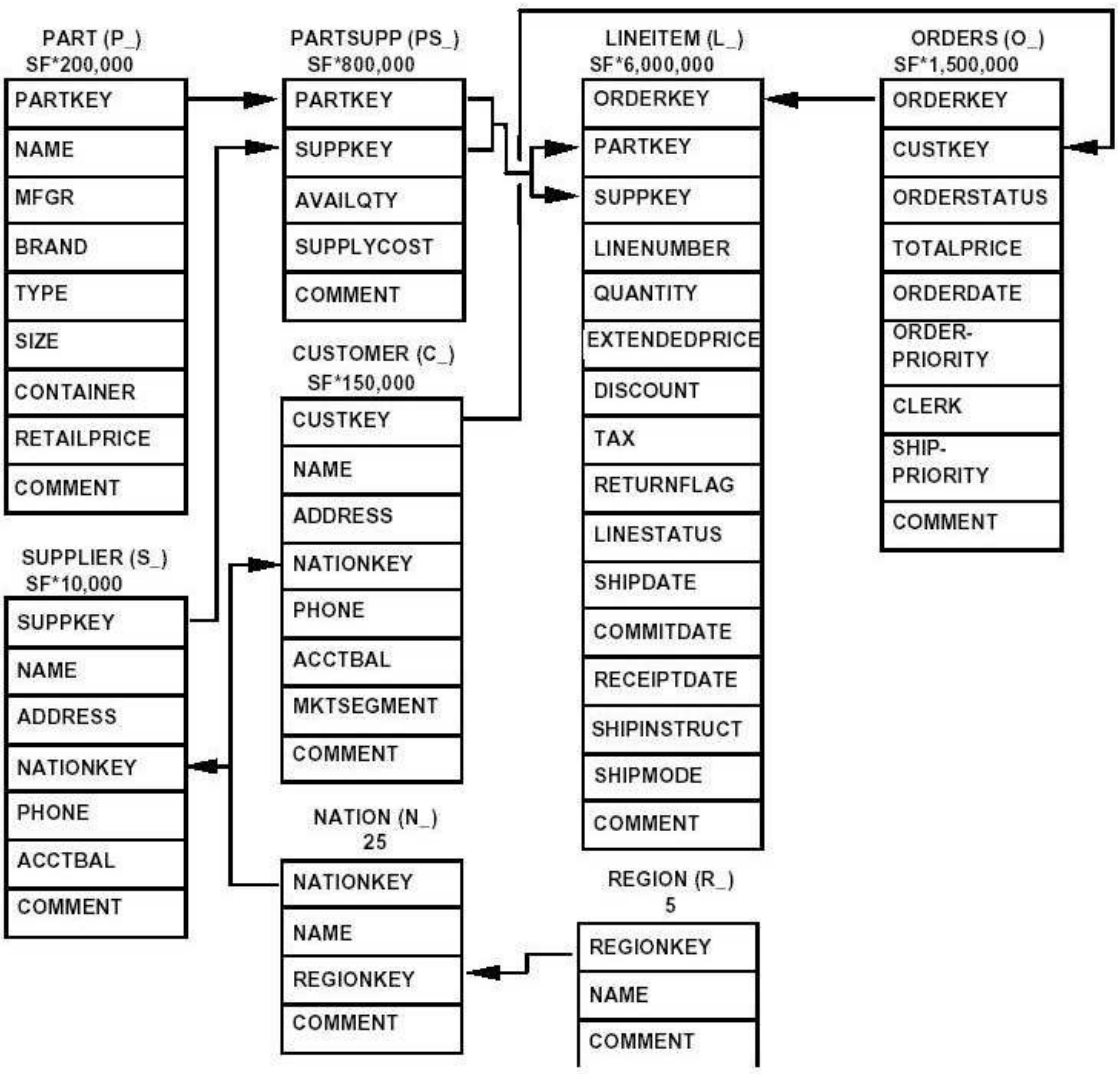

Figura E.2: Modelo Relacional do TPC-H.

A base de dados sofre acessos de um conjunto de consultas possuindo características *ad-hoc*, ou seja, não se conhece nem a ordem de execução, nem os parâmetros de cada uma das 22 consultas (leitura) e duas funções (uma insere dados e a outra elimina) (Morelli06a).

