Uma abordagem de código único para aplicações independentes de provedor de bases de dados relacionais

Willian Eduardo de Moura Casante[1]

[1]Fundação CPqD – Centro de Pesquisa e Desenvolvimento em Telecomunicações
Rod. SP-340 km 118,5 CEP 13.086-902 – Campinas – SP – Brasil

wcasante@cpqd.com.br

Abstract. *Relational databases uses specific SQL extensions for their product functions, which makes the development of complex applications unfeasible mainly because compatibility with more than one relational database would require specific codes for each database. This paper describes the development of multi-database applications based on a methodology of creation of the installation artifacts for the database and a technique to develop multi-database sources codes.*

Resumo. *Os bancos de dados relacionais fornecem funcionalidades específicas de produtos através de extensões da linguagem SQL, inviabilizando o desenvolvimento de aplicações complexas e compatíveis com mais de um banco de dados relacional sem que essas aplicações tenham código específico para cada um deles. Este artigo aborda o desenvolvimento de aplicações totalmente independentes de banco de dados através de uma metodologia de criação de artefatos de instalação da estrutura de dados e de uma técnica de desenvolvimento de código-fonte independentemente de produto.*

1. Introdução

A linguagem SQL (*Structured Query Language*) [Chamberlin et al. 1981] é uma linguagem de pesquisa e definição de estrutura de dados, inicialmente criada para SGBDs (Sistemas Gerenciadores de Banco de Dados), fortemente inspirada na álgebra relacional [Elmasri et al. 2005].

Necessidades não mapeadas no padrão SQL desenvolvido e aceito pela indústria exigiram dos fabricantes de SGBDs a definição de construções SQL específicas para seus produtos e a criação de extensões da linguagem que possibilitam o acesso das suas funcionalidades proprietárias.

Essas diferenças entre as diversas implementações da linguagem SQL prejudicam o desenvolvimento de um sistema computacional compatível com dois ou mais SGBDs pois, para cada SGBD atendido, um conjunto de arquivos específicos para aquele deverá ser mantido, tanto para a instalação do sistema quanto para o funcionamento correto de seu código-fonte.

2. Análise da incompatibilidade entre SGBDs

Um sistema computacional que realiza o armazenamento de suas informações em SGBDs possui seus arquivos de instalação organizados em DDLs (*Data Definition Language*) e DMLs (*Data Manipulation Language*).

DDL é um arquivo de metadados, que define a estrutura das tabelas de um banco de dados, composto por comandos SQL como, por exemplo, *CREATE*, *ALTER* e *DROP*. Os arquivos do tipo DDL são os primeiros a serem executados durante a instalação de um sistema computacional, visto que criam o arcabouço do sistema no SGBD.

DML é um arquivo de dados, cuja estrutura respeita fielmente a estrutura definida em seu respectivo arquivo DDL. Durante o desenvolvimento de um sistema pode, se necessário, inicializar o banco de dados; essa inicialização é colocada em arquivos do tipo DML. Os arquivos DML possuem comandos de *INSERT*, *UPDATE*, *DELETE* e *SELECT*, também utilizados de forma embutida dentro do código-fonte do sistema.

2.1. A estrutura dos dados em um SGBD

Durante a análise de um sistema, uma das tarefas é a definição de seu modelo de banco de dados. Essa definição normalmente envolve a construção de entidades básicas do sistema, em uma linguagem visual de alto nível, como, por exemplo, o modelo de entidades e relacionamentos [Barbieri 1994] ou a UML (*Unified Modeling Language*) [Jacobson et al. 1996]. Um exemplo simples de modelo UML pode ser observado na Figura 1. Através desse modelo inicial, os analistas de sistemas realizam um aprofundamento na modelagem, através da definição de todas as entidades e de seus relacionamentos, para que o sistema possa ser construído sobre esse modelo proposto. Conforme o sistema é desenvolvido, novas entidades e relacionamentos são incluídos e estes, detalhados e então o sistema evoluído para os contemplar.

