UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE TECNOLOGIA

MESTRADO EM TECNOLOGIA

Gustavo Bartz Guedes

Projeto evolutivo de bases de dados: uma abordagem iterativa e incremental usando modularização de bases de dados

Limeira, 2014

Gustavo Bartz Guedes

Projeto evolutivo de bases de dados: uma abordagem iterativa e incremental usando modularização de bases de dados

Evolutionary database design: an iterative and incremental approach using database modularization

Dissertação apresentada ao Curso de Mestrado da Faculdade de Tecnologia da Universidade Estadual de Campinas, como requisito para a obtenção do título de Mestre em Tecnologia.

Área de Concentração: Tecnologia e Inovação

Orientadora: Profa. Dra. Gisele Busichia Baioco

Co-orientadora: Profa. Dra. Regina Lúcia de Oliveira Moraes

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO GUSTAVO BARTZ GUEDES, E ORIENTADA PELA PROFA. DRA. GISELE BUSICHIA BAIOCO.

____

Limeira, 2014.

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Faculdade de Tecnologia
Vanessa Evelyn Costa - CRB 8/8295

G934p	Guedes, Gustavo Bartz, 1983- Projeto evolutivo de bases de dados : uma abordagem iterativa e incremental usando modularização de bases de dados / Gustavo Bartz Guedes. – Limeira, SP : [s.n.], 2014. Orientador: Gisele Busichia Baioco. Coorientador: Regina Lúcia de Oliveira Moraes. Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Tecnologia. 1. Métodos ágeis. 2. Bases de dados evolutivas. 3. Evolução de bases de dados. I. Baioco, Gisele Busichia. II. Moraes, Regina Lúcia de Oliveira. III. Universidade Estadual de Campinas. Faculdade de Tecnologia. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Evolutionary database design : an iterative and incremental approach using database modularization
Palavras-chave em inglês:
Agile methods
Evolutionary databases
Database evolution
Área de concentração: Tecnologia e Inovação
Titulação: Mestre em Tecnologia
Banca examinadora:
Gisele Busichia Baioco [Orientador]
João Eduardo Ferreira
Hilda Carvalho de Oliveira
Data de defesa: 11-02-2014
Programa de Pós-Graduação: Tecnologia

DISSERTAÇÃO DE MESTRADO EM TECNOLOGIA

ÁREA DE CONCENTRAÇÃO: TECNOLOGIA E INOVAÇÃO

Projeto evolutivo de bases de dados: uma abordagem iterativa e incremental usando modularização de bases de dados

Gustavo Bartz Guedes

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Gisele Busichia Baioco
FT-Unicamp
Presidente

Prof. Dr. João Eduardo Ferreira
IME-USP

Profa. Dra. Hilda Carvalho de Oliveira
UNESP

Resumo

Sistemas de software evoluem ao longo do tempo devido a novos requisitos ou a alterações nos já existentes. As mudanças são ainda mais presentes nos métodos de desenvolvimento de software iterativos e incrementais, como os métodos ágeis, que pressupõem a entrega contínua de módulos operacionais de software. Os métodos ágeis, como o Scrum e a Programação Extrema, são baseados em aspectos gerenciais do projeto e em técnicas de codificação do sistema. Entretanto, mudanças nos requisitos provavelmente terão reflexo no esquema da base de dados, que deverá ser alterado para suportá-los. Quando o sistema se encontra em produção, alterações no esquema da base de dados são onerosas, pois é necessário manter a semântica dos dados em relação à aplicação. Portanto, este trabalho de mestrado apresenta o processo evolutivo de modularização de bases de dados, uma abordagem para projetar a base de dados de modo iterativo e incremental. A modularização é executada no projeto conceitual e amplia a capacidade de abstração do esquema de dados gerado facilitando as evoluções futuras. Por fim, foi desenvolvida uma ferramenta que automatiza o processo evolutivo de modularização de bases de dados, chamada de *Evolutio DB Designer*. Essa ferramenta permite modularizar o esquema da base de dados e gerar automaticamente o esquema relacional a partir dos módulos de bases de dados.

Palavras-chave: evolução de bases de dados, bases de dados evolutivas, métodos ágeis.

Abstract

Software systems evolve through time due to new requirements or changing in the existing ones. The need for constant changes is even more present on the iterative and incremental software development methods, such as those based on the agile methodology, that demand continuous delivery of operational software modules. The agile development methods, like Scrum and Extreme Programming, are based on management aspects of the project and techniques for software coding. However, changes in the requirements will probably affect the database schema, which will have to be modified to accommodate them. In a production system, changes to the database schema are costly, because from the application's perspective the data semantics needs to be maintained. Therefore, the present work presents the evolutionary database modularization design process, an approach for the iterative and incremental design of the database. The modularization process is executed during the conceptual design improving the abstraction capacity of the generated data schema resulting in a graceful schema evolution. In addition, a tool that automates the evolutionary database modularization design process was developed, called Evolutio DB Designer. It allows the modular design of the database schema and automatically generates the relational data schema based on the database modules.

Keywords: database evolution, evolutionary databases, agile methods.

Sumário

1 Introdução 1

1.1 Motivações e Justificativas 1

1.2 Objetivo e principais contribuições 3

1.3 Organização do trabalho 4

2 Trabalhos correlatos 5

2.1 Evolução de esquemas de bases de dados 5

2.2 Refatoração de bases de dados 6

2.3 Coevolução de esquemas de dados 9

2.4 Análise do impacto de alterações em esquemas de bases de dados 11

2.5 Operadores de modificação de esquemas 13

2.6 Desenvolvimento iterativo e incremental 16

2.7 Considerações finais 20

3 Referencial teórico 23

3.1 Sistema de gerenciamento de bases de dados, base de dados, esquema de dados e modelo de dados 23

3.2 Projeto de base de dados 24

3.3 Projeto conceitual, modelo entidade-relacionamento estendido 26

3.4 Projeto lógico - modelo relacional 30

3.5 Engenharia reversa do modelo relacional para o modelo entidade-relacionamento 32

3.6 Considerações finais ... 34

4 Modularização de bases de dados ... 35

4.1 Visão geral sobre modularização de bases de dados ... 35

4.2 Projeto de modularização de bases de dados ... 37

4.3 Integração de módulos de bases de dados ... 40

4.4 Considerações finais ... 44

5 Processo evolutivo de modularização de bases de dados ... 45

5.1 Fases do processo evolutivo de modularização de bases de dados ... 45

5.2 Análise Evolutiva dos Requisitos de Modularização ... 48

5.3 Projeto Evolutivo de Modularização ... 49

5.4 Integração incremental dos módulos de bases de dados ... 52

5.5 Considerações finais ... 55

6 Aplicação do processo evolutivo de modularização de bases de dados ... 57

6.1 Descrição do projeto ... 57

6.2 Processo evolutivo de modularização de bases de dados ... 58

6.2.1 Primeira iteração ... 59

6.2.2 Segunda iteração ... 61

6.2.3 Terceira e quarta iterações ... 63

6.2.4 Quinta iteração ... 66

6.2.5 Sexta iteração ... 68

6.3 Considerações finais ... 70

7 *Evolutio DB Designer*: ferramenta de modularização de bases de dados 73

7.1 Arquitetura da *Evolutio DB Designer* 73

7.2 Recursos da *Evolutio DB Designer* 75

7.3 Telas e formulários da *Evolutio DB Designer* 77

7.4 Definição automatizada dos módulos de bases de dados e sua implantação 81

7.5 Considerações finais 83

8 Estudo de caso 85

8.1 Escolha do estudo de caso 85

8.2 Descrição do estudo de caso 86

8.3 Execução do estudo de caso 90

8.3.1 Versão inicial - v01284 91

8.3.2 Versão v04925 93

8.3.3 Versão v05648 94

8.3.4 Versão v06072 98

8.3.5 Versão v06710 100

8.3.6 Versão v09367 104

8.3.7 Versão v20301 105

8.3.8 Versão v20468 106

8.4 Considerações finais 107

9 Conclusão 111

10 Referências bibliográficas 115

Apêndice A - PHP e Twig Template Engine 121

Apêndice B - Mapeamento objeto-relacional com Doctrine... 125

Apêndice C - API de Abstração de Base de Dados Relacional 131

Apêndice D - Publicação.. 135

Dedico este trabalho aos meus pais, que sempre me apoiaram e me incentivaram a estudar.

Agradecimentos

Agradeço à Profa. Dra. Gisele Busichia Baioco por ter acreditado em mim, ter confiado em meu trabalho e por ter conduzido brilhantemente seu papel de orientadora.

À minha co-orientadora Profa. Dra. Regina Lúcia de Oliveira Moraes pelas valiosas contribuições, disponibilidade e disposição.

Agradeço também por terem ido além e me apoiado quando optei pela carreira acadêmica, ingressando como professor no Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP). Saibam que fizeram (e fazem) parte dessa conquista. Muito obrigado!

À Faculdade de Tecnologia, por ter me recebido de braços abertos, quando decidi retornar à vida acadêmica.

À Regina Bernardo da Luz, da Diretoria Geral de Recursos Humanos da Unicamp, que me deu a liberdade necessária para iniciar os meus estudos na pós-graduação.

Aos colegas do IFSP, pelas palavras de incentivo e conforto nos momentos difíceis.

Aos meus pais e meu irmão Fernando, sem vocês jamais teria chegado aqui. Devo grande parte de minhas conquistas a vocês.

Agradeço à Mirian, pelo incentivo, pela paciência, pelo amor e carinho.

Lista de Figuras

Figura 2.1 - Aplicação interagindo diretamente com o esquema relacional 10

Figura 2.2 - Aplicação interagindo por meio do esquema conceitual 10

Figura 2.3 - Arquitetura da *Hecataeus* (PAPASTEFANATOS et al., 2008). 12

Figura 2.4 - Arquitetura PRISM (CURINO et al., 2009). 15

Figura 2.5 - Exemplo de evolução de esquema (CURINO et al., 2009). 15

Figura 2.6 - Manifesto ágil publicado pela Agile Alliance traduzido para o português brasileiro (BECK, et al., 2001). 17

Figura 2.7 - Processo Scrum (Scrum Alliance apud Patuci, 2013). 18

Figura 3.1 - Fases de um projeto de base de dados (ELMASRI; NAVATHE, 2003). 25

Figura 3.2 - Exemplo de diagrama entidade-relacionamento. 27

Figura 3.3 - Representação de generalização/especialização (Busichia, 1999). 29

Figura 3.4 - Representação de agregação (HEUSER, 2009). 29

Figura 3.5 - Representação textual de um esquema de uma relação. 31

Figura 3.6 - Atributos e tuplas da relação *Empregado*. 31

Figura 3.7 - Atributos e tuplas da relação *Departamento*. 31

Figura 4.1 - Processo de projeto de base de dados incluindo a modularização (BUSICHIA, 1999). 36

Figura 4.2 - Representação de um módulo de base de dados (BUSICHIA, 1999). 40

Figura 4.3 - Objeto integrador para integração de módulos de bases de dados (BUSICHIA, 1999). 41

Figura 4.4 - Diagrama de fluxo da arquitetura do objeto integrador (BUSICHIA, 1999). 42

Figura 4.5 - Esquema de dados do objeto integrador (BUSICHIA, 1999). 43

Figura 5.1 - Processo evolutivo de modularização de bases de dados. 46

Figura 5.2 - Diagrama de atividades: Análise Evolutiva dos Requisitos de Modularização. 48

Figura 5.3 - Representação de uma inter-generalização. 50

Figura 5.4 - Mapeamento tradicional de inter-relacionamento. 50

Figura 5.5 - Mapeamento flexível de inter-relacionamento. 50

Figura 5.6 - Esquema conceitual estendido do catálogo do objeto integrador. 53

Figura 5.7 - Arquitetura de um Sistema de Bases de Dados incluindo a Camada do Objeto Integrador. 54

Figura 6.1 - Esquema conceitual de dados da primeira iteração mostrando os subesquemas. 60

Figura 6.2 - Tipo de compartilhamento de cada elemento do esquema da primeira iteração. 60

Figura 6.3 - Definição dos módulos da primeira iteração. 61

Figura 6.4 - Módulos da base de dados gerados na primeira iteração. 61

Figura 6.5 - Esquema conceitual de dados da segunda iteração mostrando os subesquemas. 62

Figura 6.6 - Tipo de compartilhamento de cada elemento do esquema da segunda iteração. 62

Figura 6.7 - Definição dos módulos da segunda iteração. 63

Figura 6.8 - Módulos de bases de dados gerados na segunda iteração. 63

Figura 6.9 - Resumo do Projeto Evolutivo de Modularização na quarta iteração. 64

Figura 6.10 - Esquema de dados do módulo M1 na quarta iteração. 65

Figura 6.11 - Esquema de dados do módulo M2 na quarta iteração. 65

Figura 6.12 - Esquema de dados do módulo M3 na quarta iteração. 65

Figura 6.13 - Esquema de dados do módulo M4 na quarta iteração. 66

Figura 6.14 - Resumo do Projeto Evolutivo de Modularização na quinta iteração. 67

Figura 6.15 - Esquema de dados do módulo M5 na quinta iteração. 67

Figura 6.16 - Esquema de dados do módulo M1, que foi estendido na quinta iteração..... 68

Figura 6.17 - Resumo do Projeto Evolutivo de Modularização na sexta iteração. 69

Figura 6.18 - Esquema de dados do módulo M1 na sexta iteração. 69

Figura 6.19 - Esquema de dados do módulo M6 na sexta iteração. 70

Figura 6.20 - Esquema de dados do módulo M7 na sexta iteração. 70

Figura 7.1 - Arquitetura da *Evolutio DB Designer*. 74

Figura 7.2 - Fluxo de uso dos recursos da *Evolutio DB Designer*. 75

Figura 7.3 - Formulário para criação de conjunto de entidades. 78

Figura 7.4 - Formulário para criação de conjunto de relacionamentos. 78

Figura 7.5 - Formulário para criação de uma generalização. 79

Figura 7.6 - Definição de um subsistema. 79

Figura 7.7 - Matriz de compartilhamento da informação. 80

Figura 7.8 - Definição dos módulos de bases de dados. 80

Figura 7.9 - Registro de SGBD. 82

Figura 8.1 - Arquitetura do MediaWiki (CURINO et al., 2008). 87

Figura 8.2 - Comparação de esquemas na ferramenta DB Solo.. 88

Figura 8.3 - Tabela *user*, exemplos de desnormalização no esquema relacional do MediaWiki.. 91

Figura 8.4 - Esquema conceitual parcial de S1 e S2. 91

Figura 8.5 - Versão v01284: compartilhamento da informação (gerada pela ferramenta *Evolutio DB Designer*). 92

Figura 8.6 - Versão v01284: Módulos da bases de dados da primeira versão do MediaWiki (gerada pela ferramenta *Evolutio DB Designer*). 93

Figura 8.7 - Inclusão do CR *participates* no esquema conceitual de dados (formulário da ferramenta *Evolutio DB Designer*)... 94

Figura 8.8 - Versão v05648: edição do subsistema S1 (formulário da ferramenta *Evolutio DB Designer*).. 95

Figura 8.9 - Versão v05648: esquema conceitual parcial de S1 e S8. 95

Figura 8.10 - Versão v05648: compartilhamento da informação (gerada pela ferramenta *Evolutio DB Designer*). 96

Figura 8.11 - Versão v05648: módulos de BD (gerada pela ferramenta *Evolutio DB Designer*). 97

Figura 8.12 - Esquema relacional gerado com base no esquema conceitual da Figura 8.9. 98

Figura 8.13 - Versão v06072: inclusão do CR *has8* (formulário da ferramenta *Evolutio DB Designer*). 98

Figura 8.14 - Versão v06072: edição do subsistema S8 (formulário da ferramenta *Evolutio DB Designer*).. 99

Figura 8.15 - Compartilhamento da informação (gerada pela ferramenta *Evolutio DB Designer*)......99

Figura 8.16 - Versão v06072: módulos de BD (gerada pela ferramenta Evolutio DB Designer). 100

Figura 8.17 - Estrutura das tabelas *cur* e *old*. 101

Figura 8.18 - Versão v06710: esquema físico das tabelas que mantém os dados de artigos. 101

Figura 8.19 - Versão v06710: esquema conceitual de dados, para manutenção dos artigos, modificado. 103

Figura 8.20 - Versão v06710: extensão do módulo M4 (gerada pela ferramenta *Evolutio DB Designer*). 104

Figura 8.21 - Esquema físico de parte do módulo M4, responsável pelo armazenamento das páginas. 104

Figura 8.22 - Versão v20301: extensão do módulo M4 (gerada pela ferramenta *Evolutio DB Designer*). 105

Figura 8.23 - Versão v20301: esquema físico atualizado das tabelas que mantém os artigos. 106

Figura A.1 – Arquitetura da MVC (SOMMERVILLE, 2011) 121

Figura A.2 - Arquitetura de um *Template Engine* 122

Figura A.3 - Exemplo de uso do Twig 123

Figura B.4 - Exemplo de Diagrama Entidade-Relacionamento. 127

Figura B.5 - Mapeamento no Doctrine do Conjunto de Entidades Funcionário 128

Figura B.6 - Mapeamento no Doctrine do Conjunto de Entidades Departamento 129

Figura C.7 - Mapeamento no DBAL 131

Figura C.8 - Comparação de esquemas com Comparator 133

Lista de Quadros

Quadro 2.1 - Estratégias de evolução de bases de dados (AMBLER, 2003). 7

Quadro 2.2 - Operadores modificadores de esquemas (CURINO; MOON; ZANIOLO, 2008)... 14

Quadro 3.1- Conceitos do modelo entidade-relacionamento (BATINI; CERI; NAVATHE, 1991)... 27

Quadro 3.2 - Mapeamento entre o ME-R e o modelo relacional. 32

Quadro 3.3 - Regras para engenharia reversa do modelo relacional para o modelo entidade-relacionamento (HEUSER, 2009). 33

Quadro 6.1 - Requisitos de sistema para escola de caratê, adaptado de AMBLER (2004). 57

Quadro 6.2 - Subsistemas definidos em cada iteração, de acordo com a saída da fase de “Análise Evolutiva dos Requisitos de Modularização”. 58

Quadro 7.1- Lista de ferramentas e aplicações utilizadas no desenvolvimento da Evolutio DB Designer. 73

Quadro 7.2 - Recursos da *Evolutio DB Designer* e as fases ou etapas correspondentes no processo evolutivo de modularização de bases de dados. 77

Quadro 7.3 - Mapeamento dos elementos do esquema conceitual de dados de cada módulo de bases de dados para o esquema relacional. 82

Quadro 8.1 - Número de alterações realizadas no esquema de dados (CURINO et al., 2008). 87

Quadro 8.2 - Versões do MediaWiki utilizadas no estudo de caso. 89

Quadro 8.3 - Agrupamento de tabelas por funcionalidades associadas (CURINO et al., 2008)... 90

Quadro 8.4 - Lista de subsistemas do MediaWiki da versão inicial (v01284)................... 92

Quadro 8.5 - Correspondência entre colunas das tabelas cur, old, page, revision e text. 102

Quadro B.1 - Lista de anotações mais comuns (WAGE et. al, 2013b)............................ 125

Quadro B.2 - Tipos do Doctrine... 126

Quadro C.3 - Principais métodos da classe SchemaManager.. 132

Lista de Abreviaturas e Siglas

Siglas

API: *Application programming interface*

BD: Base de Dados

CE: Conjunto de Entidades

CR: Conjunto de Relacionamentos

DBAL: *Database Abstraction & Access Layer*

DDL: *Data Definition Language*

DER: Diagrama Entidade-Relacionamento

ME-R: Modelo Entidade-Relacionamento

MVC: Modelo-Visão-Controlador

ORM: *Object-Relational Mapping*

PO: *Product Owner*

SGBD: Sistema de Gerenciamento de Bases de Dados

SMO: *Schema Modification Operator*

SQL: *Structured Query Language*

XP: Extreme Programming

1 Introdução

A maioria dos sistemas de software depende de uma camada de persistência para garantir a durabilidade dos dados. Essa camada é apoiada por um Sistema de Gerenciamento de Bases de Dados (SGBD), que proporciona os recursos necessários para a definição de esquemas de bases de dados, além de fornecer uma interface para realizar operações de leitura, inclusão, atualização e remoção de dados. Com isso um sistema de software manipula dados sem necessitar dos detalhes do armazenamento físico nos dispositivos de hardware.

Um sistema de software não é estático, estando sujeito a modificações durante seu ciclo de vida, que são provocadas por novos requisitos. Essas alterações podem ser encaradas como uma evolução que ocorre no sistema.

Os métodos de desenvolvimento de sistemas existentes são mais voltados para a codificação. Alguns exemplos de técnicas utilizadas incluem a "programação em pares" e o "desenvolvimento orientado a testes" que são aplicados nos métodos ágeis de desenvolvimento de software (SOMMERVILLE, 2011). Cada evolução requer a análise das mudanças necessárias e seus impactos em cada nível do sistema de software, o que inclui o esquema da base de dados, que deverá ser alterado.

Uma vez que a base de dados é o repositório de persistência dos dados da aplicação, é preciso que as alterações sejam realizadas de modo seguro, para não comprometer a consistência dos dados, principalmente das partes que já se encontram em produção.

1.1 Motivações e Justificativas

As bases de dados são partes integrantes de um sistema de software, portanto é necessário estabelecer diretrizes para projetá-las de modo flexível para que sejam flexíveis às mudanças futuras.

O cenário de mudanças constantes é ainda mais intenso quando métodos de desenvolvimento iterativos e incrementais são utilizados, em que a entrega do sistema é feita de maneira

modular, o que pode demandar modificações no esquema da base de dados a cada novo incremento de software.

Em comparação com os métodos ágeis, o modelo tradicional de projeto de base de dados, executado serialmente, com as fases de projeto conceitual, lógico e físico, não considera dois fatores importantes: a complexidade do domínio da aplicação e a velocidade das mudanças nos requisitos de negócio (DOMINGUES; KON; FERREIRA, 2009).

Atualmente, a refatoração é a técnica utilizada para a evolução do esquema da base de dados e consiste em pequenas alterações com o objetivo de melhorar o projeto da base de dados (AMBLER; SADALAGE, 2006).

O estudo realizado por Curino et al. (2008) analisou as alterações realizadas durante quatro anos e meio na base de dados da Wikipedia. A pesquisa detectou que no período houve 171 versões de esquemas da base de dados. A análise dessas versões apontou que 55% das alterações foram realizadas diretamente no esquema de dados. Essas modificações resultaram em até 70% de falhas nas consultas realizadas, exigindo intervenção de desenvolvedores diretamente no código do software. Portanto, alterações no esquema de base de dados não podem ser a fonte de introdução de falhas no sistema de software.

A evolução da base de dados deve considerar uma análise que tenha como objetivo minimizar as mudanças estruturais no esquema da base de dados que possam interromper o uso da parte da aplicação que já se encontra em produção. Sem essa análise, poderá haver um comprometimento do cotidiano das organizações que utilizam o sistema de software.

Parte da dificuldade em se realizar uma evolução na base de dados se deve ao acoplamento entre o esquema de dados e o código do software. Portanto, alterações no esquema de dados terão impacto na aplicação.

Outro fator que dificulta a evolução é a escolha de um modelo de dados com baixa capacidade de abstração. Pois, a representatividade do modelo de dados escolhido tem relação direta com as atividades envolvidas na evolução do esquema da base de dados. Esquemas de dados baseados

em modelos de dados pouco representativos dificultam a evolução futura, pois possuem baixa qualidade na representação da realidade.

Com isso, constata-se que as alterações no esquema físico da base de dados devem ser minimizadas ou, quando necessárias, conduzidas de modo padronizado e seguro, evitando ocorrências de inconsistências nos dados e falhas no sistema de software.

1.2 Objetivo e principais contribuições

A modularização de sistemas de software e a consequente interoperabilidade entre os módulos é uma técnica já conhecida e largamente utilizada (SOMMERVILLE, 2011). Já a modularização de bases de dados, de Busichia (1999), é executada no projeto conceitual e visa subdividir o esquema de dados de acordo com as transações executadas pela aplicação, gerando subesquemas coesos e com baixo acoplamento.

Com o objetivo de facilitar a evolução futura, o esquema da base de dados deve ser rico em semântica, para que sua capacidade de abstração seja maior. Isso só é possível quando o esquema de dados é capaz de capturar o maior nível de detalhes possível. Portanto, o projeto da base de dados deve ser executado de modo a ampliar a capacidade de abstração do esquema de dados que será gerado.

A modularização de bases de dados é realizada no esquema conceitual de dados, que é resultado da fase conceitual do projeto tradicional de bases de dados. O modelo de dados utilizado é o de entidade-relacionamento, definido por Peter Chen (1976), sendo um modelo semântico que está mais relacionado com a representação da informação do que nas estruturas de armazenamento, aproximando-o da realidade do usuário, simplificando seu entendimento e consequentemente sua evolução.

Desse modo, este trabalho de mestrado apresenta o processo evolutivo de modularização de bases de dados, diminuindo o número de refatorações demandadas pela evolução do esquema de dados, estendendo o trabalho de Busichia (1999), produzindo esquemas de dados mais ricos em semântica.

A principal contribuição deste trabalho é a definição de diretrizes que estendem o processo de modularização de bases de dados para apoiar o projeto evolutivo de bases de dados dentro de uma abordagem iterativa e incremental.

Outra contribuição é a automatização do processo estendido resultando na ferramenta *Evolutio DB Designer*. Essa ferramenta permite modelar a base de dados de maneira modular e, a partir do esquema conceitual modularizado, gerar automaticamente o esquema físico para implantação em um SGBD relacional.

1.3 Organização do trabalho

A presente dissertação de mestrado está organizada da seguinte maneira:

- no Capítulo 2 são apresentados trabalhos correlatos na área de evolução de bases de dados, tais como a refatoração de esquemas, coevolução de esquema conceitual e físico, simulação e análise de impacto em esquemas de dados e extensão de operadores de modificação de esquemas;
- o Capítulo 3 apresenta o referencial teórico e contempla os modelos de dados entidade-relacionamento e relacional;
- a modularização de bases de dados, como proposta por Busichia (1999), é apresentada no Capítulo 4;
- o processo evolutivo de modularização de bases de dados é descrito no Capítulo 5;
- o Capítulo 6 mostra a aplicação do processo evolutivo de modularização de bases de dados, descrevendo sua aplicação ao longo de seis iterações;
- a ferramenta *Evolutio DB Designer* é apresentada no Capítulo 7, detalhando sua arquitetura e os recursos providos para a aplicação do processo evolutivo de modularização de bases de dados;
- o estudo de caso apresentado no Capítulo 8 demonstra o uso da *Evolutio DB Designer* na evolução do esquema de dados do MediaWiki, o software utilizado pela Wikipedia;
- o Capítulo 9 contém a conclusão deste trabalho de mestrado.

2 Trabalhos correlatos

Este capítulo apresenta os trabalhos correlatos na área de evolução de esquemas de base de dados, por meio do levantamento bibliográfico realizado neste trabalho de mestrado.

Os temas abordados têm início na Seção 2.1, com a definição de evolução de esquemas de bases de dados. A Seção 2.2 descreve a técnica de refatoração de base de dados. Em seguida, a Seção 2.3 apresenta um trabalho sobre a evolução do esquema conceitual de base de dados e o reflexo no esquema físico. A Seção 2.4 descreve o trabalho que resultou na ferramenta *Hecataeus*, que possibilita simular e analisar o impacto de alterações em esquemas relacionais de bases de dados. O PRISM *Workbench* é apresentado na Seção 2.5, tratando-se de uma ferramenta que apoia a extensão dos comandos de alteração de esquemas físicos para realizar operações de evolução. A Seção 2.6 descreve o desenvolvimento iterativo e incremental de sistemas, exemplificado por meio do método Scrum. Por fim, a Seção 2.7 apresenta as considerações finais do capítulo.

2.1 Evolução de esquemas de bases de dados

A evolução de esquemas de bases de dados não é um tópico recente, surgiu inicialmente pelos termos de reorganização ou reestruturação da base de dados, abordados por Shamkant e James (1976) e por Sockut e Goldberg (1979). O termo reorganização, segundo Sockut e Goldberg (1979), é definido como a alteração de aspectos relacionados à maneira como uma base de dados é organizada lógica ou fisicamente, como por exemplo, a inclusão de um atributo ou a alteração na multiplicidade de relacionamentos.

Algumas definições de evolução de esquemas de bases de dados são apresentadas a seguir.

> A evolução de um esquema é alcançada quando um sistema de base de dados permite modificações no esquema de base de dados sem a perda do conteúdo semântico dos dados existentes. (RODDICK; CRASKE; RICHARDS, 1994, p. 138)

> Evolução de esquema é a capacidade de se mudar esquemas implantados, ou seja, alterar estruturas de metadados que descrevem formalmente artefatos complexos, tais como bases de dados, mensagens, aplicativos e workflows (RAHM; BERNSTEIN, 2006, p. 30)

A evolução de uma base de dados resulta de uma ou mais transformações aplicadas ao esquema de dados. Formalmente, segundo Batini, Ceri e Navathe (1991), uma transformação recebe como entrada um esquema E_1 e, após sua aplicação, resulta na saída de um esquema E_2. Uma transformação pode ser classificada como:

- **Transformação que preserva a informação:** o conteúdo da informação não é alterado pela transformação;
- **Transformação que altera a informação**, sendo classificadas em:
    - **Com aumento da informação:** o conteúdo de informação é maior no novo esquema;
    - **Com redução na informação:** o conteúdo de informação é menor no novo esquema;
    - **Transformações que não permitem a comparação**: nos casos em que não é possível determinar se houve ou não aumento de informação.

Já a comparação entre dois esquemas, segundo Batini, Ceri e Navathe (1991), pode ser realizada por meio de suas habilidades em responder a consultas que sejam realizadas. Dados dois esquemas E_1 e E_2, se para cada consulta C expressa em E_1 existir uma consulta C' que pode ser expressa em E_2 e que retorna o mesmo resultado, então pode-se dizer que esses esquemas são equivalentes. Se não existir uma consulta C' correspondente em E_2, então é dito que o conteúdo de informação no esquema E_1 é maior que no esquema E_2.

2.2 Refatoração de bases de dados

Em Ambler e Sadalage (2006), uma refatoração é definida como uma pequena alteração no esquema de base de dados que visa melhorar seu projeto, sem alteração na semântica informacional e que mantém a semântica comportamental em relação à aplicação que a utiliza.

A semântica informacional está ligada à visão que o usuário possui da base de dados, ou seja, após uma refatoração, a informação deverá continuar existindo sem alteração de significado para o usuário da base de dados, mesmo que em termos estruturais mudanças tenham ocorrido no esquema de dados.

A semântica comportamental refere-se às funcionalidades da aplicação como um todo. Portanto, após uma refatoração, as funcionalidades já existentes devem ser preservadas e interoperar com as alterações realizadas.

O Quadro 2.1 mostra as estratégias de evolução descrevendo os pontos fortes e fracos de cada abordagem.

Quadro 2.1 - Estratégias de evolução de bases de dados (AMBLER, 2003).

Estratégia	Pontos Fortes	Pontos Fracos
Desistir	É de fácil aceitação.	• O esquema não evolui sem interferência externa. • Não surgirá uma ferramenta para resolver todos os problemas pela falta de atualização do esquema. • Os novos projetos de aplicações terão sua qualidade comprometida, pois devem ser adaptados ao esquema de dados inalterado. • A dificuldade de se incluir novas funcionalidades irá aumentar ao longo do tempo.
Release Big-bang	Entrega um novo esquema da base de dados.	• Abordagem de risco, pois será necessário projetar o sistema novamente e fazer a nova entrega de uma só vez. Dificilmente uma organização está disposta a refazer o projeto. • Difícil de realizar em paralelo ao desenvolvimento de novas funcionalidades. • Requer infraestrutura e processos sofisticados de testes.
Entrega Incremental	• Entrega um novo esquema de base de dados. • Provê um mecanismo para evoluir a base de dados continuamente. • Reduz o risco ao se definir um processo de mudança contínuo.	• Requer infraestrutura e processos sofisticados de testes. • É necessário alterar a cultura organizacional para apoiar o desenvolvimento incremental.

O trabalho de Ambler e Sadalage (2006) traz diretrizes que estão concentradas tanto nos aspectos gerenciais e interpessoais do projeto como em técnicas para o processo de refatoração. Sobre esse último, a abordagem está mais direcionada para o modelo de dados relacional, entretanto o modelo conceitual é utilizado para a comunicação entre a equipe do projeto e o cliente. O motivo é que a simplicidade e o poder semântico do modelo conceitual o aproximam da visão do usuário em relação aos dados.

A lista a seguir classifica os tipos de refatorações, segundo Ambler e Sadalage (2006). Os exemplos são baseados no modelo relacional:

- **Qualidade dos dados:** são alterações que visam melhorar a qualidade dos dados. Exemplo: adicionar restrições às colunas;
- **Mudanças estruturais:** alterações no esquema de base de dados. Exemplo: alterar o nome de uma coluna;
- **Arquitetural:** quando um determinado objeto é convertido em outro tipo. Exemplo: substituir tabelas por visões;
- **Desempenho:** alteração com o objetivo de se melhorar o desempenho da aplicação. Exemplo: criar coluna com valores pré-calculados;
- **Integridade referencial: v**isa garantir a integridade referencial. Exemplo: criar tabela de associação.

Uma refatoração segundo Ambler e Sadalage (2006) é composta pelas seguintes etapas:

1. **Alteração do esquema da base de dados.** Nessa tarefa são feitas alterações nos objetos da base de dados (tabelas, visões, gatilhos, *stored procedures*, etc.);
2. **Migração de dados para o novo esquema.** Essa tarefa é responsável pela migração dos dados do esquema antigo para o novo esquema. É comum existir a necessidade de transformação dos dados, como conversões de tipos e correções nas restrições de integridade referencial;
3. **Alteração no código fonte.** O código fonte deve refletir as alterações realizadas no esquema de dados. É importante ressaltar que, nessa tarefa, falhas podem ser introduzidas se não forem substituídas as referências ao antigo esquema ou ao fazê-lo de forma incorreta.

É importante diferenciar a refatoração da extensão da base de dados. A adição de novos itens, como tabelas e colunas, não configura uma refatoração, mas sim uma extensão da base de dados.

Refatorações em um esquema de base de dados que já está em produção é uma tarefa que exige atenção, pois pode introduzir falhas no sistema de software. Para evitá-las, os projetistas de

base de dados tendem a fazer uma modelagem excessiva na primeira versão do esquema de base de dados, muitas vezes introduzindo elementos adicionais com o objetivo de se prever e acomodar futuras alterações. O problema é que não há garantia de que a flexibilidade introduzida será útil, pois não há como antever novos requisitos e quais alterações serão realizadas nos requisitos já existentes. Logo, a inclusão prévia de requisitos pode acarretar em um nível de complexidade desnecessário que pode até mesmo dificultar e prejudicar a evolução do esquema no futuro. Este problema também está presente nos métodos ágeis, pois o esquema completo e final da base de dados não é conhecido a priori.

2.3 Coevolução de esquemas de dados

O trabalho de Cleve, Brogneaux e Hainaut (2010) aborda os impactos causados na aplicação, resultantes de alterações na base de dados. O objetivo da pesquisa é detectar e analisar o impacto ocasionado pela evolução da base de dados na aplicação. Segundo os autores o problema é agravado quando a aplicação gera, dinamicamente, suas consultas à base de dados.

Outra questão levantada refere-se ao uso de mapeamento objeto-relacional (*object-relational mapping*, ORM), comuns em *frameworks* do tipo modelo-visão-controlador (MVC). O ORM é uma técnica que visa eliminar as diferenças de paradigmas existentes entre aplicações desenvolvidas com orientação a objetos e bases de dados relacionais. O ORM fornece uma camada para a manipulação da base de dados por meio de objetos. Esta camada realiza, sob demanda, a conversão dos registros existentes em uma base de dados para objetos e o inverso quando os dados são enviados da aplicação para a base de dados. Para estabelecer a correspondência entre a base de dados e as classes, o desenvolvedor realiza um mapeamento através de metadados, que pode variar desde um repositório paralelo até a introdução de anotações diretamente no código da aplicação. Cleve, Brogneaux e Hainaut (2010) argumentam que a dificuldade em se evoluir uma aplicação que utiliza ORM reside no fato de que o esquema da base de dados e o mapeamento da camada de ORM podem evoluir independentemente, o que pode gerar inconsistências.

A solução de Cleve, Brogneaux e Hainaut (2010) está na aplicação de sucessivas transformações no esquema conceitual para a geração automática dos esquemas lógico e físico, além de estabelecer um mapeamento entre esses diferentes níveis de esquemas de dados. O trabalho de

Hainaut (2005) apresenta um modelo entidade-relacionamento genérico próprio e os operadores de transformações aplicados para se obter os esquemas lógico e físico.

Em Cleve, Brogneaux e Hainaut (2010), ao final das transformações, uma API (*Application Programming Interface*) provendo uma visão conceitual da base de dados, é gerada automaticamente. Ao invés da aplicação acessar a base de dados diretamente pelo esquema relacional (Figura 2.1) a API é utilizada para a manipulação dos dados pela visão conceitual (Figura 2.2). Com isso, alterações no esquema conceitual que preservam a semântica não demandam alterações no código da aplicação. Essa abordagem permite que as evoluções seguintes sejam automaticamente propagadas para os esquemas lógico e físico.

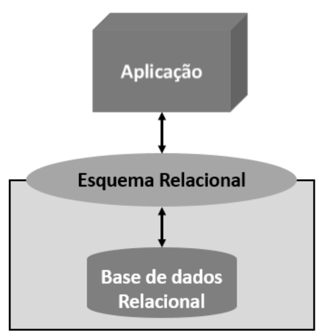


Figura 2.1 - Aplicação interagindo diretamente com o esquema relacional

(CLEVE; BROGNEAUX; HAINAUT, 2010).