O *benchmark* TPC-H compõe-se por vários testes, como revela a tabela E.1.

Antes de executar os testes, alimenta-se o banco de dados com base em arquivos texto previamente gerados. Isto acontece graças a uma ferramenta normalmente apelidada por *DBGEN* (*Database Generator*), seguindo preceitos de (TPC). A ordem na qual são lidas as vinte e duas consultas, bem como seus parâmetros, também são gerados por um programa gerador denominado: *QGEN* (*Query Generator*).

Teste	Detalhes
Refresh Function 1 (RF1)	Insere *tuplas* nas duas maiores tabelas: orders e lineitem.
Refresh Function 2 (RF2)	Elimina tuplas das tabelas orders e lineitem.
Power	Compreendido pela *Refresh Function 1*, grupo completo de análises de consultas e *Refresh Function 2*.
Throughput	Dispara vários fluxos (*streams*) de comandos em paralelo. Invoca as vinte e duas consultas seguidas pelas *Refresh Functions*.
Performance	Reúne os testes *Power* e *Throughput*.

Tabela E.1: Testes que Compõem o *benchmark* TPC-H.

SF	Fluxos
1	2
10	3
30	4
100	5
300	6

Tabela E.2: Quantidade de Fluxos Utilizados no Teste *Throughput*.

A quantidade de fluxos de comandos utilizados no teste *Throughput* deve acompanhar o fator de escalabilidade da base de dados. A especificação oficial do *benchmark* TPC-H apresenta a seguinte correlação (Tabela E.2):

O teste *Throughput* produz duas medidas: a vazão (quantos comandos por segundo) e tempo transcorrido.

As funções de atualização (RF1 e RF2) trabalham sobre as maiores tabelas, orders e lineitem; a primeira função insere *tuplas* e a segunda exclui. Vale ressaltar que, como a execução de uma RF sempre vem seguida da outra, a quantidade de *tuplas* permanece estável, já que RF1 insere 0,1% de tuplas, enquanto RF2 elimina outras 0,1% de *tuplas* (Morelli06a).

Tanto RF1 quanto RF2 recebem por argumento o fator de escalabilidade, coerente com o tamanho atual das tabelas. Assim, na tabela lineitem, quando SF=1, a execução de RF1 causará a inserção de seis mil tuplas: 6.000.000 × 0,01. Caso fosse necessário aumentar o percentual de *tuplas* afetadas, as funções poderiam ser executadas com argumento SF=30, o que, em bases com fator de escalabilidade um, teríamos o aumento trinta vezes maior (de 0,1% para 3% : 180.000 *tuplas*) (Morelli06a).

E.1
O Ambiente OLAP

OLAP (*On-Line Analytical Processing*) é uma tecnologia que permite aos analistas de negócios, gerentes e executivos analisar e visualizar dados corporativos de forma rápida, consistente e principalmente interativa.

A funcionalidade OLAP é inicialmente caracterizada pela análise dinâmica e multidimensional dos dados consolidados de uma organização permitindo que as atividades do usuário final sejam tanto analíticas quanto navegacionais. A tecnologia OLAP é geralmente implementada em ambiente multiusuário e cliente/servidor, oferecendo assim respostas rápidas às consultas *ad-hoc*, não importando o tamanho do banco de dados nem sua complexidade. Hoje em dia, OLAP também vem sendo disponibilizada em ambiente *Web*.

A tecnologia OLAP auxilia o usuário a sintetizar informações corporativas por meio de visões comparativas e personalizadas, análises históricas, projeções e elaborações de cenários.