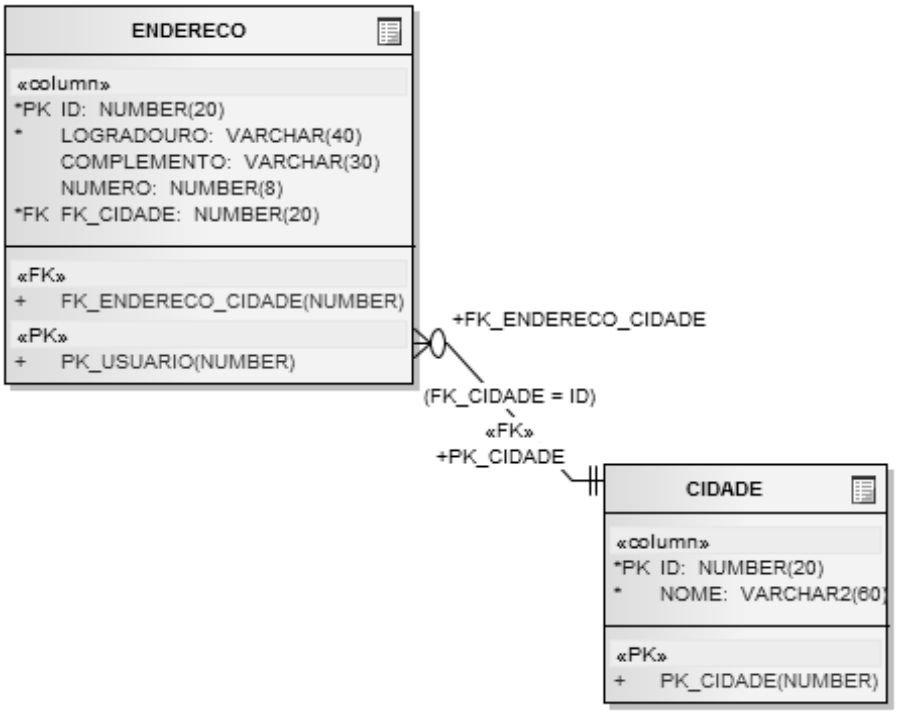

Figura 1. Exemplo de modelagem via UML

A partir da definição gráfica das entidades do modelo de banco de dados, um especialista deverá gerar um ou mais arquivos DDL, contendo os comandos SQL equivalentes

para a criação daquele modelo em um SGBD específico. Existem ferramentas que geram automaticamente esses arquivos a partir dos diagramas propostos, que estão fora do escopo deste artigo.

Um sistema que se propõe a ser instalado em mais de um SGBD apresenta algumas estratégias que podem ser adotadas, para a criação de seus arquivos DDL:

- Criação de um ou mais arquivos DDL por SGBD – exige a replicação de artefatos de trabalho (arquivos DDL) para cada um dos SGBDs; adiciona risco de erro na conversão, caso realizada de forma manual, como o risco da falta de sincronia entre as atualizações dos artefatos. Como exemplo, um profissional obtém uma DDL para o SGBD Oracle e deve convertê-la para funcionar no SGBD PostgreSQL e causa um erro ao não trocar NUMBER(20) por NUMERIC(20);
- Criação de um único arquivo DDL para um SGBD e utilização de uma ferramenta de conversão para os outros SGBDs – a conversão por ferramenta mitiga os possíveis erros porém adiciona um passo à geração do pacote de instalação do sistema: a conversão dos arquivos DDL. A existência de uma ferramenta que realize essa conversão, bem como sua aquisição, caso seja comercial, é um requisito primário, que pode ser substituído pelo desenvolvimento da própria ferramenta, como realizado no desenvolvimento da ferramenta JExodus [Neto and Passos];
- Geração automática de modelo através de código-fonte – alguns arcabouços de programação (por exemplo a especificação EJB (*Entity JavaBean*) do Java EE [Patel et al. 2006]) permitem a definição de entidades programáticas e a geração do modelo de banco de dados de forma transparente. Esse formato não é aproveitável em sistemas que necessitam controlar com precisão seus modelos de banco de dados ou realizar atualizações pois a instalação e a definição do modelo ficam a encargo do arcabouço. Pode haver conflito no que diz respeito à segurança, onde a instalação da estrutura do banco de dados ficar a cargo de um administrador do SGBD, ou durante a execução, se o sistema obtiver uma conexão na qual apenas controla a atualização dos dados neste SGBD.

2.2. A manutenção dos dados em um SGBD

A definição da estrutura, ou modelo, de um sistema em um SGBD, conforme descrito na Subseção 2.1, deverá permitir a inserção, alteração, exclusão e consulta dos dados presentes nessa estrutura. Para realizar essas operações existem os comandos DML.

