UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Samuel Matias Finkler

GERAÇÃO DE BANCO DE DADOS A PARTIR DE CONSULTAS SQL

Santa Maria, RS
2023

Samuel Matias Finkler

GERAÇÃO DE BANCO DE DADOS A PARTIR DE CONSULTAS SQL

Monografia apresentada ao Curso de Graduação em Ciência da Computação, Área de Concentração em Ciência da Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharelado em Ciência da Computação**.

Orientador: Prof. Sergio Luis Sardi Mergen

Santa Maria, RS
2023

Samuel Matias Finkler

GERAÇÃO DE BANCO DE DADOS A PARTIR DE CONSULTAS SQL

Monografia apresentada ao Curso de Graduação em Ciência da Computação, Área de Concentração em Ciência da Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharelado em Ciência da Computação**.

Aprovado em 12 de dezembro de 2023:

Sergio Luis Sardi Mergen, Dr. (UFSM)
(Presidente/Orientador)

Giovani Rubert Librelotto, Dr. (UFSM)

Deise de Brum Saccol, Dr. (UFSM)

Santa Maria, RS
2023

AGRADECIMENTOS

Gostaria de agradecer inicialmente a toda a minha família, aos meus pais, Aloisio Finkler e Marcia Inês Stulp Finkler, pelo apoio e incentivos durante todo o curso e também à minha irmã Martina Finkler que serviu de exemplo pela sua trajetória acadêmica.

Agradeço a minha namorada Andreza Evaldt de Lima, por me incentivar a realizar todas as minhas tarefas mesmo em momentos de sobrecarga e desânimo, e também por se fazer sempre presente.

Agradeço aos colegas de curso e amigos que fiz durante a graduação, especialmente ao Guilherme Pretto e ao Thiago Sordi, que estiveram comigo na maioria dos trabalhos em grupo e durante grande parte da carga horária do curso.

Agradeço ao meu orientador Sergio Mergen pelas oportunidades de projetos de pesquisa ofertadas durante a graduação, pelas reuniões semanais e pela paciência nos momentos em que o projeto não evoluiu no ritmo esperado.

Agradeço aos representantes da banca Deise e Giovani, por terem aceito o convite e pelas valiosas sugestões para melhorar esse trabalho.

Agradeço a UFSM, aos professores e ao curso de Computação por todo apoio, suporte e aprendizagens, com destaque especial ao Programa de Educação Tutorial (PET) e ao professor Giovani Librelotto, tutor do programa durante a minha participação.

RESUMO

GERAÇÃO DE BANCO DE DADOS A PARTIR DE CONSULTAS SQL

AUTOR: Samuel Matias Finkler
Orientador: Sergio Luis Sardi Mergen

A evolução dos sistemas computacionais impulsionou a importância da gestão eficiente de dados, onde os bancos de dados se tornaram essenciais. A linguagem SQL é fundamental no ensino de Ciência da Computação, mas os alunos enfrentam desafios na redação de consultas, dificultando a correção para os professores. Para abordar essa dificuldade, um projeto de pesquisa foi criado para verificar se as consultas dos alunos coincidem com o gabarito dos professores, garantindo que produzam os mesmos resultados. Isso será realizado por meio de uma ferramenta gráfica que auxiliará tanto docentes na correção quanto alunos na revisão prévia dos trabalhos. O projeto se divide em dois módulos independentes: geração de dados e comparação de consultas. Este trabalho se concentra na geração automática de bancos de dados preenchidos com dados sintéticos para auxiliar na prática e correção de consultas, deixando a comparação das consultas para uma fase subsequente do projeto. Destaca-se a importância da geração criteriosa de dados sintéticos para permitir a identificação de diferenças durante a correção, considerando condições impostas pelas consultas, como filtros aplicados.

Palavras-chave: Banco de dados. SQL. Ensino.

ABSTRACT

DATABASE GENERATION FROM SQL QUERIES

AUTHOR: Samuel Matias Finkler
ADVISOR: Sergio Luis Sardi Mergen

The continuous evolution of computational systems has underscored the critical need for efficient data management, where databases have become essential. The SQL language is pivotal in Computer Science education, yet students face challenges in formulating queries, making it difficult for teachers to assess. To address this challenge, a research project was initiated to verify if students' queries align with teachers' benchmarks, ensuring they yield the same results. This will be facilitated through a graphical tool aiding both educators in grading and students in pre-submission review. The project comprises two independent modules: data generation and query comparison. This work focuses on automatically generating databases filled with synthetic data to aid in query practice and grading, leaving query comparison for a subsequent phase. It emphasizes the importance of judicious synthetic data generation to allow for identifying differences during grading, considering conditions imposed by queries such as applied filters.

Keywords: Database. SQL. Education.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

CSV	Comma-separated Values
DDL	Data Definition Language
DML	Data Manipulation Language
RQP	Reverse Query Processing
RRA	Reverse Relational Algebra
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
SQP	Symbolic Query Processing

SUMÁRIO

1 INTRODUÇÃO

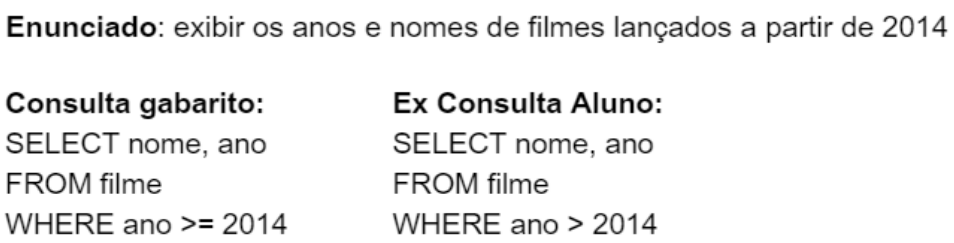
No âmbito da evolução contínua dos sistemas computacionais, a gestão eficaz de grandes volumes de dados tornou-se uma necessidade premente. Os bancos de dados emergiram como pilares essenciais para o armazenamento, recuperação e manipulação precisa de informações. A linguagem SQL (Structured Query Language) destacou-se como uma ferramenta fundamental para interagir com bancos de dados relacionais, viabilizando consultas e operações complexas.

No contexto acadêmico de cursos de Ciência da Computação, a disciplina de Banco de Dados assume papel crucial ao abordar conceitos e técnicas fundamentais para a manipulação e gestão de informações. Contudo, um desafio recorrente para estudantes reside na redação de consultas SQL, exigindo um entendimento profundo dessas consultas e a capacidade de visualizar seus efeitos em um ambiente de banco de dados real.

É papel do professor guiar os alunos por meio de aulas teóricas e práticas. No entanto, surge uma dificuldade considerável ao corrigir manualmente os trabalhos dos alunos, especialmente quando se trata de avaliar consultas SQL. A diversidade e complexidade das soluções possíveis tornam essa correção um processo demorado e sujeito a interpretações, o que por vezes prejudica a precisão da avaliação.

Para ilustrar, considere um professor de uma disciplina de banco de dados desafiando seus alunos com uma tarefa que requer a elaboração de consultas SQL a partir de enunciados descritivos, como por exemplo: "apresentar os anos e títulos de filmes lançados a partir de 2014". O professor conta com uma consulta de resposta usada como gabarito para a correção. A partir disso, os alunos são incumbidos de tentar reproduzir essa consulta. A Figura 1 apresenta uma das possibilidades de resultado.

Figura 1 – Exemplo de enunciado, gabarito e resolução de uma consulta



Fonte: Próprio autor.

Perceba que existe uma diferença tênue na formulação das duas consultas, mas que é suficiente para que sejam gerados resultados diferentes. Pode ser difícil perceber essa diferença, ainda mais considerando o volume de trabalhos a ser corrigido e a complexidade das consultas exigidas.