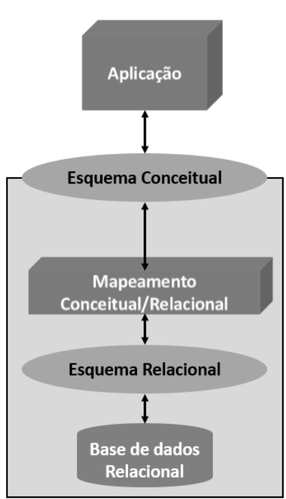


Figura 2.2 - Aplicação interagindo por meio do esquema conceitual

(CLEVE; BROGNEAUX; HAINAUT, 2010).

Cleve, Brogneaux e Hainaut (2010) classificam e fazem uma análise das alterações em relação aos impactos causados na aplicação e nos diferentes níveis de esquemas de base de dados, confrontando-os com a solução proposta:

1. **Alteração no esquema conceitual sem impacto na aplicação.** São mudanças que estão fora do escopo da visão externa que é utilizada pela aplicação. Em teoria esse tipo de evolução não tem impacto na aplicação. Entretanto, a extensão dessa propriedade depende da qualidade do projeto lógico e da tecnologia de mapeamento entre o esquema lógico e a visão externa e como a adaptação entre esses dois é feita, seja manual ou automática;

2. **Alteração no esquema conceitual com impacto na aplicação.** Esta situação ocorre quando alterações são realizadas em partes do esquema conceitual que afetam a visão externa utilizada pela aplicação. Nesse caso, a API pode ser recriada, mas é necessário alterar manualmente o código da aplicação para as novas referências existentes na API. A vantagem é que os locais em que as alterações devem ser feitas são facilmente identificados, uma vez que podem ser detectados em tempo de compilação, devido aos erros de sintaxe resultantes;
3. **Alteração no esquema lógico sem alteração no esquema conceitual.** Esta situação pode decorrer de alterações relacionadas aos requisitos não funcionais. Neste caso, apenas a API conceitual precisa ser recriada;
4. **Alteração no esquema físico sem alteração nos esquemas lógico e conceitual.** Estas alterações são realizadas e controladas no nível do SGBD, portanto não possui impacto na aplicação.

Um ponto importante a destacar no trabalho de Cleve, Brogneaux e Hainaut (2010) é que os impactos das alterações feitas no modelo conceitual sobre a aplicação podem ser detectados no momento da compilação. A razão é que as consultas são criadas estaticamente com referências explícitas no código da aplicação para o código da API gerada, logo uma quebra na referência será detectada na compilação do código da aplicação.

2.4 Análise do impacto de alterações em esquemas de bases de dados

A ferramenta *Hecateus* apresentada nos trabalhos de Papastefanatos et al. (2008) e Papastefanatos et al. (2010) provê mecanismos para simular alterações em um esquema de base de dados relacional e analisar a propagação dessas alterações. Também fornece um conjunto de métricas que possibilita avaliar o impacto em potencial das mudanças simuladas.

A *Hecataeus* mapeia elementos do esquema de dados e construções de base de dados relacionais, tais como consultas e visões, em um grafo orientado. Cada item é representando por um nó e as dependências são representadas por arestas. Também é possível adicionar metadados que

descrevem regras de restrições que são consideradas nas simulações de adições, remoções ou modificações realizadas diretamente no grafo. Um exemplo seria estabelecer que uma relação não pode ter seus atributos removidos.

A ferramenta possui uma interface gráfica para a visualização do grafo, permitindo ao usuário a manipulação direta através da interface. Os usuários podem elaborar cenários de evolução no grafo, especificando eventos hipotéticos (PAPASTEFANATOS et al., 2010). As partes afetadas pela simulação de evolução são destacadas visualmente no grafo, respeitando-se ainda as restrições definidas em cada nó.

Um conjunto de métricas foi definido para se avaliar a flexibilidade do esquema em relação a sua capacidade de evoluir. São elas:

- **O grau de um nó**, definido como a soma da quantidade de arestas de entrada e saída;
- **O grau de transitividade**, que é uma medida que representa as dependências de outros nós em relação ao nó em questão;
- **Grau de elemento agregado**, como por exemplo, uma visão, composta por diversos outros elementos.

A arquitetura da *Hecataeus*, apresentada na Figura 2.3, possui cinco componentes: *parser*, catálogo, gerenciador de evolução, visualizador de grafo e gerenciador de métricas.

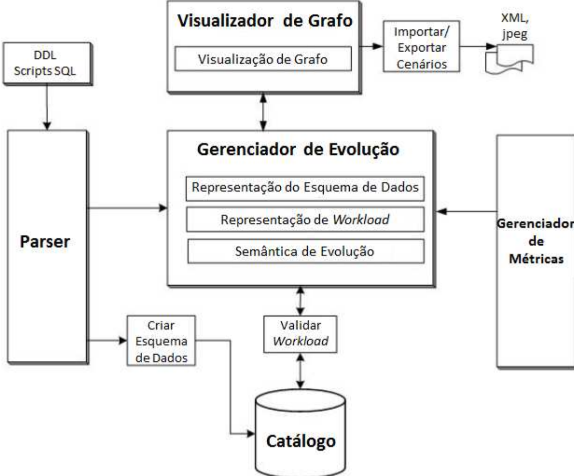


Figura 2.3 - Arquitetura da *Hecataeus* (PAPASTEFANATOS et al., 2008).

O *workload* se refere tanto ao esquema de dados representado por comandos de criação de estruturas de uma base de dados relacional por meio da *Data Definition Language* (DDL), quanto as construções (consultas, visões, etc).

A lista a seguir descreve cada um dos componentes:

- ***Parser*:** analisa um arquivo de entrada, que contém a estrutura do esquema de dados e envia cada comando para o catálogo e posteriormente para o gerenciador de evolução;
- **Catálogo:** mantém o esquema de dados e valida a sintaxe das instruções passadas. Estes dados são utilizados pelo gerenciador de evolução;
- **Gerenciador de evolução:** transpõe a representação do esquema da base de dados para um grafo. Também é responsável por associar cada nó e aresta ao seu respectivo tipo (tabela, consulta, etc);
- **Visualizador de grafo:** é responsável pela visualização do grafo e por fornecer uma interface ao usuário para efetuar alterações. As modificações realizadas pelo usuário são refletidas nos nós e arestas;
- **Gerenciador de métricas:** responsável por manter as métricas. Cada métrica é implementada como uma funcionalidade independente. Com isso, é possível estender, com maior facilidade, as funcionalidades da *Hecataeus*.

A *Hecataeus* permite apenas simular as alterações no grafo, não sendo possível aplica-las diretamente na base de dados. Esse recurso é indicado como trabalhos futuros no trabalho de Papastefanatos et al. (2008).

2.5 Operadores de modificação de esquemas

Um bom exemplo de operadores que alteram um esquema de dados são os comandos existentes na linguagem de definição de dados, a *Data Definition Language* (DDL), um subconjunto da linguagem de consulta estruturada, a *Structured Query Language* (SQL). Na DDL existem três classes de comandos: CREATE, DROP e ALTER, utilizados respectivamente na criação, remoção e alteração de elementos contidos em um esquema físico de uma base de dados relacional. Uma

evolução do esquema geralmente requer a execução de uma lista de comandos DDL, ou seja, carece de comandos únicos e de execução atômica, voltados para evolução do esquema. Muitas vezes é necessário realizar uma refatoração, combinando comandos e migrando os dados para as novas estruturas. É nesse contexto, da ausência de operadores que contemplem a evolução da base de dados, que o estudo de Curino, Moon e Zaniolo (2008) está inserido.

O estudo, realizado por Curino et al. (2008) analisou as alterações que ocorreram em um período de quatro anos e meio na base de dados da Wikipedia, em que 171 versões de esquemas de base de dados foram geradas. Os estudos apontaram que 55% das alterações foram realizadas diretamente no esquema de dados, enquanto 40% foram realizadas nos índices e chaves para se melhorar o desempenho da aplicação. Em algumas versões houve ocorrência de falha em cerca de 70% das consultas realizadas à base de dados, o que exigiu a intervenção de um desenvolvedor no código do software.

A definição de operadores de modificação de esquemas, *Schema Modification Operator* (SMO), apresentada por Curino, Moon e Zaniolo (2008), é em essência uma extensão dos comandos DDLs. O intuito é realizar as operações de evolução de esquemas de bases de dados por operações atômicas de alterações nesses esquemas. Os operadores são apresentados no Quadro 2.2 e foram utilizados na criação de uma ferramenta protótipo, a PRISM *workbench*, introduzida por Curino, Moon e Zaniolo (2008) e Curino et al. (2009).

Quadro 2.2 - Operadores modificadores de esquemas (CURINO; MOON; ZANIOLO, 2008).

Operador/Sintaxe	**Descrição**
CREATE TABLE R(A)	Cria a tabela R com os atributos A na base de dados.
DROP TABLE R	Remove a tabela R da base de dados.
RENAME TABLE R INTO T	Altera o nome da tabela R para T.
COPY TABLE R INTO T	Cria uma cópia da tabela R em T.
MERGE TABLE R, S INTO T	Funde as estruturas das tabelas R e S em T.
PARTITION TABLE R INTO S WITH *cond*, T	Particiona a tabela R em S e T. A tabela S terá os registros baseado em *cond*, já a tabela T em ¬*cond*.
DECOMPOSE TABLE R INTO $S(\overline{A},\overline{B})$, $T(\overline{A}, \overline{C})$	Decompõe a tabela R em S e T, com os respetivos atributos.
JOIN TABLE R, S INTO T WHERE *cond*	Funde as tabelas R e S em T com os registros de acordo com a condição especificada.
ADD COLUMN C INTO R	Adiciona a coluna C na tabela R.
DROP COLUMN C FROM R	Remove a coluna C da tabela R.
RENAME COLUMN B IN R TO C	Altera a coluna B da tabela R para C.

O objetivo da ferramenta é facilitar a evolução do esquema de dados, auxiliando o administrador da base de dados no planejamento e avaliação das alterações no esquema; realizar a reescrita automática de consultas legadas na versão mais atual do esquema; prover um repositório dos metadados que armazena o histórico das alterações, dando suporte a consultas legadas à base de dados. A Figura 2.4 mostra a arquitetura da PRISM, em que as alterações no esquema de dados são feitas em uma interface *web*, e são armazenadas em uma base de dados de metadados. As consultas passam a ser executadas no *Prism Runtime*, que se utiliza do repositório de metadados para reescrevê-las de acordo com as alterações realizadas no esquema de dados.

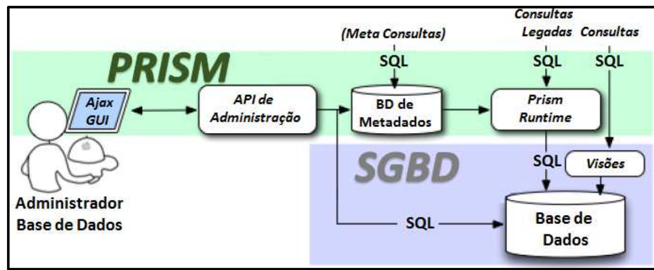


Figura 2.4 - Arquitetura PRISM (CURINO et al., 2009).

Existem cinco processos que guiam o usuário na utilização do PRISM. São eles:

1. **Configuração:** o usuário configura a ferramenta para se comunicar com um SGBD específico, sendo o esquema da base de dados carregado em um repositório de metadados.
2. **Aplicação dos SMOs:** na sequência o usuário poderá entrar com instruções SMO. A ferramenta apresenta os cenários de evolução com as alterações pretendias antes de aplicá-las no esquema da base de dados. Para exemplificar o uso dos operadores a Figura 2.5 mostra duas versões de esquema e as alterações realizadas com um SMO: `DECOMPOSE TABLE user INTO user rights(id, rights), user(id, name, password, email);`

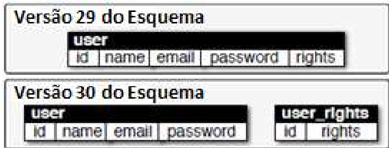


Figura 2.5 - Exemplo de evolução de esquema (CURINO et al., 2009).

3. **Plano de reversão de SMOs**: posteriormente, uma sequência de mudanças para reversão das alterações é automaticamente gerada. Esse passo é necessário para a reescrita de consultas legadas em versões anteriores do esquema. O usuário também pode fazer uma intervenção no plano de reversão gerado, com o objetivo de resolver problemas de ambiguidade quando existe mais de um SMO possível para a reversão. No exemplo da Figura 2.5, o SMO que inverte a alteração é: `JOIN TABLE user, user rights INTO user WHERE user.id = user rights.id`;
4. **Validação:** nesta fase, a ferramenta apresenta o mapeamento lógico entre as versões dos esquemas e possibilita testar se a reescrita das consultas legadas foi feita. No caso do exemplo da Figura 2.5, a consulta legada `SELECT * FROM user` é reescrita para `SELECT * FROM user, rights WHERE user.id = user rights.id`;
5. **Implantação:** esta é a última etapa, em que *scripts* para alteração do esquema e dos dados são gerados. O sistema provê duas maneiras para executar as consultas legadas. A primeira é através de um mecanismo de reescrita de consultas. A segunda é com a criação de visões que representam as versões antigas do esquema de dados.

Na ausência de ferramentas que abordem de forma prática a evolução de bases de dados, a PRISM supre muito bem essa falta ao fornecer um conjunto de instruções, os SMOs, de fácil compreensão, que permitem alterar o esquema da base de dados. Outro recurso importante é prover suporte às consultas legadas na versão mais atual do esquema. Isso é possível pelo suporte dado pelo catálogo de metadados, que mantém os procedimentos de evolução realizados pelos SMOs aplicados, a cada evolução, no esquema de dados da aplicação.

2.6 Desenvolvimento iterativo e incremental

Os métodos de desenvolvimento de sistemas de software existentes até meados da década de 90 possuem processos rigorosos e controlados, baseados em um extenso planejamento que trata os requisitos do sistema de maneira estática. Entretanto, esses processos dificilmente podem ser aplicados em sistemas corporativos de pequeno e médio porte, em que os requisitos não são estáveis, muitas vezes devendo ser alterados, mesmo durante seu desenvolvimento, para refletir novas necessidades de negócio. Ainda, esses métodos tendem a consumir mais recursos em atividades

que não estão ligadas diretamente ao desenvolvimento do sistema de software em si, mas sim no planejamento e na elaboração da documentação.

Segundo Sommerville (2011), a abordagem incremental para o desenvolvimento de um sistema de software "intercala as atividades de especificação, desenvolvimento e validação". Nessa abordagem a entrega é feita cumulativamente e continuamente com incrementos de software operacional, que adicionam novas funcionalidades às entregas anteriores.

O Manifesto Ágil foi concebido por um grupo de desenvolvedores, consultores e líderes de comunidade de software, insatisfeitos com as abordagens existentes. Estabeleceram então, as diretrizes que constam na Figura 2.6, a respeito do desenvolvimento ágil de software.

Os métodos ágeis possuem o mérito de reconhecer que existem sistemas de software que sofrem alterações em seus requisitos, evoluindo ao longo do tempo. Esses métodos partem de especificações de alto nível para a criação de um sistema inicial. Posteriormente, essas especificações são refinadas com informações provenientes do cliente. Esse ciclo é repetido até que o sistema resultante atenda às necessidades do cliente.

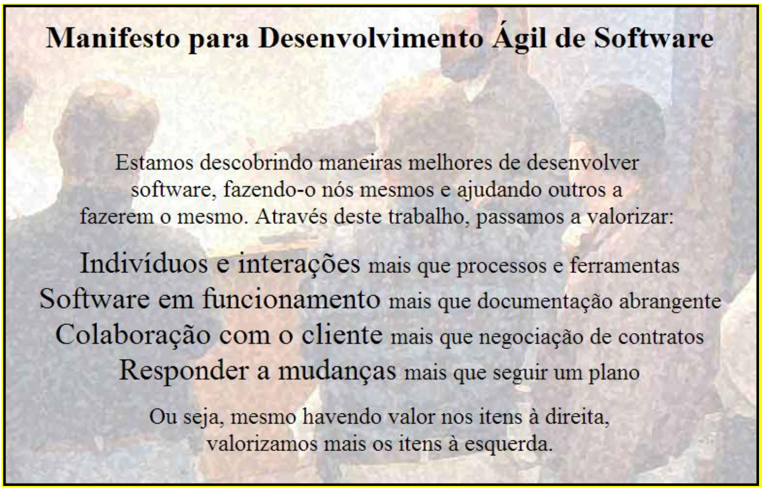


Figura 2.6 - Manifesto ágil publicado pela Agile Alliance traduzido para o português brasileiro (BECK, et al., 2001).

Sommerville (2008) sustenta que apesar das críticas, acredita-se que a abordagem ágil incorpora boas práticas de engenharia de software. Também possui processos bem definidos, leva em consideração as especificações do sistema, os requisitos dos usuários e tem altos padrões de

qualidade. Entretanto, aponta que é preciso analisar se os métodos ágeis e suas técnicas são adequados para o tipo de sistema que será desenvolvido e se é necessário realizar adaptações.

> [...] entusiastas dos métodos ágeis argumentam [...] que a chave para a implementação de software manutenível é a produção de códigos de alta qualidade, legíveis. Práticas ágeis, portanto, enfatizam a importância de se escrever códigos bem-estruturados e investir na melhoria do código. (SOMMERVILLE, 2011, p.41)

Existem diferentes métodos ágeis de desenvolvimento de software, fundamentados nos princípios do Manifesto Ágil, similares em essência, mas com processos diferentes para se atingir o desenvolvimento ágil. Scrum, Schwaber e Beedle (2001) e Schwaber (2004) e *Extreme Programming* (XP) de Beck (1999a, 1999b) são os exemplos mais conhecidos de métodos ágeis. O Scrum é voltado para práticas gerenciais, enquanto o XP aborda os aspectos técnicos do desenvolvimento. O processo do Scrum é mostrado na Figura 2.7.

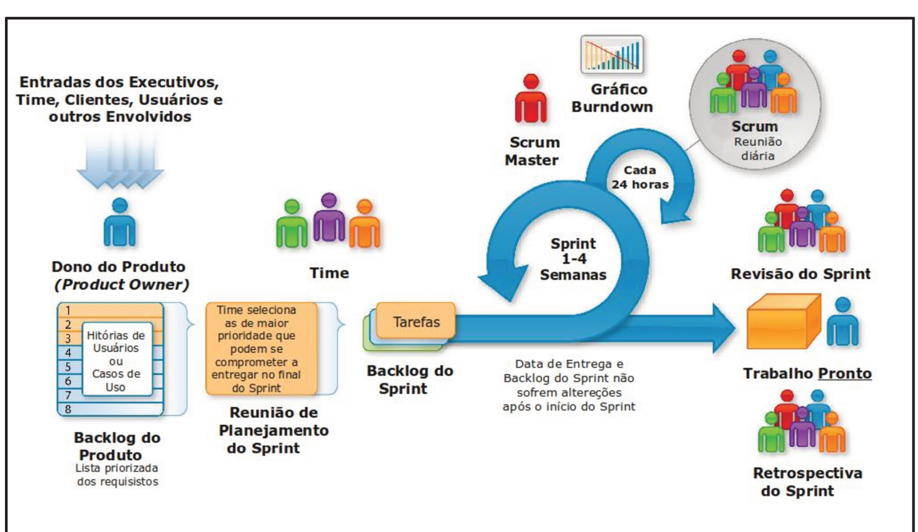


Figura 2.7 - Processo Scrum (Scrum Alliance apud Patuci, 2013).

O Scrum não especifica práticas de programação. Entretanto, descreve alguns papéis que devem ser atribuídos aos envolvidos no desenvolvimento do sistema:

- **Product Owner (PO)** é o principal *stakeholder* do projeto, sendo responsável por fornecer a visão do produto para a equipe de desenvolvimento. Também é sua função priorizar as funcionalidades que serão implementadas, de acordo com as

necessidades do negócio. Essas funcionalidades são mantidas em um artefato chamado de *backlog* do produto (*Product Backlog*);

- **Scrum Master** é responsável por garantir o sucesso do projeto, atuando como facilitador. Trabalha para que a cultura organizacional e o projeto estejam otimizados;
- A **equipe de desenvolvimento**, também chamada de **time**, é responsável por especificar e implementar as funcionalidades, respeitando as prioridades definidas pelo PO.

No Scrum cada ciclo de desenvolvimento, denominado de *Sprint*, possui um intervalo fixo de tempo, geralmente de uma a quatro semanas. Uma reunião de planejamento é conduzida com o objetivo de se elencar as funcionalidades que serão implementadas em um *Sprint*, dando origem ao *Backlog* do *Sprint*, uma lista priorizada de requisitos. Uma rápida reunião diária, chamada de *Daily Meeting*, é conduzida com o objetivo de se analisar o progresso e possíveis contratempos. Ao final de um *Sprint*, um produto parcial, mas operacional, é entregue e novos aportes são planejados para um novo período. Cada *Sprint* deve ser autocontido para que o produto possa ser entregue ao cliente de maneira modular.

Os *Sprints* deverão levar em consideração as entregas realizadas previamente. Os novos aportes deverão ser integrados com os módulos de sistema de software que estão operacionais e em uso pelo cliente.

Na prática, o Scrum é utilizado para o gerenciamento do projeto, enquanto o XP é aplicado ao desenvolvimento do sistema de software. Portanto, em termos de desenvolvimento, estão voltados para práticas de codificação, como a programação em pares, adoção de padrões de codificação e desenvolvimento orientado a testes. O projeto de base de dados acaba ocupando um papel secundário.

Portanto, uma vez que a base de dados é parte integrante da aplicação e responsável por manter os dados, é necessário estabelecer práticas que permitam evoluí-la de modo seguro, minimizando o impacto no código do sistema de software.

2.7 Considerações finais

Este capítulo apresentou trabalhos correlatos na área de evolução de bases de dados e também o desenvolvimento de software iterativo e incremental, exemplificado pelo método Scrum.

O trabalho de Ambler e Sadalage (2006) aborda a evolução de um esquema de base de dados por meio de pequenas alterações, chamadas de refatoração. Uma refatoração requer que a semântica informacional da base de dados seja preservada. Entretanto, uma refatoração pode introduzir falhas, devido ao acoplamento existente entre o esquema de dados e a aplicação.

Já o trabalho de Cleve, Brogneaux e Hainaut (2010) mostra que as alterações em um projeto de base de dados tem sua origem no projeto conceitual. Portanto, a solução apresentada define regras de transformações de um esquema conceitual, que sofreu alterações, para os esquemas lógico e físico. Portanto, o esquema conceitual, lógico e físico coevoluem. O acoplamento com a aplicação é resolvido pela geração de uma API, que permite o acesso à base de dados pelo esquema conceitual ao invés do esquema relacional. Entretanto, ainda é necessário fazer mudanças no código do software nos casos em que as alterações no esquema conceitual de dados afetam a visão externa que é utilizada pela aplicação.

A *Hecateus*, introduzida por Papastefanatos et al. (2008), permite simular as alterações em um esquema de base de dados relacional. As simulações são feitas em uma interface gráfica, colocando os elementos da base de dados em um grafo orientado, que representa a dependência entre os elementos que compõem o esquema relacional. A *Hecateus* simula as alterações, mas não provê mecanismos para alterar o esquema da base de dados.

Curino, Moon e Zaniolo (2008), apresentam a PRISM *workbench*, que implementa operadores, de execução atômica, para alteração de um esquema de dados relacional. Esses operadores são chamados de *Schema Modification Operators*. As alterações realizadas no esquema de dados são armazenadas em um repositório de dados como metadados, utilizados nas consultas legadas no novo esquema de dados evoluído.

O próximo capítulo apresenta o referencial teórico deste trabalho de mestrado. São discutidos os conceitos de bases de dados, sistemas de gerenciamento de bases de dados, modelos de

dados e esquema de dados. Posteriormente, são descritos o modelo de dados conceitual de entidade-relacionamento e o modelo relacional, utilizados respectivamente no projeto conceitual e lógico em um projeto de base de dados.

3 Referencial teórico

Neste capítulo são apresentadas as referências teóricas que serviram de base para a realização deste trabalho de mestrado.

Os temas a serem abordados têm seu início na Seção 3.1 com a definição de base de dados, modelo de dados e esquema de dados ressaltando características que influenciam na qualidade de um modelo de dados. Em seguida a Seção 3.2 descreve o projeto genérico de base de dados. A Seção 3.3 descreve o modelo conceitual de entidade-relacionamento estendido. Já a Seção 3.4 apresenta o modelo relacional, que é a utilizado na implementação de muitos SGBDs existentes. As regras da engenharia reversa do modelo relacional para o modelo entidade-relacionamento são apresentadas na Seção 3.5. Por fim, a Seção 3.6 refere-se as considerações finais do capítulo.

3.1 Sistema de gerenciamento de bases de dados, base de dados, esquema de dados e modelo de dados

Base de Dados é uma coleção de dados inter-relacionados, que representa conceitos e objetos do mundo real (ELMASRI; NAVATHE, 2003). Em computação, uma base de dados é projetada, implementada e possui um grupo de usuários que a utiliza, realizando as operações básicas de leitura, inclusão, remoção e atualização de dados. Essas operações são realizadas por meio de um SGBD, que proporciona os recursos necessários para se definir, construir e manipular bases de dados de modo transparente. Com isso, uma aplicação realiza operações sobre os dados sem necessitar de detalhes de como estão fisicamente armazenados nos dispositivos de armazenamento permanente.

Uma fase importante do projeto de base de dados é a elaboração do esquema de dados, resultado da transposição dos requisitos do sistema para um modelo de dados específico. Esse último fornece um conjunto de conceitos para representar as abstrações do mundo real em estruturas próprias pertencentes a um domínio, definido pelo tipo de modelo de dados escolhido. Logo, o poder de abstração das estruturas oferecidas por um modelo de dados contribui para a qualidade

do mesmo, o que por consequência influencia o processo de evolução da base de dados. Portanto, é necessário analisar as características que contribuem para a qualidade de um modelo de dados.

Segundo Batini, Ceri e Navathe (1991), as características listadas a seguir contribuem para determinar a representatividade de um modelo de dados:

- **Expressividade:** é a capacidade que um modelo tem de expressar abstrações por meio de conceitos bem definidos;
- **Simplicidade:** refere-se à facilidade de se compreender o modelo de dados. É possível que um modelo seja semanticamente rico, mas de difícil compreensão;
- **Ser mínimo:** indica se cada conceito no modelo tem significado distinto dos demais, não podendo ser expresso por meio da composição de outros;
- **Formalidade:** deve haver formalidade nas especificações do modelo. Os conceitos devem ser únicos, precisos e, principalmente, de interpretação bem definida. Além de que conceitos formais podem ser matematicamente manipulados.

Representações gráficas são utilizadas para aumentar a representatividade dos modelos. Os critérios listados a seguir influenciam na qualidade das representações gráficas, de acordo Batini, Ceri e Navathe (1991):

- **Completude:** cada conceito deve estar associado a uma representação gráfica, do contrário será necessário recorrer a recursos não gráficos para suprir essa falta. Essa ausência, muitas vezes, é suprida com o uso de recursos linguísticos, que podem levar a interpretações subjetivas e ambíguas;
- **Legibilidade:** deve haver distinção clara entre os símbolos gráficos do modelo para evitar interpretações errôneas.

3.2 Projeto de base de dados

Um projeto de base de dados tem seu início com a elicitação dos requisitos, dando origem aos requisitos de dados. Em paralelo, os requisitos funcionais do sistema são especificados, consistindo em um conjunto de transações que são enviadas à base de dados. Posteriormente, três fases,

em série, são executadas com o objetivo de se produzir esquemas de base de dados com diferentes níveis de detalhamento, até que se faça a implantação em um SGBD específico. A Figura 3.1 mostra as fases de um projeto de base de dados, segundo Elmasri e Navathe (2003).

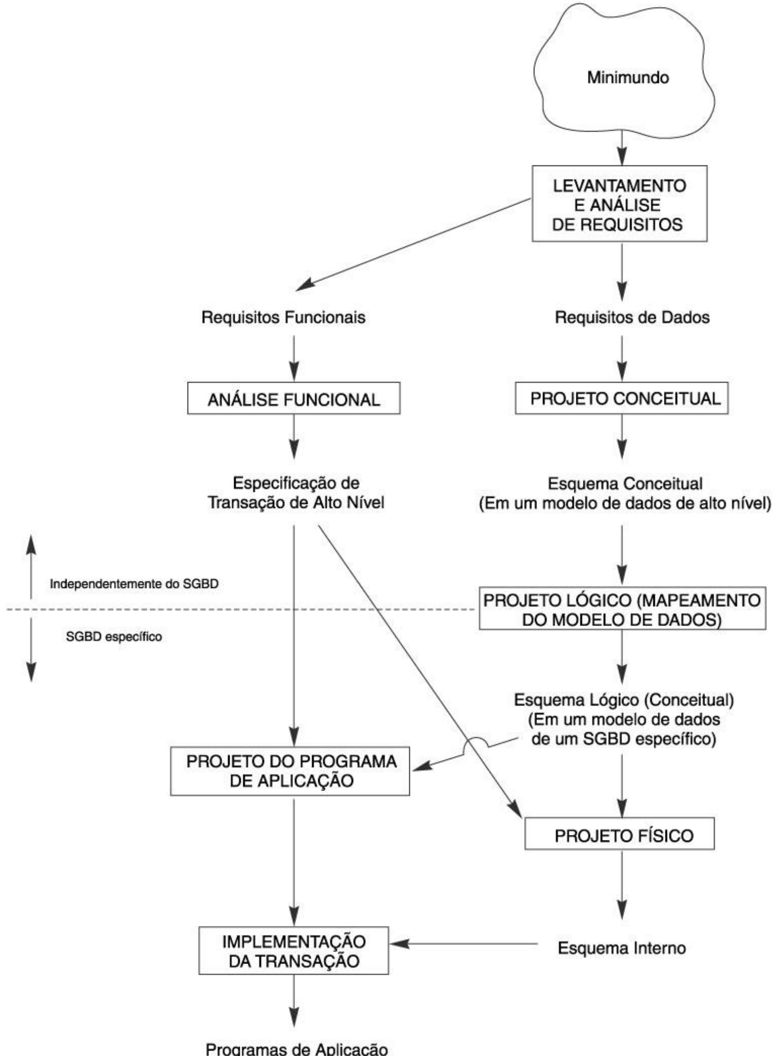


Figura 3.1 - Fases de um projeto de base de dados (ELMASRI; NAVATHE, 2003).

A lista a seguir apresenta a sequência de esquemas projetados ao longo de um projeto de base de dados:

- **Conceitual:** esquema de alto nível que omite detalhes de implementação e é independente de SGBD. Rico em construções semânticas e mais próximo das abstrações do mundo real;

- **Lógico:** é a transformação do esquema conceitual em um esquema que pode ser processado por uma categoria de SGBDs (ELMASRI; NAVATHE, 2003). Uma classe de SGBD é determinada pelo modelo de dados que este implementa. Atualmente, as classes mais comuns são: relacional (CODD, 1970), objeto-relacional (STONEBRAKER; MOORE, 1996) e orientado a objetos (ATWOOD et al., 1985), sendo o modelo relacional o mais utilizado nas implementações de SGBDs. Portanto, um esquema lógico não é construído para um SGBD específico, mas sim para uma classe de SGBDs;
- **Físico:** esquema elaborado de acordo com as características de um SGBD específico, além de possuir transformações que visam atender requisitos não funcionais da aplicação.

O projeto de uma base de dados pode ser conduzido através de duas abordagens. Uma delas, a ascendente (*bottom-up*), parte dos conceitos mais detalhados, dos níveis mais baixos de abstração, que são agrupados até se chegar ao esquema de dados. Já a estratégia descendente (*top-down*) inicia a partir dos conceitos mais abstratos, que são refinados até se chegar aos níveis mais baixos de abstração, portanto é o caminho inverso da estratégia ascendente.

3.3 Projeto conceitual, modelo entidade-relacionamento estendido

O Modelo Entidade-Relacionamento (ME-R), introduzido por Chen (1976), possui alta expressividade e é muito utilizado no projeto conceitual da base de dados.

O ME-R descreve os dados por meio dos conceitos de:

- **Entidade:** representa um objeto ou conceito do mundo real;
- **Relacionamento:** representa uma ligação (relacionamento) entre entidades;
- **Atributos:** descrevem as propriedades de entidades e relacionamentos.

Os conceitos do ME-R são utilizados na criação de uma representação gráfica, na forma de um diagrama, chamado de Diagrama Entidade-Relacionamento (DER). Os construtores básicos

oferecidos pelo ME-R, sua descrição e símbolos gráficos são apresentados no Quadro 3.1. Um exemplo de DER é apresentado na Figura 3.2.

Quadro 3.1- Conceitos do modelo entidade-relacionamento (BATINI; CERI; NAVATHE, 1991).

Conceito	Descrição	Símbolo Gráfico
Conjunto de Entidades (CE)	Representa uma coleção de entidades que possuem propriedades semelhantes.	
Conjunto de Relacionamentos (CR)	Representa uma coleção de relacionamentos que possuem propriedades semelhantes entre dois ou mais conjuntos de entidades.	
Atributos	Têm o objetivo de descrever as propriedades das entidades e relacionamentos, por meio de valores. São ligados aos seus respectivos CEs ou CRs.	

Um Conjunto de Entidades (CE) deve possuir um identificador, que é composto de um ou mais atributos que têm a característica de determinar unicamente cada uma das entidades contidas no CE. Os identificadores também são chamados de chaves ou de chaves candidatas. Na Figura 3.2 o atributo sublinhado *CPF* indica o identificador do CE *Empregado*.

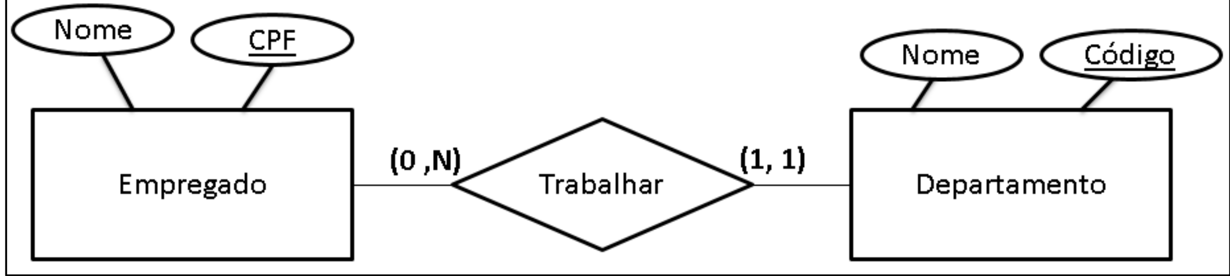


Figura 3.2 - Exemplo de diagrama entidade-relacionamento.

Em um Conjunto de Relacionamentos (CR) é possível atribuir uma restrição que define o número mínimo e máximo de entidades que podem participar do relacionamento. Essa restrição determina a multiplicidade no CR. Este tipo de restrição permite aumentar o potencial de representação dos requisitos dentro do modelo conceitual. A multiplicidade é definida por meio de um número inteiro, em que 0 (zero) indica que não existem ocorrências de entidades e N determina uma ocorrência ilimitada. Também é possível atribuir qualquer número inteiro positivo para determinar a restrição de multiplicidade mínima.

No DER apresentado na Figura 3.2, os números entre parênteses próximos ao CR *Trabalhar* indicam a multiplicidade mínima e máxima, respectivamente. No caso pode-se afirmar que um empregado deve estar associado a somente um departamento. Já um departamento pode estar associado a nenhum ou vários empregados.

Com a evolução dos paradigmas de programação, novos conceitos foram introduzidos, enriquecendo a semântica da representação da informação. A seguir, são apresentados os conceitos de generalização e especialização, que foram adicionados pelo ME-R estendido, os quais estão relacionados aos conceitos de superclasse e subclasse do paradigma de orientação a objetos.

Um CE representa uma classe genérica de dados. Entretanto, podem existir subconjuntos de entidades em um CE, que pertencem a classes distintas, podendo ser representadas separadamente em um CE específico com atributos próprios. Cada CE específico contém uma subclasse das entidades. Já as propriedades em comum a um grupo de subclasses são reunidas em um CE genérico, chamado de superclasse. A ligação estabelecida entre o CE genérico e os CEs específicos faz com que esses últimos herdem os atributos do CE genérico. Portanto, os atributos do CE genérico não devem ser representados nos CEs específicos.

O conceito de generalização é o processo de se identificar uma superclasse que representa genericamente um determinado subconjunto de entidades, resultando na criação de um CE que agrega os atributos em comum desse grupo. Já a especialização consiste em definir um ou mais CE específicos, com o objetivo de se agregar grupos de entidades com atributos próprios, determinados pelas especificidades de cada grupo. A generalização pode ser aplicada em vários níveis e possui característica hierárquica, logo é chamada de hierarquia de generalização.

A abstração de generalização ainda possui quatro restrições, divididas em duas categorias:

- **Disjunção**
    - **Exclusão mútua:** para cada entidade do CE genérico só pode haver uma ocorrência nos CEs específicos;
    - **Sobreposição:** permite que uma entidade do CE genérico tenha mais de uma ocorrência nos CEs específicos;

- **Completude**
    - **Participação total:** indica que para cada entidade no CE genérico deve haver pelo menos uma ocorrência em um dos CEs específicos;
    - **Participação parcial:** indica que uma entidade no CE genérico não necessita de ocorrências nos CEs específicos.

A Figura 3.3 mostra as representações utilizadas nas abstrações de generalização/especialização dentro do diagrama entidade-relacionamento estendido. As hastes indicam o tipo de completude: duplas para a participação total e simples para a participação parcial. Já o triângulo indica o tipo de disjunção, quando preenchido indica sobreposição e quando não preenchido indica exclusão mútua.