A instalação de um sistema pode necessitar de dados iniciais, inseridos durante a própria instalação ou configurados por uma equipe de implantação, dados estes presentes em scripts DML e também, durante a utilização do sistema, realiza esses comandos no SGBD para que os dados sejam manipulados na citada estrutura.

Portanto, podemos listar dois cenários a serem tratados:

- A criação dos dados iniciais em uma estrutura, durante a instalação de um sistema, cujos comandos DML deverão ser compatíveis com diversos SGBDs ou convertidos conforme necessidade;
- O ciclo de produção do sistema, pós-instalação, no qual o sistema emitirá comandos DML para a manipulação dos dados presentes na estrutura instalada. Esses comandos deverão ser genéricos ou estruturados de forma que o sistema não

conheça as diferenças entre os diversos SGBDs e suas peculiaridades, permitindo que o código-fonte, geralmente em linguagem de alto nível (como o Java), seja único e independente de SGBD.

2.3. Linguagens específicas

Cada SGBD pode prover uma ou mais linguagens de programação específicas, conhecidas como linguagem de *script*, nas quais é possível adicionar lógica de programação além dos comandos DDL e DML. Essas linguagens (por exemplo, o PL/SQL do SGBD Oracle [Feuerstein 2002]) possuem sintaxe proprietária e muitas vezes de difícil conversão automática entre os diversos SGBDs.

Podem ser definidas funções (*Stored Procedures*), diretamente no SGBD, utilizando-se essas linguagens de programação e realizar a chamada dessas funções diretamente do código-fonte do sistema.

3. Arquiteturas passíveis de utilização

Enumeram-se algumas possíveis arquiteturas de acesso e manutenção dos dados de múltiplos SGBDs, para um sistema de software:

- Desenvolvimento tradicional – com projeto de banco de dados, geração de arquivos DDL e DML, controle manual das alterações e controle de arquivos por SGBD;
- Arcabouço de desenvolvimento, como o Ruby on Rails [St.Laurent et al. 2012] ou o Java Persistence API [Patel et al. 2006], nos quais o próprio arcabouço trata a criação e a evolução do banco de dados;
- Metodologia proposta neste artigo, que unifica os itens citados anteriormente na manutenção de uma única base de artefatos relativos à base de dados.

4. Adotando uma metodologia única

A metodologia, proposta na Figura 2, une o desenvolvimento tradicional aos arcabouços de desenvolvimento, aproveitando o seu melhor sem limitar-se às fronteiras impostas pelos arcabouços, facilitando assim o uso em sistemas de grande porte, que exigem a modelagem de bases de dados extremamente complexas.

A premissa para o desenvolvimento do sistema é a capacidade de este ser executado em dois ou mais SGBDs. Se essa necessidade não existir, o desenvolvimento tradicional poderá ser adotado. Do mesmo modo, se a complexidade da estrutura de dados for baixa, independentemente do tamanho do sistema, será possível utilizar por completo e com todas as vantagens, um dos arcabouços de desenvolvimento ágil (como, por exemplo, o Ruby on Rails).

4.1. Definição de um SGBD "base"

Antes de iniciar o desenvolvimento, para determinar o SGBD "base", no qual os arquivos DML e DDL serão escritos, deverão ser realizados uma pesquisa e um estudo das ferramentas disponíveis para conversão de cada SGBD suportado pelo sistema. Muitas ferramentas requerem de licença de uso, o que deve ser levado em conta na análise e na tomada de decisão. Pode-se chegar à conclusão de que não há ferramenta compatível com os requisitos impostos, e, nesse caso, será utilizada a conversão manual. É recomendável adicionar ao projeto o risco de defeitos causados pela conversão manual dos artefatos.