E.2
Cargas de Trabalho TPC-H

Como mencionado anteriormente, o *benchmark* TPC-H é composto por um conjunto de consultas *ad-hoc* que simulam as atividades encontradas em um ambiente OLAP. A seguir, apresentamos as consultas que compõem este *benchmark*:

```
1.  SELECT  l_returnflag,l_linestatus, sum(l_quantity) as sum_qty,
            sum(l_extendedprice) as sum_base_price,
            sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
            sum(l_extendedprice * (1 - l_discount) * (1 + l_tax))
                as sum_charge,
            avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
            avg(l_discount) as avg_disc, count(*) as count_order
    FROM lineitem
    WHERE l_shipdate <= date'1998-12-01' - interval '10 days'
    GROUP BY l_returnflag, l_linestatus
    ORDER BY l_returnflag, l_linestatus


2.  SELECT  s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address,
            s_phone, s_comment
    FROM    part, supplier, partsupp, nation, region
```

```
    WHERE   p_partkey = ps_partkey and s_suppkey = ps_suppkey and p_size = 20
            and p_type like '%COPPER' and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey and r_name = 'AMERICA'
            and ps_supplycost = ( SELECT min(ps_supplycost)
                                  FROM partsupp, supplier, nation, region
                                  WHERE p_partkey = ps_partkey
                                        and s_suppkey = ps_suppkey
                                        and s_nationkey = n_nationkey
                                        and n_regionkey = r_regionkey
                                        and r_name = 'AMERICA'
                                )
    ORDER BY s_acctbal desc, n_name, s_name, p_partkey
```

```
3.  SELECT  l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue,
            o_orderdate, o_shippriority
    FROM    customer, orders, lineitem
    WHERE   c_mktsegment = 'AUTOMOBILE' and c_custkey = o_custkey
            and l_orderkey = o_orderkey and o_orderdate < date '19981231'
            and l_shipdate > date '19910101'
    GROUP BY l_orderkey, o_orderdate, o_shippriority
    ORDER BY revenue desc, o_orderdate
```

```
4.  SELECT o_orderpriority, count(*) as order_count
    FROM   orders
    WHERE  o_orderdate >= date '19980801'
           and o_orderdate < date '19980808' + interval '3 month'
           and exists ( SELECT *
                        FROM lineitem
                        WHERE l_orderkey = o_orderkey
                              and l_commitdate < l_receiptdate
                      )
    GROUP BY o_orderpriority
    ORDER BY o_orderpriority
```

```
5.  SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
    FROM   customer, orders, lineitem, supplier, nation, region
```

```
    WHERE  c_custkey = o_custkey and l_orderkey = o_orderkey
           and l_suppkey = s_suppkey and c_nationkey = s_nationkey
           and s_nationkey = n_nationkey and n_regionkey = r_regionkey
           and r_name = 'AMERICA' and o_orderdate >= date '19910801'
           and o_orderdate < date '19910801' + interval '1 year'
    GROUP BY n_name
    ORDER BY revenue desc
```

6.
```
    SELECT sum(l_extendedprice * l_discount) as revenue
    FROM   lineitem
    WHERE  l_shipdate >= date '19980107'
           and l_shipdate < date '19980107' + interval '1 year'
           and l_discount between 2 - 0.01 and 2 + 0.01
           and l_quantity < 5
```

7.
```
    SELECT supp_nation, cust_nation, l_year, sum(volume) as revenue
    FROM ( SELECT n1.n_name as supp_nation, n2.n_name as cust_nation,
                  extract(year from l_shipdate) as l_year,
                  l_extendedprice * (1 - l_discount) as volume
           FROM   supplier, lineitem, orders, customer, nation n1, nation n2
           WHERE  s_suppkey = l_suppkey  and o_orderkey = l_orderkey
                  and c_custkey = o_custkey and s_nationkey = n1.n_nationkey
                  and c_nationkey = n2.n_nationkey
                  and (
                         (n1.n_name = 'ARGENTINA' and n2.n_name = 'ARGENTINA')
                         or
                         (n1.n_name = 'BRAZIL' and n2.n_name = 'BRAZIL')
                      )
                  and l_shipdate between date '1995-01-01'
                  and date '1996-12-31'
         ) as shipping
    GROUP BY supp_nation, cust_nation, l_year
    ORDER BY supp_nation, cust_nation, l_year
```