Nesse cenário, foi criado um projeto de pesquisa que visa conferir se a consulta

submetida pelo aluno é equivalente à consulta usada como gabarito pelo professor. A equivalência significa que ambas as consultas retornam sempre os mesmos registros, independente do estado do banco de dados (quais registros são armazenados). A intenção é disponibilizar essa comparação por meio de uma ferramenta gráfica, com aplicabilidade tanto para os docentes, como uma forma de agilizar a avaliação dos trabalhos, quanto para os alunos, como uma assistência que permita a correção prévia antes do envio da versão final para avaliação.

Esse projeto é composto por dois módulos que podem ser trabalhados de forma independente:

- **Geração dos dados:** este trabalho envolve a geração dos registros e, se necessário, do esquema, para que as consultas formuladas possam ser executadas.
- **Comparação das consultas:** este trabalho envolve comparar o resultado de duas consultas e destacar, por meio de mensagens, os ajustes que precisam ser feitos para que a consulta enviada pelo aluno seja considerada correta.

Este trabalho final de curso concentra-se especificamente no primeiro objetivo, ou seja, a geração automática de bancos de dados, preenchidos com dados sintéticos. A etapa de comparação dos resultados entre consultas será desenvolvida em um módulo separado, em uma fase subsequente do projeto.

Cabe salientar que a geração de dados sintéticos deve ser feita com critério, para possibilitar identificação das diferenças dos resultados durante a comparação. Para isso, será fundamental analisar as condições impostas pela consulta, como por exemplo, quais filtros foram realizados.

Este trabalho está estruturado da seguinte forma: no capítulo 2 será apresentada a fundamentação teórica, que compreende um resumo dos principais conceitos que servirão como base para esta proposta, bem como trabalhos relacionados. No capítulo 3 será discutida a arquitetura proposta e destacados os desafios a serem enfrentados especificamente na geração de dados sintéticos. O capítulo 4 apresenta alguns resultados experimentais. O capítulo 5 traz as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão abordados os conceitos teóricos que fundamentam o desenvolvimento do presente trabalho, apresentando conceitos fundamentais para a sua compreensão. Inicialmente serão discutidos temas como banco de dados, linguagem de consulta estruturada, parsing, extração de esquema, extração de tipos de dados e testes de software. Pom fim, serão apresentados trabalhos relacionados com os temas previstos.

2.1 BANCO DE DADOS

Para a escrita do presente capítulo e seus subcapítulos, com o intuito de contextualizar sobre temas como banco de dados, estrutura de bancos de dados relacionais e linguagem de consulta estruturada (SQL), serão utilizados três livros: Introdução a sistemas de bancos de dados (DATE, 2004); Sistema de Banco de Dados (SILBERSCHATZ; KORTH; SUDARSHAN, 2020) e Sistemas de Gerenciamento de Bancos de Dados (RAMAKRISHNAN; GEHRKE, 2008).

Um sistema de banco de dados é um sistema computadorizado com o objetivo de armazenar registros com informações relevantes a algum propósito e prover ao usuário formas de consultar e atualizar os seus dados quando necessário. Esse sistema é formado por quatro componentes principais: dados, hardware, software e usuários. Os dados são as informações relevantes ao usuário ou empresa, que são armazenados de forma persistente. Hardware é a estrutura física para comportar o armazenamento de informações, tal como processador, memória principal e volumes de memória secundária. Software representa o sistema de gerenciamento de bancos de dados (SGBD), que é a camada intermediária entre os dados armazenados e o usuário, provendo ao usuário uma interface para realização de operações como consultas e alterações de dados. Já os usuários referidos podem ser os programadores de aplicação responsáveis por desenvolver aplicações, os usuários finais que vão utilizar o sistema e os administradores de banco de dados (DBA) que são responsáveis pela criação e manutenção do banco (DATE, 2004).

Junto a um banco de dados, temos um modelo de dados, que define a forma como esses dados são descritos, as relações entre os dados, a semântica de dados e as restrições de consistência. Tomaremos como destaque o modelo relacional que é o modelo de dados utilizado pela grande maioria dos sistemas de banco de dados atuais e foco do presente trabalho. O modelo relacional é baseado em tabelas para representar os dados e as suas relações. Cada tabela possui uma estrutura padronizada para um tipo específico de registro e cada registro possui um número fixo de atributos, caracterizando um modelo baseado em registros (SILBERSCHATZ; KORTH; SUDARSHAN, 2020).

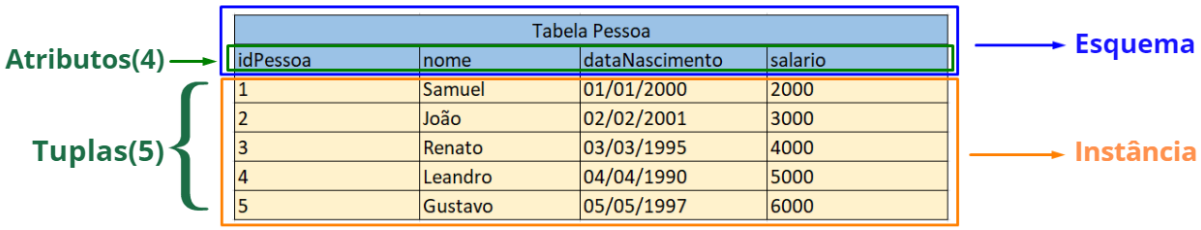
2.1.1 Estrutura de banco de dados relacional

O modelo de banco de dados relacional pode ser definido como uma coleção de uma ou mais tabelas, em que cada tabela possui linhas e colunas. No decorrer do texto, tabelas também serão citadas como relações. As linhas da tabela serão citadas como tuplas ou registros e as colunas da tabela também serão citadas como atributos.

Ao abordar a estrutura de um banco de dados, é importante destacar a diferença entre esquema de banco de dados e instância de banco de dados. O esquema do banco de dados é a sua representação lógica e a instância do banco de dados é a representação de seus dados em determinado momento.

Na Figura 2 é feita uma demonstração de um esquema de banco de dados e sua instância em determinado momento. O termo instância de relação é utilizado para se referir a uma instância específica de uma tabela, que seria um conjunto específico de tuplas. Já a ordem em que os registros são representados em uma relação não é relevante. Cada atributo de uma relação possui um domínio, que é o conjunto de valores permitidos. Também é possível que se tenha valores nulos em casos em que os valores são desconhecidos ou indisponíveis.

Figura 2 – Representação de esquema e instância



Fonte: Próprio autor.

É preciso que uma tabela tenha definida uma chave primária, que identifica unicamente as suas tupla. Comumente, é utilizado um campo identificador (Id) único como chave primária de uma tabela. Por outro lado, não devem ser utilizados para este propósito campos que podem se repetir entre uma tupla e outra da mesma instância, como no caso de um campo que armazena um nome.

Como a chave primária de uma relação identifica uma tupla unicamente em uma relação, é imposta a restrição de que duas tuplas não podem ter ao mesmo tempo os mesmos valores nos campos que compõem a chave primária. Também é recomendado que a chave primária adotada em uma tabela utilize campos que não sejam frequentemente alterados.