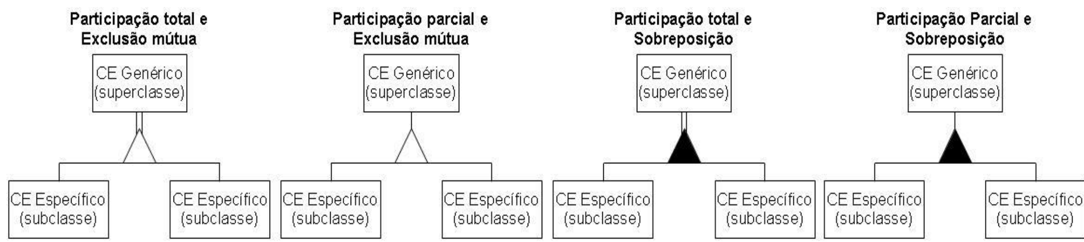


Figura 3.3 - Representação de generalização/especialização (Busichia, 1999).

No ME-R não é permitido associar dois conjuntos de entidades diretamente. Entretanto, essa situação é desejável em determinados casos quando da elaboração do DER. No ME-R estendido, a agregação consiste na criação de um CE baseado em um CR, conforme mostra a Figura 3.4. Esse novo CE possui as mesmas características de um CE do ME-R, podendo ter atributos próprios e se relacionar com outros CEs, por exemplo. Na literatura, o termo CE Associativo também é usado para denotar uma agregação (HEUSER, 2009).

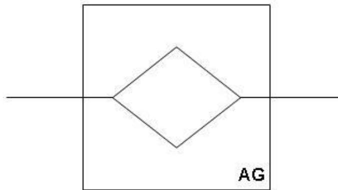


Figura 3.4 - Representação de agregação (HEUSER, 2009).

Formalmente, segundo Mesquita e Finger (1998) um DER estendido pode ser representado formalmente da seguinte maneira:

DER = {E, A, R, HG, AG, C} onde:

E = conjunto de entidades

A = conjunto de atributos

R = conjunto de relacionamentos

HG = conjunto de hierarquias de generalização

AG = conjunto de abstrações de agregação

C = conjunto de configurações do diagrama de acordo com:

- para cada entidade $e \in E$, $C(e) = <A_d, A_{nd}>$, onde $A_d \subseteq A$ é o conjunto de atributos determinantes de *e*, e $A_{nd} \subseteq A$ o conjunto de atributos não determinantes de *e*;
- para cada relacionamento $r \in R$, $C(r) = <e_1, e_2, c_{12}, c_{21}>$, onde c_{12} e c_{21} correspondem às restrições de multiplicidade;
- para cada hierarquia de generalização $hg \in HG$, $C(hg) = < e, \{e_i\}, tipo>$, onde $e \in E$ corresponde à classe de entidades genéricas da hierarquia, $\{e_i\} \subseteq R$ corresponde ao conjunto de entidades específicas e *tipo* indica se a hierarquia é total/parcial e disjunta/sobreponível;
- para cada abstração de agregação $\mathrm{ag} \in \mathrm{AG}$, $\mathrm{C(ag)} = < \mathrm{e}_1, \mathrm{e}_2, \mathrm{r}, \mathrm{A}_{\mathrm{ag}}>$, onde $\mathrm{e}_1, \mathrm{e}_2 \in \mathrm{E}$ correspondem a classes de entidades agregadas, $\mathrm{r} \in \mathrm{R}$ é o relacionamento original de ag e $\mathrm{A}_{\mathrm{ag}} \subseteq \mathrm{A}$ é o conjunto de atributos de ag.

3.4 Projeto lógico - modelo relacional

O projeto lógico de uma base de dados é comumente realizado usando o modelo relacional, introduzido por Codd (1970) e que se tornou um padrão dos SGBDs desenvolvidos. A adoção desse modelo se deve a sua simplicidade, pelo uso de sólidos conceitos matemáticos, por ser baseado na

teoria dos conjuntos e na lógica de primeira ordem (ELMASRI; NAVATHE, 2003). Ainda no artigo de Codd (1970) são apresentadas definições que abordam os problemas de redundância e consistência dos dados.

No modelo relacional a base de dados é representada como uma coleção de relações, comumente chamadas de tabela fora do meio acadêmico. Uma relação é composta por um conjunto de atributos, em que cada atributo possui um valor atômico. A Figura 3.5 é uma representação textual do esquema de uma relação.

Empregado(<u>*matricula*</u>: inteiro, *nome*: texto, *salario*: real, *num_dep*: inteiro)

Figura 3.5 - Representação textual de um esquema de uma relação.

No exemplo, o esquema da relação *Empregado* é definido com os atributos *matricula*, *nome*, *idade* e *num_dep*, cujos domínios são respectivamente inteiro, texto, real e inteiro. O conceito de tupla refere-se à instância de uma entidade pertencente à relação, respeitando o esquema definido. A quantidade e os valores contidos nas tuplas são variáveis e podem ser alteradas ao longo do tempo. As Figuras 3.6 e 3.7 mostram exemplos de relações e os valores contidos nas tuplas. Uma relação, portanto, é parecida com uma tabela de valores.

Empregado	<u>matricula</u>	nome	salario	num_dep
	123	Jorge	5000	10
	321	Sophia	4000	20
	455	Maria	2000	10
	987	Ronaldo	1500	30

Figura 3.6 - Atributos e tuplas da relação *Empregado*.

Departamento	<u>num_dep</u>	nome	local
	10	Vendas	São Paulo
	20	Produção	Campinas
	30	Compras	São Paulo

Figura 3.7 - Atributos e tuplas da relação *Departamento*.

O modelo relacional provê ainda algumas restrições, sendo as principais definidas por dois tipos de chaves:

- **Chave Primária:** um atributo ou combinação de atributos utilizados para identificar cada tupla. Por consequência não são permitidos valores duplicados para o atributo.

Os atributos que participam da chave primária são sublinhados no esquema textual do modelo relacional. No exemplo da Figura 3.6 o atributo *matricula* é a chave primária da relação *Empregado*;

- **Chave Estrangeira:** um atributo ou combinação de atributos, que indica que os valores a serem atribuídos para um atributo, devem, primeiramente, pertencer à chave primária em outra relação. Nos exemplos das Figuras 3.6 e 3.7 o atributo *num_dep* da relação *Empregado* é uma chave estrangeira que referencia o atributo *num_dep* da relação *Departamento*.

Por fim, o projeto lógico consiste na transformação do esquema conceitual, geralmente elaborado no ME-R, para o esquema lógico relacional, de acordo com regras de mapeamento. O Quadro 3.2 apresenta simplificadamente as regras para o mapeamento entre o ME-R e o modelo relacional.

Quadro 3.2 - Mapeamento entre o ME-R e o modelo relacional.

Modelo Entidade Relacionamento	**Modelo Relacional**
Conjunto de Entidades	Relação
Atributos	Atributos de Relação
Conjunto de Relacionamentos	Determinado pela multiplicidade máxima: • 1:1, um atributo é adicionado em qualquer uma das relações participantes do relacionamento no ME-R. • 1:N, um atributo é adicionado como chave estrangeira na relação que, como CE, participa do relacionamento com a multiplicidade N. • M:N, uma relação é criada com as chaves dos CEs, que participam do relacionamento.

3.5 Engenharia reversa do modelo relacional para o modelo entidade-relacionamento

A engenharia reversa pode ser aplicada em um esquema relacional já implementado para se obter o esquema de dados conceitual correspondente. Essa operação é útil quando a base de dados foi desenvolvida apenas com base na experiência do projetista e o esquema conceitual de dados não foi elaborado (HEUSER, 2009).

A engenharia reversa consiste, segundo Heuser (2009), de quatro passos:

1. Identificação da construção ER correspondente a cada relação;
2. Definição de relacionamentos 1:N e 1:1;
3. Definição de atributos;
4. Definição de identificadores.

O Quadro 3.3 apresenta as regras para aplicar a engenharia reversa em um esquema de base de dados relacional e se obter o esquema conceitual no modelo entidade-relacionamento.

Quadro 3.3 - Regras para engenharia reversa do modelo relacional para o modelo entidade-relacionamento (HEUSER, 2009).

Passo	**Relacional**	**Entidade-Relacionamento**
Identificação da construção ER correspondente a cada tabela	Chave primária composta por mais de uma chave estrangeira.	Resulta em um conjunto de relacionamentos com multiplicidade M:N.
	Toda a chave primária é também uma chave estrangeira.	Representa um conjunto de entidades especializado, em que a relação referenciada é o conjunto de entidades genérico.
	Demais casos.	Nos demais casos cada relação resulta em um conjunto de entidades.
Definição de relacionamentos 1:N e 1:1	Chave estrangeira que não foi definida como um conjunto de relacionamentos M:N.	Conjunto de relacionamentos com multiplicidade 1:N ou 1:1. Para se definir a multiplicidade com certeza é necessário analisar o conteúdo da base de dados.
Definição de atributos	Atributo/atributos definidos como chave primária, mas que não são chaves estrangeiras.	Definidos como atributos identificadores em cada conjunto de entidades correspondente.
	Atributos que não fazem parte da chave primária.	Definidos como atributos em cada conjunto de entidades correspondente.
Definição de identificadores de entidades e relacionamentos	Chaves primárias, que não fazem parte da chave estrangeira.	Definido como atributo identificador.

Apesar das regras definidas em Heuser (2009) é importante destacar que o esquema conceitual resultante pode não representar com exatidão os requisitos de dados. Isso ocorre porque o modelo entidade-relacionamento possui mais semântica do que o modelo relacional. Um exemplo é a detecção de generalizações, que não possuem correspondente no modelo relacional. Portanto, a regra “Toda a chave primária é também uma chave estrangeira”, do Quadro 3.3, nem sempre corresponde a uma generalização, podendo ser um conjunto de relacionamentos.

3.6 Considerações finais

Este capítulo apresentou o referencial teórico deste projeto de mestrado, abordando o projeto genérico de bases de dados, os projetos conceitual e lógico, por meio do modelo entidade-relacionamento e do modelo relacional, respectivamente.

A escolha de um modelo de dados deve se basear em sua representatividade, pois é sobre este modelo que o esquema será elaborado. Portanto, em relação à evolução de esquemas de bases de dados essa escolha é de extrema importância, pois influencia na facilidade ou dificuldade em se evoluir o esquema de dados.

O modelo entidade-relacionamento possui alta representatividade, por expressar os requisitos de dados por meio de conceitos bem definidos. Também é independente da tecnologia de SGBD. Ainda, a simplicidade da forma gráfica para a representação do esquema de dados, o diagrama entidade-relacionamento, facilita a compreensão do esquema por parte do usuário.

O modelo relacional foi apresentado, pois é padrão na implementação da maioria dos SGBDs utilizados atualmente. No projeto genérico de bases de dados o esquema relacional é produzido no projeto lógico pela transformação do esquema conceitual, por meio de regras de mapeamento. Essas regras influenciam na evolução do esquema de bases de dados, pois podem tornar o esquema mais ou menos tolerante a alterações futuras.

Em seguida, regras para o processo de engenharia reversa do modelo relacional para o modelo entidade-relacionamento foram apresentadas, indicando que nem sempre é possível se obter um esquema conceitual de dados fiel aos requisitos de dados originais, utilizados na elaboração do esquema conceitual.

O próximo capítulo apresenta a modularização de bases de dados definida por Busichia (1999), que é conduzida na fase conceitual do projeto de base de dados, com o intuito de se particionar o esquema de dados em módulos autônomos de dados.

4 Modularização de bases de dados

Este capítulo descreve o processo de modularização de bases de dados, definido por Busichia (1999), que neste trabalho de mestrado foi estendido para ser conduzido de modo iterativo e incremental, visando facilitar a evolução da base de dados.

A Seção 4.1 apresenta uma visão geral sobre o processo de modularização de bases de dados. A Seção 4.2 detalha a fase do "Projeto de Modularização", que foi incorporada ao processo genérico de projeto de base de dados, e as etapas que são responsáveis pela definição dos módulos de bases de dados. Já a Seção 4.3 trata da integração de módulos de bases de dados, pelo uso de objetos integradores. Ao final, a Seção 4.4 traz as considerações finais sobre a modularização de bases de dados.

4.1 Visão geral sobre modularização de bases de dados

O desenvolvimento de um sistema de software pode ser feito de maneira modular, em que partes são desenvolvidas de modo independente e, posteriormente, são integradas dando origem a uma versão completa e operacional.

Um dos atributos de qualidade do software apontado por Sommerville (2011) é a modularidade. Portanto, é interessante que esse conceito seja aplicado ao projeto da base de dados, pois com isso é possível obter maior controle de como a informação está compartimentada ao longo do esquema de dados e como se dá a interoperabilidade dos dados pelos módulos de software.

A decomposição do esquema global da base de dados em módulos de dados resulta em unidades autônomas de informação, possuindo baixo acoplamento e alta coesão entre os elementos que compõem o esquema de dados de cada módulo, o que simplifica a evolução do esquema global.

Em sistemas de software desenvolvidos de maneira incremental, o projeto completo da base de dados não é conhecido de antemão, pois está em constante evolução. Portanto, módulos de dados podem ser criados, integrados e estendidos em incrementos ulteriores.

O trabalho de Busichia (1999) propõe diretrizes para a modularização de bases de dados com o uso do ME-R estendido para a elaboração do esquema conceitual de dados. Essa característica permite que a modularização ocorra independente da tecnologia de SGBD escolhida. As fases referentes à modularização foram incorporadas ao processo genérico de projeto de base de dados com a adição de duas novas fases específicas, "Coleta e Análise de Requisitos de Modularização" e "Projeto de Modularização", conforme mostra a Figura 4.1.

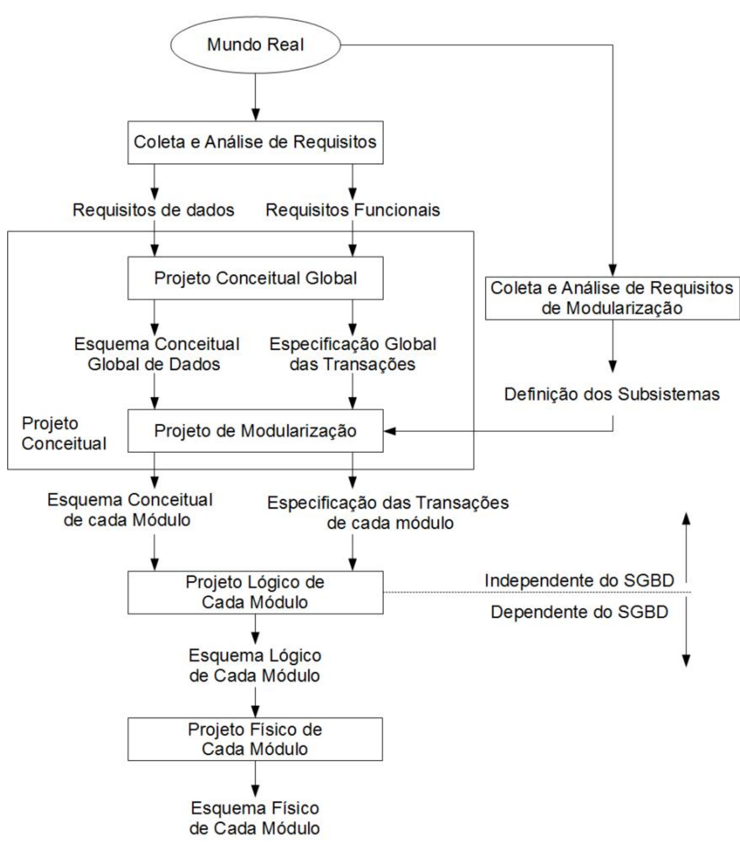


Figura 4.1 - Processo de projeto de base de dados incluindo a modularização (BUSICHIA, 1999).

A fase de "Coleta e Análise de Requisitos de Modularização" parte dos requisitos funcionais definidos no domínio da aplicação. As funcionalidades são agrupadas em subsistemas, de acordo com as transações executadas pela aplicação. Já a fase de "Projeto de Modularização" é responsável pela definição dos módulos da base de dados propriamente ditos e é abordada com mais detalhes na Seção 4.2.

Além da modularização, o trabalho de Busichia (1999) também trata da integração de módulos de bases de dados, descrita na Seção 4.3, necessária no nível físico para prover interoperabilidade dos dados entre os módulos de bases de dados.

Uma vantagem em se trabalhar com a modularização de bases de dados está no fato de que a decomposição do esquema de dados permite representar e associar as funcionalidades e os aspectos informacionais de cada módulo, com o uso de uma representação própria demonstrada na Seção 4.2. Outro ponto a ressaltar é que a leitura do esquema conceitual de dados é simplificada, pois está decomposto em módulos, definidos de acordo com seu contexto na aplicação.

4.2 Projeto de modularização de bases de dados

No processo de modularização de bases de dados proposto por Busichia (1999), os módulos da base de dados são definidos na fase de "Projeto de Modularização" (Figura 4.1). A definição dos módulos é baseada na análise do tipo de operação realizado a cada elemento do esquema conceitual global de dados pelas transações de cada um dos subsistemas da aplicação. Cada módulo possui um esquema conceitual próprio, além de representar o conjunto das funcionalidades pertencentes a cada subsistema que fazem acesso aos seus elementos.

De acordo com Busichia (1999) a tarefa de definição dos módulos é composta por quatro etapas apresentadas a seguir, executadas de modo serial:

1. **Particionamento do esquema conceitual global de dados**: tem como entrada o esquema conceitual global de dados da aplicação e os subsistemas definidos na fase de "Coleta e Análise de Requisitos de Modularização". O objetivo desta etapa é decompor o esquema conceitual global em um subesquema para cada subsistema, composto pela união dos elementos que são acessados por suas transações. Busichia (1999) define que a decomposição requer que a informação esteja modelada da maneira mais atômica possível, o que só ocorre no nível mais baixo em uma hierarquia de abstrações. Formalmente, a decomposição do esquema pode ser representa por:

$$DER = \cup_{i=1..n} DER_i$$

2. **Tratamento do compartilhamento da informação:** a etapa anterior pode resultar na sobreposição entre subesquemas, ou seja, um elemento pode pertencer a mais de um subesquema. Portanto, cada elemento deve ser analisado de acordo com o tipo de operação realizado por cada subsistema. As operações podem ser de leitura, que consistem apenas em consultas, ou de escrita, que englobam a criação, remoção e atualização de dados. Por fim, o elemento é classificado como:

    a. **Não-compartilhado:** o elemento pertence a um único subesquema e, portanto, um único subsistema executa as operações tanto de leitura quanto de escrita.
    b. **Compartilhamento unidirecional:** o elemento pertence a dois ou mais subesquemas, mas apenas um único subsistema executa operações de escrita no elemento e os demais apenas acessos de leitura.
    c. **Compartilhamento multidirecional:** o elemento pertence a dois ou mais subesquemas, sendo que mais de um subsistema executa operações de escrita no elemento.

    Formalmente, para cada subsistema S_i tem-se a utilização de um subesquema DER_i. Assim:

    S1 utiliza $DER_1 = \{E_1, A_1, R_1, HG_1, AG_1, C_1\}$

    S2 utiliza $DER_2 = \{E_2, A_2, R_2, HG_2, AG_2, C_2\}$

    ...

    Sn utiliza $DER_n = \{E_n, A_n, R_n, HG_n, Ag_n, C_n\}$

    Define-se G_{Sn} como sendo o grupo dos elementos dos respectivos subesquemas e operações para sintetizar cada um dos DER_n utilizados por um subsistema S_n.

3. **Definição dos módulos da base de dados:** consiste em definir os módulos de base de dados de acordo com as informações obtidas na etapa anterior. Primeiramente os elementos são agrupados de acordo com os subsistemas que os mantêm, ou seja,

aqueles que realizam as operações de escrita. Posteriormente, o resultado da interseção entre esses grupos é o que determina a definição dos módulos de base de dados:

a. **Interseção vazia:** origina diretamente um módulo para cada grupo. Ou seja, se:

$G_{S1} \cap G_{S2} = \varnothing$ e

$G_{S1} \cap G_{S3} = \varnothing$ e

...

$G_{S1} \cap G_{S2} \cap ... G_{Sn} = \varnothing$, então cria-se um módulo Mi para cada G_{Si}

b. **Interseção diferente de vazia:** os elementos dos grupos são unidos, gerando um módulo que os agrega. Dessa maneira, se:

$G_{S1} \cap G_{S2} \neq \varnothing$, então cria-se um módulo $Mi = G_{S1} \cup G_{S2}$ ou

$G_{S1} \cap G_{S3} \neq \varnothing$, então cria-se um módulo $Mi = G_{S1} \cup G_{S3}$ ou

$G_{S1} \cap G_{S2} \cap ... G_{Sn} \neq \varnothing$, então cria-se um módulo $Mi = G_{S1} \cup G_{S2} \cup ... G_{Sn}$

Outra opção é gerar um módulo excluindo-se o conjunto de elementos contidos na interseção, colocando-os em um novo módulo. Essa opção é válida se os elementos participantes da interseção não pertencerem a nenhuma hierarquia de generalização, abstração de agregação ou a qualquer outra hierarquia complexa. Por exemplo:

se $G_{S1} \cap G_{S2} = G_x$, então

$G_{S1} - G_x \Rightarrow M1$

$G_{S2} - G_x \Rightarrow M2$

$G_x \Rightarrow M3$

M1 é acessado somente por S1, M2 é acessado somente por S2 e M3 é acessado por S1 e S2.

4. **Definição da interface dos módulos da base de dados:** consiste em definir uma interface de acesso para cada um dos módulos de base de dados. Para tanto, deve-se criar procedimentos que encapsulem as operações de acesso aos dados de cada um dos módulos da base de dados. As operações são divididas em procedimentos públicos, somente de leitura e privados, de escrita. A Figura 4.2 apresenta uma representação gráfica para um módulo de base de dados.

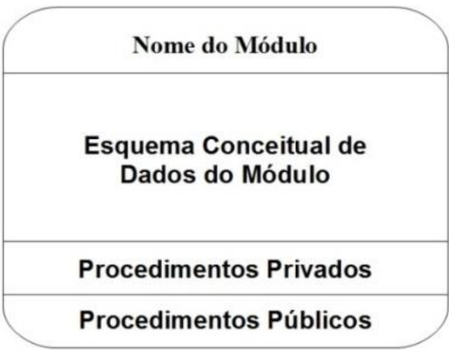


Figura 4.2 - Representação de um módulo de base de dados (BUSICHIA, 1999).

4.3 Integração de módulos de bases de dados

Após o particionamento do esquema conceitual em módulos de bases de dados, as fases de projeto lógico e físico devem tratar da integração destes. Isso se dá devido ao fato de um módulo poder ser acessado por vários subsistemas e da necessidade de se manter a integridade referencial entre os módulos.

O trabalho de Busichia (1999) traz as especificações de integração considerando recursos de SGBDs relacionais. A implementação dos procedimentos dos módulos pode ser realizada pelo uso de *stored procedures* e o acesso público ou privado aos procedimentos fica por conta do controle de acesso do SGBD.

É possível que cada módulo seja gerado para diferentes tipos de SGBDs, configurando um ambiente heterogêneo de gerenciamento de dados. Portanto, o projeto lógico deve ser executado com o intuito de se gerar esquemas de dados para cada uma das classes de SGBDs correspondentes.

Uma vez terminado o projeto lógico, a execução do projeto físico deve considerar os recursos específicos de cada SGBD, além de tratar da heterogeneidade ou homogeneidade do ambiente em que os módulos serão implementados.

Em um ambiente de gerenciamento de dados homogêneo apenas um SGBD é utilizado pela aplicação, sendo que "na maioria dos casos, os próprios recursos providos pelos SGBDs são suficientes para a integração dos módulos em ambiente homogêneo." (BUSICHIA, 1999, p. 46). Portanto, "no caso de ambiente de gerenciamento de dados homogêneo, não há a necessidade da utilização de um objeto integrador, pois os subsistemas podem acessar diretamente as *stored procedures* armazenadas nos módulos [...]" (BUSHICIA, 1999, p. 49).

No caso de ambientes heterogêneos a solução proposta para a integração é através de objetos integradores, que possibilitam a interoperabilidade da informação entre os módulos de bases de dados (BUSICHIA; FERREIRA, 1999a, 1999b). A Figura 4.3 mostra as diferenças de acesso aos módulos de base de dados de acordo com os ambientes, homogêneo ou heterogêneo.

Os objetos integradores possuem diversas finalidades, conforme definição apresentada por Busichia (1999, p. 48) os "objetos integradores podem ser caracterizados e utilizados em vários tipos de conversões, tais como: dados, estruturas de dados, valor semântico dos dados, regras de validação de dados, procedimentos, interfaces e SGBDs."

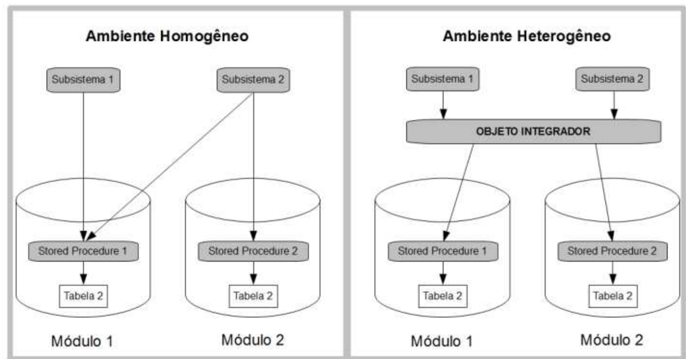


Figura 4.3 - Objeto integrador para integração de módulos de bases de dados (BUSICHIA, 1999).

A arquitetura do objeto integrador conta com seis serviços: identificação, métodos, mapeamento, regras ativas, conectividade e execução. O diagrama de fluxo da arquitetura do objeto integrador está representado na Figura 4.4.

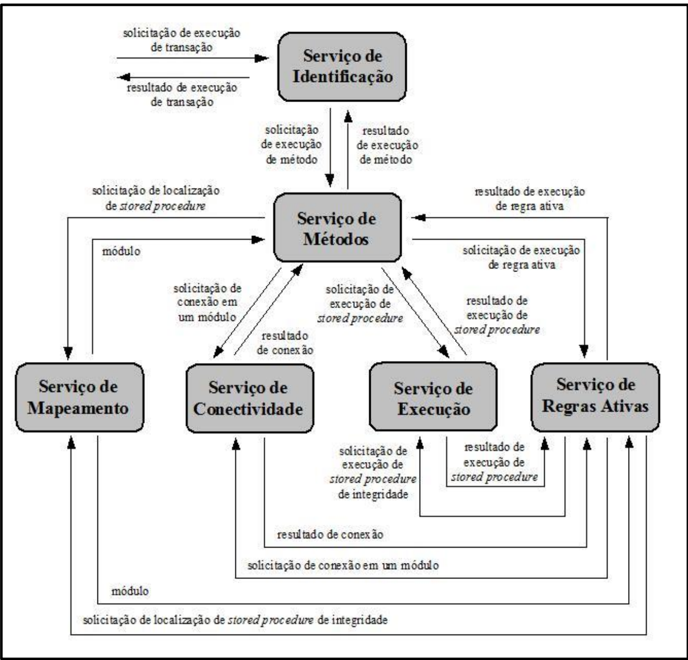


Figura 4.4 - Diagrama de fluxo da arquitetura do objeto integrador (BUSICHIA, 1999).

Os serviços do objeto integrador são descritos a seguir:

- **Serviço de Identificação:** responsável por tratar das requisições de transações realizadas pelos subsistemas. Logo, provê uma visão transparente dos módulos da base de dados. O serviço indica se houve sucesso ou falha na realização de uma transação de escrita e retorna uma lista de registros no caso de uma leitura;
- **Serviço de Métodos:** responsável por encapsular as transações que serão executadas nos módulos da base de dados. Esse encapsulamento consiste em um método (ou função) para cada transação;
- **Serviço de Mapeamento:** responsável por localizar os procedimentos associados com cada módulo da base de dados;
- **Serviço de Regras Ativas:** responsável por manter a integridade referencial entre módulos;

- **Serviço de Conectividade:** gerencia as conexões com a base de dados. Em um ambiente heterogêneo, a conexão com diversos tipos de SGBDs deve ser possível;
- **Serviço de Execução:** responsável por executar os procedimentos de acordo com as informações obtidas pelos serviços de identificação e de métodos.

O objeto integrador possui um repositório que armazena os metadados referentes às informações dos módulos, cujo esquema de dados é apresentado na Figura 4.5. Esse repositório fica disponível para os serviços do objeto integrador que necessitam de informações relativas aos módulos da base de dados para executar suas ações.

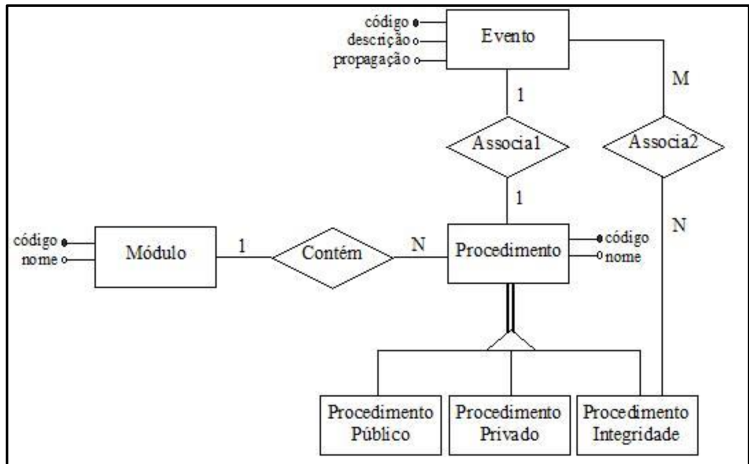


Figura 4.5 - Esquema de dados do objeto integrador (BUSICHIA, 1999).

Considerando uma abordagem orientada a eventos, pode-se considerar que uma transação dispara um ou mais eventos. Cada evento associa um procedimento armazenado em determinado módulo, como ilustrado no esquema de dados da Figura 4.5. Um evento pode ser propagável ou não, indicando, respectivamente, se há ou não necessidade de manutenção de integridade referencial entre módulos.

Como pode ser observado na Figura 4.5, há uma especialização do procedimento em público, privado e de integridade. Os públicos e privados foram abordados na Seção 4.2. Já os de integridade implementam as consistências necessárias para garantir os vários tipos de integridade referencial que podem ou não estarem associadas ao evento através do relacionamento Associa2 (N-M).

4.4 Considerações finais

Este capítulo apresentou o processo de modularização de bases de dados e o tratamento da interoperabilidade de informação entre módulos, de acordo com Busichia (1999).

No processo de modularização de bases de dados duas novas fases foram introduzidas ao projeto genérico de base de dados. O processo para se definir os módulos de bases de dados considera os dados e as transações executadas pela aplicação, determinando o grau de compartilhamento entre os elementos que compõem o esquema conceitual global de dados.

Com o particionamento do esquema conceitual global de dados em módulos, é necessário considerar a interoperabilidade de informações entre eles. Duas soluções são apresentadas: 1) uso dos próprios recursos do SGBD para ambiente de gerenciamento de dados homogêneo; 2) uso de objeto integrador, uma abordagem genérica de integração e utilizada em ambiente de gerenciamento de dados heterogêneo. O uso do objeto integrador torna a integração dos módulos de bases de dados mais flexível, permitindo que ajustes sejam feitos apenas nos dados do repositório de metadados, cujo esquema foi apresentado na Figura 4.5.

A modularização de bases de dados visa obter módulos coesos e com baixo acoplamento. Essas características são desejáveis em métodos de desenvolvimento de software iterativos e incrementais, em que a entrega do sistema de software é realizada de maneira modular.

O próximo capítulo apresenta o processo evolutivo de modularização de bases de dados com adaptações e extensões das diretrizes de modularização de bases de dados apresentadas neste capítulo, visando prover um meio de projetar bases de dados de modo iterativo e incremental.

5 Processo evolutivo de modularização de bases de dados

Este capítulo descreve a abordagem evolutiva do processo de modularização de bases de dados proposta neste trabalho.

A Seção 5.1 descreve as fases do processo evolutivo de modularização de bases de dados. A Seção 5.2, detalha a fase de “Análise Evolutiva dos Requisitos de Modularização”, responsável por verificar a completude dos novos requisitos em relação aos módulos de bases de dados já existentes. Em seguida, a Seção 5.3 apresenta a fase “Projeto Evolutivo de Modularização”, adaptada para a evolução dos módulos de bases de dados. A integração dos módulos de bases de dados, com uma abordagem evolutiva, é descrita na Seção 5.4. Por fim, a Seção 5.5 contém as considerações finais do capítulo.

5.1 Fases do processo evolutivo de modularização de bases de dados

O trabalho original de modularização, apresentado no Capítulo 4, define um processo de projeto de base de dados que utiliza a estratégia descendente, em que um esquema conceitual global de dados é inicialmente projetado e particionado (modularizado). Por outro lado, este trabalho aborda projetos de software iterativos e incrementais, em que os ciclos de desenvolvimento subsequentes se iniciam a partir de um esquema já implantado e com dados. Como exemplo, esse é o caso de projetos desenvolvidos com métodos ágeis.

Desse modo, o processo evolutivo de modularização de bases de dados é conduzido considerando-se tanto a estratégia descendente quanto a ascendente. A descendente é aplicada em cada iteração para se definir os novos subsistemas ou para associar novos requisitos e funcionalidades aos já existentes. Entretanto, é necessário considerar a existência de módulos de bases de dados criados em iterações anteriores, que podem contemplar totalmente ou parcialmente as novas funcionalidades, caracterizando uma abordagem ascendente.

O processo evolutivo de modularização de bases de dados é composto por seis fases de acordo com a Figura 5.1. Inclui ao processo de modularização de bases de dados abordado no Capítulo 4 (Figura 4.1), uma nova fase denominada “Análise Evolutiva dos Requisitos de Modularização” e uma extensão da fase original de “Projeto de Modularização”, descrita na Seção 4.2, que passa a se chamar “Projeto Evolutivo de Modularização”. Essas novas fases estão destacadas em cinza na Figura 5.1.

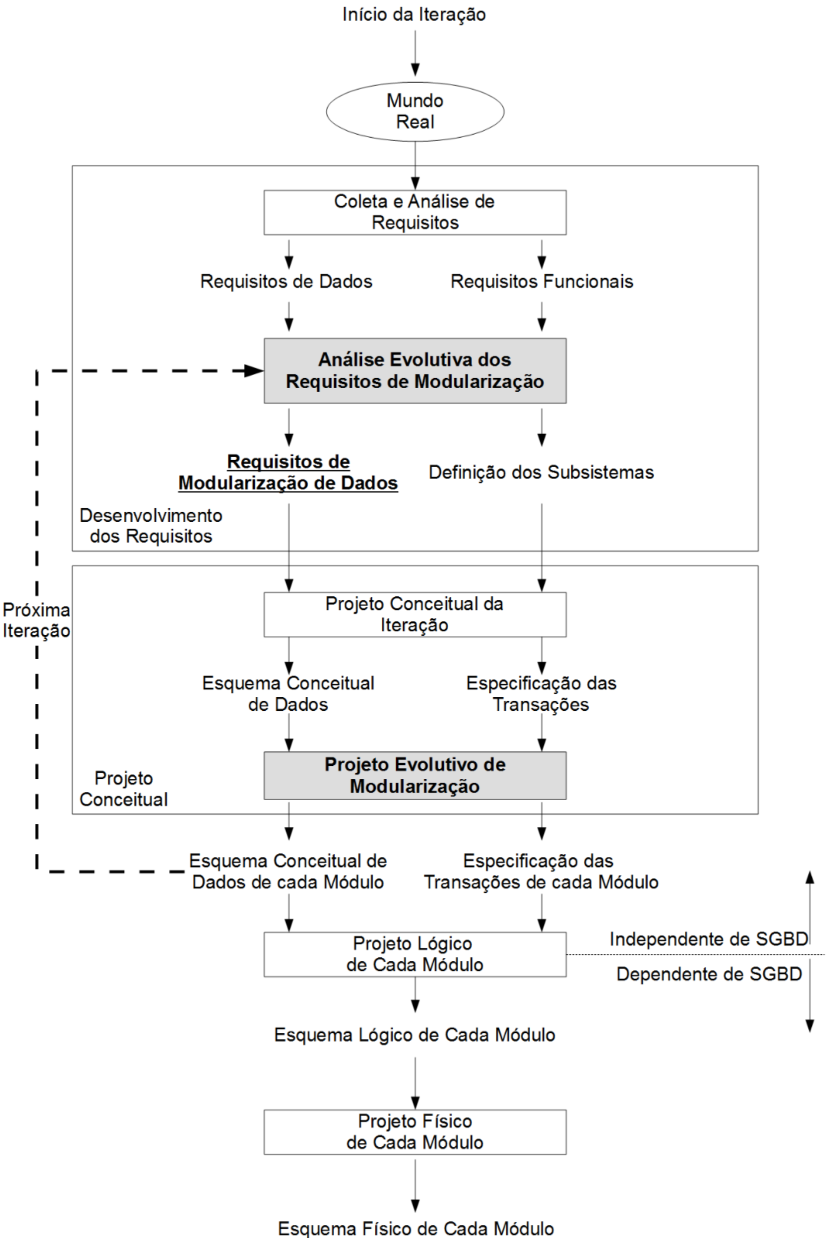


Figura 5.1 - Processo evolutivo de modularização de bases de dados.