8.
```
    SELECT o_year, sum(case when nation = 'UNITED STATES'
                        then volume else 0 end) / sum(volume) as mkt_share
```

```
    FROM ( SELECT extract(year from o_orderdate) as o_year,
                  l_extendedprice * (1 - l_discount) as volume,
                  n2.n_name as nation
           FROM   part, supplier, lineitem, orders, customer,
                  nation n1, nation n2, region
           WHERE  p_partkey = l_partkey and s_suppkey = l_suppkey
                  and l_orderkey = o_orderkey and o_custkey = c_custkey
                  and c_nationkey = n1.n_nationkey
                  and n1.n_regionkey = r_regionkey and r_name = 'AFRICA'
                  and s_nationkey = n2.n_nationkey
                  and o_orderdate between date '1995-01-01' a
                  and date '1996-12-31' and p_type = 'ECONOMY BRUSHED COPPER'
         ) as all_nations
    GROUP BY o_year
    ORDER BY o_year
```

```
9.  SELECT nation, o_year, sum(amount) as sum_profit
    FROM ( SELECT n_name as nation, extract(year from o_orderdate) as o_year,
                  l_extendedprice * (1 - l_discount) -
                  ps_supplycost * l_quantity as amount
           FROM   part, supplier, lineitem, partsupp, orders, nation
           WHERE  s_suppkey = l_suppkey and ps_suppkey = l_suppkey
                  and ps_partkey = l_partkey and p_partkey = l_partkey
                  and o_orderkey = l_orderkey and s_nationkey = n_nationkey
                  and p_name like '%blush%'
         ) as profit
    GROUP BY nation, o_year
    ORDER BY nation, o_year desc
```

```
10. SELECT  c_custkey, c_name,
            sum(l_extendedprice * (1 - l_discount)) as revenue,
            c_acctbal, n_name, c_address, c_phone, c_comment
    FROM    customer, orders, lineitem, nation
    WHERE   c_custkey = o_custkey and l_orderkey = o_orderkey
            and o_orderdate >= date '19920801'
            and o_orderdate < date '19920801' + interval '3 month'
            and l_returnflag = 'N' and c_nationkey = n_nationkey
```

```
    GROUP BY c_custkey, c_name, c_acctbal, c_phone, n_name, c_address,
             c_comment order by revenue desc
```

```
11. SELECT ps_partkey, sum(ps_supplycost * ps_availqty) as value
    FROM   partsupp, supplier, nation
    WHERE  ps_suppkey = s_suppkey and s_nationkey = n_nationkey
           and n_name = 'BRAZIL'
    GROUP BY ps_partkey
    HAVING sum(ps_supplycost * ps_availqty) >
                    (
                      SELECT sum(ps_supplycost * ps_availqty) * 2
                      FROM   partsupp, supplier, nation
                      WHERE  ps_suppkey = s_suppkey and s_nationkey = n_nationkey
                             and n_name = 'BRAZIL'
                    )
    ORDER BY value desc
```

```
12. SELECT l_shipmode, sum(case when o_orderpriority = '1-URGENT'
                                    or o_orderpriority = '2-HIGH'
                               then 1 else 0 end) as high_line_count,
                       sum(case when o_orderpriority <> '1-URGENT'
                                    and o_orderpriority <> '2-HIGH'
                               then 1 else 0 end) as low_line_count
    FROM   orders, lineitem
    WHERE  o_orderkey = l_orderkey and l_shipmode in ('TRUCK', 'AIR')
           and l_commitdate < l_receiptdate and l_shipdate < l_commitdate
           and l_receiptdate >= date '19960101'
           and l_receiptdate < date '19960101' + interval '1 year'
    GROUP BY l_shipmode
    ORDER BY l_shipmode
```

```
13. SELECT c_count, count(*) as custdist
    FROM ( SELECT c_custkey, count(o_orderkey)
           FROM   customer
                  left outer join orders on c_custkey = o_custkey
                  and o_comment not like '%even%deposits%'
```

```
        GROUP BY c_custkey
      ) as c_orders (c_custkey, c_count)
GROUP BY c_count
ORDER BY custdist desc, c_count desc
```

```
14. SELECT 100.00 * sum(case when p_type like 'PROMO%'
                        then l_extendedprice * (1 - l_discount)
                        else 0 end
                       ) / sum(l_extendedprice * (1 - l_discount))
                  as promo_revenue
    FROM  lineitem, part
    WHEER l_partkey = p_partkey and l_shipdate >= date '19960201'
          and l_shipdate < date '19960201' + interval '1 month'
```

```
15. CREATE VIEW revenue (supplier_no, total_revenue) as
        SELECT l_suppkey, sum(l_extendedprice * (1 - l_discount))
        FROM lineitem
        WHERE l_shipdate >= '19960201' and l_shipdate < '19960501'