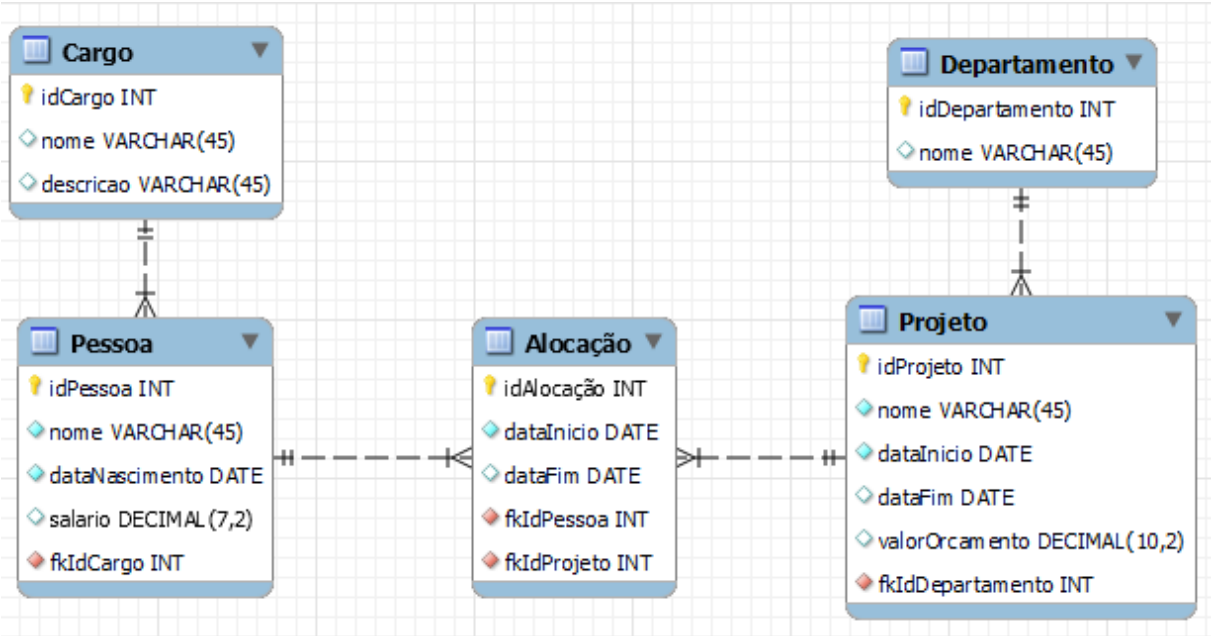
Uma chave estrangeira é uma chave em uma tabela que referencia uma chave primária da própria tabela ou de outra tabela presente no banco de dados. Assim, a tabela que possui a chave estrangeira pode ser chamada de relação referenciadora, que faz refe-

rência a uma relação referenciada.

No contexto de sistemas de banco de dados, a distinção entre os modelos conceitual e lógico é fundamental. O modelo conceitual foca na representação de alto nível das estruturas de dados, capturando a essência do que deve ser armazenado, sem se preocupar com detalhes específicos de implementação. Este modelo é frequentemente representado através de diagramas entidade-relacionamento, destacando entidades, seus atributos e relações. Por outro lado, o modelo lógico transpõe esta visão abstrata para uma estrutura mais próxima da implementação, detalhando como os dados serão efetivamente armazenados em um sistema de banco de dados. Ele define tabelas, chaves, tipos de dados e relações, servindo como uma ponte entre o modelo conceitual abstrato e a implementação física do banco de dados. Essa distinção permite que os desenvolvedores planejem cuidadosamente a estrutura dos dados, mantendo uma visão clara da arquitetura do banco de dados em diferentes níveis de abstração.

A seguir, é demonstrado na Figura 3, um esquema de banco de dados representado por um diagrama lógico. O diagrama possui cinco entidades, sendo elas cargo, pessoa, alocação, projeto e departamento. Cada uma das entidades possui sua chave primária representada por um id. Há alguns relacionamentos entre as entidades, como entre 'pessoa' e 'cargo', em que 'pessoa' é a relação referenciadora e 'cargo' é a relação referenciada. Portanto, um dos atributos de 'pessoa' é uma chave estrangeira que referencia uma tupla da relação 'cargo'.

Figura 3 – Exemplo de diagrama lógico de banco de dados



Fonte: Próprio autor.

2.1.2 Álgebra relacional

Uma linguagem de consulta é uma linguagem utilizada para que dados sejam buscados em um banco de dados por um usuário. A linguagem mais utilizada para consultas em bancos de dados relacionais é a linguagem de consulta estruturada (SQL - Structured Query Language), que será discutida com mais detalhes na sequência. Neste subcapítulo, será apresentada a álgebra relacional, que forma a base teórica da linguagem de consulta SQL.

Álgebra relacional é uma forma de representar operações a serem realizadas em uma ou mais relações em um banco de dados. Cada uma dessas operações produz um resultado para a sequência das operações. Essas operações podem ser unárias, que operam sobre uma relação apenas, ou binárias, que operam sobre pares de relações. Exemplos de operações unárias são seleção, projeção e renomeação. Exemplos de operações binárias são união, produto cartesiano e diferença.

A operação 'seleção' seleciona tuplas que atendem a determinadas condições. Essas condições podem ser feitas utilizando comparações entre atributos e valores, ou entre dois atributos de uma relação e podem utilizar operadores relacionais como: igual (=), diferente (<>), menor que (<), menor ou igual a (<=), maior que (>), maior ou igual a (>=). Em caso de junção de mais condições, podemos utilizar os conectores 'e', 'ou' e 'não'.

A operação 'projeção' é uma operação unária que permite filtrar colunas de uma relação, para que sejam obtidos apenas os atributos que interessam e omitidos os demais.

A operação 'produto cartesiano' é utilizada para combinar as informações de duas relações. O resultado do produto cartesiano de duas relações é uma única relação que contém todas as combinações possíveis entre os elementos das duas relações originais. Assim, o número de colunas da relação resultante da operação é equivalente a soma das duas colunas das duas relações originais e o número de tuplas da relação resultante é igual ao produto do número de linhas das relações originais.

Na operação 'união' são juntadas as informações que estão em duas relações distintas em uma única relação após a operação. A relação resultante possui o mesmo número de colunas das relações originais e possui, no máximo, número de linhas equivalente às linhas das relações originais somadas. Para a união, é um requisito que as relações que serão unidas sejam compatíveis.

A operação 'interseção' é outra operação binária que permite encontrar tuplas presentes nas duas relações de entrada. As restrições de compatibilidade entre as relações de entrada são as mesmas da operação de união.

A operação 'diferença de conjuntos' permite encontrar tuplas que estejam presentes em uma relação e ausentes em outra. Essa é mais uma operação que deve considerar relações compatíveis.

A operação 'atribuição' não fornece capacidades adicionais para a álgebra, mas é importante para simplificar a representação de uma relação formada por operações com-

plexas. Assim, é possível representar uma relação proveniente de operações anteriores com uma variável de relação temporária nomeada de forma intuitiva. Essa variável que representa a relação pode ser usada em relações subsequentes.

A operação 'renomeação' é usada para renomear a saída de uma relação. Assim, é possível salvar o resultado de uma expressão complexa como uma relação com um nome específico para utilização subsequente. Outro uso importante e frequente da renomeação é quando trabalhamos com o relacionamento de uma tabela com ela mesma, em que poderiam surgir nomes iguais para as colunas.

A operação 'agregação' permite que uma função seja calculada sobre o conjunto de valores retornados por uma consulta. Essas funções incluem média, soma, mínimo e máximo, dentre outras. A operação também permite que as agregações sejam feitas após agrupamento.

É importante destacar que na álgebra relacional, é possível alcançar um mesmo resultado por meio de consultas diferentes. Com isso exposto, podemos citar uma área de estudos em banco de dados que trata da questão da otimização de consultas, em que é importante a análise da equivalência entre consultas. Para que seja possível dizer que duas consultas são equivalentes, estas devem atender a critérios específicos, sendo necessário retornar o mesmo conjunto de resultados para qualquer conjunto de dados de entrada.

2.1.3 Linguagem de consulta estruturada (SQL)

Este segmento do trabalho dedica-se a explorar a linguagem SQL (Structured Query Language), fundamental no gerenciamento de bancos de dados. A SQL é categorizada em duas partes principais: a linguagem de definição de dados (DDL - Data Definition Language) e a linguagem de manipulação de dados (DML - Data Manipulation Language), cada uma com funções distintas e complementares.