A seguir encontra-se a descrição das fases do processo evolutivo de modularização de bases de dados:

1. **Coleta e Análise de Requisitos:** consiste em identificar e documentar os requisitos de dados e funcionais;
2. **Análise Evolutiva dos Requisitos de Modularização:** em uma abordagem evolutiva todas as fases são executadas a cada ciclo, resultando em um novo incremento à base de dados. Portanto, esta fase tem o objetivo de analisar os novos requisitos considerando os módulos já implementados em iterações anteriores. A saída são os requisitos de modularização de dados, que indicam o impacto da iteração nos subsistemas e módulos. Esta fase é abordada com detalhes na Seção 5.2;
3. **Projeto Conceitual da Iteração:** os requisitos são analisados para a elaboração do esquema conceitual de dados da iteração, utilizando-se o modelo entidade-relacionamento estendido;
4. **Projeto Evolutivo de Modularização:** esta fase continua composta por quatro etapas, mas foi estendida para que a modularização seja realizada de maneira evolutiva. Logo, a etapa de “definição dos módulos da base de dados” foi estendida para “definição **evolutiva** dos módulos da base de dados”. A elaboração do esquema conceitual de cada módulo é alterada para apoiar as futuras evoluções. A abordagem completa dessa fase é apresentada na Seção 5.3;
5. **Projeto Lógico de cada Módulo:** nesta fase o esquema conceitual de cada módulo é transformado em um esquema lógico;
6. **Projeto Físico de cada Módulo:** os esquemas lógicos são implantados fisicamente em uma base de dados, utilizando-se um SGBD específico.

Por fim, o particionamento da base de dados em módulos, faz com que seja necessário considerar a integração destes, com o objetivo de prover interoperabilidade da informação entre os módulos. Esse assunto é abordado na Seção 5.4, em que o esquema de dados do objeto integrador proposto por Busichia (1999), contido na Figura 4.5, foi estendido.

5.2 Análise Evolutiva dos Requisitos de Modularização

No processo evolutivo de modularização de bases de dados (Figura 5.1) a fase de "Análise Evolutiva dos Requisitos de Modularização" tem como objetivo analisar a completude dos módulos da base de dados em relação às funcionalidades da iteração em vigor. A entrada dessa fase são os requisitos de dados e funcionais, coletados por meio de algum método de elicitação de requisitos, que dão origem aos requisitos de modularização de dados e a definição dos subsistemas.

O diagrama de atividades da Figura 5.2 representa a análise dos requisitos em relação à modularização e também as possíveis saídas, que resultam nos requisitos de modularização de dados, que podem ser:

1. Os módulos já existentes atendem os novos requisitos, portanto nenhuma alteração é necessária e as funcionalidades são implementadas nos módulos correspondentes;
2. Não há suporte, por parte dos módulos já existentes, para os novos requisitos, logo novos módulos devem ser criados com os elementos e funcionalidades correspondentes a cada um deles;
3. Se os módulos existentes contemplam parcialmente os novos requisitos, pode ocorrer tanto a criação de novos módulos, quanto a extensão dos já existentes, para dar suporte às novas funcionalidades.

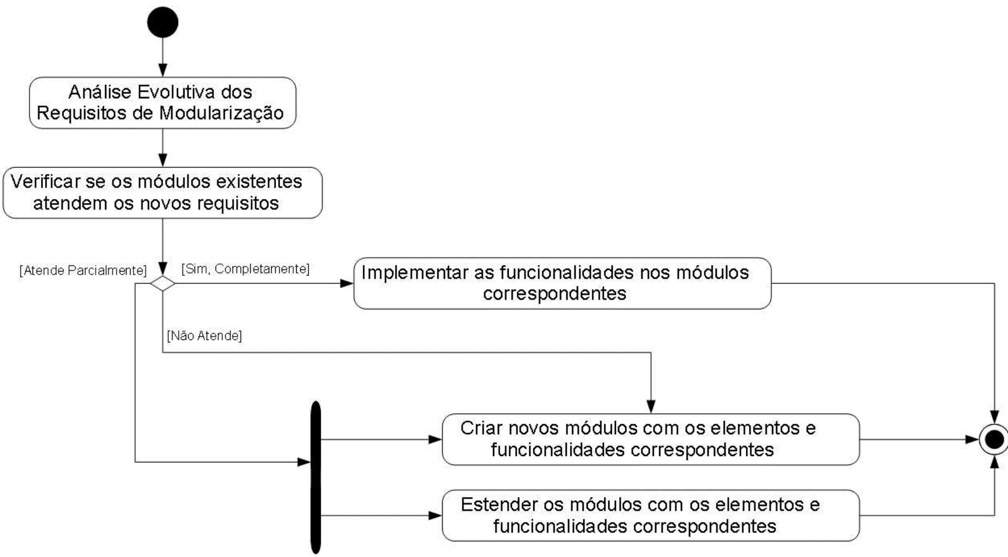


Figura 5.2 - Diagrama de atividades: Análise Evolutiva dos Requisitos de Modularização.

É importante destacar que na primeira iteração não existem módulos, portanto o resultado será a criação dos primeiros módulos da base de dados, ou seja, a saída é representada pela atividade “Criar novos módulos com os elementos e funcionalidades correspondentes" da Figura 5.2. A partir da segunda iteração a fase em questão recebe como entrada os módulos já existentes, ou seja, os esquemas conceituais de cada módulo gerados nas iterações anteriores. Essa entrada é representada pela seta tracejada na Figura 5.1.

5.3 Projeto Evolutivo de Modularização

Para produzir módulos de bases de dados coesos visando a evolução futura é necessário considerar o acoplamento existente entre eles, o que ocorre nos inter-relacionamentos definidos por Busichia (1999) e Ferreira e Busichia (1999). Outro aspecto são as hierarquias de generalização que envolvem mais de um módulo. Esse último conceito é denominado por este trabalho de inter-generalização.

Uma inter-generalização ocorre quando os requisitos de uma iteração são parcialmente atendidos pelos módulos já existentes e, conforme o diagrama de atividades da Figura 5.2, um novo módulo será gerado. Além disso, o novo módulo necessita especializar um CE de um módulo já implementado. Portanto, o CE específico é criado no novo módulo, que será responsável por mantê-lo. O objetivo é evitar um acoplamento desnecessário entre o CE especializado e o genérico, e respeitar que “[...] para um esquema conceitual de dados poder ser decomposto é essencial que a informação modelada esteja representada da forma mais atômica possível.” (BUSICHIA, 1999, p. 41).

A hierarquia de generalização com as linhas tracejadas na Figura 5.3 representa uma inter-generalização, em que o CE *E1* genérico e o CE *E1’* específico, estão em módulos distintos. Em relação ao Módulo X, é possível notar que os CEs *E1’* e *EX* possuem o mesmo atributo $A_{E1'}$, mas não foram representados na mesma hierarquia de generalização. O intuito é permitir que cada CE evolua de maneira independente, uma vez que o acoplamento gerado pela hierarquia de generalização entre módulos foi reduzido com a introdução de uma inter-generalização.

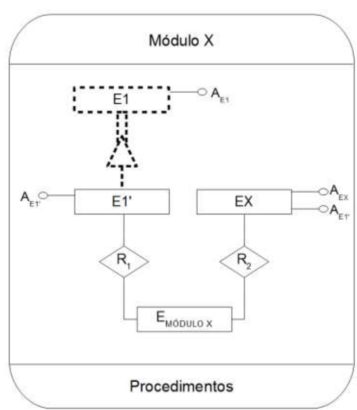


Figura 5.3 - Representação de uma inter-generalização.

Já os inter-relacionamentos, representam o local em que dois módulos compartilham dados, conforme o CR *R* da Figura 5.4, que relaciona os CEs *E* e $E_{MÓDULO\ W}$. Em um projeto evolutivo, existe uma alta possibilidade de que a multiplicidade do relacionamento se altere em iterações futuras, o que exigiria refatoração. Dessa maneira, o ideal é torná-los mais flexíveis e protegê-los de possíveis alterações, considerando o mapeamento dos inter-relacionamentos sempre como muitos-para-muitos (M:N), como ilustrado na Figura 5.5.

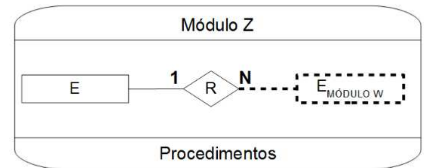


Figura 5.4 - Mapeamento tradicional de inter-relacionamento.

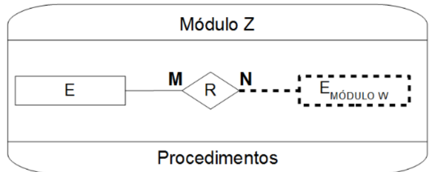


Figura 5.5 - Mapeamento flexível de inter-relacionamento.

Com a definição do conceito de inter-generalização e do mapeamento de inter-relacionamentos, em seguida são apresentadas as etapas da fase de “Projeto Evolutivo de Modularização”, do processo evolutivo de modularização de bases de dados (Figura 5.1) e que consiste de alterações e extensões no projeto de modularização de bases de dados descrito na Seção 4.2:

Etapa 1 - Particionamento do esquema conceitual global de dados

O esquema conceitual continua sendo particionado em um subesquema por subsistema. Entretanto, podem ocorrer modificações em subsistemas já existentes bem como novos subsistemas podem surgir, o que pode resultar em um conjunto diferente de subesquemas em relação aos existentes em iterações anteriores.

Etapa 2 - Tratamento do compartilhamento da informação

As sobreposições de elementos entre os subesquemas gerados na Etapa 1 podem ser alteraras de uma iteração para outra. A adição de novos subsistemas e a modificação dos já existentes pode mudar a classificação de um elemento de duas maneiras:

1. **De não-compartilhado para unidirecional:** se um subsistema passa a ler dados de um elemento que já é mantido por outro subsistema;
2. **De unidirecional para multidirecional:** se um subsistema passa a escrever dados em um elemento que já é mantido por outro subsistema.

Etapa 3 - Definição evolutiva dos módulos da base de dados

Nesta etapa os elementos podem ter tido sua classificação alterada de acordo com os resultados da Etapa 2. Isso significa que as operações de interseção podem gerar novos inter-relacionamentos.

Existem duas opções para se definir um módulo, conforme a Etapa 3 descrita na Seção 4.2, a primeira é gerar um módulo separado com os elementos resultantes da interseção; a segunda é realizar uma união entre todos os elementos dos subesquemas sobrepostos. Considerando uma abordagem evolutiva, a primeira opção é mais adequada, pois gera módulos com baixo acoplamento com os demais, por meio dos inter-relacionamentos.

Entre uma iteração e outra a multiplicidade entre CRs pode mudar de 1:N para M:N, devido a novos requisitos. Mas, se essa alteração ocorrer em um inter-relacionamento não há necessidade de refatoração, pois o mapeamento já estará como M:N. O mesmo acontece com hierarquias de generalização entre módulos, que serão mapeadas como inter-generalizações, em que o CE especializado será mantido pelo novo módulo.

Esta etapa define os módulos de bases de dados, seguindo os requisitos de modularização de dados, saída da fase “Análise Evolutiva dos Requisitos de Modularização”, descrita na Seção 5.2.

Duas situações são possíveis:

1. Novos módulos são criados quando:
    a. Um elemento ou um conjunto de elementos que era do tipo "não-compartilhado" ou de "compartilhamento unidirecional" passa a ser de "compartilhamento multidirecional";
    b. Quando os novos elementos não são associados aos módulos já existentes.
2. Um módulo é estendido quando um novo elemento ou um grupo de elementos introduzido foi associado aos módulos já existentes e não são compartilhados ou possuem apenas compartilhamento unidirecional.

Etapa 4 - Definição da interface dos módulos da base de dados
Os procedimentos públicos e privados são associados a cada um dos módulos definidos na Etapa 3.

As alterações na fase "Projeto Evolutivo de Modularização" ocorrem nas Etapas 2 e 3, pois devem considerar a existência de módulos de bases de dados criados em iterações anteriores. O impacto na geração dos módulos é determinado pelas alterações no tipo de compartilhamento dos elementos que compõem cada módulo.

O impacto causado por alterações é diminuído ao se considerar a multiplicidade dos inter-relacionamentos e do mapeamento das hierarquias de generalização, com o uso de inter-generalizações. Em seguida, após a definição dos módulos, é necessário considerar como o acesso a estes é feito pela aplicação.

A próxima seção propõe um catálogo para manter os metadados referentes ao processo evolutivo de modularização de bases de dados, que pode ser utilizado para apoiar a execução do processo a cada iteração.

5.4 Integração incremental dos módulos de bases de dados

O trabalho de Busichia (1999) propõe o uso de um objeto integrador como solução para a integração dos módulos de bases de dados, como abordado na Seção 4.3.

Neste trabalho é proposta uma extensão do catálogo do objeto integrador, descrito na Seção 4.3, que passa a ser alimentado com os metadados gerados pelo processo evolutivo de modularização de bases de dados a cada iteração, com as definições dos conjuntos de entidades, conjuntos de relacionamentos, atributos, subsistemas, e os módulos de bases de dados.

O esquema conceitual estendido do catálogo é apresentado na Figura 5.6. A parte destacada representa os elementos que foram inseridos no esquema de dados apresentado na Figura 4.5, elaborado por Busichia (1999). Os dados do esquema são atualizados a cada iteração de acordo com as saídas das fases do processo evolutivo de modularização de bases de dados, refletindo inclusive mudanças que podem ter ocorrido nos módulos.

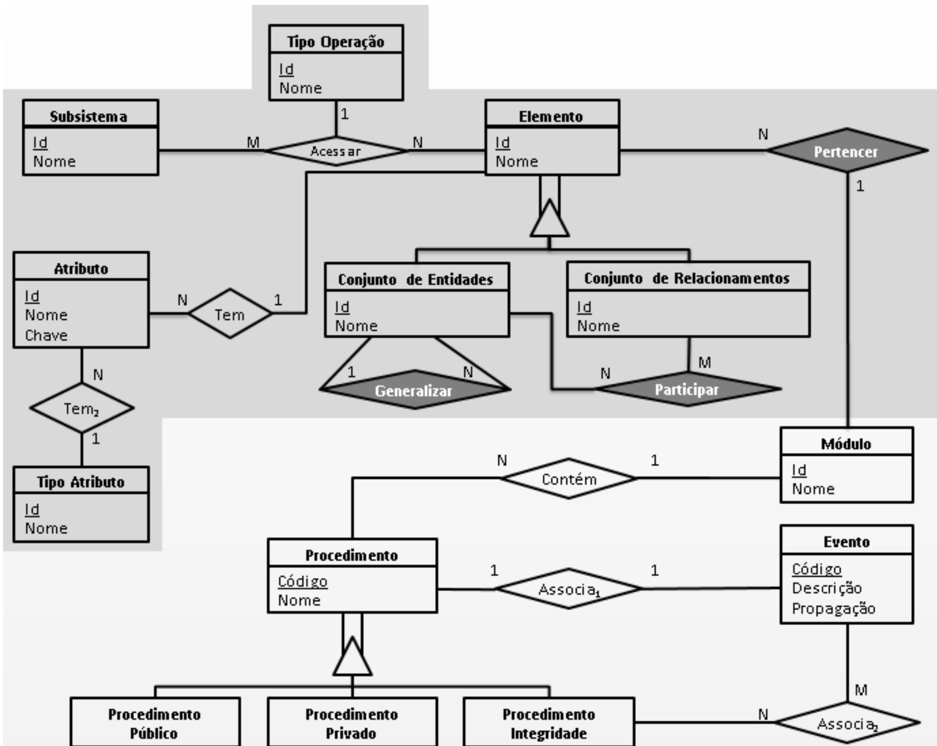


Figura 5.6 - Esquema conceitual estendido do catálogo do objeto integrador.

No esquema da Figura 5.6, conforme os destaques em cinza escuro, um inter-relacionamento pode ser detectado pelos relacionamentos contidos nos CRs *Pertencer* e *Participar*. O primeiro mantém os elementos que compõem um módulo, enquanto o segundo mantém os dados referentes aos CRs do esquema conceitual de dados da aplicação. Portanto, um inter-relacionamento é identificado quando um CR relaciona dois CEs que estão em módulos distintos. Entretanto,

a multiplicidade definida no nível de projeto conceitual da base de dados, é armazenada no catálogo, ainda que, como inter-relacionamento, o mapeamento seja M:N. Com isso as regras de integridade podem ser respeitadas, enquanto o esquema se mantém flexível para evoluções subsequentes.

Similarmente, uma inter-generalização pode ser detectada pelos CRs *Pertencer* e *Generalizar*, também destacados em cinza escuro na Figura 5.6, sendo que este último indica os CEs da aplicação, que participam de uma hierarquia de generalização. A inter-generalização é caracterizada quando um CE genérico e um CE específico pertencem a dois módulos distintos.

O catálogo possui um papel importante na automatização do processo evolutivo de modularização de bases de dados sendo utilizado como entrada para a fase “Análise Evolutiva dos Requisitos de Modularização” (Figura 5.1). Entretanto, é necessário manter ao menos a instância do catálogo da iteração imediatamente anterior, com o intuito de utilizá-la para a comparação dos esquemas de dados entre iterações. Desse modo, é possível automatizar a geração dos módulos pela análise das alterações nos subsistemas e nas operações realizadas aos elementos do esquema de dados da aplicação. O Capítulo 7 apresenta a *Evolutio DB Designer*, ferramenta desenvolvida para a execução do processo evolutivo de modularização de bases de dados.

O catálogo do objeto integrador também pode ser usado para proporcionar uma visão transparente da aplicação em relação aos módulos de bases de dados. Para tanto, propõe-se uma camada intermediária entre a aplicação e o SGBD, chamada de “Camada do Objeto Integrador”. A Figura 5.7 mostra a arquitetura de um sistema de bases de dados incluindo a camada do objeto integrador.

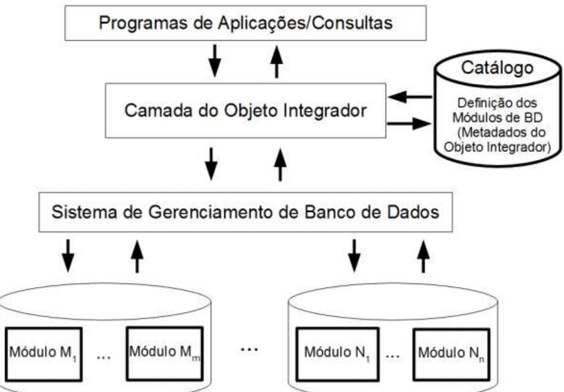


Figura 5.7 - Arquitetura de um Sistema de Bases de Dados incluindo a Camada do Objeto Integrador.

A aplicação envia as consultas para a camada do objeto integrador, que por sua vez recupera os metadados do catálogo. Com isso, as transações são enviadas ao SGBD de acordo com as definições dos módulos de bases de dados. Essa abordagem atende tanto ambientes homogêneos quanto heterogêneos, possibilitando a distribuição dos módulos por diversas bases de dados.

5.5 Considerações finais

A entrega modular de um sistema de software é a característica principal dos métodos de desenvolvimento iterativos e incrementais. Entretanto, grande parte desses métodos é voltada para a codificação do sistema, deixando o projeto da base de dados em segundo plano. Diante desse cenário, o processo de modularização de bases de dados apresentado no Capítulo 4 foi adaptado e estendido, para se obter um esquema de dados mais flexível e tolerante a mudanças futuras, demandadas pelo surgimento de novos requisitos.

Cada módulo de base de dados gerado pelo processo evolutivo de modularização contém um esquema de dados representado por um diagrama entidade-relacionamento e os procedimentos públicos e privados associados. Essas duas características aumentam a capacidade de representação dos requisitos de dados (esquema de dados) e funcionais (procedimentos). Portanto, facilitam a análise e evolução futura da base de dados.

Uma nova fase, “Análise Evolutiva dos Requisitos de Modularização”, e a extensão da fase “Projeto de Modularização”, agora chamada “Projeto Evolutivo de Modularização” foram apresentadas. A primeira tem como objetivo verificar se os módulos de bases de dados existentes atendem os requisitos de uma nova iteração, enquanto a segunda visa gerar módulos de bases de dados mais coesos e diminuir o acoplamento, facilitando a evolução futura. Isso é feito pelo mapeamento de inter-relacionamentos ser sempre M:N e pela introdução do conceito de inter-generalização.

A introdução da inter-generalização e do mapeamento de inter-relacionamentos resulta em esquemas de dados com maior capacidade de abstração, o que facilita as evoluções futuras.

O esquema de dados do objeto integrador, proposto por Busichia (1999) e Busichia e Ferreira (1999), foi estendido para armazenar os dados referentes ao processo evolutivo de modularização de bases de dados. Dessa maneira é possível automatizar a geração dos módulos da base de dados.

O próximo capítulo apresenta uma aplicação do processo evolutivo de modularização de bases de dados em um projeto de software iterativo e incremental.

6 Aplicação do processo evolutivo de modularização de bases de dados

Neste capítulo, as especificações do sistema descrito por Ambler (2004) são utilizadas para demonstrar o processo evolutivo de modularização de bases de dados.

A Seção 6.1 descreve o projeto, apresentando os requisitos definidos em cada iteração. Já a Seção 6.2 contém a aplicação do processo evolutivo de modularização de bases de dados, em que cada subseção representa uma iteração do desenvolvimento do projeto. Por fim, a Seção 6.3 traz as considerações finais, discutindo os resultados da aplicação do processo evolutivo de modularização de bases de dados no projeto apresentado.

6.1 Descrição do projeto

Os requisitos iniciais especificavam um sistema para uma escola de caratê, posteriormente estendido para outras artes marciais. A descrição sucinta dos requisitos de Ambler (2004) é apresentada no Quadro 6.1. Cada linha da tabela representa os requisitos que eram conhecidos em cada iteração de desenvolvimento do produto.

Quadro 6.1 - Requisitos de sistema para escola de caratê, adaptado de AMBLER (2004).

(continua)

Iteração	Requisitos
1	• Manter a informação de contato do aluno • Matricular aluno • Cancelar matrícula do aluno • Gravar pagamento
2	• Promover aluno para uma faixa mais alta • Enviar convite de exame para passagem de faixa • Enviar e-mail com os dados da matrícula para o aluno • Imprimir a matrícula para o aluno
3	• Agendar exame de faixa • Imprimir certificado • Suspender temporariamente a matrícula
4	• Matricular aluno do tipo criança • Oferecer plano familiar • Sistema de faixa para aluno do tipo criança

Quadro 6.1 - Requisitos de sistema para escola de caratê, adaptado de AMBLER (2004).
(conclusão)

Iteração	Requisitos
5	• Cadastro de cursos de artes marciais o Matricular aluno no curso de Tai chi o Sistema de faixas de Tai chi o Matricular aluno no curso de Kick Boxing o Ordem das faixas de cada curso
6	• Manter informação de produtos • Vender produto

6.2 Processo evolutivo de modularização de bases de dados

Nas seções seguintes, o processo evolutivo de modularização de bases de dados é aplicado a cada iteração para se obter os módulos da base de dados. Nos esquemas de dados apresentados, os atributos identificadores foram omitidos, pois é possível atribuir chaves artificias em todos os CEs.

O Quadro 6.2 apresenta os requisitos, contidos no Quadro 6.1, relacionados a cada subsistema e com a respectiva iteração em que foram adicionados. Os subsistemas foram definidos pelo agrupamento das funcionalidades, após a execução da fase de “Análise Evolutiva dos Requisitos de Modularização” em cada iteração.

Quadro 6.2 - Subsistemas definidos em cada iteração, de acordo com a saída da fase de “Análise Evolutiva dos Requisitos de Modularização”.
(continua)

Subsistema	Iteração	Requisitos
S1	1	• Manter a informação de contato do aluno • Matricular aluno • Cancelar matrícula do aluno
	3	• Suspender temporariamente a matrícula
	4	• Matricular aluno do tipo criança • Oferecer plano familiar
	5	• Matricular aluno no curso de Tai chi • Matricular aluno no curso de Kick Boxing
S2	1	• Gravar pagamento
S3	2	• Promover aluno para uma faixa mais alta • Enviar convite de exame para passagem de faixa • Enviar e-mail com os dados da matrícula para o aluno • Imprimir a matrícula para o aluno
	4	• Sistema de faixa para aluno do tipo criança
	5	• Sistema de faixas de Tai chi • Ordem das faixas de cada curso

Quadro 6.2 - Subsistemas definidos em cada iteração, de acordo com a saída da fase de “Análise Evolutiva dos Requisitos de Modularização”.

(conclusão)

Subsistema	Iteração	Requisitos
S4	3	• Agendar exame de faixa • Imprimir certificado
S5	5	• Cadastro de cursos de artes marciais
S6	6	• Manter informação de produtos • Vender produto

6.2.1 Primeira iteração

Na primeira iteração, a saída da fase “Análise Evolutiva dos Requisitos de Modularização” resulta em dois subsistemas. O Subsistema S1 mantém os dados do aluno e o S2 os pagamentos. Os requisitos de modularização de dados indicam que novos módulos devem ser criados para dar suporte às funcionalidades de S1 e S2, uma vez que trata-se da primeira iteração e não existem módulos de bases de dados já implantados.

De acordo com as fases do processo evolutivo de modularização de bases de dados apresentadas na Figura 5.1, o próximo passo é a elaboração do esquema conceitual de dados da iteração na fase de “Projeto Conceitual da Iteração”. Em princípio, a elaboração desse esquema conceitual segue as regras tradicionais de modelagem, usando o modelo entidade-relacionamento estendido. Desse modo, detalhes relevantes sobre esse esquema conceitual de dados serão abordados juntamente com a próxima fase “Projeto Evolutivo de Modularização”.

Com os subsistemas definidos e o esquema conceitual de dados da iteração elaborado, parte-se para a fase de “Projeto Evolutivo de Modularização”, onde devem ser executadas as seguintes etapas especificadas na Seção 5.3:

Etapa 1 - Particionamento do esquema conceitual global de dados: um subesquema é gerado para cada um dos subsistemas. A Figura 6.1 mostra o esquema conceitual de dados da primeira iteração com os subesquemas correspondentes aos subsistemas S1 e S2. Existem duas diferenças em relação ao esquema elaborado por Ambler (2004):

1. A hierarquia de generalização, em que participam o CE genérico *Pessoa* e o específico *Aluno* não é criada. No esquema elaborado nesta etapa, consta apenas o CE

Aluno. O motivo é que os requisitos da primeira iteração indicam apenas a necessidade de dados de alunos; introduzir uma hierarquia de generalização configuraria modelagem excessiva. Ainda, segundo o processo de modularização de bases de dados, a decomposição do esquema de dados requer que a informação esteja modelada da forma mais atômica possível, o que só ocorre no nível mais baixo em uma hierarquia de abstrações;

2. Os dados de endereço são mapeados como atributos de *Aluno*, já no esquema elaborado na Figura 6.1 o CE *Endereço* é criado, pois representa um conceito bem definido.

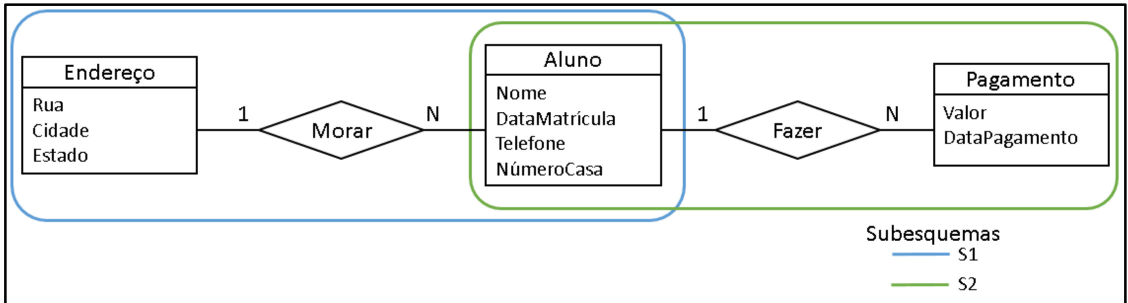


Figura 6.1 - Esquema conceitual de dados da primeira iteração mostrando os subesquemas.

Etapa 2 - Tratamento do compartilhamento da informação: de acordo com o esquema da Figura 6.1, a sobreposição entre os Esquemas S1 e S2 demonstra que há compartilhamento de dados. A classificação do tipo de compartilhamento de cada elemento é apresentada na Figura 6.2.

Elemento	Pertencente ao(s) subesquema(s)		Tipo da Operação (L - Leitura / E - Escrita)		Tipo de Compartilhamento
	S1	S2	S1	S2	
Aluno			L/E	L	Unidirecional
Endereço			L/E		Não-Compartilhado
Morar			L/E		Não-Compartilhado
Pagamento				L/E	Não-Compartilhado
Fazer				L/E	Não-Compartilhado

Figura 6.2 - Tipo de compartilhamento de cada elemento do esquema da primeira iteração.

Etapa 3 - Definição evolutiva dos módulos da base de dados: a partir do compartilhamento de dados os elementos são agrupados de acordo com os subsistemas que realizam as operações de escrita e é realizada a interseção dos grupos obtidos.

G_{S1} = {Aluno, Endereço, Morar}

G_{S2} = {Pagamento, Fazer}

$G_{S1} \cap G_{S2} = \varnothing \therefore G_{S1}$, dá origem ao módulo M1 e G_{S2} ao módulo M2.

A matriz da Figura 6.3 mostra a definição dos módulos, a matriz apresentada possuiu uma coluna para classificar o tipo do elemento com base nos conceitos existentes no processo evolutivo de modularização de bases de dados. O elemento *Fazer* é classificado como um inter-relacionamento, pois relaciona o CE *Pagamento* do módulo M2 com o CE *Aluno* do módulo M1.

Módulo	Elemento	É mantido por		Tipo do Elemento
		S1	S2	
M1	Aluno			Conjunto de Entidades
	Endereço			Conjunto de Entidades
	Morar			Conjunto de Relacionamentos
M2	Pagamento			Conjunto de Entidades
	Fazer			**Inter-Relacionamento**

Figura 6.3 - Definição dos módulos da primeira iteração.

Etapa 4 - Definição da interface dos módulos da base de dados: a interface de cada módulo é definida, conforme a Figura 6.4. O CE *Aluno* pontilhado no módulo M2 representa o inter-relacionamento entre este módulo e M1. O destaque em vermelho mostra que a multiplicidade é M:N, por se tratar de um inter-relacionamento.

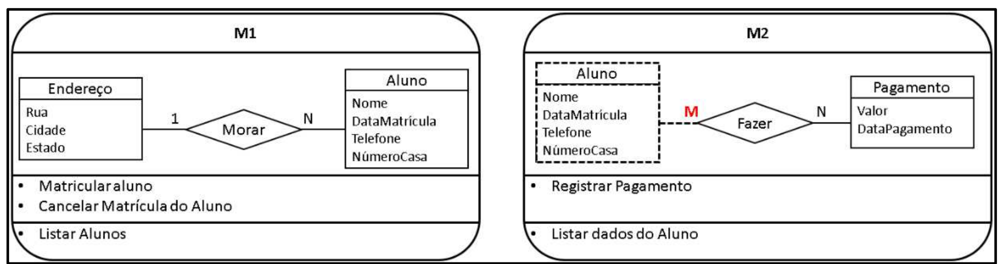


Figura 6.4 - Módulos da base de dados gerados na primeira iteração.

6.2.2 Segunda iteração

Os requisitos da segunda iteração exigem que um controle das faixas de cada aluno seja implementado, sendo que as funcionalidades são agrupadas em um novo Subsistema S3, conforme definido no Quadro 6.2.

A saída da fase de “Análise Evolutiva dos Requisitos de Modularização” indica que os requisitos dessa iteração são parcialmente atendidos, pois os dados dos alunos são representados no módulo M1. A seguir são apresentadas as etapas da fase de “Projeto Evolutivo de Modularização”:

Etapa 1 - Particionamento do esquema conceitual global de dados: na Figura 6.5 o subesquema de S3 é definido levando-se em consideração o esquema conceitual já existente.

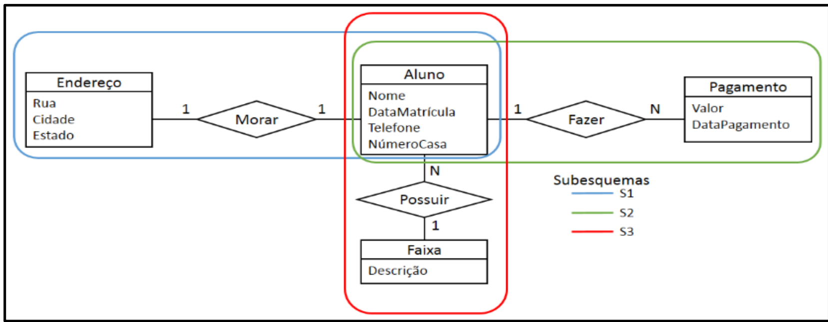


Figura 6.5 - Esquema conceitual de dados da segunda iteração mostrando os subesquemas.

Etapa 2 - Tratamento do compartilhamento da informação: os tipos de compartilhamento de cada elemento são novamente classificados, considerando os novos elementos que dão suporte as funcionalidades de S3. Nenhum elemento teve seu tipo de compartilhamento alterado, conforme resultados apresentados na matriz da Figura 6.6.

Elemento	Pertencente ao(s) subesquema(s)			Tipo da Operação (L - Leitura / E - Escrita)			Tipo de Compartilhamento
	S1	S2	S3	S1	S2	S3	
Aluno				L/E	L		Unidirecional
Endereço				L/E			Não-Compartilhado
Morar				L/E			Não-Compartilhado
Pagamento					L/E		Não-Compartilhado
Fazer					L/E		Não-Compartilhado
Faixa						L/E	Não-Compartilhado
Possuir						L/E	Não-Compartilhado

Figura 6.6 - Tipo de compartilhamento de cada elemento do esquema da segunda iteração.

Etapa 3 - Definição evolutiva dos módulos da base de dados: os grupos são novamente gerados de acordo com o resultado da etapa 2, incluindo agora o grupo 3, referente ao subsistema S3.

G_{S1} = {Aluno, Endereço, Morar}

G_{S2} = {Pagamento, Fazer}

G_{S3} = {Faixa, Possuir}

$G_{S1} \cap G_{S2} = \varnothing \therefore$ M1 e M2 são mantidos.

$G_{S1} \cap G_{S3} = \varnothing \wedge G_{S2} \cap G_{S3} = \varnothing \therefore G_{S3}$, dá origem ao módulo M3.

Os módulos dessa iteração são apresentados na Figura 6.7.

Módulo	Elemento	É mantido por			Tipo do Elemento
		S1	S2	S3	
M1	Aluno				Conjunto de Entidades
	Endereço				Conjunto de Entidades
	Morar				Conjunto de Relacionamentos
M2	Pagamento				Conjunto de Entidades
	Fazer				**Inter-Relacionamento**
M3	Faixa				Conjunto de Entidades
	Possuir				**Inter-Relacionamento**

Figura 6.7 - Definição dos módulos da segunda iteração.

Etapa 4 - Definição da interface dos módulos da base de dados: a interface do novo módulo M3 é definida. A Figura 6.8 apresenta os módulos da iteração 3.

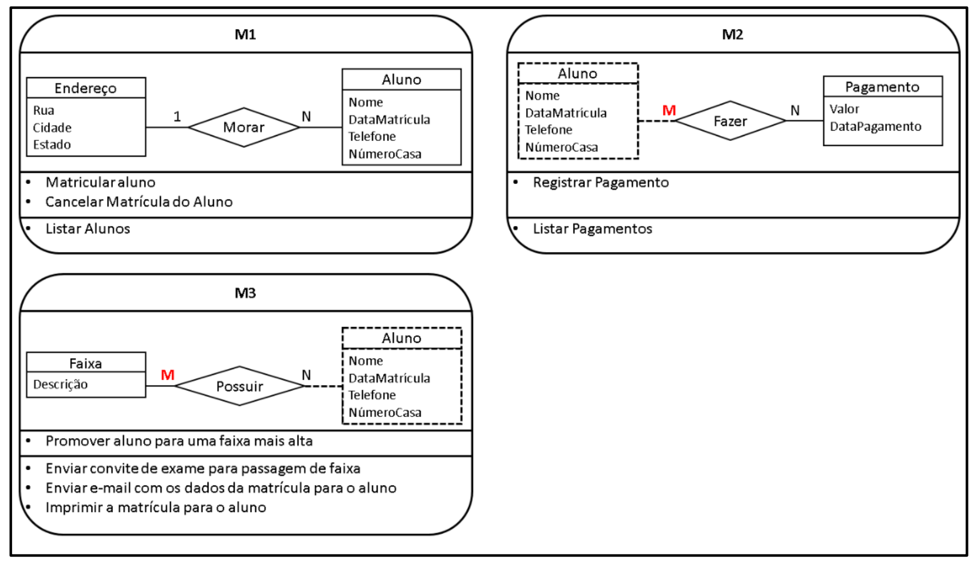


Figura 6.8 - Módulos de bases de dados gerados na segunda iteração.

6.2.3 Terceira e quarta iterações

Os requisitos da terceira iteração incluem o agendamento de exames com o intuito de promover os alunos para faixas mais elevadas. Essas funcionalidades são agrupadas em um novo sub-

sistema S4. Além disso, permite que um aluno possa suspender sua matrícula por um período determinado. Essa funcionalidade é agrupada em S1, responsável pelos requisitos de cadastro e da matrícula do estudante.

A quarta iteração requer implementar o sistema de faixa para crianças, o que demanda diferenciar os alunos entre crianças e adultos. Também permite ao aluno aderir a um plano familiar.

Os requisitos da terceira iteração fizeram com que o módulo M1 fosse estendido pela adição do CE *Suspensão Matrícula*. Enquanto as funcionalidades de S4 deram origem ao novo módulo M4.

Já a quarta iteração estendeu M1 pela inclusão do CE *Família* e do CR *Membro de*, além do atributo *TipoCriança* no CE Aluno. O módulo M3 também foi estendido com a inclusão do atributo *TipoCriança* no CE *Faixa*.

As Figuras 6.10, 6.11, 6.12 e 6.13 apresentam os módulos gerados até a quarta iteração, a partir dos resultados do "Projeto Evolutivo de Modularização" apresentados na Figura 6.9.

