Implementação de Banco de Dados para uso como laboratório experimental

Luiz Henrique B. Lago[1], **Sérgio Luís S. Mergen** [1]

[1]Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brazil

lhlago@inf.ufrgs.br,mergen@inf.ufsm.br

Abstract. *This paper describes the implementation and operation of a database intended for use as an experimental laboratory, in particular for teaching and researching the implementation of a DBMS. In this paper, we discuss the layer architecture in which the database was implemented and some basic notions of each system layer, as well as the communication interfaces between these layers.*

Resumo. *Este artigo descreve a implementação e o funcionamento de um banco dados voltado para o uso como um laboratório experimental, em especial para o ensino e pesquisa da implementação de um SGBD. Neste artigo, serão discutidas a arquitetura de camadas na qual o banco de dados foi implementado e algumas noções básicas de cada camada do sistema, assim como as interfaces de comunicação entre estas camadas.*

1. Introdução

O banco de dados é um dos principais elementos da computação moderna, usado para armazenar, organizar e manipular grandes quantidades de informações. Uma das áreas de estudo em computação é o funcionamento interno de bancos de dados, que visa explicar a sua arquitetura, estruturas de dados e algoritmos que consigam armazenar, buscar e filtrar de forma eficiente, reduzindo os custos de comunicação entre a memória principal e a memória secundária.

No ensino e aprendizado sobre bancos de dados relacionais é possível encontrar diversas ferramentas que possibilitam tanto auxiliar na construção de consultas em banco de dados reais [Greubel et al. 2020] como também ferramentas com um foco mais teórico no ensino da linguagem SQL [Andreas Grillenberger 2012]. Em contrapartida, o presente trabalho se centraliza na execução, processamento e análise de diferentes algoritmos no desenvolvimento de um banco de dados.

Nesse sentido, este artigo propõe um arquitetura extensível de banco de dados, cujo propósito é ser usado como um laboratório para o aprendizado e a elaboração de experimentos e testes, assim como o sistema Minix foi para a área de sistemas operacionais [Tanenbaum and Woodhull 2006]. Com o sistema projetado, é possível exemplificar os princípios de um banco de dados diretamente em código, em uma arquitetura modular que facilita a criação e substituição de algoritmos, de modo a analisar como diferentes alternativas afetam o funcionamento do sistema gerenciador de banco de dados.

Vale também destacar os softwares SimpleDB [Sciore 2007] e Minibase [Ramakrishnan and Gehrke 2000] que apresenta uma proposta semelhante. Existem

diferenças de recursos entre as ferramentas, porém o maior diferencial é em relação a forma que é estruturado o programa. No banco de dados desenvolvido nesse trabalho, cada camada do sistema funciona de maneira independente e a estrutura do motor de armazenamento e o SGBD são pacotes isolados um do outro. O objetivo dessa arquitetura é poder atingir uma extensibilidade maior do software, ou seja, possibilitar a modificação das camadas, adicionar flexibilidade no desenvolvimento do software, e permitir o software funcionar sem as camadas mais altas de abstração, útil para fazer estudos focados na *storage engine*.

2. Arquitetura Organizacional

Como o banco de dados desenvolvido tem seu uso voltado para ser um laboratório prático, escolheu-se para a implementação uma arquitetura que divida o software em camadas, sendo cada camada responsável por uma tarefa específica. Essa estrutura de organização de código é essencial para dividir o estudo complexo de um banco de dados em estruturas especializadas menores.

Mais especificamente, o sistema hoje possível seis camadas principais, conforme ilustrado na Figura 1, que constituem a estrutura do banco. Os objetos de comunicação entre cada nível identificam o papel de cada camada, sendo ela responsável por fazer a transformação do dado do nível acima para o dado da nível abaixo, e vice-versa. Vale destacar que, analogamente a bancos de dados convencionais, as três primeiras camadas do topo compõem a *storage engine*, enquanto as abaixo estabelecem o funcionamento do SGBD.

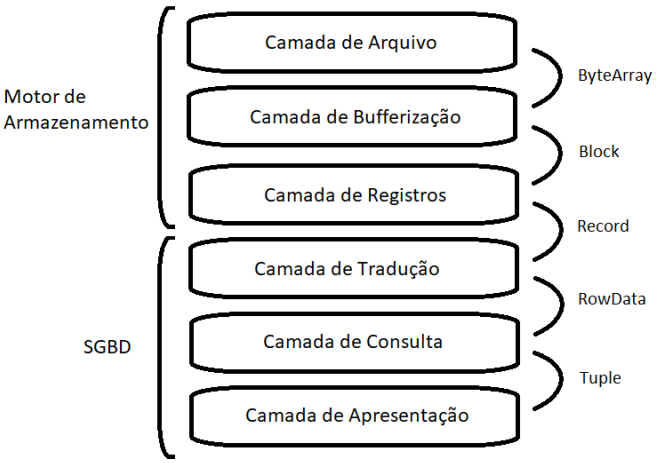

Figura 1. Divisão das principais camadas e os tipos de dados que se comunicam.

A linguagem escolhida para o desenvolvimento do sistema foi Java por ter um suporte nativo à orientação a objetos, o que proporciona o desenvolvimento de um sistema de camadas que se comunicam via interfaces. Além disso, Java é uma linguagem rápida por fazer uso de um compilador *JIT* na execução da *JVM*. Todo o código fonte foi publicado online na plataforma *Github* pelo nome de *Fy-Database* [Lago 2022], e está disponível para acesso público.