A DDL é responsável pela criação e estruturação do esquema de um banco de dados. Ela define o esquema de cada relação, os tipos de dados para cada atributo, e estabelece restrições de integridade, índices, e parâmetros de segurança e autorização. Por outro lado, a DML é utilizada para realizar consultas e operações nos dados armazenados. As consultas em SQL são estruturadas em torno de três cláusulas fundamentais: 'SELECT', 'FROM' e 'WHERE'. A cláusula 'SELECT' especifica os campos a serem exibidos, 'FROM' indica as tabelas de onde os dados serão retirados, e 'WHERE' define as condições que os dados devem satisfazer.

Um aspecto interessante da SQL é que ela não remove duplicatas por padrão, devido à carga de processamento envolvida. Para contornar isso, a palavra-chave 'DISTINCT' pode ser usada após 'SELECT' para eliminar duplicatas, enquanto 'ALL' mantém as dupli-

catas na consulta. A cláusula 'DISTINCT' tem seu uso demonstrado no **Exemplo 1**. O exemplo também mostra como usar a cláusula 'ORDER BY' para ordenar os resultados de uma consulta. No caso, os filmes são ordenados pelo seu ano de forma ascendente.

Exemplo 1: SELECT DISTINCT(ano) FROM filme ORDER BY ano.

As funções agregadas, como 'MAX', 'MIN', 'SUM', 'AVG', e 'COUNT', são usadas para realizar cálculos sobre conjuntos de dados. A consulta do exemplo **Exemplo 2** usa 'MAX' para encontrar o maior salário.

Exemplo 2: SELECT MAX(salario), nome FROM pessoa.

As consultas em SQL podem variar de simples a complexas. As mais simples lidam com uma única tabela, enquanto as mais complexas podem envolver múltiplas tabelas e condições. Vemos no **Exemplo 3** uma consulta mais complexa do que as apresentadas anteriormente.

Exemplo 3: SELECT DISTINCT pessoa.nome AS 'Funcionário', pessoa.salario AS 'Salário' FROM pessoa JOIN cargo ON pessoa.fkIdCargo = cargo.idCargo WHERE cargo.nome LIKE 'Desenvolvedor Jr' AND pessoa.salario < 3000.

A operação de renomeação, indicada pela palavra-chave 'AS', permite alterar o nome de colunas ou tabelas nos resultados. No **Exemplo 3**, 'pessoa.nome' e 'pessoa.salario' são renomeados para 'Funcionário' e 'Salário', respectivamente.

Em SQL, a diferenciação entre maiúsculas e minúsculas (case-sensitive) é padrão, mas alguns sistemas de gerenciamento de banco de dados, como MySQL e SQL Server, não fazem essa distinção. O operador 'LIKE', usado no **Exemplo 3**, é útil para buscar padrões em strings, com os caracteres especiais '%' e '_' representando qualquer substring e qualquer caractere, respectivamente.

Outra consulta diferente das vistas até o momento é exposta no **Exemplo 4** a seguir. Esse exemplo demonstra uma subconsulta aninhada. Subconsultas permitem a realização de consultas dentro de outras consultas, proporcionando uma poderosa ferramenta para comparações complexas e análises detalhadas. Operadores como 'IN', 'NOT IN', 'SOME', 'ALL', 'EXISTS', e 'NOT EXISTS' são usados em conjunto com subconsultas para refinar ainda mais os critérios de busca e comparação. De acordo com Ramakrishnan e Gehrke (2008), os operadores 'IN' e 'NOT IN' são equivalentes a '= ANY' e '< > ALL', respectivamente.

Exemplo 4: SELECT idPessoa, nome, salario FROM pessoa WHERE salario > SOME(SELECT salario FROM pessoa WHERE fkIdCargo = 2)

Em resumo, a linguagem SQL oferece uma gama diversificada de ferramentas e funcionalidades para a manipulação eficiente de dados em bancos de dados relacionais. Através de suas várias cláusulas, funções e operadores, a SQL permite a realização de consultas simples e complexas, atendendo às necessidades variadas de usuários e aplicações.

2.2 ANÁLISE SINTÁTICA - PARSING

Neste subcapítulo será abordado o tema de análise sintática (parsing), que será importante para o desenvolvimento do presente trabalho. Para isso, será utilizado o livro Compiladores (BARBOSA et al., 2021).

Para chegar ao tema de análise sintática, é importante falar sobre compiladores e suas estruturas. Um compilador é um componente que faz parte de uma cadeia de ferramentas de programas que são utilizados para a criação de executáveis a partir do código-fonte.

O processo de compilação é composto por diversas etapas sequenciais, passando pelos seguintes módulos de um compilador: analisador léxico, analisador sintático, analisador semântico, gerador de código intermediário, otimizador de código independente de máquina, gerador de código e otimizador de código dependente de máquina.

O processo de compilação é iniciado pelo analisador léxico, que é responsável por dividir o código de entrada em partes chamadas de tokens (símbolos léxicos) para que a etapa da análise sintática seja realizada em seguida. A análise léxica realiza a remoção de espaços em branco, espaços vazios e comentários, além de fazer o agrupamento de caracteres em tokens, que são um par formado por nome e valor do atributo.

A análise sintática, também chamada de parsing, se refere à análise de uma sentença para obtenção de sua estrutura sintática, que é armazenada em uma árvore conhecida como árvore sintática. Para realizar essa análise, verifica-se a partir de uma sentença e de uma dada gramática, se esta sentença pertence à linguagem gerada pela gramática. Para sentenças válidas, o parser produz a árvore de derivação, caso contrário, é emitido um erro.

As demais etapas do processo de compilação não são relevantes ao presente trabalho, pois é a árvore sintática gerada por um parser que será utilizada para extrair informações de consultas SQL para posterior geração de banco de dados.

2.3 TESTES DE SOFTWARE

Neste subcapítulo, será abordado o tema de testes de software. Para isso, será utilizado o livro Testes de software e gerência de configuração, (GONÇALVEZ et al., 2019).

Ao se falar do assunto desenvolvimento de software, uma importante etapa do desenvolvimento se concentra nos testes para garantir que a funcionalidade esteja de acordo com o esperado. O teste de software tem como objetivo encontrar erros e produzir softwares com a maior qualidade possível, tornando-os seguros e confiáveis para utilização.

Na área da engenharia de software, o teste de software é considerado como uma atividade de grande importância e é ideal que seja realizada ao longo de todo o ciclo de desenvolvimento, para que sejam antecipadas as descobertas de problemas. Quanto mais tarde se descobre um erro no software, maior pode ser a demanda de tempo ou custos para realizar uma correção.

No contexto deste trabalho, os testes são indispensáveis para validar a correta extração de informações de consultas SQL e a subsequente geração de um esquema de banco de dados relacional. Ao analisar consultas SQL e usar um parser para extrair informações, é vital garantir que o esquema de banco de dados gerado seja preciso e refletivo da consulta original. Por exemplo, se uma consulta SQL especifica uma relação entre duas tabelas, o esquema gerado deve refletir essa relação corretamente. Testes automatizados podem ajudar a validar essa e outras funcionalidades, garantindo que o software funcione conforme o esperado.

Para este projeto, os casos de teste podem ser criados considerando diferentes cenários, como: consultas SQL básicas que envolvam seleções simples e verificações de campos específico; consultas complexas com joins, subconsultas, funções agregadas, entre outras características avançadas da SQL; testes com diferentes estruturas de tabelas, como tabelas com múltiplas chaves estrangeiras, tabelas sem chaves, entre outros.

Os testes unitários também podem ajudar na etapa de criação das instâncias, levando em consideração alguma regra relevante que tenha sido definida nas consultas de entrada. Por exemplo, ao filtrar pessoas com salário maior ou igual a algum valor, é esperado que o banco de dados possua instâncias com valores de salário menores do que o valor definidos, iguais ao valor definido e maiores do que o valor definido.