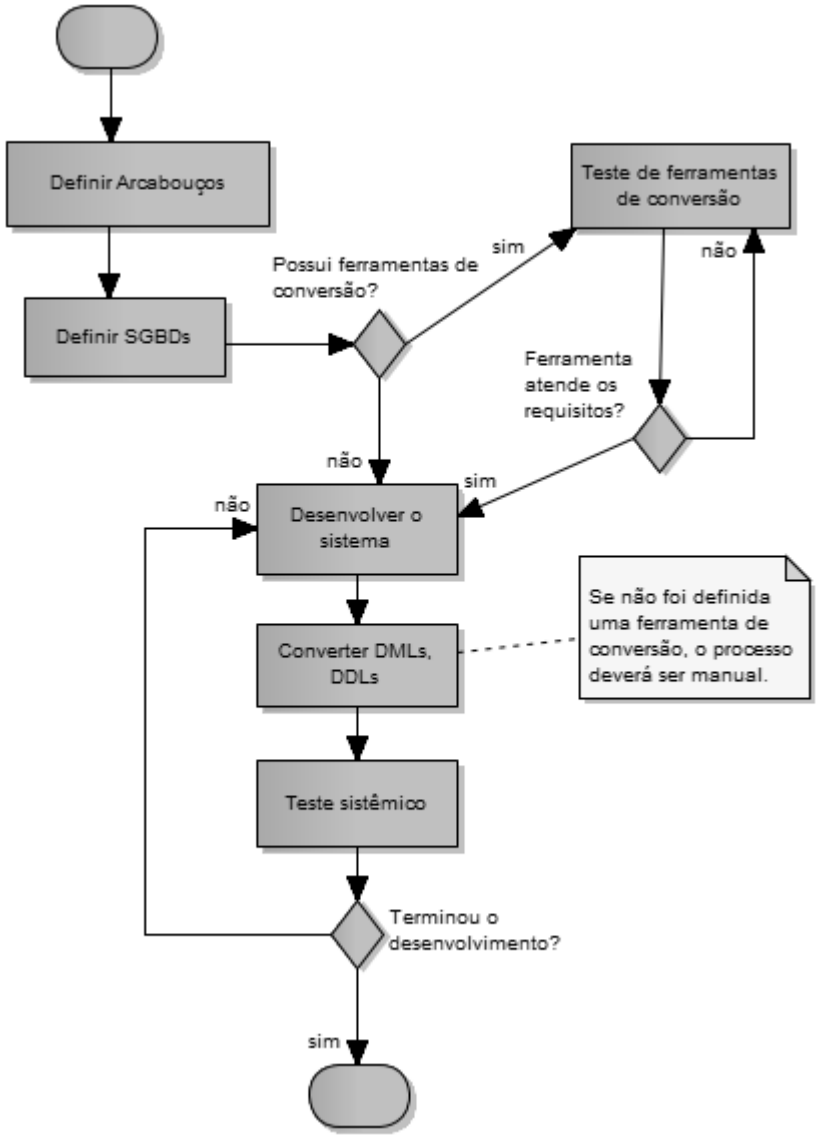

Figura 2. Metodologia sugerida para o desenvolvimento do sistema

4.2. Definição dos arcabouços de programação

Uma vez conhecida a linguagem de programação na qual o sistema será desenvolvido, aconselha-se escolher também um arcabouço de abstração de acesso aos dados. Esses arcabouços abstraem detalhes dos diversos SGBDs suportados, padronizam o código-fonte, e facilitam o desenvolvimento do acesso aos dados, pois normalmente oferecem facilidades para tal.

Deve ser feito um estudo de uso, conforme feito por Viana e Borba [Viana and Borba 1999], a partir de arcabouços atuais da linguagem de programação escolhida (por exemplo, para o Java, o Hibernate [Bauer and King 2005] ou a especificação Java Persistence API [Patel et al. 2006]), para determinar o melhor conjunto de arcabouços. Caso os SGBDs escolhidos suportem a orientação à objetos [Boscarioli et al. 2006], deve-se observar se os arcabouços de programação escolhidos possuem o suporte necessário para as funcionalidades que serão utilizadas.

4.3. Ciclo de vida do sistema

Definidos o SGBD "base" e a ferramenta de conversão para os outros SGBDs (se for o caso), são gerados os scripts DML e DDL e realizadas as conversões necessárias, conforme o sistema é desenvolvido. Eventuais scripts de atualização de versão do sistema precisarão ser escritos para cada SGBD, pois muitas vezes envolvem migração de dados.

O uso de funções SQL diretamente em banco de dados deve ser evitado, pois aumenta a complexidade da solução, exigindo sua manutenção para cada SGBD, bem como a coerência entre as diferentes codificações, inerentes a cada SGBD.

É importante que todos os envolvidos no desenvolvimento do sistema sigam o processo a partir das soluções adotadas, tanto para a conversão entre os diversos SGBDs quanto para o desenvolvimento do próprio sistema na sua linguagem de programação. Esse cuidado garante que todos trabalhem para que a aplicação continue compatível com cada SGBD escolhido.