O software tem implementado atualmente um conjunto de classes que constituem o funcionamento do banco, e a partir de código é possível interagir com o sistema. Na

Figura 2 podemos ver um exemplo da construção de uma operação de álgebra relacional utilzando as classes do sistema.

```
Table users = Table.loadFromHeader( headerPath: "users.head");
Table cidades = Table.loadFromHeader( headerPath: "cidades.head");

users.open();
cidades.open();

Operator selectUsers = new TableScan(users, Arrays.asList("id","idade","nome", "idCidade","anoNascimento"));
Operator where = new FilterOperator(selectUsers,(Tuple t)->{
    return t.getContent( name: "users").getInt( column: "idade") >=18;
});

Operator selectAllCidades = new TableScan(cidades);
Operator join = new BlockNestedLoopJoin(where,selectAllCidades,(t1, t2) -> {
    return t1.getContent( name: "users").getInt( column: "idCidade") == t2.getContent( name: "cidades").getInt( column: "id");
});

Operator executor=join;
executor.open();
while(executor.hasNext()) {
    Tuple t = executor.next();
    String str = t.getContent( name: "cidades").getString( column: "name")+" - "
            +t.getContent( name: "users").getString( column: "name");
}
```

Figura 2. Código para a execução de uma operação de busca no sistema.

3. Atribuições das Camadas

Sobre a função de cada camada que compõe a arquitetura:

Camada de arquivo: A camada de arquivo é responsável por criar, deletar e ler dados a partir de um arquivo. Através de recursos de mapeamento, essa camada acessa os dados diretamente do arquivo, sem utilizar recursos de bufferização do sistema operacional, o que leva a um melhor aproveitamento de memória. Essa camada retorna os dados solicitados pela camada em um objeto do tipo *ByteBuffer* existente na biblioteca padrão da linguagem.

Camada de bufferização: A camada de bufferização implementa uma estratégia de gerenciamento de blocos, cujo principal objetivo é lidar com situações em que a quantidade de dados acessados supera a quantidade de memória disponível. Nessa camada foram implementados dois tipos distintos de algoritmos (*FIFO* e *LIFO*) e uma versão sem buffer em memória. O tamanho dos blocos é uma informação parametrizável. Os usuários podem analisar como modificações no tamanho afetam o desempenho geral no acesso aos dados.

Camada de registros: A camada de registros é responsável pela organização física dos dados em registros. Cada registro é um objeto do tipo *Records*, composto por uma chave primária de tamanho fixo e uma sequência de bytes de tamanho variável. Essa camada não faz distinção entre colunas, e armazena todas as colunas como se fosse uma sequência de bytes.

Camada de tradução: A camada de tradução tem como função transformar um *Record* em uma estrutura separadas por colunas, indexadores e tipos. É nessa camada que a distinção acontece, separando o identificador em múltiplas colunas de chave primária, e o corpo dos dados nas colunas definidas na tabela.

Camada de consulta: A camada de consulta é responsável por montar e executar a estrutura de consulta. O plano de consulta é montado a partir do padrão de projeto

Composition. Todas operações do plano de consulta herdam de uma classe genérica que disponibiliza funções de iteradores. Ao combinar os operadores, que podem ser unários ou binários, é gerado um plano de consulta composto pelas operações típicas estudadas em álgebra relacional, como junção, seleção e projeção.

De modo geral, o banco de dados permite o armazenamento em arquivo de estruturas complexas com colunas e tipos de coluna, bem como colunas com tamanho variável, colunas nulas e outros tipos de indexadores para colunas. Cada tabela fica armazenada em um arquivo separado, o que permite que se use estratégias diferentes para cada tabela. Por exemplo, uma tabela pode usar o método *heap* de organização de arquivos, enquanto outra tabela pode manter os registros em ordem.

4. Conclusão

Como considerações finais, destaca-se que todo o fluxo de comunicação dos dados entre as camadas já foi validado, ou seja, já existe uma versão do banco de dados completamente funcional, que serve como um laboratório experimental para estudo ou pesquisa. No entanto, a versão atual ainda carece de algoritmos distintos dentro de cada camada. Por exemplo, a camada de bufferização disponibiliza apenas dois algoritmos (*FIFO* e *LIFO*). Essa limitação pode ser vista como uma oportunidade, pois abre caminho para que alunos implementem estratégias diferentes como atividades dentro de disciplinas voltadas ao estudo de bancos de dados.

Outro ponto que precisa ser melhorado é o suporte ao acesso concorrente, pois a versão atual é mono-usuário. Dessa forma, ainda é preciso incorporar módulos de gerenciamento de acesso concorrente à arquitetura. Assim como as demais camadas, pretende-se atingir um baixo acoplamento, para que seja fácil substituir os algoritmos e observar como diferentes estratégias influenciam no tempo de resposta e em outras métricas de qualidade.

Referências

Andreas Grillenberger, T. B. (2012). EledSQL - a new web-based learning environment for teaching databases and sql at secondary school level. *WiPSCE*.

Greubel, A., Rudolph, T., and Hennecke, M. (2020). VeraSQL: An educational client for relational databases. In Zender, R., Ifenthaler, D., Leonhardt, T., and Schumacher, C., editors, *DELFI 2020 – Die 18. Fachtagung Bildungstechnologien der Gesellschaft für Informatik e.V.*, pages 351–352, Bonn. Gesellschaft für Informatik e.V.

Lago, L. H. B. (2022). FyDatabase. `https://github.com/crazynds/FyDatabase-Java`. [Online; accessed 03-Janeiro-2023].

Ramakrishnan, R. and Gehrke, J. (2000). *Database Management Systems*. McGraw-Hill, Inc., USA, 2nd edition.

Sciore, E. (2007). Simpledb: A simple java-based multiuser syst for teaching database internals. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, page 561–565, New York, NY, USA. Association for Computing Machinery.

Tanenbaum, A. S. and Woodhull, A. S. (2006). *Operating Systems Design and Implementation*. Pearson Prentice Hall, Upper Saddle River, NJ, 3 edition.