        GROUP BY l_suppkey
```

```
16. SELECT p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt
    FROM   partsupp, part
    WHERE  p_partkey = ps_partkey and p_brand <> 'Brand#13'
           and p_type not like 'STANDARD%'
           and p_size in (7, 12, 14, 16, 21, 23, 32, 43)
           and ps_suppkey not in ( SELECT s_suppkey
                                   FROM supplier
                                   WHERE s_comment like '%Customer%Complaints%'
                                  )
    GROUP BY p_brand, p_type, p_size
    ORDER BY supplier_cnt desc, p_brand, p_type, p_size
```

```
17. SELECT sum(l_extendedprice) / 7.0 as avg_yearly
    FROM   lineitem, part
    WHERE  p_partkey = l_partkey and p_brand = 'Brand#13'
```

```
        and p_container = 'JUMBO PKG'
        and l_quantity < ( SELECT 0.2 * avg(l_quantity)
                           FROM lineitem
                           WHERE l_partkey = p_partkey
                          )
```

```
18. SELECT c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice,
        sum(l_quantity)
    FROM   customer, orders, lineitem
    WHERE  o_orderkey in ( SELECT l_orderkey
                           FROM lineitem
                           GROUP BY l_orderkey
                           HAVING sum(l_quantity) > 3
                          )
        and c_custkey = o_custkey
        and o_orderkey = l_orderkey
    GROUP BY c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
    ORDER BY o_totalprice desc, o_orderdate
```

```
19. SELECT sum(l_extendedprice* (1 - l_discount)) as revenue
    FROM   lineitem, part
    WHERE  ( p_partkey = l_partkey and p_brand = 'Brand#13'
             and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
             and l_quantity >= 4 and l_quantity <= 14
             and p_size between 1 and 5
             and l_shipmode in ('AIR', 'AIR REG')
             and l_shipinstruct = 'DELIVER IN PERSON'
           )
           or
           ( p_partkey = l_partkey and p_brand = 'Brand#44'
             and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
             and l_quantity >= 5 and l_quantity <= 15
             and p_size between 1 and 10 and l_shipmode in ('AIR', 'AIR REG')
             and l_shipinstruct = 'DELIVER IN PERSON'
           )
           or
           ( p_partkey = l_partkey and p_brand = 'Brand#53'
```

```
            and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
            and l_quantity >= 6 and l_quantity <= 16
            and p_size between 1 and 15 and l_shipmode in ('AIR', 'AIR REG')
            and l_shipinstruct = 'DELIVER IN PERSON'
          )
```

```
20. SELECT s_name, s_address
    FROM   supplier, nation
    WHERE  s_suppkey in ( SELECT distinct (ps_suppkey)
                          FROM partsupp, part
                          WHERE ps_partkey=p_partkey and p_name like 'dim%'
                                and ps_availqty > ( SELECT 0.5 * sum(l_quantity)
                                                    FROM lineitem
                                                    WHERE l_partkey = ps_partkey
                                                      and l_suppkey = ps_suppkey
                                                      and l_shipdate >= '19970301'
                                                      and l_shipdate <
                                                      date '19970301' +
                                                           interval '1 year'
                                                  )
                        )
           and s_nationkey = n_nationkey and n_name = 'ARGENTINA'
    ORDER BY s_name
```

```
21. SELECT s_name, count(*) as numwait
    FROM   supplier, lineitem l1, orders, nation
    WHERE  s_suppkey = l1.l_suppkey and o_orderkey = l1.l_orderkey
           and o_orderstatus = 'F' and l1.l_receiptdate > l1.l_commitdate
           and exists ( SELECT *
                        FROM lineitem l2
                        WHERE l2.l_orderkey = l1.l_orderkey
                              and l2.l_suppkey <> l1.l_suppkey
                      )
           and not exists ( SELECT *
                            FROM lineitem l3
                            WHERE l3.l_orderkey = l1.l_orderkey
                                  and l3.l_suppkey <> l1.l_suppkey
```

```
                                and l3.l_receiptdate > l3.l_commitdate
                        )
        and s_nationkey = n_nationkey and n_name = 'BRAZIL'
    GROUP BY s_name
    ORDER BY numwait desc, s_name
```