Módulo	**Iteração**	**Elemento**	**Tipo do Elemento**	**Tipo da Operação**				**Tipo de Compartilhamento**	**É mantido por**			
				S1	**S2**	**S3**	**S4**		**S1**	**S2**	**S3**	**S4**
M1	1	Aluno	CE	L/E	L	L	L	Unidirecional				
		Endereço	CE	L/E				Não-Compartilhado				
		Morar	CR	L/E				Não-Compartilhado				
	3	Supensão Matrícula	CE	L/E				Não-Compartilhado				
		Fazer2	CR	L/E				Não-Compartilhado				
	4	Família	CE	L/E				Não-Compartilhado				
		Membro De	CR	L/E				Não-Compartilhado				
M2	1	Pagamento	CE		L/E			Não-Compartilhado				
		Fazer	**Inter-Relacionamento**		L/E			Não-Compartilhado				
M3	2	Faixa	CE			L/E	L	Unidirecional				
		Possuir	**Inter-Relacionamento**			L/E		Não-Compartilhado				
M4	3	Exame	CE				L/E	Não-Compartilhado				
		Agendar	**Inter-Relacionamento**				L/E	Não-Compartilhado				
		Participar	**Inter-Relacionamento**				L/E	Não-Compartilhado				

Figura 6.9 - Resumo do Projeto Evolutivo de Modularização na quarta iteração.

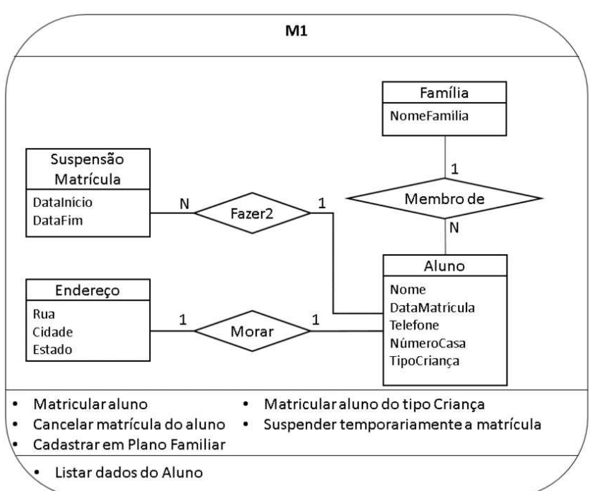


Figura 6.10 - Esquema de dados do módulo M1 na quarta iteração.

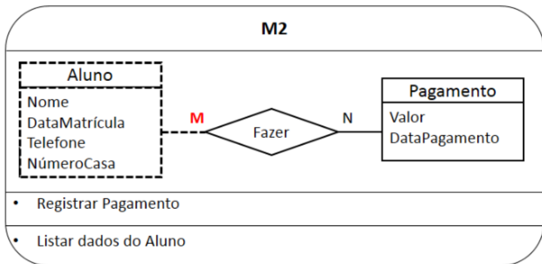


Figura 6.11 - Esquema de dados do módulo M2 na quarta iteração.

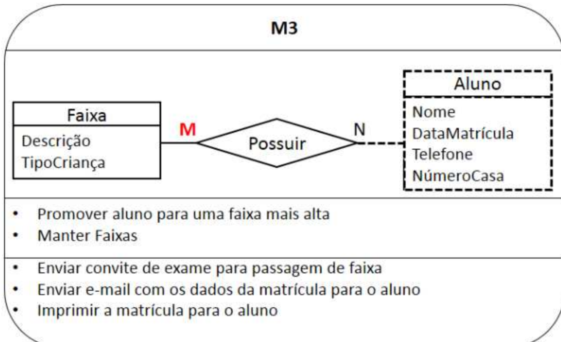


Figura 6.12 - Esquema de dados do módulo M3 na quarta iteração.

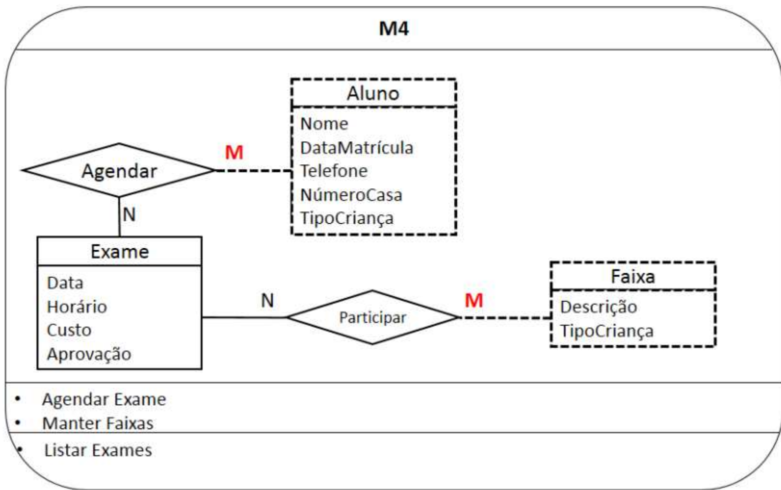


Figura 6.13 - Esquema de dados do módulo M4 na quarta iteração.

Portanto, essas iterações são exemplos de quando novos módulos são criados, como é o caso de M4, e quando módulos já existentes são estendidos, como M1 e M3. Tanto a criação como a extensão de módulos tem como objetivo atender as novas funcionalidades.

6.2.4 Quinta iteração

A princípio, o sistema foi desenvolvido para apenas um tipo de arte marcial, o caratê, mas os requisitos da quinta iteração introduzem uma mudança significativa, pois outros cursos foram introduzidos. Esses cursos possuem sistemas diferentes de faixas e alguns nem mesmo possuem faixas.

As funcionalidades para os cursos são agrupadas no subsistema S5, enquanto as matrículas nos cursos são colocadas em S1. Pois, a fase de "Análise Evolutiva dos Requisitos de Modularização" indica que os módulos M1 e M3 atendem parcialmente os requisitos, em que o primeiro mantém os dados do aluno e o segundo os dados das faixas. A Figura 6.14 resume a execução da fase de "Projeto Evolutivo de Modularização".

A adição dos novos cursos implica que um aluno passa a poder ter mais de uma faixa. Caso o mapeamento tivesse sido realizado com o projeto tradicional de bases de dados o esquema resul-

tante seria o da Figura 6.5, em que a multiplicidade do CR *Possuir* é 1:N. Entretanto, na modularização esse CR foi mapeado como um inter-relacionamento, logo a multiplicidade é M:N, conforme definição do módulo M3 (Figura 6.12).

Módulo	Iteração	Elemento	Tipo do Elemento	Tipo da Operação (L - Leitura / E - Escrita)					Tipo de Compartilhamento	É mantido por				
				S1	S2	S3	S4	S5		S1	S2	S3	S4	S5
M1	1	Aluno	CE	L/E	L	L	L	L	Unidirecional					
		Endereço	CE	L/E					Não-Compartilhado					
		Morar	CR	L/E					Não-Compartilhado					
	3	Supensão Matrícula	CE	L/E					Não-Compartilhado					
		Fazer2	CR	L/E					Não-Compartilhado					
	4	Família	CE	L/E					Não-Compartilhado					
		Membro De	CR	L/E					Não-Compartilhado					
	5	Cursar	**Inter-Relacionamento**	L/E				L	Unidirecional					
M2	1	Pagamento	CE		L/E				Não-Compartilhado					
		Fazer	**Inter-Relacionamento**		L/E				Não-Compartilhado					
M3	2	Faixa	CE			L/E	L	L	Unidirecional					
		Possuir	**Inter-Relacionamento**			L/E			Não-Compartilhado					
M4	3	Exame	CE				L/E		Não-Compartilhado					
		Agendar	**Inter-Relacionamento**				L/E		Não-Compartilhado					
		Participar	**Inter-Relacionamento**				L/E		Não-Compartilhado					
M5	5	Curso	CE					L/E	Não-Compartilhado					
		Ter	**Inter-Relacionamento**					L/E	Não-Compartilhado					

Figura 6.14 - Resumo do Projeto Evolutivo de Modularização na quinta iteração.

Os novos módulos M5 e a extensão do módulo M1, com a inclusão do CR *Cursar*, são apresentados na Figura 6.15 e Figura 6.16, respectivamente. É importante destacar que o CR *Cursar* é um inter-relacionamento entre o CE *Aluno*, do módulo M1 e o CE *Curso* do novo módulo M5.

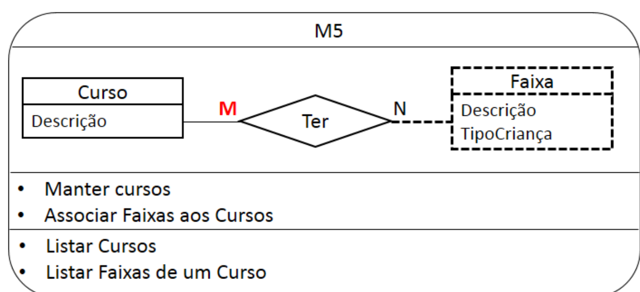


Figura 6.15 - Esquema de dados do módulo M5 na quinta iteração.

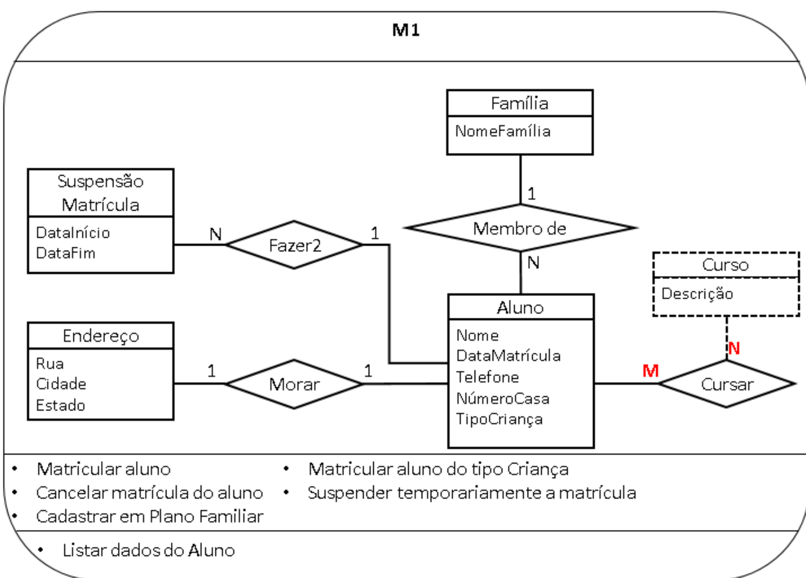


Figura 6.16 - Esquema de dados do módulo M1, que foi estendido na quinta iteração.

6.2.5 Sexta iteração

Os requisitos da sexta iteração introduzem funcionalidades de uma loja para venda de produtos tanto para os estudantes como para consumidores em geral que frequentam a escola. Essas funcionalidades são agrupadas no subsistema S6. A Figura 6.17 apresenta o resumo do "Projeto Evolutivo de Modularização", cujos resultados são discutidos a seguir.

O novo módulo M6 dá suporte às funcionalidades do Subsistema S6 e, na abordagem evolutiva, uma inter-generalização, representada por *AlunoCliente*, é utilizada ao invés da criação de uma hierarquia de generalização. Ambos os CEs *AlunoCliente* e *Cliente* possuem o atributo *CartãoCrédito*, entretanto o CE *AlunoCliente* possui o atributo *Desconto* que se aplica somente aos clientes do tipo *Aluno*. Com isso, a inter-generalização permite que os CEs evoluam independentemente, estando cada um em módulos diferentes.

Já o módulo M7 surge do compartilhamento multidirecional no CE *Endereço* em que tanto S1 quanto S6 fazem operações de escrita. A Figura 6.18 mostra a alteração no módulo M1, os módulos M6 e M7 são apresentados na Figura 6.19 e Figura 6.20, respectivamente. Os relacionamentos existentes entre o CE *Endereço* com os demais módulos são então transformados em inter-relacionamentos, conforme mostra a Figura 6.20.

Módulo	Iteração	Elemento	Tipo do Elemento	Tipo da Operação (L - Leitura / E - Escrita)						Tipo de Compartilhamento	É mantido por					
				S1	S2	S3	S4	S5	S6		S1	S2	S3	S4	S5	S6
M1	1	Aluno	CE	L/E	L	L	L	L	L	Unidirecional						
	6	Morar	**Inter-Relacionamento**	L/E						Não-Compartilhado						
	3	Supensão Matrícula	CE	L/E						Não-Compartilhado						
		Fazer2	CR	L/E						Não-Compartilhado						
	4	Família	CE	L/E						Não-Compartilhado						
		Membro De	CR	L/E						Não-Compartilhado						
	5	Cursar	**Inter-Relacionamento**	L/E				L		Unidirecional						
M2	1	Pagamento	CE		L/E					Não-Compartilhado						
		Fazer	**Inter-Relacionamento**		L/E					Não-Compartilhado						
M3	2	Faixa	CE			L/E	L	L		Unidirecional						
		Possuir	**Inter-Relacionamento**			L/E				Não-Compartilhado						
M4	3	Exame	CE				L/E			Não-Compartilhado						
		Agendar	**Inter-Relacionamento**				L/E			Não-Compartilhado						
		Participar	**Inter-Relacionamento**				L/E			Não-Compartilhado						
M5	5	Curso	CE					L/E		Não-Compartilhado						
		Ter	**Inter-Relacionamento**					L/E		Não-Compartilhado						
M6	6	AlunoCliente	**Inter-Generalização**	L					L/E	Unidirecional						
		Entregar	**Inter-Relacionamento**						L/E	Não-Compartilhado						
		Entregar2	**Inter-Relacionamento**						L/E	Não-Compartilhado						
		Fazer3	CR						L/E	Não-Compartilhado						
		Fazer4	CR						L/E	Não-Compartilhado						
		Cliente	CE						L/E	Não-Compartilhado						
		Pedido	CE						L/E	Não-Compartilhado						
		Item	CE						L/E	Não-Compartilhado						
M7	6	Endereço	CE	L/E					L/E	Multidirecional						

Figura 6.17 - Resumo do Projeto Evolutivo de Modularização na sexta iteração.

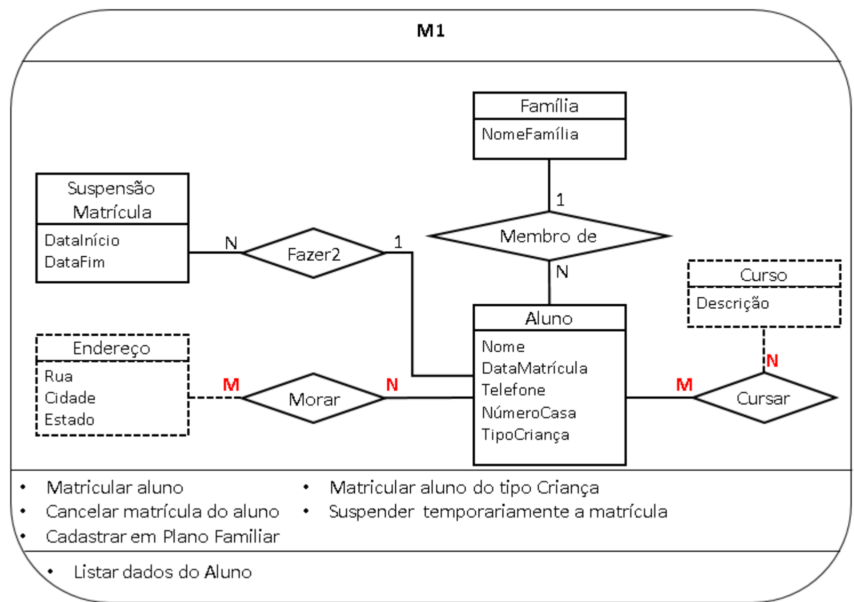


Figura 6.18 - Esquema de dados do módulo M1 na sexta iteração.

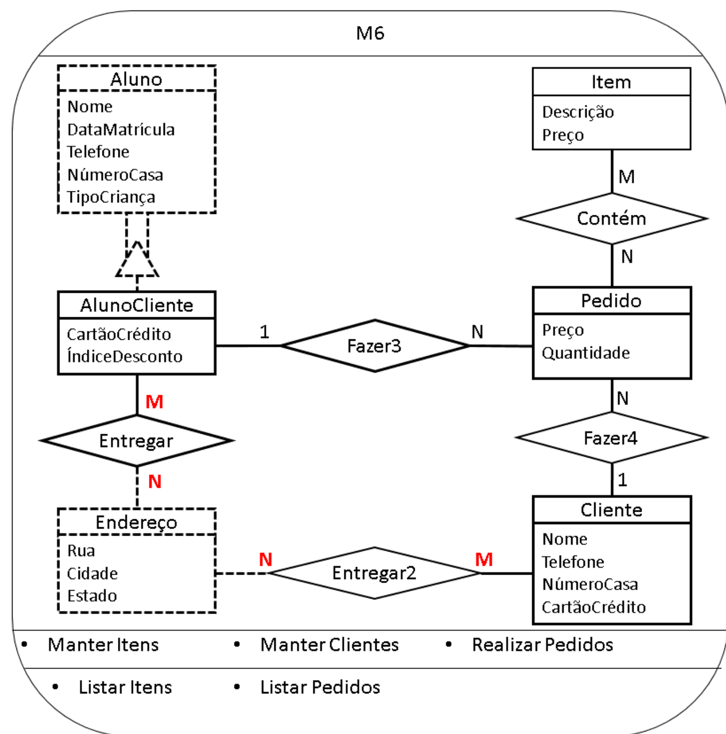


Figura 6.19 - Esquema de dados do módulo M6 na sexta iteração.

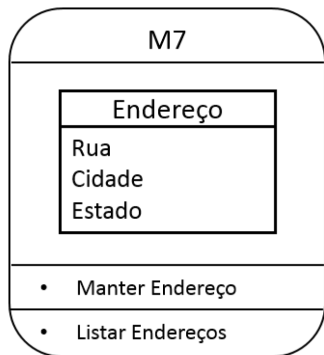


Figura 6.20 - Esquema de dados do módulo M7 na sexta iteração.

6.3 Considerações finais

Neste capítulo, o processo evolutivo de modularização de bases de dados, descrito no Capítulo 5 foi aplicado em um projeto desenvolvido de maneira iterativa e incremental. A padronização na condução das evoluções no esquema de dados reduziu o escopo de análise aos subsistemas e aos módulos de dados que seriam afetados a cada iteração. Essa característica também facilita a

rastreabilidade, uma vez que um módulo é composto por um esquema de dados e as transações que são executadas em seu esquema.

O uso de inter-relacionamentos evitou que refatorações fossem realizadas, quando da mudança de multiplicidade entre relacionamentos que envolviam dois módulos distintos. Já o uso de inter-generalização reduz o acoplamento que existe entre elementos da base de dados, ao não produzir extensas hierarquias de generalização.

O processo evolutivo de modularização de bases de dados se mostrou uma alternativa para o desenvolvimento de projetos iterativos e incrementais de bases de dados.

No próximo capítulo, a ferramenta *Evolutio DB Designer*, que implementa o processo evolutivo de modularização de bases de dados é apresentada, contemplando desde a elaboração do esquema conceitual até a geração física dos módulos de base de dados.

7 *Evolutio DB Designer*: ferramenta de modularização de bases de dados

Neste capítulo é apresentada a ferramenta *Evolutio DB Designer*, desenvolvida no trabalho. A *Evolutio DB Designer* implementa o processo evolutivo de modularização de bases de dados apresentado no Capítulo 5, permitindo especificar o esquema conceitual da base de dados a cada iteração, definir os subsistemas e gerar automaticamente os módulos de bases de dados.

Este capítulo está organizado da seguinte maneira: a Seção 7.1 descreve a arquitetura da *Evolutio DB Designer* e as ferramentas e aplicações utilizadas em seu desenvolvimento. A Seção 7.2 relaciona os recursos da *Evolutio DB Designer* com as fases e etapas do processo evolutivo de modularização de bases de dados. As telas e formulários da *Evolutio DB Designer* são apresentadas na Seção 7.3. O processo de geração automatizada dos módulos de bases de dados é descrito na 0. Enfim, a Seção 7.5 traz as considerações finais.

7.1 Arquitetura da *Evolutio DB Designer*

A *Evolutio DB Designer* foi desenvolvida com as ferramentas e aplicações descritas no Quadro 7.1, todas de distribuição livre. Os apêndices A, B e C descrevem o uso do *Twig, Doctrine* e *DBAL*, respectivamente.

Quadro 7.1- Lista de ferramentas e aplicações utilizadas no desenvolvimento da Evolutio DB Designer.

Descrição	Ferramenta / Aplicação	Versão
Linguagem de programação	PHP	5.4.9
Servidor de páginas *Web*	Apache	2.2.22
Template engine	Twig	1.12.1
Mapeamento objeto-relacional	Doctrine	2.3.2
API de abstração de bases de dados	*Database Abstraction & Access Layer*	2.3.2
Framework javascript	jQuery	1.9.2

A *Evolutio DB Designer* foi desenvolvida para implementar especificamente o processo evolutivo de modularização de bases de dados. Portanto, a integração dos módulos é feita por meio do recurso de engenharia reversa provida pelo mapeamento objeto-relacional do *Doctrine*. Desse modo, o esquema de dados do catálogo do objeto integrador possui apenas a extensão do esquema

de dados proposto por Busichia (1999). Portanto, o esquema implementado na *Evolutio DB Designer* corresponde ao subesquema destacado na Figura 5.6. Além disso, também é mantida uma instância do catálogo referente à iteração imediatamente anterior, para a comparação dos esquemas de dados entre as iterações. A arquitetura da *Evolutio DB Designer* apresentada na Figura 7.1.

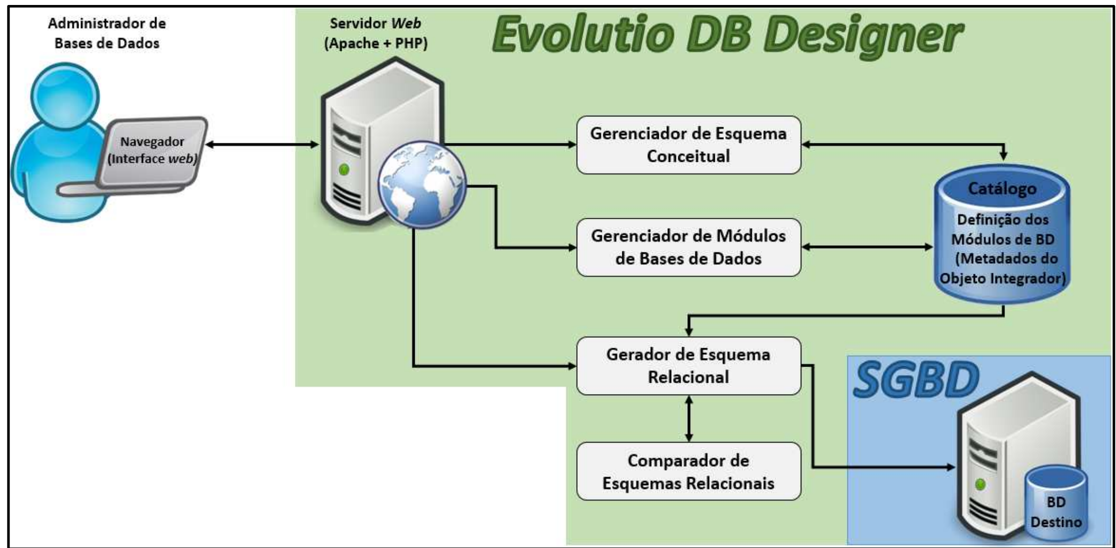


Figura 7.1 - Arquitetura da *Evolutio DB Designer*.

A lista a seguir descreve os quatro componentes principais da *Evolutio DB Designer*:

1. **Gerenciador de Esquema Conceitual:** responsável por manter o esquema conceitual global da aplicação. Possui métodos para a criação e edição de CEs, CRs e hierarquias de generalização. Portanto, o esquema conceitual de dados de cada iteração é alterado por este componente;
2. **Gerenciador de Módulos de Bases de Dados:** responsável por manter os subsistemas e definir e manter os módulos de bases de dados. Possui métodos para a criação e edição de subsistemas. Por fim, utiliza a definição dos subsistemas para definir os módulos de bases de dados. Os detalhes da definição dos módulos de bases de dados são descritos na Seção 7.4;
3. **Gerador de Esquema Relacional:** gera um esquema relacional correspondente aos módulos de bases de dados definidos pelo “Gerenciador de Módulos de Bases de Dados”. A API de abstração de bases de dados relacionais *DBAL* (ref. Apêndice C) é utilizada na geração do esquema relacional. Em seguida, encaminha o esquema

relacional gerado ao “Comparador de Esquemas Relacionais”. As regras de transformação do esquema conceitual de cada módulo para o relacional são detalhadas na Seção 7.4;

4. **Comparador de Esquemas Relacionais:** responsável por comparar a versão atual e anterior do esquema relacional. Portanto, gera os comandos necessários para realizar a evolução do esquema físico da base de dados de destino.

7.2 Recursos da *Evolutio DB Designer*

Ao todo a *Evolutio DB Designer* possui cinco recursos principais, apresentados Figura 7.2, para a execução do processo evolutivo de modularização de bases de dados.

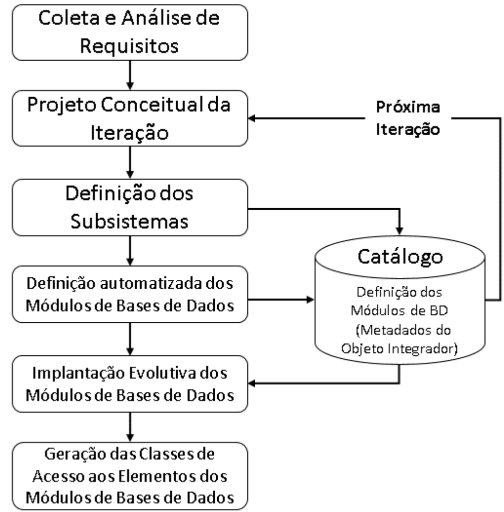


Figura 7.2 - Fluxo de uso dos recursos da *Evolutio DB Designer*.

A lista a seguir descreve os cinco recursos principais apresentados na Figura 7.2:

1. **Coleta e Análise de Requisitos:** os requisitos são coletados de acordo com o método de elicitação escolhido. Posteriormente, os requisitos de dados são utilizados para a elaboração do esquema conceitual de dados da iteração;
2. **Projeto Conceitual da Iteração:** o esquema conceitual da base de dados da iteração é elaborado com os recursos providos pela *Evolutio DB Designer*, sendo gravado no catálogo do objeto integrador, por meio do componente “Gerenciador de

Esquema Conceitual". A seta que parte do catálogo para o projeto conceitual da aplicação indica que nas iterações subsequentes os esquemas são elaborados partindo-se dos metadados já armazenados no catálogo do objeto integrador;

3. **Definição dos subsistemas:** após a definição do esquema conceitual de dados da iteração, os subesquemas são definidos pela seleção dos elementos que os compõem juntamente com o tipo de operação (leitura ou escrita) realizado em cada um deles, de acordo com os subsistemas correspondentes. A definição de cada subsistema também é gravada no catálogo do objeto integrador, sendo o componente "Gerenciador de Módulos de Bases de Dados" responsável por essas operações;
4. **Definição automatizada dos módulos de bases de dados:** com os subsistemas definidos é possível gerar a definição dos módulos de bases de dados da iteração de maneira automática. O componente "Gerenciador de Módulos de Bases de Dados" é responsável por automatizar a definição. Os detalhes da automatização são apresentados na Seção 7.4. A definição de cada módulo é armazenada no catálogo do objeto integrador;
5. **Implantação evolutiva dos módulos de bases de dados:** o esquema conceitual modularizado da iteração é transformado em um esquema relacional, por meio do componente "Gerador de Esquema Relacional". Em seguida, o "Comparador de Esquemas Relacionais" gera o esquema relacional correspondente às definições dos módulos da iteração anterior. Depois, compara os dois esquemas, gerando os comandos necessários para a aplicação das diferenças encontradas na base de dados de destino, que se encontra operacional;
6. **Geração das classes de acesso aos elementos dos módulos de bases de dados:** com o esquema modularizado de dados implantado é necessário prover acesso, para a aplicação, aos elementos que o compõe. Desse modo, esse componente permite que a *Evolutio DB Designer* gere classes que contenham o mapeamento objeto-relacional, com base na definição dos módulos de bases de dados, por meio do recurso de engenharia reversa do *Doctrine*. Portanto, são definidas interfaces de acesso aos módulos de bases de dados.

O Quadro 7.2 mostra a relação entre os recursos da *Evolutio DB Designer* com as fases e etapas do processo evolutivo de modularização de bases de dados, descritas no Capítulo 5.

Quadro 7.2 - Recursos da *Evolutio DB Designer* e as fases ou etapas correspondentes no processo evolutivo de modularização de bases de dados.

Recurso da *Evolutio DB Designer*	**Fase/Etapa do processo evolutivo de modularização de bases de dados**
Coleta e análise de requisitos	Coleta e análise de requisitos
Projeto Conceitual da Iteração	Projeto Conceitual da Iteração
Definição dos subsistemas	Análise Evolutiva dos Requisitos de Modularização - *Primeira Execução*
	Particionamento do esquema conceitual global de dados
	Tratamento do compartilhamento da informação
Definição automatizada dos módulos de bases de dados	Análise Evolutiva dos Requisitos de Modularização - *Segunda Execução*
	Definição evolutiva dos módulos da base de dados
Implantação evolutiva dos módulos de bases de dados	Projeto Lógico de cada módulo
	Projeto Físico de cada módulo
Geração das classes de acesso aos elementos dos módulos	Definição da interface dos módulos da base de dados

Em relação às correspondências entre o uso da *Evolutio DB Designer* e o processo evolutivo de modularização de bases de dados é importante destacar que a fase de “Análise Evolutiva dos Requisitos de Modularização” é utilizada em dois momentos. Primeiro na definição dos subsistemas. Posteriormente, os componentes “Gerador de Esquema Relacional” e “Comparador de Esquemas Relacionais”, verificam o compartilhamento da informação de cada subsistema da iteração atual e anterior, gerando os requisitos de modularização de dados, que determinam se haverá a criação ou extensão de módulos de bases de dados.

7.3 Telas e formulários da *Evolutio DB Designer*

Esta seção apresenta as telas e formulários criados na *Evolutio DB Designer* para a execução do processo evolutivo de modularização de bases de dados.

Para a elaboração do esquema conceitual de cada iteração a *Evolutio DB Designer* provê os conceitos de conjunto de entidades (CE), conjunto de relacionamentos (CR), atributos e hierarquias de generalização, do modelo entidade-relacionamento estendido. Entretanto, a abstração de agregação não foi incluída na *Evolutio DB Designer*, ficando para implementação futura.

O formulário para a criação de um CE é apresentado na Figura 7.3. O componente "Gerador de Esquema Relacional" requer que os atributos tenham um domínio definido. Portanto, os tipos dos atributos são escolhidos dentre os tipos de dados existentes no *Doctrine*. A relação dos tipos do *Doctrine* são descritos no Quadro B.2 do Apêndice B. De acordo com a Figura 7.3, a adição dos atributos é feita na opção *Add Atribute* e as chaves são definidas pela marcação no campo *Key?*.

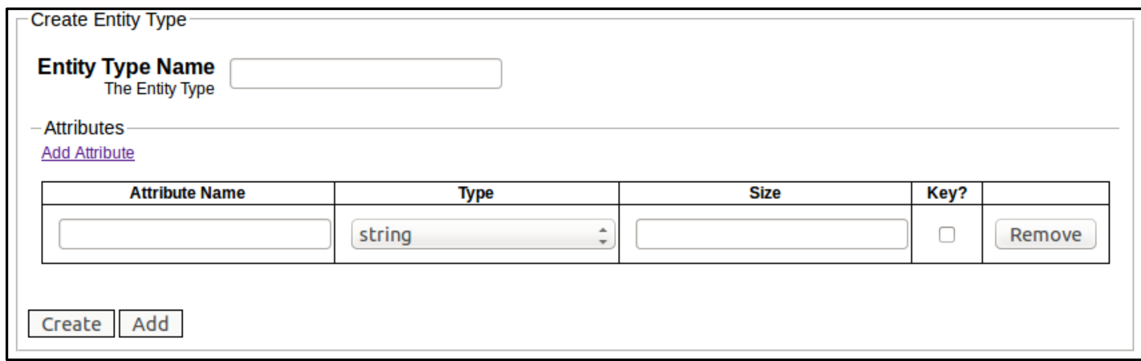


Figura 7.3 - Formulário para criação de conjunto de entidades.

O formulário para criação de um CR é apresentado na Figura 7.4. A definição de um CR está restrita a relacionamentos binários e os atributos devem pertencer a um dos tipos de dados do *Doctrine*, assim como é feito para a definição dos atributos de um CE.

Figura 7.4 - Formulário para criação de conjunto de relacionamentos.

Uma generalização é definida pela seleção de um CE já existente como genérico, seguida pela informação do nome e dos atributos do CE específico. A Figura 7.5 apresenta o formulário para a criação de uma generalização.

Figura 7.5 - Formulário para criação de uma generalização.

Após a elaboração do esquema conceitual da iteração, o próximo passo é a definição dos subsistemas. Um subsistema é definido por meio da seleção dos CRs que compõem o seu subesquema correspondente. A seleção de um CR faz com que os CEs participantes do relacionamento sejam incluídos no subesquema acessado pelo subsistema, além de liberar os campos que indicam os tipos de operações, leitura ou escrita (*read* e *write*, respectivamente), realizados pelo subsistema aos elementos selecionados, conforme mostra a Figura 7.6.

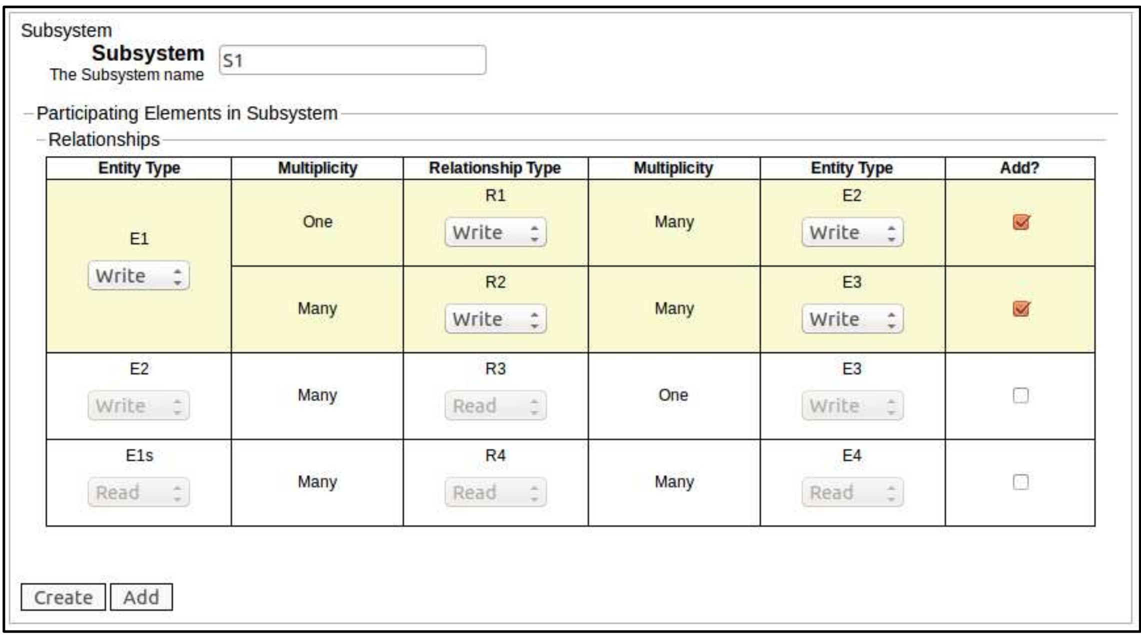


Figura 7.6 - Definição de um subsistema.

Com os subsistemas definidos é possível apresentar uma matriz com o resultado da etapa de "Tratamento do compartilhamento da informação", parte da fase de "Projeto Evolutivo de Modularização", descrita na Seção 5.3. A matriz apresenta o tipo de operação realizada por cada subsistema aos elementos do esquema conceitual de dados, o tipo de compartilhamento resultante (não-compartilhado, unidirecional ou multidirecional) e, por fim, indica os subsistemas responsáveis pela manutenção de cada elemento. Os resultados apresentados são gerados dinamicamente. Portanto, caso um subsistema seja alterado, em uma iteração subsequente, pela inclusão de novos elementos ou mudanças nos tipos de operações realizadas, a matriz irá refletir essas alterações.

Information Sharing Matrix

Element	Type of Operation				Type of Sharing	Mantained By			
	S1	S2	S3	S4		S1	S2	S3	S4
E1	Write	Read	-	-	Unidirectional	S1	-	-	-
E2	Write	-	-	-	Non-Shared	S1	-	-	-
E3	Write	Write	-	-	Multidirectional	S1	S2	-	-
R1	Write	-	-	-	Non-Shared	S1	-	-	-
R2	Write	Read	-	-	Unidirectional	S1	-	-	-
E1s	-	-	Write	Write	Multidirectional	-	-	S3	S4
E4	-	-	Write	Write	Multidirectional	-	-	S3	S4
R4	-	-	Write	Write	Multidirectional	-	-	S3	S4

Figura 7.7 - Matriz de compartilhamento da informação.

O pré-requisito para a definição automatizada dos módulos de bases de dados é que os subsistemas estejam definidos. Entretanto, não é necessário que a matriz de compartilhamento da informação seja gerada, pois os algoritmos foram implementados de modo independente. A Figura 7.8 apresenta a tela com o resultado da geração dos módulos, indicando também os casos de inter-relacionamentos e inter-generalizações.

Database Modules

Module	Element	Element Type	Interrelationship?	Intergeneralization?
M0	E3	Entity Type	No	No
M1	R2	Relationship	Yes	No
	R1	Relationship	No	No
	E2	Entity Type	No	No
	E1	Entity Type	No	No
M2	R4	Relationship	No	No
	E4	Entity Type	No	No
	E1s	Sub Entity Type	No	Yes

Figura 7.8 - Definição dos módulos de bases de dados.