O teste sistêmico deverá ser realizado com o sistema instalado em cada SGBD, garantindo assim que esteja compatível com todos.

É importante entender que, quanto maior o nível de abstração oferecido pelo arcabouço escolhido e de automação na conversão entre os SGBDs menor o risco de incompatibilidade entre eles. Para um sistema totalmente abstraído em relação ao seu modelo relacional, o teste sistêmico poderá ser realizado por amostragem, sendo que os casos de teste são executados parcialmente em um SGBD e parcialmente em outro, evitando que seja necessário realizar todos os casos de teste em todos os SGBDs, com expressivo ganho de tempo durante seu desenvolvimento.

5. Prova de conceito

Para a realização da prova de conceito foi considerado um sistema inicialmente construído para o SGBD Oracle, que deveria ser compatível também com o SGBD PostgreSQL. O requisito de "portabilidade" entre ambos SGBDs, durante a vida do sistema e suas evoluções, possibilitou a aplicação da metodologia definida na Seção 4, garantindo a qualidade dos artefatos gerados e a ausência de impacto nos prazos de entrega, eliminando também o risco produzido por conversões manuais realizadas pelos desenvolvedores.

A estrutura adotada, para os artefatos gerados para o projeto é apresentada na Figura 3.

Conforme apresentado na Seção 2, existem arquivos DDL, de instalação da estrutura de tabelas do sistema e arquivos DML, que realizam a população dos dados iniciais necessários para o funcionamento do sistema, bem como os comandos DML presentes em código-fonte, no caso, escrito em Java.

Os comandos DDL, originalmente em Oracle, foram convertidos utilizando-se a ferramenta Ora2Pg [Ora2Pg 2012]. Esses arquivos precisam inicialmente ser instalados em um SGBD Oracle para que a ferramenta consiga extrair as DDLs no formato PostgreSQL. Da mesma forma que as DDLs, as DMLs de carga do sistema foram instaladas no banco de dados Oracle e extraídas com a mesma ferramenta para o formato PostgreSQL.

Os arquivos DDL e DML gerados para o PostgreSQL são colocados em controle

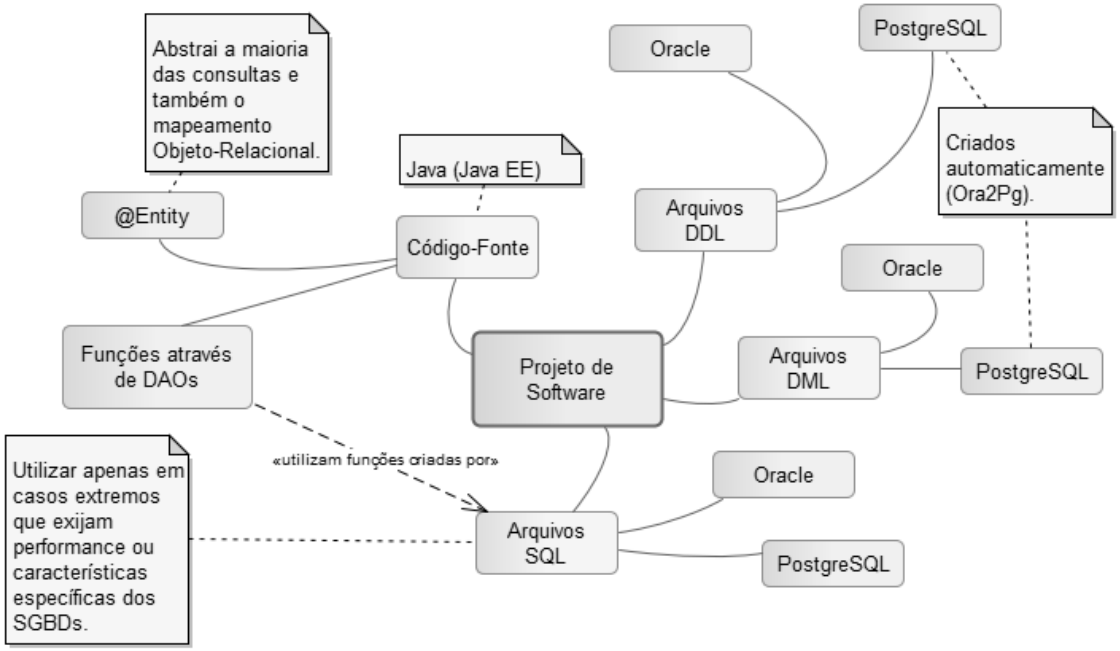

Figura 3. Estrutura do projeto da prova de conceito

de versão, sendo disponibilizados para a instalação, juntamente com o produto. Dessa forma o instalador poderá optar por fazer a instalação em um SGBD Oracle ou PostgreSQL, utilizando para isso um único pacote de instalação.