Os testes são uma parte integrante deste projeto, garantindo que as consultas SQL sejam analisadas corretamente, que o esquema de banco de dados gerado seja um reflexo preciso da consulta original e que as instâncias geradas sejam coerentes com os filtros aplicados nas consultas.

2.4 TRABALHOS RELACIONADOS

Neste subcapítulo, serão apresentados trabalhos e ferramentas que possuem alguma relação com o objetivo do presente trabalho, abordando temas como geração de banco de dados para testes e geração de dados sintéticos. Não foi encontrada uma aplicação real e acessível que seja semelhante à proposta do presente trabalho.

2.4.1 Reverse Query Processing

O artigo "Reverse Query Processing", de BINNIG, KOSSMANN e LO (2011), propõe uma técnica inovadora chamada Processamento de Consulta Reversa (RQP). Esta técnica é projetada para gerar instâncias de banco de dados que podem produzir resultados específicos para uma dada consulta, sendo útil para testar aplicações de banco de dados e depurar consultas SQL.

O RQP tem várias aplicações práticas, como a geração de bancos de dados de teste baseados em consultas individuais de um programa de aplicação para realizar testes funcionais. Ele pode gerar diferentes bancos de dados para testes, considerando diferentes resultados para uma consulta. Além disso, pode ser usado para estudar a performance de um Sistema de Gerenciamento de Banco de Dados (SGBD), permitindo que os usuários especifiquem o tamanho dos resultados da consulta e a seletividade de cada predicado.

O RQP opera com base em uma Álgebra Relacional Reversa (RRA), onde cada operador da álgebra relacional tem um operador correspondente na RRA. O RQP processa consultas de forma reversa, começando pela análise da consulta resultante e empurrando cada tupla para as folhas (ou seja, as tabelas base) da árvore de consulta. O processo de RQP envolve quatro etapas principais: análise de consulta, anotação de consulta de baixo para cima, otimização de consulta e instanciação de dados de cima para baixo.

A arquitetura do RQP processa consultas de maneira reversa em quatro etapas principais, descritas anteriormente. Inicialmente, um parser traduz a consulta SQL em uma árvore de consulta que consiste em operadores da álgebra relacional. Este parsing é realizado da mesma forma que em um processador SQL tradicional. O que torna o RQP especial é que esta árvore de consulta é traduzida em uma árvore de consulta reversa. Em seguida, ocorre a fase da anotação de consulta de baixo para cima, que anota cada operador de uma árvore de consulta reversa com um esquema de entrada e um esquema de saída. Após essa fase, temos o otimizador de consulta, que transforma a árvore de consulta reversa em uma equivalente que satisfaça um certo objetivo de otimização. E por fim, a etapa de instanciação de dados de cima para baixo, em que a árvore de consulta reversa anotada é interpretada usando a Tabela R (tabela de resultados que se deseja obter de uma consulta SQL dada) como entrada. A instanciação de dados é realizada

de cima para baixo, começando com uma tabela de resultados e produzindo relações de saída que respeitam as restrições de integridade do esquema do banco de dados.

O artigo também exemplifica o processo de RQP usando um esquema de banco de dados e uma consulta SQL, demonstrando como a árvore de álgebra relacional reversa é construída e como os dados fluem durante a execução. A RRA tem leis de associatividade, comutatividade, etc., semelhantes à álgebra relacional tradicional, e a implementação física da RRA é independente da aplicação, formando a base para a geração de bancos de dados de teste para testes funcionais.

2.4.2 QAGen: Generating query-aware test databases

O artigo "QAGen: Generating Query-Aware Test Databases", de Binnig et al. (2011), propõe uma metodologia inovadora para testar sistemas de gerenciamento de banco de dados (SGBD). O foco é na geração de bancos de dados de teste que são conscientes das consultas que serão executadas sobre eles, ao contrário das abordagens tradicionais que geram bancos de dados de teste independentemente das consultas.

A ideia central é criar um gerador de banco de dados de teste chamado QAGen, que, além do esquema do banco de dados e um conjunto de restrições básicas definidas nas tabelas base, leva a consulta e um conjunto de restrições definidas na consulta como entrada, e gera um banco de dados de teste consciente da consulta como saída. O banco de dados gerado garante que a consulta de teste pode obter os resultados de consulta desejados conforme definido no caso de teste, facilitando diversas tarefas de teste de SGBD, como testar gerenciadores de memória e estimar componentes de cardinalidade de otimizadores de consulta.

O processo de geração de dados do QAGen é dividido em duas fases: a fase de processamento de consulta simbólica (SQP) e a fase de instanciação de dados. Na fase SQP, o QAGen integra o conceito de execução simbólica do campo da engenharia de software ao processamento de consulta tradicional, criando um banco de dados simbólico que contém um conjunto de símbolos ao invés de dados concretos. Este banco de dados simbólico é então processado de maneira semelhante a uma consulta tradicional, mas manipulando dados simbólicos em vez de dados concretos. Na fase de instanciação de dados, os símbolos nas tuplas do banco de dados simbólico são instanciados por um resolvedor de restrições, e as tuplas instanciadas são inseridas no banco de dados de teste final.

O QAGen permite que os usuários definam diferentes casos de teste para a mesma consulta, fornecendo uma maneira de especificar distribuições e outras restrições diretamente nos operadores e tabelas base da consulta, tornando o banco de dados de teste gerado muito mais relevante e útil para o teste de DBMS.

2.4.3 IBM DB2 Test Database Generator

De acordo com IBM (2023), o IBM Db2 Test Database Generator é uma ferramenta da IBM projetada para ajudar na criação de conjuntos de dados de teste para o IBM Db2, um sistema de gerenciamento de banco de dados relacional. Ele permite que os desenvolvedores criem facilmente dados de teste realistas para testar aplicativos e consultas sem afetar o ambiente de produção.

O funcionamento da ferramenta pode ser dividido em três etapas, sendo elas: configuração de dados, geração de dados e exportação de dados. Na etapa de configuração de dados, a ferramenta permite que seja definido como os dados de teste serão gerados. É possível especificar o tamanho do banco de dados, as tabelas que devem ser incluídas, os tipos de dados a serem usados e até mesmo regras de geração de dados específicas. A etapa de geração de dados toma como base essas configurações que foram definidas, criando dados fictícios que se assemelham a dados reais de um banco de dados de produção. Ele pode preencher tabelas com informações realistas, como nomes, datas, números, entre outros. Na etapa de exportação de dados, é possível exportar os dados gerados para o Db2 ou para outros formatos, como arquivos CSV, para uso em testes.

O Db2 Test Database Generator não é uma ferramenta gratuita. Geralmente, faz parte da suíte de produtos Db2 da IBM, e o custo pode variar dependendo do acordo de licenciamento e da versão do Db2 utilizada. A IBM oferece diferentes opções de licenciamento e preços para suas ferramentas de banco de dados, e o custo específico dependerá de vários fatores, como o tamanho do ambiente e os recursos necessários.

2.4.4 DTM Data Generator

De acordo com DTM soft (2023), o DTM Data Generator é uma ferramenta que permite criar conjuntos de dados fictícios para testes e desenvolvimento de bancos de dados. Essa ferramenta é projetada para ajudar os desenvolvedores a gerar dados de teste realistas que podem ser usados em ambientes de teste e desenvolvimento, sem a necessidade de dados sensíveis ou reais. A ferramenta possui conexão com os bancos de dados mais populares como SQL Server, MySQL, Firebird, IBM DB2, PostgreSQL e Oracle Database.