7.4 Definição automatizada dos módulos de bases de dados e sua implantação

A definição automatizada dos módulos de bases de dados é feita baseada nas definições dos subsistemas contidos no catálogo do objeto integrador.

O Algoritmo 1 é responsável por definir os módulos de bases de dados. A entrada do algoritmo é um mapa ordenado com cada um dos subsistemas e uma lista dos elementos que cada um deles mantém, portanto, aqueles elementos em que o subsistema realiza as operações de escrita. A saída do algoritmo faz com que um elemento pertença a apenas um módulo de bases de dados. Também faz com que elementos que possuam compartilhamento multidirecional pertençam a um novo módulo de bases de dados, com os devidos inter-relacionamentos. O objetivo é manter o baixo acoplamento entre os módulos.

Algoritmo 1 Definição dos módulos de bases de dados.

Entrada: Mapa ordenado contendo os subsistemas e o conjunto de elementos que eles mantém *S*.

Saída: Mapa ordenado contendo os módulos de bases de dados *M*, com seus respectivos elementos.

```
1:  início
2:    M ← Ø
3:    n = 0
4:    i = 0
5:    enquanto recuperarElementos(S) != Ø faz
6:      j = i + 1
7:      X ← S[i]
8:      Y ← S[j]
9:      enquanto recuperarElementos(Y) != Ø faz
10:       X ← X ∩ Y
11:       Y ← S[j]
12:       j ← j + 1
13:     fimenquanto
14:     M[n] = X
15:     n ← n + 1
16:     para cada (chave, mapaElementosDeS) em S faça
17:       S[chave] ← mapaElementosDeS - M[n]
18:     fimpara
19:     se recuperarElementos(S) == Ø então
20:       i ← i + 1
21:     fimse
22:   fimenquanto
23: fim
```

Com os módulos definidos o próximo passo é implantá-los em uma base de dados de destino. O SGBD responsável pelo gerenciamento da base de dados de destino é registrado na *Evolutio DB Designer* pelo preenchimento dos campos do formulário apresentado na Figura 7.9. A *Evolutio DB Designer* é compatível com os SGBDs PostgreSQL e MySQL.

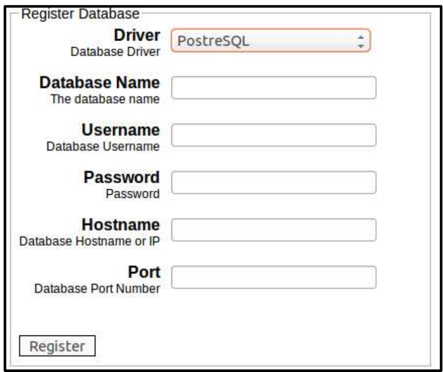


Figura 7.9 - Registro de SGBD.

A implantação do esquema relacional segue a seguinte ordem:

1. O componente “Gerador de Esquema Relacional” é acionado para gerar o esquema relacional com base nas definições do esquema conceitual de dados de cada módulo de bases de dados, que estão armazenadas no catálogo de metadados do objeto integrador. As regras de mapeamento estão definidas no Quadro 7.3;
2. Em seguida, o “Comparador de Esquemas Relacionais” gera o esquema relacional da iteração anterior e o compara com o esquema relacional da iteração atual. Por fim, as diferenças encontradas são aplicadas no esquema relacional da base de dados de destino.

Quadro 7.3 - Mapeamento dos elementos do esquema conceitual de dados de cada módulo de bases de dados para o esquema relacional.

(continua)

Tipo do elemento no *Evolutio DB Designer*	Tipo correspondente no esquema relacional
Conjunto de entidades	Tabela, contendo uma coluna adicional do tipo inteiro com uma chave artificial sequencial.
Atributos	Mapeados como colunas. O tipo de dados *Doctrine* definido para o atributo é mapeado para o tipo correspondente de acordo com o SGBD de destino escolhido.

Quadro 7.3 - Mapeamento dos elementos do esquema conceitual de dados de cada módulo de bases de dados para o esquema relacional.

(conclusão)

Tipo do elemento no *Evolutio DB Designer*	Tipo correspondente no esquema relacional
Conjunto de relacionamentos	Determinado pela multiplicidade: • 1:1, uma coluna é adicionada em qualquer uma das tabelas participantes do relacionamento. • 1:N, uma coluna é adicionada como chave estrangeira na tabela que, como CE, participa do relacionamento com a multiplicidade N. • M:N, uma tabela é criada com as chaves das tabelas que, como CEs, participam do relacionamento.
Hierarquias de generalização	Cada CE é mapeado para uma tabela. Uma chave estrangeira é criada, em cada tabela que corresponde a um CE específico, referenciando a tabela correspondente ao CE genérico.
Inter-relacionamento	Mapeado como M:N, portanto uma tabela é criada com as chaves das tabelas que, como CEs, participam do relacionamento.
Inter-generalização	CE específico é mapeado como uma tabela, pertencente ao módulo que realiza as operações de escrita.

A *Evolutio DB Designer* não permite a implantação de módulos de bases de dados em mais de um de SGBD, portanto é uma solução para ambiente homogêneo. Entretanto, é possível estender o componente “Gerador de Esquema Relacional” para ambientes heterogêneos, para que atenda a mais de uma base de dados de destino. Também é possível adicionar novos componentes que atendam outros tipos de bases de dados além das que implementam o modelo relacional.

Por fim, após a implantação do esquema relacional na base de dados de destino a *Evolutio DB Designer* permite, por meio do recurso de engenharia reversa do *Doctrine*, a geração de classes com o mapeamento objeto-relacional correspondente ao esquema relacional implantado. Com isso, o acesso aos elementos da base de dados modularizada é disponibilizado.

7.5 Considerações finais

A evolução de um esquema de base de dados, quando do uso de um método de desenvolvimento iterativo e incremental, exige que o projeto da base de dados seja documentado e que as alterações realizadas nas iterações ulteriores sejam conduzidas de maneira segura.

Nesse cenário, a *Evolutio DB Designer* implementa o processo evolutivo de modularização de bases de dados, descrito no Capítulo 5. Trata-se de um processo que gera um incremento à base

de dados, em que as informações de cada iteração são armazenadas como metadados no catálogo do objeto integrador.

A flexibilidade do esquema de dados gerado e implantado pela *Evolutio DB Designer* é atingida, pois implementa as regras de mapeamento definidas no processo evolutivo de modularização de bases de dados, especialmente os inter-relacionamentos e inter-generalizações.

A aplicação do processo evolutivo de modularização de bases de dados pode se tornar complexa em esquemas de dados com muitos elementos. Entretanto, com o uso da *Evolutio DB Designer*, a documentação do projeto é mantida no catálogo do objeto integrador e parte das alterações no esquema ficam limitadas apenas às alterações nas definições dos subsistemas.

As interfaces de acesso aos módulos de bases de dados são geradas pelo recurso de engenharia reversa do *Doctrine*, provendo acesso ao esquema modularizado de dados implantado pelo componente de "Gerador de Esquema Relacional".

Por fim, a arquitetura da *Evolutio DB Designer* é baseada em componentes, portanto é possível estendê-la pela inclusão de novos componentes, com o objetivo de dar suporte a novas funcionalidades.

No próximo capítulo, a *Evolutio DB Designer* é utilizada em um estudo de caso que considera as alterações realizadas na base de dados de um sistema de gerenciamento de conteúdo, o MediaWiki, que é o software utilizado pela Wikipedia.

8 Estudo de caso

Este capítulo apresenta um estudo de caso da aplicação do processo evolutivo de modularização de bases de dados por meio da ferramenta *Evolutio DB Designer*.

A Seção 8.1 apresenta os projetos analisados para o estudo de caso, antes de apresentar o projeto escolhido, que é descrito na Seção 8.2. A execução do estudo de caso está na Seção 8.3. Por fim, a Seção 8.4 traz as considerações finais.

8.1 Escolha do estudo de caso

Primeiramente o objetivo era realizar o estudo de caso em um projeto desenvolvido com algum método ágil, que tivesse o histórico de requisitos separados e conservados por iteração para acompanhar o processo de desenvolvimento desde a primeira fase “Coleta e Análise de Requisitos”. A seguir, cada iteração na elaboração do sistema de software e os impactos no esquema da base de dados deveriam ser acompanhados. A busca por projetos que atendessem essas condições foi conduzida em empresas e órgãos públicos da região de Campinas, além de repositórios de projetos de código aberto como o GitHub (WERNER; WANSTRATH; HYETT, 2008).

A seguir são apresentados quatro projetos analisados e os motivos que determinaram sua recusa como estudo de caso:

1. **Sistema para cadastro de alunos de pós-graduação.** Ao analisar os requisitos, constatou-se que grande parte das alterações adicionava novas funcionalidades em nível de código, tais como: novos relatórios e regras para consistência de cadastros. As modificações na base de dados foram poucas e em sua maioria estendiam o esquema, como adição de novos atributos, não demandando modificações significativas.
2. **Sistema de gerenciamento de estoque.** O órgão responsável por este sistema disponibilizou 194 casos de uso. A especificação desses possui alto nível de detalhamento. Contudo, o sistema foi desenvolvido sobre uma base de dados legada,

sendo que as alterações foram controladas em uma ferramenta de gestão de projetos e não por meio de novos casos de uso. Os dados referentes às alterações não foram disponibilizados.

3. **Sistema de gerenciamento de materiais.** Esse sistema foi desenvolvido pelo mesmo órgão do “Sistema de gerenciamento de estoque” e 85 casos de uso foram disponibilizados. Os mesmos problemas reportados no sistema anterior se aplicam a esse sistema, ou seja, uso de base de dados legada e não foi disponibilizado acesso à ferramenta que controlava as alterações.
4. **Sistema de cadastro e gerenciamento de torres de celular.** O contato inicial mostrou-se promissor, inclusive um exemplo detalhado de requisito com alteração na base de dados foi disponibilizado. Mas houve negativa para o envio de todos os requisitos do sistema, impossibilitando seu uso como estudo de caso.

O principal desafio enfrentado foi a resistência das empresas e órgãos públicos para a liberação dos requisitos dos sistemas. Entretanto, todos esses contatos afirmaram que alterações no esquema de dados são problemas recorrentes tanto em projetos com desenvolvimento iterativo e incremental, quanto em evoluções de sistemas legados.

8.2 Descrição do estudo de caso

O estudo de caso escolhido foi baseado nas modificações realizadas no esquema de dados do MediaWiki (WIKIMEDIA FOUNDATION, 2002) entre 2003 e 2008, totalizando 171 versões de esquemas de dados. O objetivo foi aplicar o processo evolutivo de modularização de bases de dados, descrito no Capítulo 5, assim como testar a ferramenta *Evolutio DB Designer*, apresentada no Capítulo 7, na evolução incremental do esquema de dados de um sistema real.

O MediaWiki é um sistema de software de código aberto, escrito em PHP, para gerenciamento de conteúdo colaborativo para a Internet, inicialmente criado como um software para a Wikipedia (WIKIMEDIA FOUNDATION, 2001). Ainda hoje, o MediaWiki é o software utilizado na Wikipedia, mas também está disponível na Internet e é possível customizá-lo para outros ambientes.

A arquitetura do MediaWiki é apresentada na Figura 8.1.

Figura 8.1 - Arquitetura do MediaWiki (CURINO et al., 2008).

Um arquivo para cada uma das 171 versões de esquemas, utilizadas nos estudos de Curino et al. (2008), estão disponibilizados e acessíveis na Internet (CURINO, 2008). Cada arquivo contém as instruções DDL para a geração de um esquema de dados relacional de uma base dados do SGBD MySQL. A DDL das versões mais antigas dos esquemas possuíam incompatibilidades com a versão atual do MySQL, como palavras-chaves e definição de índices. Portanto, algumas adaptações foram necessárias, mas sem impactar na evolução dos esquemas.

Curino et al. (2008) agruparam as principais alterações, realizadas ao longo de quatro anos e meio, no esquema de dados relacional do MediaWiki. O Quadro 8.1 mostra o número de alterações e a porcentagem para cada tipo.

Quadro 8.1 - Número de alterações realizadas no esquema de dados (CURINO et al., 2008).

Tipo de Alteração	**# Operações de evolução**	**% Operações de evolução**
Diretamente no esquema	94	54,9%
Índice/Chave	69	40,3%
Tipo de dados	22	12,8%
Correção de sintaxe	20	11,7%
Rollback	15	8,8%
Documentação	13	7,6%
Engine	6	3,5%

As alterações do tipo *"Diretamente no esquema"* indicam modificações que ocorreram no esquema de dados. As do tipo *"Índice/Chave*" referem-se às alterações realizadas em chaves e índices, por motivos de integridade referencial e de desempenho. As alterações nos tipos de dados dos atributos são representadas por *"Tipo de dados"*. A *"Correção de sintaxe"* indica correções que tiveram que ser feitas na sintaxe da DDL, por motivo de erro de escrita nos comandos. A

“Rollback” refere-se reversões em alterações, que retornaram parte do esquema a uma versão anterior. As alterações que tiveram impacto apenas na documentação estão representadas por *“Documentação”*. Por fim, as alterações necessárias para prover compatibilidade com uma nova versão do SGBD foram agrupadas em *“Engine*”.

Os três principais fatores que levaram à extensão do esquema de dados foram: melhorias visando aumentar o desempenho, como a introdução de tabelas de *cache*; adição de novas funcionalidades, como um *log* para validação de conteúdo; necessidade de se preservar o histórico do conteúdo da base de dados, por meio de remoções lógicas antes de se apagar os dados em definitivo.

Os passos executados para a configuração do ambiente do estudo de caso são apresentados a seguir:

1. Criação de uma base de dados para cada versão de esquema disponível em Curino (2008), totalizando 171 bases de dados;
2. Cada base de dados foi comparada, com a sua versão imediatamente anterior, na ferramenta DB Solo (DB SOLO LLC, 2013). Por meio desta ferramenta é possível detectar, por exemplo, a adição ou remoção de tabelas e colunas. A Figura 8.2 mostra a comparação entre dois esquemas de dados e, de acordo com os destaques, na versão 01307 a tabela *random* está presente e na versão 01308 essa tabela está ausente;

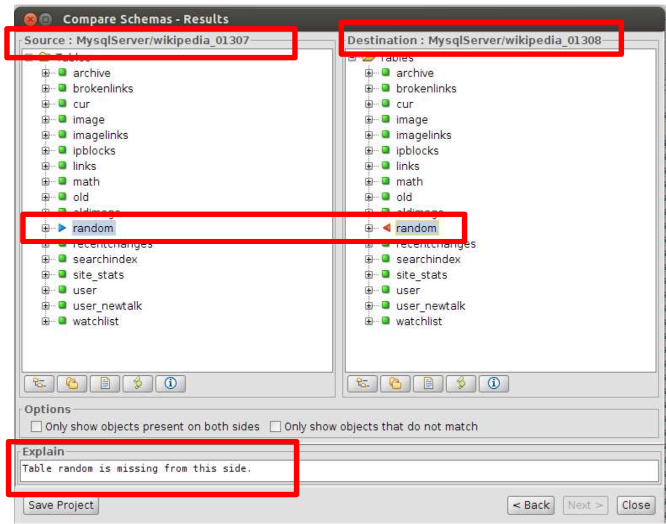


Figura 8.2 - Comparação de esquemas na ferramenta DB Solo.

3. As comparações foram realizadas com o objetivo de se detectar as versões que introduziram modificações significativas ao esquema de dados do MediaWiki. Por fim, apesar de existirem versões intermediárias, aquelas que constam no Quadro 8.2 são as que tiveram maior impacto na alteração do esquema de dados;
4. Instalação da versão mais recente do MediaWiki para analisar as funcionalidades do sistema em relação ao esquema de dados;
5. Com base no esquema lógico relacional de cada versão e na documentação do MediaWiki, a *Evolutio DB Designer* foi utilizada para a elaboração do esquema conceitual inicial da base dados e nas evoluções subsequentes. Os esquemas conceituais de dados foram obtidos por meio de engenharia reversa dos esquemas relacionais existentes. As regras aplicadas nesse processo foram as definidas por Heuser (2009), descritas na Seção 3.5 e contidas no Quadro 3.3. Além disso, foi possível detectar que o esquema relacional da base de dados do MediaWiki possui alguns pontos de desnormalização, identificados por colunas cujo nome está no plural. Provavelmente introduzidas por questões de desempenho. Esses casos originaram um novo CE, para manter os dados da coluna e um CR para o devido relacionamento. Dois desses casos são exemplificados na Seção 8.3.1.

O Quadro 8.2 apresenta as versões escolhidas para o estudo de caso, por serem as que introduziram as mudanças mais significativas no esquema de dados do MediaWiki.

Quadro 8.2 - Versões do MediaWiki utilizadas no estudo de caso.

Versão	**Principal alteração**
v01284	Versão inicial do MediaWiki.
v04925	Remoção da coluna *user_rights* da tabela *user*. Os dados dessa coluna foram copiados para uma tabela própria, com o mesmo nome de *user_rights*.
v05648	Adição da tabela *user_groups* para suporte à funcionalidade de grupos.
v06072	Adição da coluna *group_rights* na tabela *user_groups* para manter as permissões de grupos.
v06710	Os artigos, que eram armazenados em duas tabelas (*cur* e *old*), passaram a ser armazenados em três tabelas: *page*, *revision* e *text*.
v09367	Remoção da tabela *groups*. Os grupos passaram a ser armazenados no código da aplicação e não mais na base de dados.
v20301	Adição de coluna na tabela *revision*, introduzindo um auto-relacionamento.
v20468	Criação de um mecanismo para remoção lógica de alguns dados, com a inclusão de colunas com o sufixo *_deleted* em algumas tabelas.

8.3 Execução do estudo de caso

Nas subseções seguintes são apresentados os resultados das alterações nos esquemas de dados realizados com o uso da *Evolutio DB Designer*. Cada subseção se refere a uma versão que consta no Quadro 8.2.

Primeiramente, foi necessário definir os subsistemas, que é uma das saídas da fase de "Análise Evolutiva dos Requisitos de Modularização". A definição final dos subsistemas foi baseada no agrupamento das tabelas por funcionalidades associadas, conforme estabelecido por Curino et al. (2008). Esse agrupamento foi realizado com as tabelas da última versão existente do esquema de dados do MediaWiki, quando da realização do trabalho de Curino et al. (2008). O Quadro 8.3 traz o agrupamento das tabelas por funcionalidades associadas.

Quadro 8.3 - Agrupamento de tabelas por funcionalidades associadas (CURINO et al., 2008).

Funcionalidades	**Tabelas**
Gerenciamento de artigos e conteúdo	*page, revision, text, image, user_newtalk, math*
Gerenciamento de histórico e arquivamento	*archive, filearchive, oldimage e logging*
Links e estrutura do site	*categorylinks, externallinks, imagelinks, interwiki, langlinks, pagelinks, redirect, templatelinks e trackbacks*
Gerenciamento de usuários e permissões	*user, user groups, ipblocks, watchlist, page restrictions*
Desempenho e cache	*objectcache, querycache, querycache_info, job, querycachetwo, trans-cache, searchindex*
Estatísticas e suporte a recursos especiais	*recentchanges, hitcounter, site_stats*

Em relação ao agrupamento do Quadro 8.3, duas questões tiveram que ser tratadas:

1. Existem diferenças significativas entre o esquema de dados inicial (versão v01284) e a versão utilizada no agrupamento realizado por Curino et al. (2008);
2. Cada agrupamento feito por Curino et al. (2008) pode englobar mais de uma funcionalidade. Isso pode ser visto pelo uso do conectivo "e" em alguns agrupamentos, como por exemplo em "Gerenciamento de usuários **e** permissões". Apesar de estarem relacionados, são tratados como dois subsistemas. Portanto, a relação do agrupamento realizado por Curino et al. (2008) e a definição de subsistemas nem sempre será de um para um. Com isso, a lista com a definição dos subsistemas não segue a mesma definição Quadro 8.3.

8.3.1 Versão inicial - v01284

A saída da fase de "Análise Evolutiva dos Requisitos de Modularização", indicou que não as funcionalidades dessa versão não eram atendidas, pois se tratava da primeira versão. Portanto, novos módulos de bases de dados foram criados. Em seguida, o projeto conceitual dessa versão foi executado.

Os elementos com _ (*underscore*) foram introduzidos para lidar com os pontos de desnormalização. Dois exemplos de desnormalização são encontrados na coluna *user_options* e *user_rights* da tabela *user* (Figura 8.3). Na primeira são armazenadas as preferências de um usuário e na segunda suas permissões. Já no esquema conceitual elaborado na *Evolutio DB Designer*, dois CEs (*_rights* e *_options*) foram introduzidos juntamente com os CRs *has* e *has2*, conforme mostra a Figura 8.4.

Figura 8.3 - Tabela *user*, exemplos de desnormalização no esquema relacional do MediaWiki.

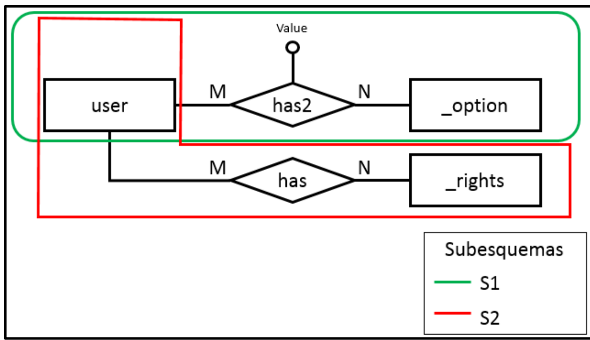


Figura 8.4 - Esquema conceitual parcial de S1 e S2.

O próximo passo foi a definição dos subsistemas. As funcionalidades de "Gerenciamento de usuários e permissões" foram dividias em três subsistemas S1, S2 e S3, respectivamente. Já o

"Gerenciamento de artigos e conteúdo" foi definido nos subsistemas S4 e S5. Enquanto o "Gerenciamento de histórico e arquivamento" foi mantido em um subsistema, pois o arquivamento é uma funcionalidade do software e refere-se a manter arquivos apagados do MediaWiki em um diretório específico no sistema de arquivos, mas em termos de esquema de dados as tabelas utilizadas são as mesmas do histórico. Por fim, as "Estatísticas e suporte a recursos especiais" são mantidas pelo subsistema S7. O Quadro 8.4 mostra os subsistemas e as funcionalidades correspondentes e a Figura 8.5 apresenta a matriz de compartilhamento da informação gerada pela *Evolutio DB Designer*.

Quadro 8.4 - Lista de subsistemas do MediaWiki da versão inicial (v01284).

Funcionalidades	**Subsistema**	**Descrição**
Gerenciamento de usuários e permissões	S1	Gerenciamento de usuários
	S2	Gerenciamento de permissões
	S3	Gerenciamento de bloqueios de acesso
Gerenciamento de artigos e conteúdo	S4	Gerenciamento de artigos
	S5	Gerenciamento de conteúdo
Gerenciamento de histórico e arquivamento	S6	Gerenciamento de histórico e arquivamento
Estatísticas e suporte a recursos especiais	S7	Suporte a recursos especiais

Information Sharing Matrix

Element	Type of Operation							Type of Sharing	Mantained By						
	S1	S2	S3	S4	S5	S6	S7		S1	S2	S3	S4	S5	S6	S7
image	Read	-	-	-	Write	-	-	Unidirectional	-	-	-	-	S5	-	-
_options	Write	-	-	-	-	-	-	Non-Shared	S1	-	-	-	-	-	-
user	Write	Read	Read	Read	Read	Read	-	Unidirectional	S1	-	-	-	-	-	-
has5	Write	-	-	-	-	-	-	Non-Shared	S1	-	-	-	-	-	-
has3	Read	-	-	-	Write	-	-	Unidirectional	-	-	-	-	S5	-	-
has2	Write	-	-	-	-	-	-	Non-Shared	S1	-	-	-	-	-	-
user_newtalk	Write	-	-	-	-	-	-	Non-Shared	S1	-	-	-	-	-	-
_rights	-	Write	-	-	-	-	-	Non-Shared	-	S2	-	-	-	-	-
has	-	Write	-	-	-	-	-	Non-Shared	-	S2	-	-	-	-	-
blocks	-	-	Write	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-
ipblocks	-	-	Write	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-
is	-	-	Write	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-
has4	-	-	-	Write	-	-	-	Non-Shared	-	-	-	S4	-	-	-
cur	-	-	-	Write	-	-	Read	Unidirectional	-	-	-	S4	-	-	-
old	-	-	-	Write	-	-	-	Non-Shared	-	-	-	S4	-	-	-
_restrictions	-	-	-	Write	-	-	-	Non-Shared	-	-	-	S4	-	-	-
_flags	-	-	-	Write	-	Read	-	Unidirectional	-	-	-	S4	-	-	-
makes	-	-	-	Write	-	-	-	Non-Shared	-	-	-	S4	-	-	-
has6	-	-	-	Write	-	-	-	Non-Shared	-	-	-	S4	-	-	-
contains	-	-	-	Write	-	-	-	Non-Shared	-	-	-	S4	-	-	-
has7	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	S6	-
contains2	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	S6	-
oldimage	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	S6	-
archive	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	S6	-
recentchanges	-	-	-	-	-	-	Write	Non-Shared	-	-	-	-	-	-	S7
belongs	-	-	-	-	-	-	Write	Non-Shared	-	-	-	-	-	-	S7
previous	-	-	-	-	-	-	Write	Non-Shared	-	-	-	-	-	-	S7
isold	-	-	-	-	-	-	Write	Non-Shared	-	-	-	-	-	-	S7

Figura 8.5 - Versão v01284: compartilhamento da informação (gerada pela ferramenta *Evolutio DB Designer*).

Com os subsistemas definidos, o passo seguinte foi a geração automatizada dos módulos de bases de dados, que é realizada pelo componente "Gerenciador de Módulos de Bases de Dados",

que implementa o Algoritmo 1 apresentado na Seção 7.4. Ao final, sete módulos de bases de dados foram gerados, conforme mostra a Figura 8.6.

Database Modules

Module	Element	Element Type	Interrelationship?	Intergeneralization?
M1	user	Entity Type	No	No
	_options	Entity Type	No	No
	has2	Relationship Type	No	No
M2	_rights	Entity Type	No	No
	has	Relationship Type	Yes	No
M3	ipblocks	Entity Type	No	No
	is	Relationship Type	Yes	No
	blocks	Relationship Type	Yes	No
M4	user_newtalk	Entity Type	No	No
	old	Entity Type	No	No
	cur	Entity Type	No	No
	_restrictions	Entity Type	No	No
	has4	Relationship Type	Yes	No
	makes	Relationship Type	Yes	No
	has5	Relationship Type	Yes	No
	has6	Relationship Type	No	No
	_flags	Entity Type	No	No
	contains	Relationship Type	No	No
M5	image	Entity Type	No	No
	has3	Relationship Type	Yes	No
M6	archive	Entity Type	No	No
	contains2	Relationship Type	Yes	No
	oldimage	Entity Type	No	No
	has7	Relationship Type	Yes	No
M7	recentchanges	Entity Type	No	No
	belongs	Relationship Type	Yes	No
	previous	Relationship Type	Yes	No
	isold	Relationship Type	Yes	No

Figura 8.6 - Versão v01284: Módulos da bases de dados da primeira versão do MediaWiki (gerada pela ferramenta *Evolutio DB Designer*).

Por fim, o esquema relacional foi implantado, em uma base de dados do MySQL, por meio do componente “Gerador de Esquema Relacional” da *Evolutio DB Designer*.

8.3.2 Versão v04925

A principal mudança na versão v04925 é que as permissões de usuários passaram a ser armazenadas em uma tabela própria, chamada de *user_rights*. Ou seja, foi feita a normalização da tabela *user*, apresentada na Figura 8.3.

A "Análise Evolutiva dos Requisitos de Modularização" indicou que os módulos de bases de dados atendiam os requisitos dessa versão. Portanto, nenhuma alteração no esquema de dados foi necessária.

Ocorre que, na versão inicial do esquema de dados elaborada na ferramenta *Evolutio DB Designer* e descrita na Seção 8.3.1, o gerenciamento de permissões foi alocado em um subsistema a parte (S2). Logo, as entidades pertencem a um CE próprio (*_rigths*). Já o relacionamento *has* entre *_rights* e *user* da Figura 8.4 foi mapeado como um inter-relacionamento (Figura 8.6), com a multiplicidade N:M. Portanto, não foi necessário particionar a tabela *user* em *user_rights*.

8.3.3 Versão v05648

Na versão v05648 foi adicionada a funcionalidade de grupos de usuários. A "Análise Evolutiva dos Requisitos de Modularização" indicou que os módulos atendiam parcialmente os requisitos. Pois já existe o CE *user*, mas não existe um CE para manter os dados dos grupos.

No projeto conceitual dessa versão as entidades dos grupos foram alocadas em um novo CE, *group*, e o novo subsistema S8 foi definido para gerenciar os grupos. Entretanto, a associação de um usuário a um grupo é uma funcionalidade tanto do subsistema S1 (gerenciamento de usuários) quanto de S8. A Figura 8.7 mostra a inclusão do CR *participates*, relacionando *user* e *group*.

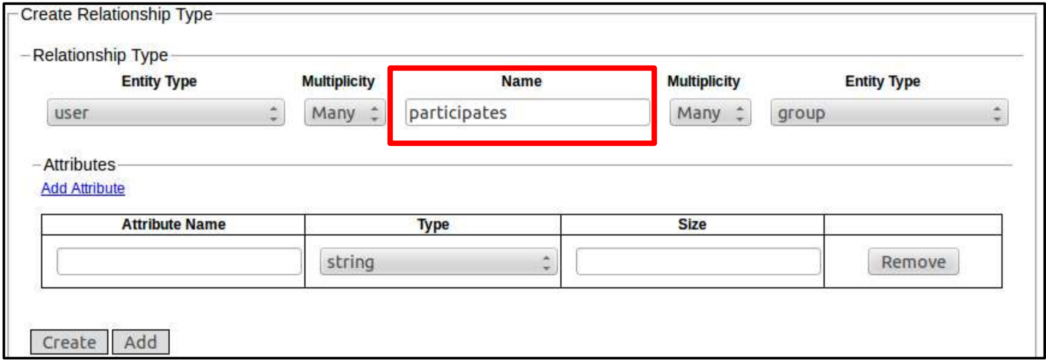
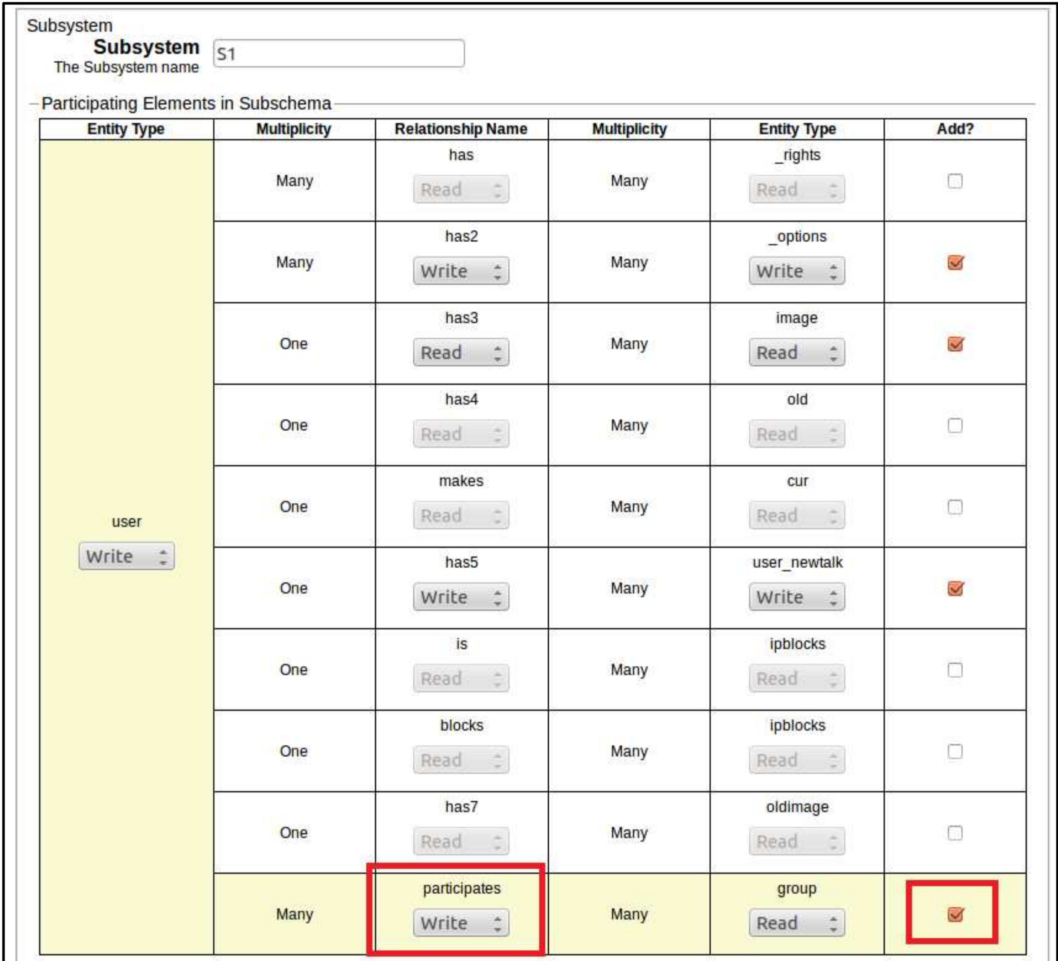


Figura 8.7 - Inclusão do CR *participates* no esquema conceitual de dados (formulário da ferramenta *Evolutio DB Designer*)

A ferramenta *Evolutio DB Designer* permite a edição de um subsistema para inclusão ou remoção de elementos ao seu subesquema, além da alteração do tipo de operação realizada aos

elementos por um subsistema. A edição do subsistema S1 é mostrada na Figura 8.8, cujos destaques representam a inclusão do CR *participates* e a operação de escrita realizada neste elemento.



Subsystem

Subsystem S1

The Subsystem name

Participating Elements in Subschema

Entity Type	Multiplicity	Relationship Name	Multiplicity	Entity Type	Add?
user Write	Many	has Read	Many	_rights Read	☐
	Many	has2 Write	Many	_options Write	☑
	One	has3 Read	Many	image Read	☑
	One	has4 Read	Many	old Read	☐
	One	makes Read	Many	cur Read	☐
	One	has5 Write	Many	user_newtalk Write	☑
	One	is Read	Many	ipblocks Read	☐
	One	blocks Read	Many	ipblocks Read	☐
	One	has7 Read	Many	oldimage Read	☐
	Many	participates Write	Many	group Read	☑

Figura 8.8 - Versão v05648: edição do subsistema S1 (formulário da ferramenta *Evolutio DB Designer*).

A Figura 8.9 mostra parte dos subesquemas de S1 e S8 e a sobreposição entre estes, que ocorre no CR *participates*.

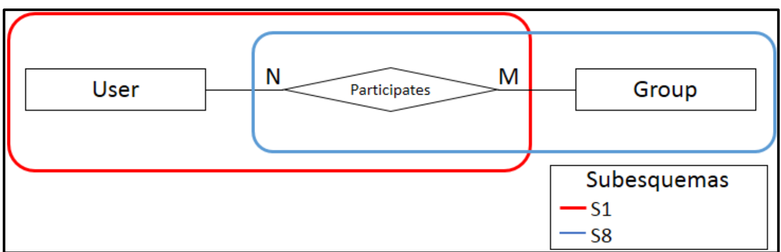


Figura 8.9 - Versão v05648: esquema conceitual parcial de S1 e S8.

A matriz de compartilhamento da informação é apresentada na Figura 8.10, em que o CR *participates* aparece com compartilhamento multidirecional.

Information Sharing Matrix																	
Element	Type of Operation								Type of Sharing	Mantained By							
	S1	S2	S3	S4	S5	S6	S7	S8		S1	S2	S3	S4	S5	S6	S7	S8
group	Read	-	-	-	-	-	-	Write	Unidirectional	-	-	-	-	-	-	-	S8
_options	Write	-	-	-	-	-	-	-	Non-Shared	S1	-	-	-	-	-	-	-
user	Write	Read	Read	Read	Read	Read	-	Read	Unidirectional	S1	-	-	-	-	-	-	-
participates	Write	-	-	-	-	-	-	Write	Multidirectional	S1	-	-	-	-	-	-	S8
has2	Write	-	-	-	-	-	-	-	Non-Shared	S1	-	-	-	-	-	-	-
_rights	-	Write	-	-	-	-	-	-	Non-Shared	-	S2	-	-	-	-	-	-
has	-	Write	-	-	-	-	-	-	Non-Shared	-	S2	-	-	-	-	-	-
ipblocks	-	-	Write	-	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-	-
is	-	-	Write	-	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-	-
blocks	-	-	Write	-	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-	-
_flags	-	-	-	Write	-	Read	-	-	Unidirectional	-	-	-	S4	-	-	-	-
has4	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
makes	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
has5	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
has6	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
contains	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
cur	-	-	-	Write	-	-	Read	-	Unidirectional	-	-	-	S4	-	-	-	-
old	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
user_newtalk	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
_restrictions	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
image	-	-	-	-	Write	-	-	-	Non-Shared	-	-	-	-	S5	-	-	-
has3	-	-	-	-	Write	-	-	-	Non-Shared	-	-	-	-	S5	-	-	-
archive	-	-	-	-	-	Write	-	-	Non-Shared	-	-	-	-	-	S6	-	-
oldimage	-	-	-	-	-	Write	-	-	Non-Shared	-	-	-	-	-	S6	-	-
has7	-	-	-	-	-	Write	-	-	Non-Shared	-	-	-	-	-	S6	-	-
contains2	-	-	-	-	-	Write	-	-	Non-Shared	-	-	-	-	-	S6	-	-
recentchanges	-	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	-	S7	-
previous	-	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	-	S7	-
isold	-	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	-	S7	-
belongs	-	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	-	S7	-

Figura 8.10 - Versão v05648: compartilhamento da informação (gerada pela ferramenta *Evolutio DB Designer*).