Funções SQL podem ser criadas diretamente em banco de dados e poderão ser utilizadas, a partir do código-fonte Java, através do padrão de projeto DAO (*Data Access Object*) [Alur et al. 2003]. Essas funções devem ser criadas com uma mesma assinatura de chamada, mesmo que para diferentes SGBDs, para que a chamada possa ser única e a construção do DAO independente do SGBD, bem como devem ser construídas para todos os SGBDs definidos.

O sistema foi construído totalmente sobre o arcabouço Java EE EJB e todas as entidades de banco de dados foram mapeadas utilizando-se as funcionalidades de abstração fornecidas por esse arcabouço. Todas as consultas a essas entidades foram construídas utilizando-se a linguagem de consulta JPQL (*Java Persistence Query Language*), que abstrai a linguagem SQL, garantindo que o código-fonte Java seja único e independente do SGBD adotado.

O resultado da prova de conceito é um sistema capaz de trabalhar tanto no SGBD Oracle quanto no SGBD PostgreSQL, com um único pacote de instalação e um único código-fonte, sendo que os processos de conversão entre esses SGBDs são completamente automatizados, validando a metodologia proposta.

6. Conclusão

Propor uma solução para a abstração de acesso aos dados de um SGBD, bem como centralizar e automatizar a geração de comandos de definição do modelo relacional, garante a longevidade do sistema desenvolvido. O sistema torna-se independente de SGBD, permitindo a redução dos custos de implantação pela adoção de soluções de software livre

sem adaptações do código-fonte original ou mesmo troca estratégica do SGBD. O conhecimento dos desenvolvedores envolvidos na produção do sistema pode ser nivelado, pois não são mais necessários especialistas para cada SGBD, os riscos são mitigados e os pontos passíveis de erro reduzidos.

Por fim, a automação da geração e do controle dos artefatos de cada SGBD evita erros relacionados à migração de código-fonte específico de determinado SGBD, que são maiores no caso de migração manual, bem como reduz o custo do desenvolvimento por eliminar as horas necessárias para a conversão.

Referências

Alur, D., Crupi, J., and Malks, D. (2003). *Core J2EE patterns: best practices and design strategies*. Prentice Hall.

Barbieri, C. (1994). *Modelagem de Dados*. IBPI Press Rio de Janeiro.

Bauer, C. and King, G. (2005). *Hibernate in action*. Manning.

Boscarioli, C., Bezerra, A., Benedicto, M., and Delmiro, G. (2006). Uma reflexão sobre banco de dados orientados a objetos. *IV CONGED*.

Chamberlin, D., Astrahan, M., Blasgen, M., Gray, J., King, W., Lindsay, B., Lorie, R., Mehl, J., Price, T., Putzolu, F., et al. (1981). A history and evaluation of system r. *Communications of the ACM*, 24(10):632–646.

Elmasri, R., Navathe, S., Pinheiro, M., Canhette, C., Melo, G., Amadeu, C., and de Oliveira Morais, R. (2005). *Sistemas de banco de dados*. Pearson Addison Wesley.

Feuerstein, S. (2002). *Oracle pl/sql Programming*. O'Reilly.

Jacobson, I., Booch, G., and Rumbaugh, J. (1996). *The Unified Modeling Language*. University Video Communications.

Neto, J. and Passos, E. Jexodus: uma ferramenta para migraçao de dados independente de sgbd.

Ora2Pg (2012). Ora2pg software. http://ora2pg.darold.net/.

Patel, R., Brose, G., and Silverman, M. (2006). *Mastering Enterprise JavaBeans 3.0*. Wiley Pub.

St.Laurent, S., Dumbill, E., and Gruber, E. (2012). *Learning Rails 3*. O'Reilly.

Viana, E. and Borba, P. (1999). Integrando java com bancos de dados relacionais. *III Simpósio Brasileiro de Linguagens de Programação, Porto Alegre Brasil*, 5.