O funcionamento da ferramenta pode ser dividido em quatro etapas, sendo elas: definição de tabelas, configuração de dados, geração de dados e exportação de dados. Na etapa de definição de tabelas, são definidas as tabelas do banco de dados para as quais serão gerados gerar dados de teste. Isso envolve especificar a estrutura das tabelas, incluindo os nomes das colunas, os tipos de dados e as restrições. Na etapa de configuração de dados, são configuradas as regras para gerar os dados. O DTM Data Generator permite que especificar como os dados serão gerados. É possível definir faixas

de valores para colunas numéricas, listas de valores para colunas de texto, geração de datas aleatórias, entre outros. A ferramenta oferece flexibilidade para criar dados que se adequem às necessidades de teste. Na etapa de geração de dados, a ferramenta preenche as tabelas com dados fictícios com base nas configurações definidas. Na etapa de exportação de dados, os dados gerados podem ser exportados para um banco de dados de teste, salvos em arquivos de texto ou em outros formatos que possam ser usados em testes.

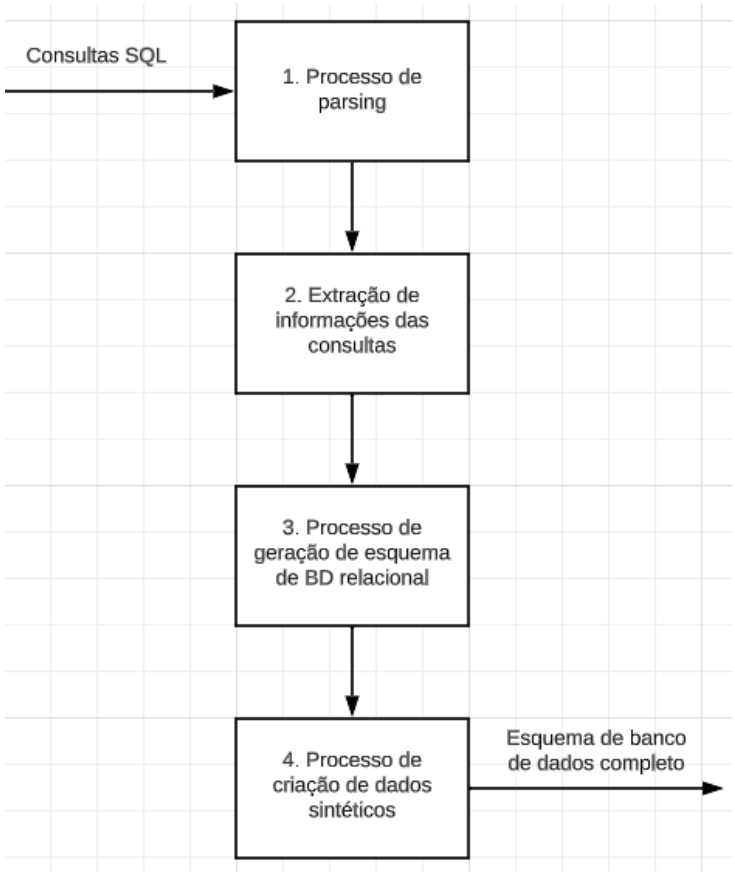
O DTM Data Generator não é uma ferramenta gratuita. A DTM soft oferece várias versões do produto com diferentes recursos e preços. O custo específico depende da edição que você escolhida e de qualquer licença adicional que for necessária, como licenças de usuário ou licenças para servidores. Apesar disso, é possível obter uma versão de demonstração gratuita, que possui limitações em relação às versões comerciais. A versão de demonstração limita o número de tuplas geradas para cada tabela a cinquenta, o número de regras para quinze por projeto e o número de tabelas para cinco.

3 DESCRIÇÃO DA ARQUITETURA PROPOSTA

O objetivo deste trabalho é criar uma aplicação simples e acessível, utilizando a linguagem de programação Java, para permitir a geração de bancos de dados completos para testes, usando consultas SQL fornecidas. Ou seja, dada uma consulta, pretende-de que, a partir dela, sejam gerados dados de teste e um esquema para que os dados sejam armazenados. Com os dados gerados, um outro módulo (não desenvolvido neste trabalho) poderia realizar comparações entre resultados de consultas. Supondo que haja consultas usadas como gabarito, a comparação entre os resultados pode servir de guia tanto para alunos (que podem corrigir as consultas à medida que elas vão sendo formuladas) tanto por professores (que ganham um assistente para a correção dos trabalhos).

Para viabilizar as etapas do sistema de geração de dados, foram elaborados diversos módulos, cujo funcionamento é ilustrado na Figura 4. Essas etapas serão detalhadas a seguir.

Figura 4 – Arquitetura da implementação do projeto



Fonte: Próprio autor.

3.1 PROCESSO DE PARSING

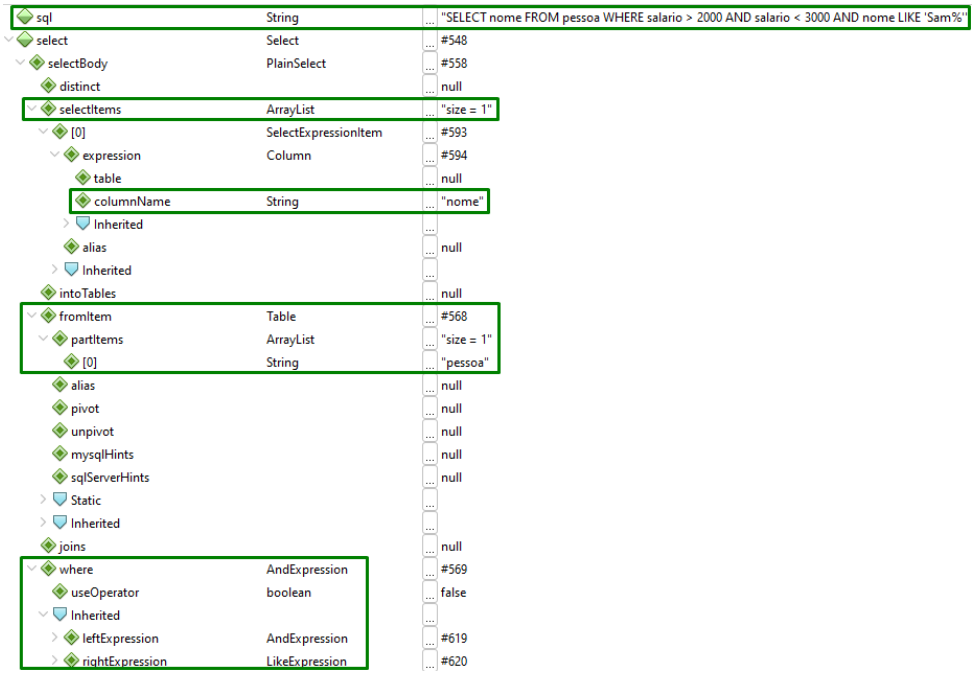
O primeiro módulo é o processo de parsing, realizado após a obtenção de consultas SQL inseridas pelo usuário da ferramenta. O objetivo é utilizar um parser SQL para a linguagem de programação Java para transformar as consultas SQL em uma estrutura de árvore, da qual seja possível obter as informações relevantes para realizar os processos subsequentes.

Para o desenvolvimento desse primeiro módulo, foi feita uma busca por bibliotecas existentes, capazes de desempenhar a função de parsing da linguagem SQL. Para isso, foi escolhido o JSQLParser 4.7, que é um analisador de instruções SQL construído a partir de JavaCC. Ele é capaz de traduzir expressões SQL em uma hierarquia percorrível de classes Java.

O exemplo da Figura 5 mostra como uma árvore de objetos Java é gerada após a realização do processo de parsing por meio de funcionalidade do JSQLParser. A biblioteca também possui implementações que utilizam o padrão 'visitor' para percorrer essa árvore.

Na imagem, foram destacadas algumas partes relevantes para a extração de informações que se dará a seguir. É possível ver a indicação de que há um item na cláusula 'SELECT', que é o campo 'nome'. Também há a indicação de um item na cláusula 'WHERE' que é a tabela 'pessoa' e a indicação de que a cláusula 'WHERE' possui um operador 'AND' com um filtro textual utilizando 'LIKE' em sua expressão da direita e outro operador 'AND' aninhado em sua expressão da esquerda.