A inclusão dos novos elementos, o CR *participates* e o CE *groups*, e a alteração no subsistema S1, resultou nos módulos de bases de dados mostrados na Figura 8.11, em que dois novos módulos foram criados. O novo módulo M8 contém o CR *participates*, que é de compartilhamento multidirecional. Já o módulo M9 possui o CE *group*, que tem compartilhamento unidirecional e foi associado a um novo subsistema (S8). É importante destacar que o CR *participates* é um inter-relacionamento, conforme mostra destaque na Figura 8.11. Por fim, o módulo M1 mantém os mesmos elementos da versão inicial (v01284).

Database Modules

Module	Element	Element Type	Interrelationship?	Intergeneralization?
M1	user	Entity Type	No	No
	_options	Entity Type	No	No
	has2	Relationship Type	No	No
M2	_rights	Entity Type	No	No
	has	Relationship Type	Yes	No
M3	ipblocks	Entity Type	No	No
	is	Relationship Type	Yes	No
	blocks	Relationship Type	Yes	No
M4	user_newtalk	Entity Type	No	No
	old	Entity Type	No	No
	cur	Entity Type	No	No
	_restrictions	Entity Type	No	No
	has4	Relationship Type	Yes	No
	makes	Relationship Type	Yes	No
	has5	Relationship Type	Yes	No
	has6	Relationship Type	No	No
	_flags	Entity Type	No	No
	contains	Relationship Type	No	No
M5	image	Entity Type	No	No
	has3	Relationship Type	Yes	No
M6	archive	Entity Type	No	No
	contains2	Relationship Type	Yes	No
	oldimage	Entity Type	No	No
	has7	Relationship Type	Yes	No
M7	recentchanges	Entity Type	No	No
	belongs	Relationship Type	Yes	No
	previous	Relationship Type	Yes	No
	isold	Relationship Type	Yes	No
M8	participates	Relationship Type	Yes	No
M9	group	Entity Type	No	No

Figura 8.11 - Versão v05648: módulos de BD (gerada pela ferramenta *Evolutio DB Designer*).

O esquema relacional foi gerado pela *Evolutio DB Designer* pelo componente "Gerador de Esquema Relacional" de acordo com os módulos de bases de dados definidos na Figura 8.11. A Figura 8.12 mostra o esquema relacional gerado correspondente ao esquema conceitual da Figura 8.9. O CR *participates*, um inter-relacionamento, foi mapeado para uma tabela.

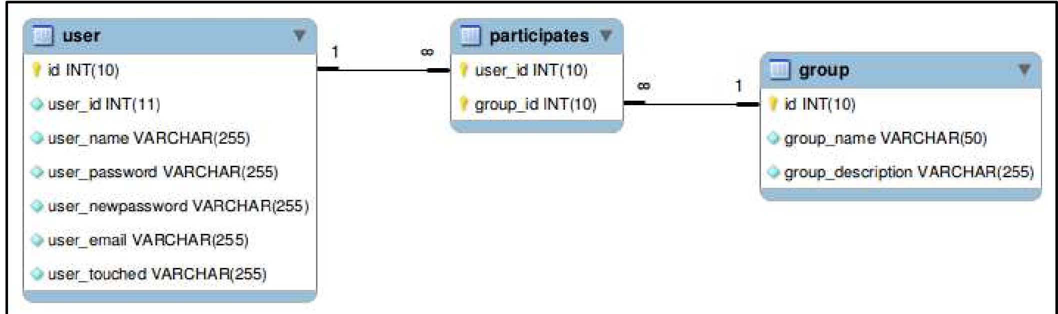


Figura 8.12 - Esquema relacional gerado com base no esquema conceitual da Figura 8.9.

8.3.4 Versão v06072

Na versão v06072 os grupos passaram a ter permissão. A saída da fase de "Análise Evolutiva dos Requisitos de Modularização" indicou que os requisitos atendiam parcialmente essa funcionalidade. Com isso, um CR com o relacionamento dos CEs *group* e *_rights* foi incluído no esquema conceitual. Na *Evolutio DB Designer* essa inclusão é apresentada na Figura 8.13 com a inclusão do CR *has8*.

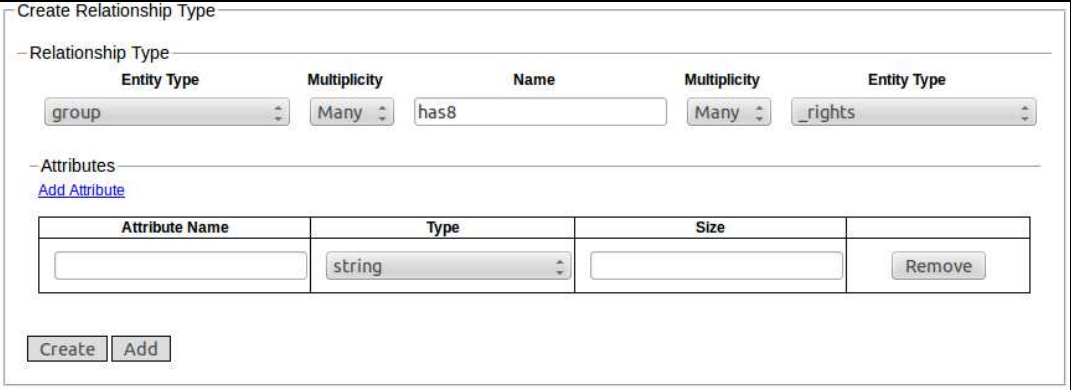
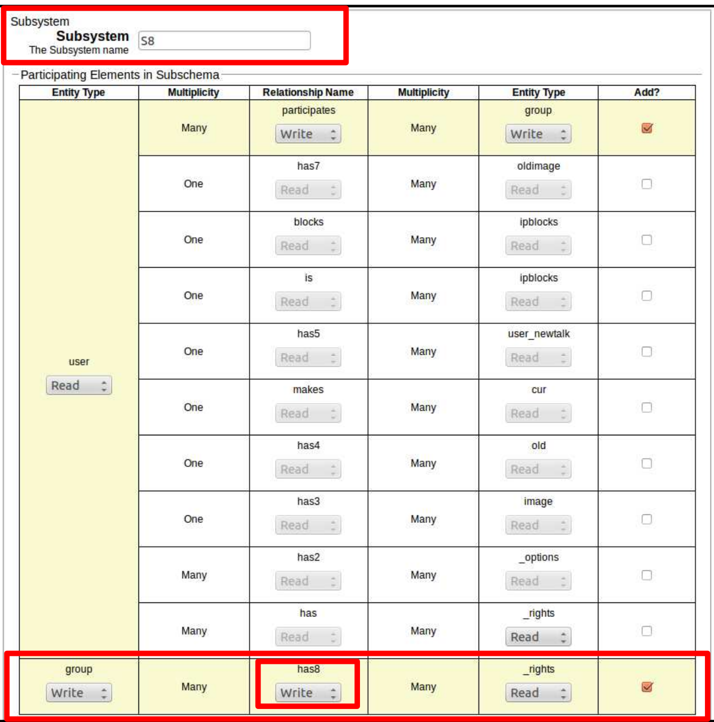


Figura 8.13 - Versão v06072: inclusão do CR *has8* (formulário da ferramenta *Evolutio DB Designer*).

O passo seguinte foi incluir o novo CR *has8* no subsistema S8 (responsável por manter os grupos), conforme mostra a Figura 8.14. Em seguida, a matriz de compartilhamento da informação gerada, apresentada na Figura 8.15, indica que o compartilhamento de *has8* é do tipo não-compartilhado.



Subsystem

Subsystem S8

The Subsystem name

Participating Elements in Subschema

Entity Type	Multiplicity	Relationship Name	Multiplicity	Entity Type	Add?
user Read	Many	participates Write	Many	group Write	☑
	One	has7 Read	Many	oldimage Read	☐
	One	blocks Read	Many	ipblocks Read	☐
	One	is Read	Many	ipblocks Read	☐
	One	has5 Read	Many	user_newtalk Read	☐
	One	makes Read	Many	cur Read	☐
	One	has4 Read	Many	old Read	☐
	One	has3 Read	Many	image Read	☐
	Many	has2 Read	Many	_options Read	☐
	Many	has Read	Many	_rights Read	☐
group Write	Many	has8 Write	Many	_rights Read	☑

Figura 8.14 - Versão v06072: edição do subsistema S8 (formulário da ferramenta *Evolutio DB Designer*).

Information Sharing Matrix

Element	Type of Operation								Type of Sharing	Mantained By							
	S1	S2	S3	S4	S5	S6	S7	S8		S1	S2	S3	S4	S5	S6	S7	S8
participates	Write	-	-	-	-	-	-	Write	Multidirectional	S1	-	-	-	-	-	-	S8
group	Read	-	-	-	-	-	-	Write	Unidirectional	-	-	-	-	-	-	-	S8
_options	Write	-	-	-	-	-	-	-	Non-Shared	S1	-	-	-	-	-	-	-
has2	Write	-	-	-	-	-	-	-	Non-Shared	S1	-	-	-	-	-	-	-
user	Write	Read	Read	Read	Read	Read	-	Read	Unidirectional	S1	-	-	-	-	-	-	-
_rights	-	Write	-	-	-	-	-	Read	Unidirectional	-	S2	-	-	-	-	-	-
has	-	Write	-	-	-	-	-	-	Non-Shared	-	S2	-	-	-	-	-	-
ipblocks	-	-	Write	-	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-	-
is	-	-	Write	-	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-	-
blocks	-	-	Write	-	-	-	-	-	Non-Shared	-	-	S3	-	-	-	-	-
has5	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
has4	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
makes	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
has6	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
contains	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
_flags	-	-	-	Write	-	Read	-	-	Unidirectional	-	-	-	S4	-	-	-	-
old	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
cur	-	-	-	Write	-	-	Read	-	Unidirectional	-	-	-	S4	-	-	-	-
user_newtalk	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
_restrictions	-	-	-	Write	-	-	-	-	Non-Shared	-	-	-	S4	-	-	-	-
image	-	-	-	-	Write	-	-	-	Non-Shared	-	-	-	-	S5	-	-	-
has3	-	-	-	-	Write	-	-	-	Non-Shared	-	-	-	-	S5	-	-	-
archive	-	-	-	-	-	Write	-	-	Non-Shared	-	-	-	-	-	S6	-	-
oldimage	-	-	-	-	-	Write	-	-	Non-Shared	-	-	-	-	-	S6	-	-
has7	-	-	-	-	-	Write	-	-	Non-Shared	-	-	-	-	-	S6	-	-
contains2	-	-	-	-	-	Write	-	-	Non-Shared	-	-	-	-	-	S6	-	-
belongs	-	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	-	S7	-
isold	-	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	-	S7	-
recentchanges	-	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	-	S7	-
previous	-	-	-	-	-	-	Write	-	Non-Shared	-	-	-	-	-	-	S7	-
has8	-	-	-	-	-	-	-	Write	Non-Shared	-	-	-	-	-	-	-	S8

Figura 8.15 - Compartilhamento da informação (gerada pela ferramenta *Evolutio DB Designer*).

Por fim, os módulos de base de dados gerados são apresentados na Figura 8.16, em que o Módulo M9 é estendido com a inclusão do CR *has8*. A extensão de M9 ocorre pelo fato do CR *has8* ter compartilhamento do tipo não-compartilhado e de ter sido associado ao subsistema S8.

Database Modules

Module	Element	Element Type	Interrelationship?	Intergeneralization?
M1	user	Entity Type	No	No
	_options	Entity Type	No	No
	has2	Relationship Type	No	No
M2	_rights	Entity Type	No	No
	has	Relationship Type	Yes	No
M3	ipblocks	Entity Type	No	No
	is	Relationship Type	Yes	No
	blocks	Relationship Type	Yes	No
M4	user_newtalk	Entity Type	No	No
	old	Entity Type	No	No
	cur	Entity Type	No	No
	_restrictions	Entity Type	No	No
	has6	Relationship Type	No	No
	_flags	Entity Type	No	No
	contains	Relationship Type	No	No
	has4	Relationship Type	Yes	No
	makes	Relationship Type	Yes	No
	has5	Relationship Type	Yes	No
M5	image	Entity Type	No	No
	has3	Relationship Type	Yes	No
M6	archive	Entity Type	No	No
	oldimage	Entity Type	No	No
	contains2	Relationship Type	Yes	No
	has7	Relationship Type	Yes	No
M7	recentchanges	Entity Type	No	No
	belongs	Relationship Type	Yes	No
	previous	Relationship Type	Yes	No
	isold	Relationship Type	Yes	No
M8	participates	Relationship Type	Yes	No
M9	group	Entity Type	No	No
	has8	Relationship Type	Yes	No

Figura 8.16 - Versão v06072: módulos de BD (gerada pela ferramenta Evolutio DB Designer).

8.3.5 Versão v06710

Na versão v06710 houve alteração em relação a manutenção dos artigos no MediaWiki, causando um impacto muito significativo no esquema de dados. Para entender as mudanças ocorridas é necessário explicar como os artigos eram armazenados e recuperados no MediaWiki nas versões anteriores e a partir da versão v06710.

Até a versão v06710 existiam duas tabelas para manter os artigos, cujas estruturas são apresentadas na Figura 8.17:

1. **cur**: armazena a versão mais recente de um artigo. A coluna *cur_text* mantém o texto de um artigo;
2. **old**: mantém o histórico dos artigos. Portanto, possui um registro para cada artigo atualizado. A coluna *old_text* mantém os textos antigos dos artigos atualizados.

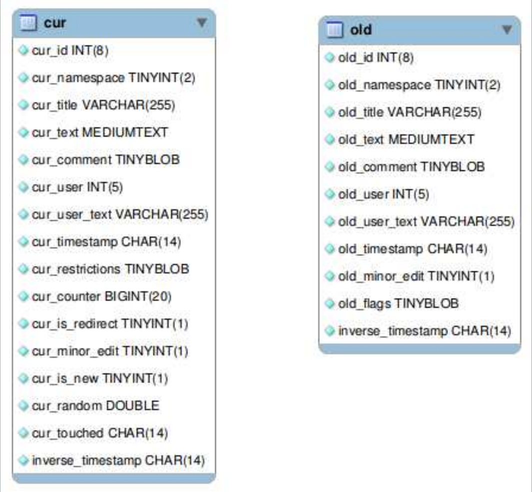


Figura 8.17 - Estrutura das tabelas *cur* e *old*.

A partir da versão v06710 as tabelas *cur* e *old* foram substituídas por três tabelas, cujo esquema físico está na Figura 8.18.

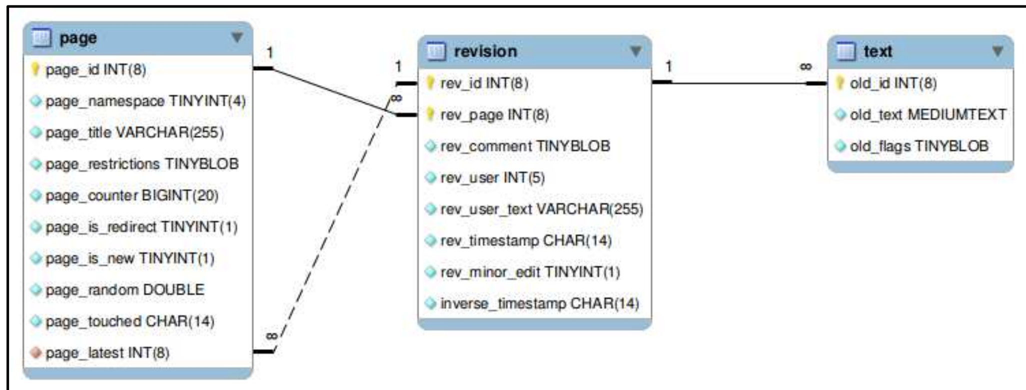


Figura 8.18 - Versão v06710: esquema físico das tabelas que mantém os dados de artigos.

O objetivo foi separar os metadados de um artigo de seu conteúdo, a fim de obter melhor desempenho em consultas. Os artigos passaram a se chamar página (*page*). A descrição do conteúdo dessas tabelas é apresentada a seguir:

1. **page:** cada página no MediaWiki, possui um único registro nessa tabela;
2. **revision:** mantém os metadados de cada alteração realizada em uma página do MediaWiki, cada modificação é chamada de revisão;
3. **text:** mantém o texto de cada revisão de uma página na coluna *old_text*.

A equivalência entre as colunas das tabelas *cur* e *old,* da versão anterior a v06710 e as tabelas *page*, *revision e text* da versão v06710 é mostrada no Quadro 8.5.

Quadro 8.5 - Correspondência entre colunas das tabelas cur, old, page, revision e text.

Tabelas do MediaWiki				
Versão < v06710		**Versão v06710**		
cur	**old**	**page**	**revision**	**text**
cur_id	old_id	page_id	-	-
cur_namespace	old_namespace	page_namespace	-	-
cur_title	old_title	page_title	-	-
cur_text	old_text	-	-	old_text
cur_user	old_user	-	rev_user	-
cur_restrictions	-	page_restrictions	-	-
cur_counter	-	page_counter	-	-
cur_is_redirect	-	page_is_redirect	-	-
cur_is_new	-	page_is_new	-	-
cur_random	-	page_random	-	-
cur_touched	-	page_touched	-	-
inverse_timestamp	-	-	inverse_timestamp	-

Para clarificar essa equivalência segue a descrição de como a recuperação de um artigo no MediaWiki é realizada a partir da versão v06710.

1. Procurar a página usando os valores contidos na coluna *page_title* da tabela *page*.
2. Caso um registro seja encontrado no passo 1, o identificador encontrado na coluna *page_latest*, que é uma chave estrangeira para a coluna *rev_id* da tabela *revision*, é utilizado para encontrar a revisão mais recente da página.
3. O valor de *rev_id* é uma chave estrangeira na tabela *text*, que contém o texto do artigo associado a revisão em questão.

Desse modo, o texto de todas as páginas passou a ser armazenado em uma tabela própria (*text*). Já o histórico de versão passa a ser obtido na tabela *revision*, em que a coluna *page_latest* da tabela *page* determina a versão mais recente de um artigo.

Para ficar em conformidade com as alterações executadas no esquema de dados nessa versão do MediaWiki, seria necessário copiar os dados das tabelas *cur* e *old* para as tabelas *page*, *revision* e *text*. Entretanto, o recurso de particionamento de CEs não foi implementado na *Evolutio DB Designer*, logo duas opções foram levantadas como possíveis soluções para a evolução:

1. Manter o texto dos artigos nos CEs *cur* e *old*. Ou seja, não particionar essas tabelas. Nesse caso a saída da "Análise Evolutiva dos Requisitos de Modularização" indicaria suporte completo dos requisitos.
2. Fazer com que o CE *old*, passe a representar *text* e adicionar os CEs *page* e *revision*. Em termos de módulos de bases de dados, a saída da fase "Análise Evolutiva dos Requisitos de Modularização" indicaria que os requisitos atendem parcialmente, mas os dados de *cur* teriam que ser copiados para *old*. Posteriormente os metadados de um artigo, contidos em *old*, devem ser copiados para *page* e *revision*.

A opção 2 foi a escolhida, mesmo com cópia de dados entre as tabelas resultantes, porque os metadados de um arquivo precisavam ficar separados por um requisito de desempenho. Também é importante ressaltar que essa é uma alteração intra-módulo em M4, responsável pelos dados das páginas.

Os CEs *page* e *revision* e os CRs *has_last*, *has_revision* e *has_text* foram criados na *Evolutio DB Designer*. O esquema conceitual é apresentado na Figura 8.19.

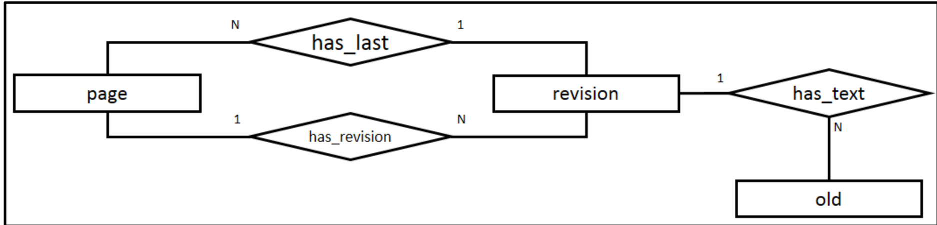


Figura 8.19 - Versão v06710: esquema conceitual de dados, para manutenção dos artigos, modificado.

O Subsistema S4 foi editado com a inclusão desses CEs e CRs, que acabou por gerar uma extensão no módulo M4, conforme mostram os destaques na Figura 8.20.

Database Modules

Module	Element	Element Type	Interrelationship?	Intergeneralization?
M4	user_newtalk	Entity Type	No	No
	old	Entity Type	No	No
	cur	Entity Type	No	No
	_restrictions	Entity Type	No	No
	has6	Relationship Type	No	No
	_flags	Entity Type	No	No
	contains	Relationship Type	No	No
	has4	Relationship Type	Yes	No
	makes	Relationship Type	Yes	No
	has5	Relationship Type	Yes	No
	page	Entity Type	No	No
	revision	Entity Type	No	No
	has_last	Relationship Type	No	No
	has_revision	Relationship Type	No	No

Figura 8.20 - Versão v06710: extensão do módulo M4 (gerada pela ferramenta *Evolutio DB Designer*).

Por fim, o esquema físico foi gerado pela *Evolutio DB Designer* pelo componente “Gerador de Esquema Relacional” e é apresentado na Figura 8.21.

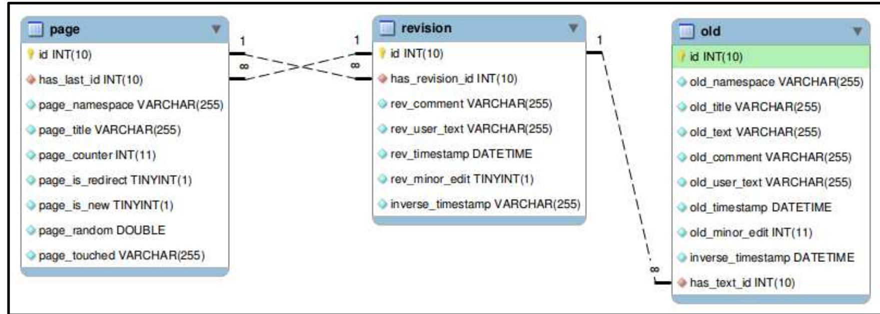


Figura 8.21 - Esquema físico de parte do módulo M4, responsável pelo armazenamento das páginas.

8.3.6 Versão v09367

A partir da versão v09367 os grupos de usuários e as permissões passaram a ser fixos e armazenados diretamente no código-fonte do MediaWiki, em um mapa ordenado de valores.

O esquema conceitual de dados da Figura 8.9, versão v05648, não precisou ser alterado, mas o CE *group* não seria mais mapeado para uma relação de um esquema relacional.

A fim de manter o uso do esquema relacional gerado pela *Evolutio DB Designer* (Figura 8.12), é necessário que os grupos introduzidos na aplicação sejam adicionados na tabela *group*, para se manter a integridade referencial que consta na tabela *participates*.

8.3.7 Versão v20301

A versão v20301 teve uma alteração simples, que é a introdução de um auto-relacionamento em *revision*. Ou seja, já havia suporte parcial para os requisitos dessa versão.

O objetivo era manter um identificador em cada revisão, referente à revisão que a antecedeu. A implementação desse auto-relacionamento foi feita através da criação de um novo CR, *has_parent*, e pela inclusão deste no subesquema do Subsistema S4. O compartilhamento desse CR é do tipo não-compartilhado, mantido apenas por S4.

O Módulo M4 foi estendido com a inclusão do CR *has_parent*, conforme mostra o destaque na Figura 8.22.

Database Modules				
Module	**Element**	**Element Type**	**Interrelationship?**	**Intergeneralization?**
M4	user_newtalk	Entity Type	No	No
	old	Entity Type	No	No
	cur	Entity Type	No	No
	_restrictions	Entity Type	No	No
	has6	Relationship Type	No	No
	_flags	Entity Type	No	No
	contains	Relationship Type	No	No
	has4	Relationship Type	Yes	No
	makes	Relationship Type	Yes	No
	has5	Relationship Type	Yes	No
	page	Entity Type	No	No
	revision	Entity Type	No	No
	has_last	Relationship Type	No	No
	has_revision	Relationship Type	No	No
	has_parent	Relationship Type	No	No

Figura 8.22 - Versão v20301: extensão do módulo M4 (gerada pela ferramenta *Evolutio DB Designer*).

Por fim, o esquema físico foi atualizado pelo "Gerador de Esquema Relacional", conforme mostra a Figura 8.23 e que corresponde à extensão de M4 pela inclusão de *has_parent*.

O preenchimento dos valores da coluna *has_parent*, referente às páginas já existentes, tiveram que ser feitos manualmente através de instruções SQL, com base nos dados já existentes na própria tabela *revision*.

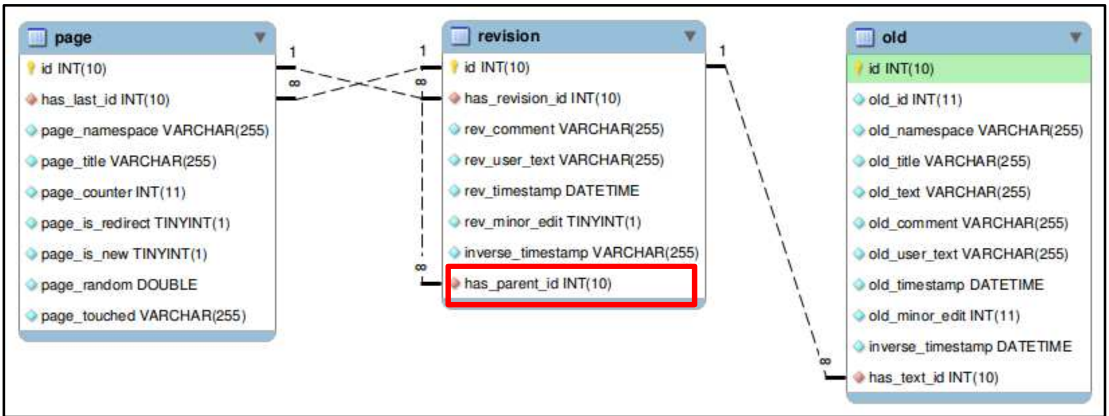


Figura 8.23 - Versão v20301: esquema físico atualizado das tabelas que mantém os artigos.

8.3.8 Versão v20468

As alterações na versão v20468 foram simples e realizadas por meio da inclusão de atributos com sufixo *_deleted*, com o objetivo de se fazer uma remoção lógica (*soft deleted*) de conteúdo. Essa funcionalidade foi introduzida, para lidar com os casos em que é necessário ocultar um conteúdo rapidamente do público em geral. Essas situações ocorrem, por exemplo, devido a questões de licenças dos conteúdos disponibilizados tais como imagens; divulgação de dados pessoais de terceiros, etc.

O esquema conceitual foi alterado pela inclusão do atributo *_deleted* nos CEs: *archive*, *ipblocks* e *recentchanges*. Como apenas atributos foram incluídos em CEs, não há alteração nos módulos de bases de dados gerados. Portanto, o componente "Gerador de Esquema Relacional" atualiza o esquema relacional já incluindo o atributo *_deleted* nas tabelas correspondentes.

8.4 Considerações finais

Este capítulo apresentou o uso da ferramenta *Evolutio DB Designer* na evolução do esquema de dados do MediaWiki, software utilizado pela Wikipedia, uma enciclopédia livre mantida na Internet, cujo conteúdo é escrito de modo colaborativo.

O cenário é diferente do apresentado no capítulo 6, pois se trata de versões de um esquema de dados e não de um projeto iterativo e incremental, que consiste de incrementos ao esquema de dados.

Ao todo 171 versões de esquemas de bases de dados foram analisadas, até se chegar nas 8 versões utilizadas no estudo de caso. Essas versões foram as que introduziram as alterações mais significativas no esquema de dados.

Primeiramente, o efeito adverso das desnormalizações inseridas no esquema de dados do MediaWiki foi o aumento no acoplamento entre os elementos que compõem o esquema de dados. Entretanto, a elaboração do esquema conceitual de dados na *Evolutio DB Designer* é realizada com a definição dos subsistemas. Com isso, a base de dados é projetada de maneira modularizada, levando à maior independência entre os elementos e gerando esquemas com maior capacidade de abstração.

As desnormalizações foram revistas ao longo da evolução do esquema de dados, como, por exemplo, a versão v04925, em que os elementos relacionados às permissões de usuários foram normalizados. Contudo, não foi necessário modificar o esquema de dados gerado pela *Evolutio DB Designer*, pois um módulo de bases de dados foi gerado para atender às permissões de usuários.

A saída da fase de "Análise Evolutiva dos Requisitos de Modularização" ficou evidente em cada iteração, resultando na geração de novos módulos de bases de dados ou na extensão dos já implementados. As saídas foram determinadas pela definição de novos subsistemas e pela associação de elementos aos subsistemas já existentes. Essas situações ocorreram na versão v05648, com a geração de dois novos módulos de bases de dados, M8 e M9. Já na versão subsequente (v06072) o Módulo M9 exemplifica a extensão de um módulo de bases de dados.

A versão v06710 mereceu uma atenção particular, pois a alteração no esquema de dados foi muito significativa. Essa alteração partiu de um requisito não-funcional de desempenho, com o objetivo de separar o conteúdo de um artigo de seus metadados. O resultado foi uma alteração na arquitetura do MediaWiki. A alteração impactava apenas o esquema de dados do módulo M4, considerada, portanto, uma alteração intra-módulo. Contudo, tanto o processo evolutivo de modularização de bases de dados quanto a *Evolutio DB Designer* são abordagens para a evolução inter-módulos. Com isso, houve a necessidade de cópia de dados no novo esquema gerado pela *Evolutio DB Designer*. Esse tipo de situação poderia ter sido evitada se o projeto conceitual inicial tivesse contemplado os requisitos não-funcionais. De qualquer maneira, em trabalhos futuros é interessante que o processo evolutivo de modularização de bases de dados e a *Evolutio DB Designer* abordem, com mais profundidade, a refatoração intra-módulo.

Na versão v09367 o esquema conceitual não é alterado, mas parte da arquitetura do MediaWiki é novamente modificada, pois os grupos deixaram de ser armazenados na base de dados e passaram para o código da aplicação. Essa decisão é questionável, pois ao mesmo tempo que elimina a necessidade de recuperação dos grupos da base de dados, remove a integridade referencial entre um usuário e seus respectivos grupos.

A adição de atributos, que ocorre na versão v20468, mostrou que o componente "Gerador de Esquema Relacional" da *Evolutio DB Designer* foi capaz de evoluir o esquema relacional, incluindo os atributos em suas tabelas correspondentes. Esse é um exemplo de refatoração intra-módulo que é tratado pela *Evolutio DB Designer*.

A conclusão é que a *Evolutio DB Designer* diminuiu a necessidade de refatoração do esquema de dados do MediaWiki pelos seguintes motivos:

1. A definição dos subsistemas faz com que o projetista analise o acoplamento entre as funcionalidades da aplicação e o esquema de dados;
2. O uso do modelo entidade-relacionamento reflete com mais semântica as alterações realizadas no esquema de dados, simplificando a evolução;

3. A geração automatizada dos módulos de bases de dados torna possível o uso do processo evolutivo de modularização de bases de dados em projetos que tenham esquemas de bases de dados com um número considerável de elementos;
4. O “Gerador de Esquema Relacional” foi capaz de evoluir o esquema relacional já operacional de acordo com as definições dos módulos de bases de dados gerados em cada versão.

Por fim, o próximo capítulo apresenta a conclusão deste trabalho descrevendo as contribuições e possíveis trabalhos futuros.

9 Conclusão

Uma evolução de um sistema de software ocorre quando surgem novos requisitos que implementam novas necessidades de negócio. Esse tipo de situação é ainda mais frequente em projetos desenvolvidos com métodos ágeis, em que o produto final é elaborado de maneira incremental e os requisitos são constantemente alterados. A evolução afeta não só o código do sistema de software, mas também a base de dados, cujo esquema de dados deve ser alterado para se adaptar aos novos requisitos.

A evolução de uma base de dados é mais complexa, devido à semântica dos dados. Quando os novos requisitos não têm como objetivo alterar a semântica da aplicação, a informação, após uma evolução, deve ser mantida. Ademais, alterações em um esquema de dados já implantado são onerosas para a equipe de desenvolvimento, devido ao risco de perda de dados e à introdução de falhas no sistema de software, comprometendo a consistência dos dados.

Nesse ambiente de mudanças constantes, em que os requisitos são instáveis, o ideal é que o projeto da base de dados produza um esquema de dados flexível para tolerar futuras alterações.

Nessa direção, a principal contribuição deste trabalho de mestrado foi a definição do processo evolutivo de modularização de bases de dados. Tal processo é uma adaptação e extensão da modularização de bases de dados proposta por Busichia (1999). O processo de modularização de bases de dados de Busichia (1999) passou, então, a ser iterativo e incremental, produzindo esquemas de dados coesos e com baixo acoplamento, por meio da geração de módulos autônomos de informação, facilitando evolução futura.

Este trabalho de mestrado incluiu uma nova fase, chamada de “Análise Evolutiva dos Requisitos de Modularização”, e adaptou a fase de “Projeto de Modularização” ao processo de modularização de bases de dados proposto por Busichia (1999). A nova fase verifica a completude dos novos requisitos em relação aos módulos de bases de dados implementados, indicando se os módulos existentes atendem os requisitos por completo, parcialmente ou se não atendem. Portanto, essa nova fase determina se os módulos de bases de dados são mantidos sem alterações, estendidos ou se novos módulos devem ser criados.

A fase adaptada, “Projeto de modularização”, passou a se chamar “Projeto Evolutivo de Modularização”. A etapa de “Definição evolutiva dos módulos da base de dados” considera a definição dos subsistemas da iteração em vigor, que determinam os módulos de bases de dados, considerando mapeamentos diferenciados para os conceitos de inter-relacionamento e inter-generalização, esse último introduzido por este trabalho. Os mapeamentos de inter-relacionamento e inter-generalização foram adaptados para diminuir o acoplamento entre os módulos de bases de dados e, consequentemente, tornando o esquema de dados mais flexível e tolerante à mudanças.

Este trabalho propôs a integração incremental dos módulos de bases de dados ao estender o esquema de dados do catálogo do objeto integrador. Esse catálogo passou a armazenar o esquema conceitual de dados da aplicação, a definição dos subesquemas dos subsistemas e dos módulos de bases de dados. O catálogo passa a dar suporte ao processo evolutivo de modularização de bases de dados, sendo fonte de consulta a cada nova iteração.

O Capítulo 6 mostrou a aplicação do processo evolutivo de modularização de bases de dados, contemplando a descrição de cada uma de suas fases e etapas, em um projeto de desenvolvimento de software iterativo e incremental especificado por Ambler (2004). O objetivo proposto de diminuição de refatoração do esquema de dados foi atingido, uma vez que os módulos de bases de dados gerados e os mapeamentos dos inter-relacionamentos e inter-generalizações não demandaram refatorações quando da chegada de novos requisitos.

Em seguida, com o objetivo de apoiar projetos de bases de dados que possuem muitos elementos, uma ferramenta que implementa o processo evolutivo de modularização de bases de dados foi desenvolvida, a *Evolutio DB Designer*.

O Capítulo 8 mostrou o uso da *Evolutio DB Designer* em um estudo de caso baseado no estudo de Curino et al. (2008) sobre as alterações ocorridas na base de dados da Wikipedia durante quatro anos e meio. Ao todo cento e setenta e uma versões de esquemas de dados foram analisadas para se detectar aquelas que introduziam as mudanças mais significativas. Ao final, oito versões foram selecionadas. O estudo de caso mostrou que houve diminuição na refatoração da base de dados, pelos seguintes motivos: 1) a definição dos módulos de bases de dados possibilitou deter-

minar o acoplamento entre as funcionalidades e os elementos do esquema de dados de cada módulo; 2) o fato do processo de modularização estar baseado em um modelo conceitual rico em semântica, o modelo entidade-relacionamento estendido, possibilitou refletir com mais precisão os requisitos e, consequentemente, as alterações na base de dados; 3) a geração automatizada dos módulos de bases de dados permitiu a aplicação do processo evolutivo de modularização em bases de dados que tenham um número grande de elementos, além de que a evolução do esquema da base de dados foi realizada de modo automático.

As sugestões para trabalhos futuros foram divididas em duas partes. Primeiramente são apresentadas sugestões relacionadas ao processo evolutivo de modularização de bases de dados. Em seguida, as sugestões para a *Evolutio DB Designer*.

Sugestões de trabalhos futuros para o processo evolutivo de modularização de bases de dados:

1. O estudo de caso do Capítulo 8 mostrou que o processo evolutivo de modularização de bases de dados atende a extensão intra-módulo apenas para a inclusão de novos conjuntos de entidades, conjuntos de relacionamentos e a inclusão de atributos. Quanto a evolução no esquema físico, é desejável, em algumas situações, que dados sejam copiados e tabelas sejam particionadas. Trabalhos futuros devem abordar a evolução intra-módulo para esses casos;
2. Uma linguagem de definição de módulos de bases de dados pode ser proposta, estendendo o catálogo do SGBD para a modularização evolutiva de bases de dados;
3. Em um ambiente coorporativo uma base de dados geralmente é utilizada por mais de uma aplicação. Portanto, cada aplicação terá que ser alterada para se ajustar as modificações no esquema de dados, entretanto, a velocidade com que cada uma é modificada não é igual. Com isso, Ambler (2004) indica que haverá um período de transição em que esquema novo e o antigo irão coexistir, devendo os dados entre esses serem sincronizados. Pode ser uma tarefa síncrona (Ambler, 2004) ou assíncrona conforme proposta de Domingues, Kon e Ferreira (2009). Para lidar com essa situação o processo evolutivo de modularização de bases de dados deve implementar a camada do objeto integrador, encapsulando o acesso à base de dados

por meio dos procedimentos públicos e privados de cada módulo. Desse modo, as aplicações só teriam acesso ao esquema de dados, pelos procedimentos dos módulos, tornando as alterações nos esquemas conceituais de cada módulo transparentes para as aplicações.