22.
```
    SELECT cntrycode, count(*) as numcust, sum(c_acctbal) as totacctbal
    FROM   ( SELECT substr(c_phone, 1, 2) as cntrycode, c_acctbal
             FROM   customer
             WHERE  substr(c_phone, 1, 2)
                      in ('25', '11', '13', '14', '30', '23', '18')
                    and c_acctbal > ( SELECT avg(c_acctbal)
                                      FROM   customer
                                      WHERE  c_acctbal > 0.00
                                             and substr(c_phone, 1, 2)
                                             in ('25', '11', '13', '14',
                                                 '30', '23','18')
                                    )
                    and not exists ( SELECT *
                                     FROM orders
                                     WHERE o_custkey = c_custkey
                                   )
           ) as vip
    GROUP BY cntrycode
    ORDER BY cntrycode
```

23.
```
    SELECT s_suppkey, s_name, s_address, s_phone, total_revenue
    FROM   supplier, revenue
    WHERE  s_suppkey = supplier_no
           and total_revenue = ( SELECT max(total_revenue) FROM revenue )
    ORDER BY s_suppkey
```

24.
```
    DROP VIEW revenue
```

F
Arquitetura Implementada

A Figura F.1 ilustra os componentes da arquitetura proposta que foram implementados no protótipo construído para validar as idéias concebidas nesta tese.

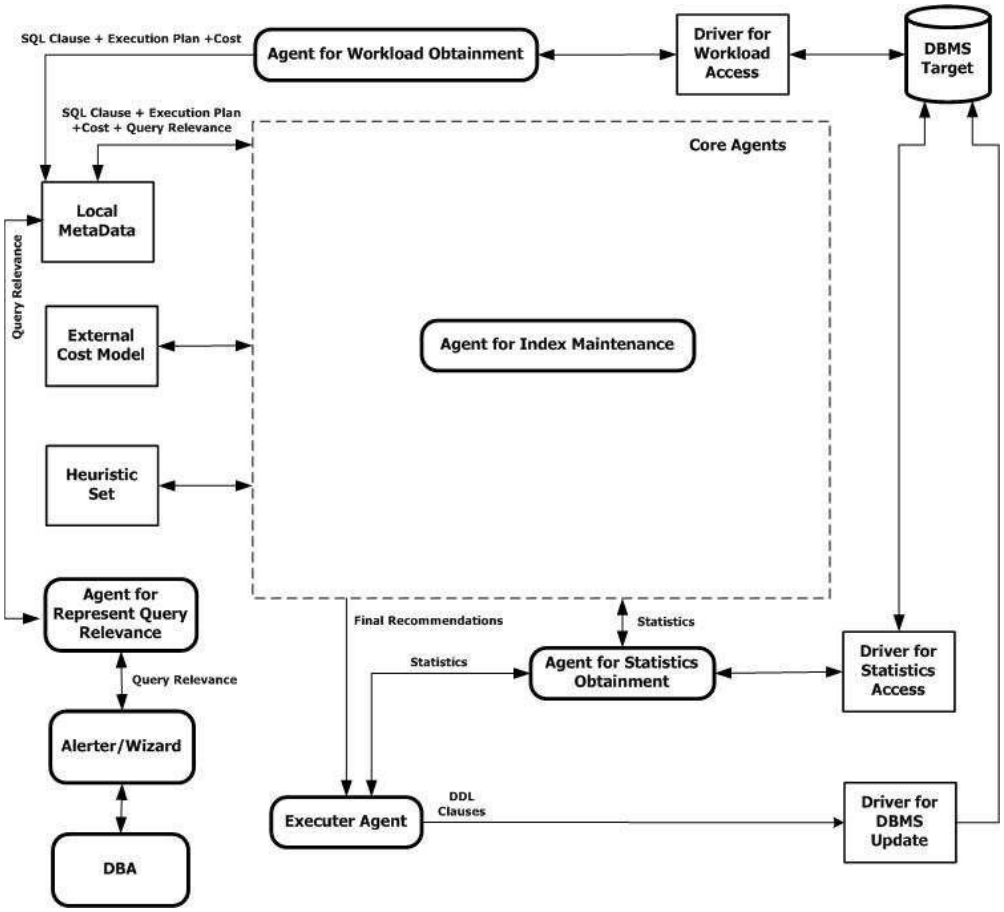

Figura F.1: Arquitetura do Protótipo Implementado.