Figura 5 – Exemplo de árvore de objetos Java após parsing



Fonte: Próprio autor.

3.2 EXTRAÇÃO DE INFORMAÇÕES DAS CONSULTAS

O segundo módulo é responsável por fazer a extração de informações das consultas SQL após o processo de parsing. A versão atual do extrator processa apenas o conteúdo presente nas cláusulas SELECT, FROM e WHERE, conseguindo recuperar informações referentes às tabelas, atributos e filtros aplicados em atributos. Os dados extraídos são usados posteriormente para gerar o esquema de banco de dados de teste e dados sintéticos para povoá-lo.

Foram utilizadas implementações do padrão visitor providas pela biblioteca do JSQLParser para percorrer a árvore de objetos gerada após o processo de parsing. O módulo desenvolvido utiliza extensivamente a estrutura de 'Map' em Java para organizar e acessar os dados. Além de outras estruturas como listas, sets e mapas, a seguir são detalhadas as três principais estruturas de dados utilizadas:

- Mapa de Campos por Tabela (fieldsByTable): armazena um conjunto (Set) de campos associados a cada tabela. É do tipo Map<String, Set<String> >.
- Mapa de Campo para Tipo por Tabela (mapTableToFieldToType): uma estrutura de mapa aninhado que relaciona cada tabela a outro mapa, onde a chave é o campo e o valor é o tipo desse campo. É do tipo Map<String, Map<String, String> >.
- Mapa de Campo para Valores por Tabela (mapTableToFieldToValues): é semelhante ao mapa anterior, mas, neste caso, associa-se cada campo a uma lista de valores que apareceram em filtros para o respectivo campo. É do tipo Map<String, Map<String, List<String> > >.

Como já destacado, apenas o conteúdo presente nas cláusulas SELECT, FROM e WHERE foi extraído. Mesmo dentro desse contexto limitado, surgem diversas situações que devem ser tratadas. No que se segue, algumas das situações possíveis são apresentadas.

Em primeiro lugar, o extrator consegue identificar corretamente as **tabelas** a partir das informações presentes na cláusula 'FROM'. Quanto aos **atributos**, o extrator os identifica a partir de informações presentes nas cláusulas 'SELECT' e 'WHERE'.

Por exemplo, na consulta apresentada no **Exemplo 5**, o extrator identifica que há uma tabela 'pessoa' contendo contendo dois atributos: 'nome' e 'salario'. Nesse caso, como há apenas uma tabela, a identificação ocorreria mesmo que um 'alias' não fosse usado.

Exemplo 5: SELECT pe.nome, pe.salario FROM pessoa pe WHERE pe.salario > 2000

Cabe salientar que, caso haja mais de uma tabela presente nessa cláusula, os atributos devem ser necessariamente prefixados com o nome da tabela ou com o 'alias' da sua respectiva tabela, para que o extrator consiga fazer a associação correta entre atributos e tabelas.

Por exemplo, na consulta apresentada no **Exemplo 6**, o extrator identifica que há uma tabela 'pessoa', contendo os atributos 'nome' e 'salario', e outra tabela 'categ' contendo os atributo 'nomeCateg'.

Exemplo 6: SELECT pe.nome, pe.salario, ca.nomeCateg FROM pessoa pe JOIN categ ca ON pe.idCateg = ca.idCateg

Para finalizar, o extrator utiliza heurísticas para definir um **tipo de dados** para cada coluna. No momento, três tipos são suportados: texto, data e numérico. Caso a coluna possua filtros, analisa-se a presença de aspas envolvendo o valor. Por exemplo, dado o filtro (situacaoProj='novo'), o atributo situacaoProj é tratado como texto. A exceção ocorre caso o valor esteja expresso no formato de data, como em (dataIni = '20200101'). Nesse caso, o tipo é tratado como data. Caso o conteúdo não esteja envolto por aspas, como em (salario = 2000), o tipo é tratado como numérico. Caso algum atributo não possua filtros, ele é considerado um tipo textual.

3.2.1 Limitações

A seguir, serão descritas alguns casos em que o extrator falha em gerar o esquema mais adequado. Todos os casos apresentados são situações que podem ser trabalhadas em trabalhos futuros.

Em primeiro lugar, no caso do **Exemplo 6**, o correto seria que as informações presentes nesse critério de junção levassem à criação de novos atributos (idCateg em pessoa e idCateg em categ). Contudo, as informações presentes na cláusula 'ON' não são consideradas. O mesmo problema ocorre caso fosse usada a cláusula 'USING' para especificar os critérios de junção.

De modo geral, os critérios de junção só levam à geração de atributos caso eles sejam especificados dentro da cláusula 'WHERE', como em 'WHERE pe.idCateg = ca.idCateg'. Mesmo assim, esses atributos não são associados a nenhuma regra de **integridade referencial**. Ou seja, os atributos não têm nenhuma vinculação com chaves primárias/chaves estrangeiras. Esse recurso não foi implementado devido à dificuldade para reconhecer qual tabela deve conter a chave primária e qual deve conter a chave estrangeira.

Além disso, não se pode assumir que existe de fato uma conexão de chave primária/estrangeira verificando unicamente a presença de um critério de junção. Por exemplo,

analise a consulta apresentada no **Exemplo 7**:

Exemplo 7: SELECT pessoa.nome FROM pessoa JOIN proj ON pessoa.salario = proj.custo

Nesse caso, os atributos 'custo' e 'salario' são corretamente gerados. No entanto, 'custo' não é chave estrangeira para pessoa, tampouco 'salario' é chave estrangeira para 'projeto'. O que ocorre é que a consulta usou um critério de junção para expressar uma solicitação de relação não referencial, relativa à existência de pessoas cujo salário seja equivalente a algum custo de projeto.

O problema de reconhecer se existe uma conexão, e caso exista, identificar qual coluna é chave primária, envolve o uso de heurísticas. Por exemplo, o nome dos atributos pode ser um bom indicativo da sua finalidade. No entanto, dada a complexidade do problema, sua implementação foi deixada para trabalhos futuros.

Outra situação problemática envolve o uso de junções naturais, como no **Exemplo 8**. Nesse caso, não é possível derivar os nomes dos atributos a partir da junção, pois a junção natural, por definição, não informa nomes de atributos. Nesse caso, a solução mais adequada seria usar algum esquema pre-existente para verificar quais colunas nas duas tabelas envolvidas possuem o mesmo nome. Essa informação poderia ser usada pela etapa de geração de dados, para garantir que sejam gerados registros adequados, como será discutido na seção 3.4.

Exemplo 8: SELECT pessoa.nome FROM pessoa NATURAL JOIN proj

As mesmas dificuldades presentes em consultas envolvendo junções aparecem em **consultas aninhadas**. O aninhamento pode ser usado como uma forma alternativa de representar conexões entre chave primária e chave estrangeira. Para exemplificar, considere o caso apresentado no **Exemplo 9**:

Exemplo 9: SELECT pessoa.nome FROM pessoa WHERE salario IN (SELECT custo FROM proj)

O extrator consegue identificar as tabelas e colunas. Por outro lado, não está preparado para reconhecer a existência de uma regra de integridade referencial. Apesar dessa limitação, é importante ressaltar que, para o objetivo em vista, a existência dessa conexão é irrelevante. O que realmente importa é que existam valores adequados gerados para as colunas 'salario' e 'custo', como será discutido na seção 3.4.

3.3 GERAÇÃO DE ESQUEMA DE BANCO DE DADOS RELACIONAL

A terceira etapa da ferramenta é o processo de criação de um esquema de banco de dados que represente as características das consultas extraídas. Para isso, são utilizadas as informações obtidas no módulo anterior, que foram armazenadas nas estruturas de dados mencionados.