Sugestões de trabalhos futuros para a ferramenta *Evolutio DB Designer*:

1. Implementar suporte à abstração de agregação;
2. Implementar recursos gráficos para a definição do esquema, dos subsistemas e dos módulos de bases de dados por meio do diagrama entidade-relacionamento estendido;
3. Implementar o catálogo do objeto integrador em sua totalidade, contemplando as funcionalidades associadas aos subsistemas;
4. Desenvolvimento da camada do objeto integrador.

Acredita-se que as pesquisas futuras contribuam ainda mais para a consolidação do processo evolutivo de modularização de bases de dados proposto por este trabalho e, principalmente, para o seu uso automatizado no desenvolvimento iterativo e incremental de sistemas de software que utilizam bases de dados.

10 Referências bibliográficas

AMBLER, S. W. **Agile Database Techniques**. Indianopolis: Wiley, 2003.

AMBLER, S. W. **Agile/Evolutionary Data Modeling**: From Domain Modeling to Physical Modeling, 2004. Disponivel em: <http://www.agiledata.org/essays/agileDataModeling.html>. Acesso em: 20 Setembro 2013.

AMBLER, S. W.; SADALAGE, P. J. **Refactoring Databases:** Evolutionary Database Design. Upper Saddle River: Pearson Addison-Wesley, 2006.

ATWOOD, T. M.; et al. An Object-Oriented DBMS for Design Support Applications. In: Proceedings of the IEEE COMPINT 85, 1985, Montreal. **Anais**…Montreal: [s.n], 1985. p. 299-307.

BATINI, C.; CERI, S.; NAVATHE, S. B. **Conceptual Database Design:** An Entity-Relationship Approach. California: Pearson Addison Wesley, 1991.

BECK, K. **Extreme Programming Explained:** Embrace Change. 2ª ed. Boston: Pearson Addison-Wesley, 1999a.

BECK, K. Embrancing Change with Extreme Programming. **IEEE Computer**, [S.l.], v. 32, 10, p. 70-78, 1999b.

BECK, K. et al. **Manifesto for Agile Software Development**, Wasatch, 11 Fevereiro 2001. Disponível em: <http://agilemanifesto.org>. Acesso em: 27 de Abril 2013.

BUSICHIA, G. **Modularização de bases de dados para a construção de sistemas de informação flexíveis**. 1999. 75f. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, 1999.

BUSICHIA, G. ; FERREIRA, J. E. . Compartilhamento de módulos heterogêneos de dados através de objetos integradores.. In: Simpósio Brasileiro de Banco de Dados, 13., 1999, Florianópolis. **Anais...**Florianópolis: [s.n], 1999a, p. 320-329.

BUSICHIA, G.; FERREIRA, J. E. Sharing of heterogeneous databases modules by integration objects. In: Third East-European Conference on Advances in Databases and Information Systems, 7., 1999, Maribor-Slovenia: **Proceedings**… Maribor-Slovenia: [s.n], 1999b. p. 1-8.

CHEN, P. P. The entity-relationship model: towards a unified view of data. In: ACM Transactions on Database Systems, 1976, Framingham. **Proceedings**… Framingham: [s.n], 1976. p. 9-36.

CLEVE, A.; BROGNEAUX, A.-F.; HAINAUT, J.-L. A Conceptual Approach to Database Applications Evolution. In: 29th international conference on Conceptual modeling, 29., 2010, Vancouver. **Proceedings**… Vancouver: [s.n], 2010. p. 132-145.

CODD, E. F. Relational Model of Data for Large Shared Data Banks. **Information Retrieval**, New York, v. 13, n. 10, p. 377-387, 1970.

CURINO, C.A. **Benchmark Downloadables**, 22 de maio de 2008. Disponível em <http://yellowstone.cs.ucla.edu/schema-evolution/index.php/Benchmark_Downloadabl-es>. Acesso em: 20 de novembro de 2013.

CURINO, C. A.; et al. Schema Evolution in Wikipedia: toward a Web Information System Benchmark. In: Tenth International Conference on Enterprise Information Systems, 10., 2008, Barcelona. **Proceedings**… Barcelona: [s.n], 2008. p. 323-332.

CURINO, C. A.; et al. The PRISM Workwench: Database Schema Evolution Without Tears. In: IEEE 25th International Conference on Data Engineering, 25., 2009, Shanghai. **Anais**… Shanghai: [s.n], 2009. p. 1523-1526.

CURINO, C. A.; MOON, H. J.; ZANIOLO, C. Graceful Database Schema Evolution: the PRISM Workbench. In: Proceedings of the VLDB, 1., 2008, San Jose. **Anais**…San Jose: [s.n], 2008. p. 761-772.

DB SOLO LLC. **DB Solo - The SQL Tool**, 2013. Disponivel em: <http://www.dbsolo.com>. Acesso em: 20 Outubro 2013.

DOMINGUES, H.; KON, F; FERREIRA, J. E. Replicação assíncrona em bancos de dados evolutivos. In: Simpósio Brasileiro de Banco de Dados, 24., 2009, Fortaleza. **Anais...**Fortaleza: Simpósio Brasileiro de Banco de Dados, 2009. p. 121-135.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 4ª. ed. Boston: Pearson Addison Wesley, 2003.

FERREIRA, J. E.; BUSICHIA, G. Database modularization design for the construction of flexible information systems. In: International Symposium on Database Engineering and Applications, 1999, Montreal. **Proceedings**… Montreal: [s.n], 1999. p. 415-422.

HAINAUT, J. L. A Generic Entity-Relationship Model. In: IFIP WG 8.1 Conference on Information System Concepts: an in-depth analysis, 1989, Holanda. **Proceedings**…Holanda: [s.n.], 1989. p. 109-138.

HAINAUT, J. L. The Transformational Approach to Database Engineering. In: Lämmel, R; Saraiva, J.; Visser, Joost. **Generative and Transformational Techniques in Software Engineering**. Braga: Springer Berlin Heidelberg, 2005. v. 4143, n. 4, p. 95-143.

HEUSER, C. A. **Projeto de Banco de Dados**. 6ª. ed. Porto Alegre: Bookman, 2009.

BECK, K. et al. Manifesto Ágil para o Desenvolvimento Ágil de Software. **Manifesto Ágil**, 2001. Disponivel em: <http://manifestoagil.com.br>. Acesso em: 20 Setembro 2013.

MESQUITA, E. J. S.; FINGER, M. Projeto de dados em bancos de dados distribuídos. In: XIII Simpósio Brasileiro de Banco de Dados, 13., 1998, Maringá. **Anais**... Maringá: [s.n], 1998. p. 87-101.

OPEN SOURCE INITIATIVE. **The BSD 2-Clause License**. Disponivel em: <http://opensource.org/licenses/bsd-license.php>, 1998. Acesso em: 15 Setembro 2013.

PAPASTEFANATOS, G. et al. Hecataeus: A What-If Analysis Tool for Database Schema Evolution. In: Software Maintenance and Reengineering, 12., 2008, Atenas. **Proceedings**… Atenas: [s.n], 2008. p. 326-328.

PAPASTEFANATOS, G. et al. HECATAEUS: Regulating Schema Evolution. In: IEEE International Conference on Data Engineering, 26., 2010, Long Beach. **Proceedings**…Long Beach: [s.n], 2010. p. 1181-1184.

PATUCI, G. **Test Backlog: nova abordagem incremental de planejamento e execução de testes para Scrum**. 2013. 104f. Dissertação (Mestrado em Tecnologia) - Universidade Estadual de Campinas, Faculdade de Tecnologia, Limeira, 2013.

RAHM, E.; BERNSTEIN, P. A. An Online Bibliography on Schema Evolution. **SIGMOD Record**, New York, v. 35, 4., p. 30-31, 2006.

RODDICK, J. F.; CRASKE, N. G.; RICHARDS, T. J. A taxonomy for schema versioning based on the relational and entity relationship models. In: International Conference on the Entity-Relationship Approach: Entity-Relationship Approach, 12., 1994, London. **Anais**… London: Springer-Verlag, 1994. p. 137-148.

SCHWABER, K. **Agile Project Management with Scrum**. Seattle: Microsoft Press, 2004.

SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. Englewood Cliffs: Prentice Hall, 2001.

SENSIO LABS. **Twig**: The flexible, fast, and secure template engine for PHP, 2013. Disponivel em: <http://twig.sensiolabs.org/>. Acesso em: 07 set. 2013.

SHAMKANT, B. N.; JAMES, P. F. Restructuring for large databases: three levels of abstraction. **ACM Transactions on Database Systems**, New York, v. 1, n. 2, p. 138-158, 1976.

SOCKUT, G. H.; GOLDBERG, R. P. Database Reorganization - Principles and Practice. **ACM Computing Surveys**, New York, v. 11, n. 4, p. 371-395, 1979.

SOMMERVILLE, I. **Engenharia de Software**. 8ª. ed. São Paulo: Pearson Addison Wesley, 2008.

SOMMERVILLE, I. **Engenharia De Software**. 9ª. ed. São Paulo: Pearson Addison Wesley, 2011.

STONEBRAKER, M.; MOORE, D. **Object-Relational DBMSs**: The Next Great Wave. 2ª ed. San Francisco: Morgan Kaufmann Pub, 1996.

THE PHP GROUP. PHP: What is PHP?, 2013. Disponivel em: <http://www.php.net/manual/en/intro-whatis.php>. Acesso em: 07 de setembro de 2013.

WAGE, J. H. et al. **26. Tools - Doctrine 2 ORM Documentation**, 2013a. Disponivel em: < http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/>. Acesso em: 15 de Setembro 2013.

WAGE, J. H. et al. **Welcome to the Doctrine Project**, 2013b. Disponivel em: <http://www.doctrine-project.org>. Acesso em: 15 de Setembro 2013.

WERNER, T.; WANSTRATH, C.; Hyett, P.; GitHub. Disponível em < https://github.com/>. Acesso em: 01/12/2013. 2008.

WIKIMEDIA FOUNDATION. **MediaWiki**, 2002. Disponivel em: <http://www.mediawiki.org>. Acesso em: 20 Outubro 2013.

WIKIMEDIA FOUNDATION. **Wikipedia**, 2001. Disponivel em: <http://www.wikipedia.org>. Acesso em: 2013 Outubro 2013.

Apêndice A - PHP e Twig Template Engine

O PHP (acrônimo recursivo de *PHP: Hypertext Preprocessor*) é uma linguagem de *script*, que pode ser integrada a um servidor de páginas *web*, portanto é majoritariamente utilizada para o desenvolvimento de sistemas de software para a Internet (THE PHP GROUP, 2013). O código é embutido diretamente no *HyperText Markup Language* (HTML), permitindo a alternância entre blocos de código PHP e de HTML. Esses dois fatores contribuem para uma rápida aprendizagem do uso da linguagem, pois não há compilação e nem comandos específicos para formatação da saída. Logo, apenas os recursos do HTML são empregados na geração dinâmica de conteúdo.

O fato do PHP ser embutido impede a separação da camada de apresentação, realizada por marcações do HTML, e da lógica do sistema de software desenvolvido. Para contornar essa situação, a arquitetura Modelo-Visão-Controlador (MVC) é adotada. A MVC é composto de três componentes lógicos que interagem entre si (Figura A.1), em que a apresentação e a interação com os dados ficam separados (SOMMERVILLE, 2011). O Modelo gerencia as operações sobre os dados. A visão é responsável pelo modo como os dados são apresentados. O controlador gerencia a interação com o usuário, repassando as entradas e saídas para o Modelo e a Visão, respectivamente. A arquitetura MVC pode ser implementada com a utilização de um *Template Engine*, responsável pelo Controlador e Visão, enquanto uma API para mapeamento objeto-relacional é utilizada para o Modelo.

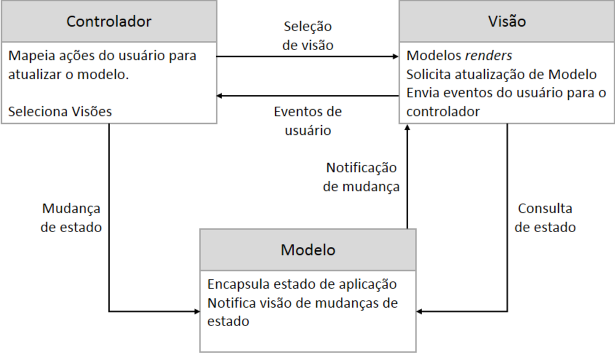


Figura A.1 – Arquitetura da MVC (SOMMERVILLE, 2011)

Um *Templage Engine* possibilita a separação da apresentação, camada Visão, e da lógica da aplicação, camada Controlador. Conforme a Figura A.2 um modelo (*template*) de uma página *web* é elaborado contendo marcações especiais que posteriormente são substituídas por dados dinamicamente gerados por meio de código ou recuperados de uma base de dados. Essa abordagem traz as seguinte vantagens: facilidade de manutenção, pois as mudanças na lógica e na apresentação são feitas separadamente; reuso, pois um mesmo modelo pode ser utilizado com diferentes dados.

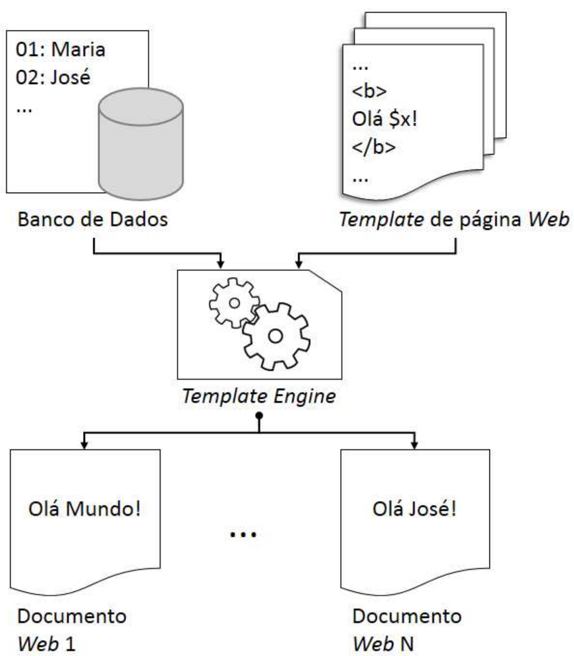


Figura A.2 - Arquitetura de um *Template Engine*

O Twig é um *Template Engine* desenvolvido pelo Sensio Labs (2013) e distribuído pela licença BSD (OPEN SOURCE INITIATIVE, 1998). O uso do Twig é feito por meio de dois arquivos: um que contém a lógica definida em código PHP e outro que contém o *layout* da página *web*, criado em HTML e contendo marcações específicas do Twig.

A Figura A.3 mostra um exemplo do Twig. O arquivo hello.twig possui um modelo de uma página *web*. Já o arquivo hello.php instancia um objeto do tipo Twig_Environment e executa o método *render*, que tem como parâmetros o nome do arquivo *template* (*hello.twig*) e um mapa ordenado de valores, que contém um conjunto de pares chave/valor, acessíveis do arquivo *template* do Twig.

Na Figura A.3 a marcação do Twig “{{nome}}” na linha 4 do arquivo hello.twig é substituída pelo valor “João”, definido no mapa ordenado de valores $vars do arquivo hello.php e passado como argumento do método render. Por fim, o HTML resultante é apresentado no navegador.

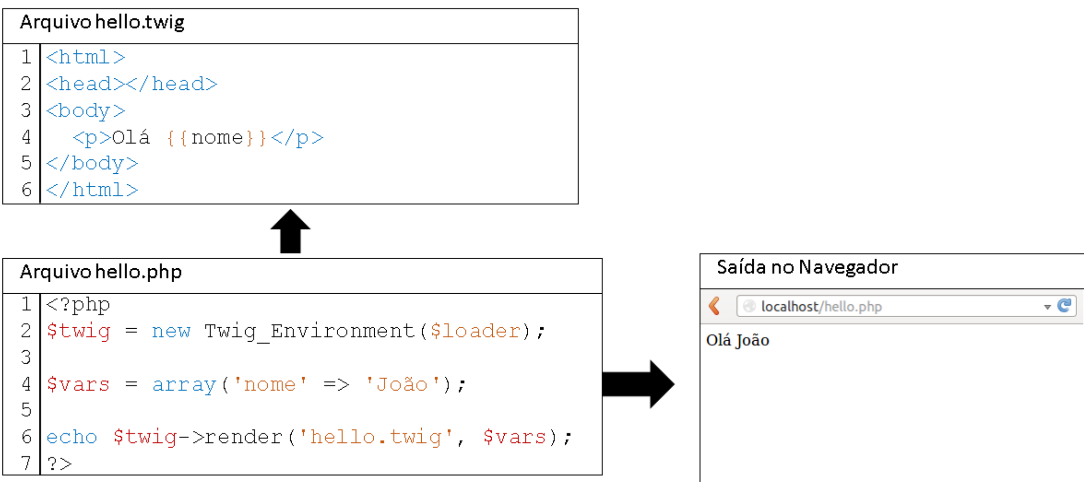


Figura A.3 - Exemplo de uso do Twig

Apêndice B - Mapeamento objeto-relacional com Doctrine

O mapeamento objeto-relacional (*Object-Relational Mapping, ORM)* tem como objetivo eliminar a incompatibilidade de paradigmas que existe entre a camada de persistência, uma base de dados relacional, e a codificação de um sistema de software, elaborado no paradigma da orientação a objetos. Desse modo, os registros da base de dados são convertidos, sob demanda, para objetos no código da aplicação.

Apesar da crítica ao uso de mapeamento objeto-relacional apresentada em por Cleve, Brogneaux e Hainaut (2010) e descrita na Seção 2.3, o ORM contribui para a portabilidade do sistema, pois é capaz de gerar automaticamente código SQL para diferentes tipos de SGBDs relacionais, sem necessitar de alterações no mapeamento. Ou seja fornece uma abstração de SGBD relacional. O Doctrine é uma API para mapeamento objeto-relacional.

No Doctrine o mapeamento do esquema da base de dados pode ser feito separadamente em arquivos nos formatos xml, yaml ou por meio de anotações inseridas diretamente no código PHP de uma classe. O formato de anotações será detalhado a seguir.

Cada classe representa uma entidade e as anotações são utilizadas para estabelecer o mapeamento objeto-relacional. As anotações são feitas com marcadores, delimitados pelo par /** e */, e possuem atributos, que agem como argumentos complementando as definições do mapeamento. O Quadro B.1 mostra as principais anotações existentes no Doctrine.

Quadro B.1 - Lista de anotações mais comuns (WAGE et. al, 2013b).
(continua)

Anotação	Descrição	Atributos
@Entity	Define a classe como sendo uma entidade.	-
@Table	Possibilita definir os parâmetros de criação de uma tabela.	- name: define o nome da tabela no esquema físico.
@Id	Define um atributo da classe como sendo o identificador, que é mapeado para uma chave primária.	-

Quadro B.1 - Lista de anotações mais comuns (WAGE et. al, 2013b).
(conclusão)

Anotação	Descrição	Atributos
@GeneratedValue	Define a estratégia utilizada para gerar o valor para o identificador da entidade.	- strategy: indica qual estratégia será utilizada. Valores possíveis: *AUTO*, *SEQUENCE*, *TABLE*, *IDENTITY*, *UUID*, *CUSTOM* ou *NONE*.
@Column	Define o atributo da classe como persistente, mapeado para uma coluna.	- type: indica o tipo Doctrine a ser utilizado, convertido para um tipo correspondente do SGBD configurado para uma conexão. - name: define o nome da coluna no esquema físico. - nullable: determina se valores nulos são admitidos. - unique: determina se o valor deve ser único dentre todos os registros da tabela.
@OneToMany	Define um relacionamento um-para-muitos.	- targetEntity: nome da classe, definida como entidade, a ser referenciada. - mappedBy: define o atributo da classe referenciada no qual o relacionamento está definido.
@ManyToOne	Define um relacionamento muitos-para-um.	- targetEntity: nome da classe, mapeada como entidade, a ser referenciada.
@OneToOne	Define um relacionamento um-para-um	
@ManyToMany	Define um relacionamento muitos-para-muitos	

O Doctrine possui tipos próprio para mapeamento, convertidos automaticamente para os tipos correspondentes de acordo com o SGBD configurado para a conexão. Os principais tipos e seus correspondentes no padrão SQL ANSI estão descritos no Quadro B.2.

Quadro B.2 - Tipos do Doctrine.

Tipo Doctrine	Tipo SQL ANSI	Descrição
string	varchar	Representa um *string*.
integer	integer (inteiro)	Número Inteiro.
smallint	smallint	Inteiro com menor capacidade de representação.
bigint	bigint	Inteiro com maior capacidade de representação.
boolean	boolean ou inteiro	Booleano ou inteiro em que 0 (zero) é validado como falso e 1 (um) como verdadeiro.
date	datetime	Representa uma data.
time	time	Representa um horário.
datetime	datetime	Representa uma data com um horário.
float	float	Número de ponto flutuante.

Nos próximos parágrafos um exemplo de mapeamento no Doctrine é apresentado, partindo-se do esquema de dados conceitual da Figura B.4.

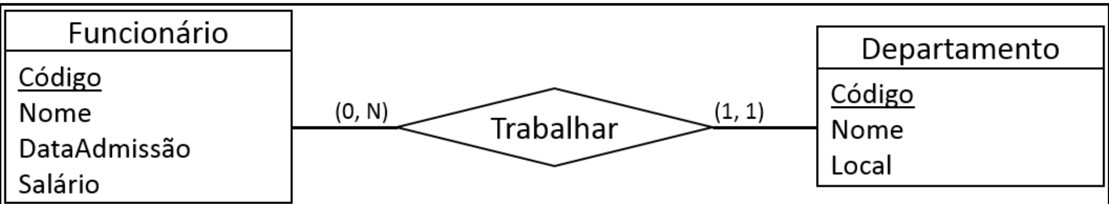


Figura B.4 - Exemplo de Diagrama Entidade-Relacionamento.

O mapeamento do CE *Funcionário* é mostrado na Figura B.5. Cada atributo do CE é mapeado como uma propriedade da classe, com exceção de $departamento, que é uma referência ao departamento de um determinado funcionário. A seguir uma descrição do mapeamento.

- Linhas 7 a 12: o atributo Código é mapeado como inteiro, é uma chave e seu identificador é gerado automaticamente.
- Linhas 14 a 17: o atributo Nome é do tipo string, tem seu tamanho máximo em 100 caracteres e não admite valores nulos.
- Linhas 19 a 22: o atributo Data Admissão é mapeado como uma data, através do tipo Doctrine date.
- Linhas 24 a 27: o atributo Salário é mapeado como um número de ponto flutuante e não admite valores nulos.
- Linhas 29 a 32: a propriedade $departamento representa o relacionamento entre uma entidade *Funcionário* e uma entidade *Departamento*. A anotação @ManyToOne indica que para várias entidades de *Funcionário* existe somente uma entidade *Departamento* correspondente. O atributo targetEntity define qual é a classe que contém o mapeamento de um *Departamento*.

```
1  <?php
2  /**
3   * @Entity
4   * @Table(name="funcionario")
5   */
6  class Funcionario {
7    /**
8     * @Id
9     * @Column(type="integer")
10    * @GeneratedValue(strategy="AUTO")
11    */
12   private $codigo;
13
14   /**
15    * @Column(type="string", length=100, name="nome", nullable=false)
16    */
17   private $nome;
18
19   /**
20    * @Column(type="date", name="data_admissao")
21    */
22   private $dataAdmissao;
23
24   /**
25    * @Column(type="float", name="salario", nullable=false)
26    */
27   private $salario;
28
29   /**
30    * @ManyToOne(targetEntity="Departamento")
31    */
32   private $departamento;
33 }
34 ?>
```

Figura B.5 - Mapeamento no Doctrine do Conjunto de Entidades Funcionário

O mapeamento do CE *Departamento* é mostrado na Figura B.6. Os destaques em relação as anotações são apresentados a seguir.

- Linha 15: o atributo nome, mapeado na propriedade $nome, deve conter valores únicos.
- Linhas 24 a 27: a anotação @OneToMany indica que para uma entidade *Departamento* existem várias entidades correspondentes em *Funcionário*. O atributo targetEntity indica que as entidades de *Funcionário* estão mapeadas na classe Funcionario. O atributo mappeBy, determina que o atributo $departamento da classe Funcionario contém o departamento correspondente.

```
1  <?php
2  /**
3   * @Entity
4   * @Table(name="departamento")
5   */
6  class Departamento {
7    /**
8     * @Id
9     * @Column(type="integer")
10    * @GeneratedValue(strategy="AUTO")
11    */
12   private $codigo;
13
14   /**
15    * @Column(type="string", length=100, name="nome", nullable=false, unique=true)
16    */
17   private $nome;
18
19   /**
20    * @Column(type="string", length=100, name="local", nullable=false)
21    */
22   private $local;
23
24   /**
25    * @OneToMany(targetEntity="Funcionario", mappedBy="departamento")
26    */
27   private $funcionarios;
28 }
29 ?>
```

Figura B.6 - Mapeamento no Doctrine do Conjunto de Entidades Departamento

O Doctrine permite realizar a engenharia reversa de uma base de dados relacional, gerando uma conjunto de classes com as anotações necessárias ao mapeamento objeto-relacional. Entretanto o processo, segundo a documentação, detecta entre 70 e 80 por cento das informações necessárias do mapeamento (WAGE et al., 2013a). Esse é o caso das hierarquias de abstração, que não possuem correspondente direto no modelo relacional. Mesmo assim as classes geradas são válida e permitem a manipulação de dados.

Apêndice C - API de Abstração de Base de Dados Relacional

O suporte a diferentes SGBDs no Doctrine é realizado por meio de uma API de abstração de bases de dados relacionais, que fornece uma série de recursos para manipulação transparente das estruturas do esquema de dados da base de dados. A API é chamada de *Database Abstraction & Access Layer* (DBAL).

Existem duas classes principais na API: Schema e SchemaManager. A primeira permite a criação de uma abstração de esquema de dados relacional, que posteriormente pode ser implantando utilizando-se uma conexão a um SGBD, já a segunda contém a representação de um esquema de dados relacional já existente, obtido através de uma conexão com um SGBD.

O diagrama entidade-relacionamento da Figura B.4 será utilizado para exemplificar o mapeamento no DBAL por meio da classe Schema.

```
1  <?php
2  $schema      = new \Doctrine\DBAL\Schema\Schema();
3
4  //Tabela Departamento
5  $departamento = $schema->createTable("departamento");
6  $departamento->addColumn("codigo", "integer");
7  $departamento->addColumn("nome", "string", array("length" => 100));
8  $departamento->addColumn("local", "string", array("length" => 100));
9  $departamento->setPrimaryKey(array("codigo"));
10
11 //Tabela Funcionário
12 $funcionario = $schema->createTable("funcionario");
13 $funcionario->addColumn("codigo", "integer");
14 $funcionario->addColumn("nome", "string", array("length" => 100));
15 $funcionario->addColumn("data_admissao", "date", array("notnull" => true));
16 $funcionario->addColumn("salario", "float");
17 $funcionario->addColumn("codigo_dep", "integer");
18 $funcionario->setPrimaryKey(array("codigo"));
19
20 $funcionario->addForeignKeyConstraint($departamento, array("codigo_dep"), array("codigo"));
21
22 //Geração do DDL
23 $listaDDL = $schema->toSql(new \Doctrine\DBAL\Platforms\PostgreSqlPlatform());
24 ?>
```

Figura C.7 - Mapeamento no DBAL

A explicação do mapeamento feito no DBAL é apresentado a seguir.

- Linha 2: cria um objeto do tipo Schema, que representa um esquema de base de dados relacional.
- Linha 5: define a tabela departamento na representação do esquema.
- Linhas 6 a 8: adiciona as colunas à tabela departamento com os respectivos tipos Doctrine.
- Linha 9: define a chave primária da tabela departamento. O parâmetro é um vetor contendo os nomes das colunas que compõe a chave primária.
- Linhas 12 a 18: definição da tabela departamento.
- Linha 20: define a chave estrangeira na tabela funcionário, que referencia a tabela departamento.
- Linha 23: o método toSql retorna um mapa ordenado de valores contendo as instruções DDL para criação da base de dados de acordo com os elementos definidos nas linhas anteriores. O método toSql, recebe como argumento uma instância de um objeto do tipo Platform, que representa um determinado SGBD. No exemplo o DDL será gerado para o SGBD PostgreSQL.

O SchemaManager constrói a abstração de um esquema de base de dados relacional já existente por meio de uma conexão ao SGBD. O Quadro C.3 descreve os principais métodos da classe SchemaManager.

Outro recurso importante no DBAL é a possibilidade de comparação de duas representações de esquemas de dados através da classe Comparator. O resultado pode ser utilizado para aplicar as diferenças de um esquema de dados em outro. O código apresentado na Figura C.8 mostra um exemplo do uso do Comparator, em que $schemaFonte e $schemaDestino são objetos do tipo Schema. Por fim, na linha 5, o método toSql gera uma lista com os comandos DDL necessários para se alterar o esquema fonte de acordo com as diferenças com o esquema destino.

Quadro C.3 - Principais métodos da classe SchemaManager.
(continua)

Assinatura do Método	**Descrição**
listSequences()	Retorna as sequências da base de dados.

Quadro C.3 - Principais métodos da classe SchemaManager.
(conclusão)

Assinatura do Método	Descrição
listTableColumns(<nome tabela>)	Retorna as colunas de uma tabela a partir do argumento <nome tabela>.
listTableDetails(<nome tabela>)	Retorna a representação completa de uma tabela a partir do argumento <nome tabela>.
listTableForeignKeys(<nome tabela>)	Retorna as chaves estrangeiras de uma tabela.
listTableIndexes(<nome tabela>)	Retorna os índices de uma tabela.
listTables()	Retorna uma lista de tabelas da base de dados.
createSchema()	Retorna um objeto do tipo Schema, com a representação do esquema da base de dados.

```
1 <?php
2 $comparator = new \Doctrine\DBAL\Schema\Comparator();
3 $schemaDiff = $comparator->compare($schemaFonte, $schemaDestino);
4
5 $listaDDL = $schemaDiff->toSql($platform);
6
7 ?>
```

Figura C.8 - Comparação de esquemas com Comparator

Com os recursos apresentados é possível obter uma representação de um esquema de base de dados já implementado, pelo método createSchema da classe SchemaManager, e compará-lo com uma representação gerada dinamicamente, por código, com a classe Schema.

Apêndice D - Publicação

GUEDES, G. B.; BAIOCO, G. B.; MORAES, R. Database Modularization applied to System Evolution. In: Sixth Latin American Symposium on Dependable Computing - LADC 2013, 6., 2013, Rio de Janeiro. **Anais…**Rio de Janeiro: UFMG, 2013. p. 81-82.

Database Modularization applied to System Evolution

Gustavo Bartz Guedes
University of Campinas
gubartz@gmail.com

Gisele Busichia Baioco
University of Campinas
gisele@ft.unicamp.br

Regina L. O. Moraes
University of Campinas
regina@ft.unicamp.br

Abstract

Evolutionary development needs to deal with constant changes in requirements, including new or complement and change existent ones. Most of time, to accommodate requirements, changes in database schema are mandatory, in addition to software code modification. This work's proposal helps to conduct these changes with minor impact in software dependability, providing a structured and trustworthy method to conduct the evolutionary process. A specification for an integration layer, supported by the modularized schema metadata is proposed reducing the need for physical changes to the database, which in turn prevents query loss, thus increasing availability and maintainability.

1. Introduction

Software systems are in constant evolution and most of them manipulate data through a persistence layer supported by a database. Between the evolution cycles new requirements may arise, demanding changes in the database schema. These changes cannot compromise the software's dependability.

The key process in database design is its definition, which is made in the form of a schema that represents the database metadata. Assessing the impacts of the required changes in all application levels is essential.

An issue in database evolution according to Cleve et al [1] is that the loss of already working queries, defined as query loss, can reach up to 70% per schema version, especially due to the technology impedance mismatch between the application and the database [2].

Our proposal towards schema evolution is to adapt the database modularization technique described in [3] reducing the need for database schema changes and to provide a structured and trustworthy method to conduct the evolution process. We also provide the specification for an integration layer, supported by the modularized schema metadata. Therefore, changes are applied at this layer reducing the need for physical changes, which in turn prevents query loss, thus increasing availability. In this way the maintainability process can be conducted in an easier and more safety way, which contributes to the system integrity.

Nowadays, database evolution is an increasingly important area due to the arising of new incremental software development methods, such as the Agile ones.

2. Database Modularization and Evolution

Database modularization decomposes the database schema into database modules that contain a conceptual view of a subschema along with its related functionalities. The resulting database modules are loosely coupled and autonomous subschemas that are defined by grouping transactions performed by the application in subsystems, thus increasing cohesion and positively impacting further maintainability.

The technique provides a way to conduct the database design in an evolutionary scenario. It narrows down the scope when assessing the required changes and impacts caused by the software evolution. Later the modules interoperability is provided by the integration object layer.

Our work includes the definition of a repository that is populated along with the process and it can be queried to support the evolution process at each cycle also providing an external view to the application of the database modules. Figure 1 shows the classical database design process including the two database modularization phases: "Collection and Analysis of Modularization Requirements" and "Modularization Design".

The "Collection and Analysis of Modularization Requirements" is responsible for grouping the software systems functionalities in subsystems, according to the transactions performed by the application.

The "Modularization Design" is conducted within the Conceptual Design and is divided into four stages. In the first stage a subschema is define for each subsystem in respect to the elements that it accesses. An overlap between these subschemas can occur, when a database element belongs to more than one subsystem's subschema. Therefore, the next stage treats the information sharing, where each element is classified considering read or write access performed by each subsystem. Next the elements are grouped according to the subsystems responsible to perform the

LADC'2013, April 2-5, 2013, Rio de Janeiro, Brazil.
ISBN 978-85-7669-274-4.

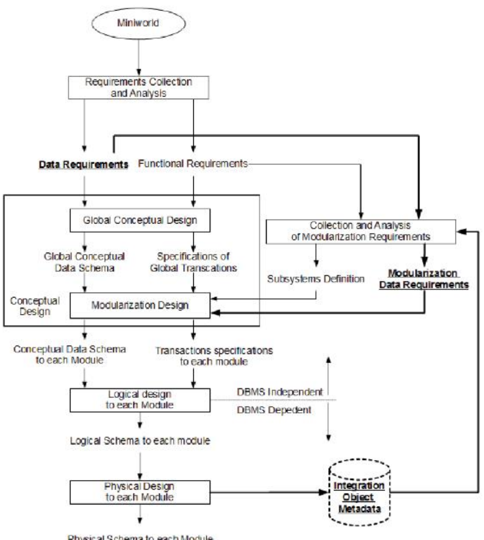


Figure 1 - Database Design Process including Modularization and Evolution

writing operations. Later, the intersection operations between these groups define the database modules. An empty intersection originates a database module with the participating elements; whereas a non-empty intersection results in a database module with the intersecting elements or an union among all elements from the groups. Finally the databases modules are encapsulated with the writing (private procedure) and reading transactions (public procedure) and the logical and physical database schemas for each module are created. A graphical representation of a database module is presented in Figure 2.

The integration of the database modules is performed by the integration object layer, which is responsible for several types of conversions, such as: data, data structures, semantic value of data, data validation rules, procedures, interfaces and data managers. In our work it consists of a metadata repository of the database modules that can be queried to check if the already existing database modules support the new requirements. It is also updated to reflect database schema changes avoiding physical

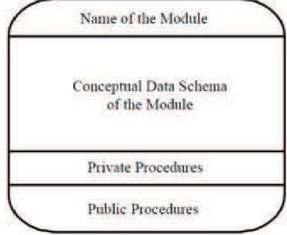


Figure 2 - A module representation [3]

schema changes.

As shown in Figure 3, the integration object approach in [4] was extended to provide a transparent view of the database modules through an execution engine that obtains information from the metadata repository. The integration object layer functionalities can be automated, by either extending the database management system metadata or through a framework.

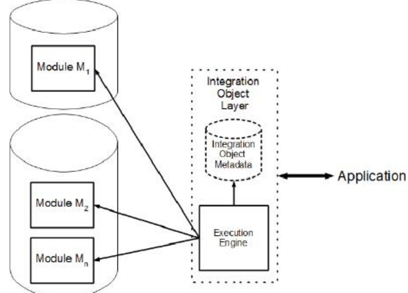


Figure 3 - Integration Object Layer

3. Conclusion and Future Work

Preserving software integrity through an evolutionary process demands a great deal of effort, especially because they involve two different paradigms: code and database designs. Database modularization is a standardized way to evolutionary database schema design.

A database module holds both the subschema and the subset of related application transactions. Ergo, it improves traceability and contributes in the maintainability process, performed at module level.

Database changes can occur at the integration object metadata level, which increase availability, since less physical schemas changes are required and the already implemented queries (already in use in the operational environment) can be preserved.

Future work includes the development of a framework to implement the integration object layer.

References

[1] A. Cleve, A. F. Brogneaux e J. Hainaut, “A Conceptual Approach to Database Applications Evolution,” ER'10 Proceedings of the 29th international conference on Conceptual modeling, pp. 132-145, 2010.

[2] S. W. Ambler, Agile Database Techniques, Wiley, 2003.

[3] J.E. Ferreira and G. Busichia, Database modularization design for the construction of flexible information systems, Database Engineering and Applications, 1999. IDEAS '99, pp. 415-422

[4] G. Busichia and J.E. Ferreira, Sharing of heterogeneous databases modules by integration objects, Advances in Databases and Information Systems, 1999, pp. 1-8.