O processo gera um script em linguagem SQL, capaz de ser executado para gerar um banco de dados relacional, com suas respectivas tabelas e atributos com os tipos adequados.

O script é organizado com a criação de um banco de dados novo denominado 'mydb'. Em seguida, são geradas as tabelas, cada uma com uma chave primária simples, cujo nome usa a string 'id' sufixada com o nome da respectiva tabela. Por exemplo, para a tabela pessoa, seria gerada uma chave primária chamada 'idPessoa'. Como a chave é auto-incremental, ela não precisa ser especificada no momento em que forem inseridos dados na tabela.

Além da chave primária, são gerados os demais campos que foram encontrados durante a etapa de extração de informações da consulta. Para a definição dos tipos de dados, usa-se o tipo INT para dados numéricos, VARCHAR(100) para os dados textuais e DATE para o tipo data. O tipo VARCHAR(100) foi escolhido por possibilitar o armazenamento de um número razoável de caracteres, considerado suficiente para uso em filtros textuais.

O gerador de esquema não leva em consideração a existência de um esquema pre-existente. Ou seja, caso já exista um esquema, ele será sobrescrito. Dependendo do contexto, pode ser interessante preservar o esquema original, apenas complementando-o com devidos ajustes estruturais (ex. adição de novos atributos a uma tabela já existente). A mescla entre um esquema original e o esquema gerado é tema para trabalhos futuros.

A etapa de geração de esquema é realizada em conjunto com a geração de dados, que será explicada na próxima seção, podendo ser considerados juntos como um módulo único, pois a cada esquema de tabela que é gerado, também são gerados os dados para povoar essa tabela, quando há alguma informação advinda de filtros nas consultas.

3.4 CRIAÇÃO DE DADOS SINTÉTICOS

A quarta etapa envolve a criação de dados sintéticos com base no esquema criado na etapa anterior e das informações extraídas após o parsing. Essa etapa ocorre como descrito a seguir:

1. Recuperação dos nomes das tabelas usadas na consulta (ex. 'pessoa').

2. Recuperação dos nomes de atributos usados na consulta, associados às tabelas recuperadas anteriormente (ex. ’nome’ e ’salario’).

3. Recuperação dos filtros existentes na consulta (ex. salario > 2000).

4. Geração de uma lista de valores para cada ocorrência de um atributo em um filtro.

5. Combinação da lista de valores para a formação dos registros.

Para recuperar as tabelas, é usado um método ’getter’. Já o mapa ’fieldsByTable’ permite recuperar os atributos de cada tabela que aparecem na consulta. Para a recuperação dos filtros é empregado o mapa ’mapTableToFieldToValuesPara’, que usa o nome da tabela e do campo como chaves da estrutura.

A próxima etapa envolve a geração de uma lista de valores que devem aparecer nos dados gerados referentes a cada filtro presente na consulta. Essa lista é armazenada em um mapa, em que a chave é o campo ao qual os dados se referem. A lista de valores gerados tem relação com o valor e o tipo de comparação usados em cada filtro.

O quadro a seguir descreve as situações tratadas neste trabalho. Para **tipos numéricos**, é gerado um valor imediatamente inferior, igual e imediatamente superior ao valor presente no filtro. Por exemplo, o filtro ’salario = 3000’ levaria aos valores 2999, 3000, 3001. **Filtros de data** seguem o mesmo princípio. Nesse caso, os valores imediatamente inferior e superior levam em consideração o componente no dia. Por exemplo, o filtro ’data > ’20231105” levaria aos valores 2023-11-04, 2023-11-05 e 2023-11-06. Já os **filtros de texto** no seu caso mais simples, geram o próprio valor e ele mesmo acrescido de um caractere ao final. Por exemplo, o filtro ’nome LIKE ’Samuel” levaria os valores Samuel e SamuelA.

Quadro 1 – Variações de dados para atributos de acordo com o seu tipo

Tipo	Variações
Numérico	3 (<, =, >)
Data	3 (<, =, >)
Texto	2 ou mais

Fonte: Próprio autor

O conjunto de regras definido no quadro acima é suficiente para tratar os casos mais básicos de filtro e garantir que variações no filtro levem à resultados diferentes. Por exemplo, mudando o tipo de comparação de ’salario > 2000’ para ’salario >= 2000’, o resultado da consulta muda, pois passaria a contemplar o registro que contém o valor de salario = 2000.

A ultima etapa envolve a formação dos registros com base nos valores obtidos pela etapa anterior. Os registros são gerados por meio da combinação de todos os valores. Ou seja, o número de registros gerados para cada tabela é igual ao produto do número de

variações de valores de cada atributo. Por exemplo, se uma consulta possui dois filtros, com cada filtro gerando três valores, o número total de tuplas geradas seria nove.

Os valores gerados a partir dos filtros têm como objetivo gerar dados que levem a diferentes resultados de consultas caso haja modificações no filtro utilizado. Ou seja, o objetivo é garantir que seja possível verificar que o uso de filtros modificados implique em consultas não equivalentes (consultas que tragam resultados diferentes).

3.4.1 Limitações

A seguir, serão descritas alguns casos em que o gerador de dados falha em gerar os registros mais adequados. Todos os casos apresentados são situações que podem ser evoluídas em trabalhos futuros.

Em primeiro lugar, a versão atual do gerador de dados não leva em consideração o **tipo de comparação** utilizada em cada filtro('>', '<', '=', ...). Caso essa análise fosse feita, seria possível reduzir a quantidade de valores gerados, sem prejuízo ao principal objetivo, que é gerar um número suficiente de registros que consiga reconhecer diferenças em consultas.

Relacionado a isso, outro ponto que pode ser aprimorado é a análise dos **conectores** usados dentro da cláusula WHERE (OR, AND, NOT). Em consultas complexas, os conectores podem ser usados para criar **expressões booleanas** intrincadas, em que possivelmente pode-se trabalhar com um número de registros menor do que o que seria gerado pelo produto da quantidade de valores gerados para cada filtro.

Outra limitação é referente à relação entre as tabelas, que pode ocorrer de diversas formas (via cláusula 'WHERE', consultas aninhadas, cláusula de junção). Foi comentado que o extrator de esquema não reconhece conexões de chave estrangeira, e que a presença dessas chaves é irrelevante para o objetivo em vista. O que importa é que sejam gerados dados que satisfaçam os critérios de correspondência informados na consulta. No entanto, a geração dos dados não leva essas conexões em consideração.

Por exemplo, observe a consulta apresentada no **Exemplo 10**:

Exemplo 10: SELECT pessoa.nome FROM pessoa WHERE salario IN (SELECT custo FROM proj)

Considere também que a mesma requisição tenha sido expressa de forma equivocada, da forma apresentada no **Exemplo 11**. Para conseguir perceber diferenças entre as duas versões, é necessário gerar registros de pessoas e de projetos de modo que existam salários iguais à custos e salários diferentes.

Exemplo 11: SELECT pessoa.nome FROM pessoa WHERE salario = ALL (SELECT custo FROM proj)

O exemplo anterior apresenta uma das possíveis situações em que a conexão entre duas tabela deveria resultar na geração cuidadosa de registros, visando produzir registros em que essas relações estejam estabelecidas.

Os casos podem se tornar gradativamente mais complexos, à medida que forem adicionados filtros e forem usados recursos que demandem tratamentos especializados, como comparações com nulo ('IS NULL', 'IS NOT NULL'), negações ('NOT IN'), comparações com subconsulta ('IN', 'EXISTS', 'SOME', 'ALL') e agrupamentos ('GROUP BY', 'HAVING'). para citar alguns. Mesmo consultas envolve uma única tabela podem possuir um grau de complexidade alto e demandar soluções mais personalizadas.

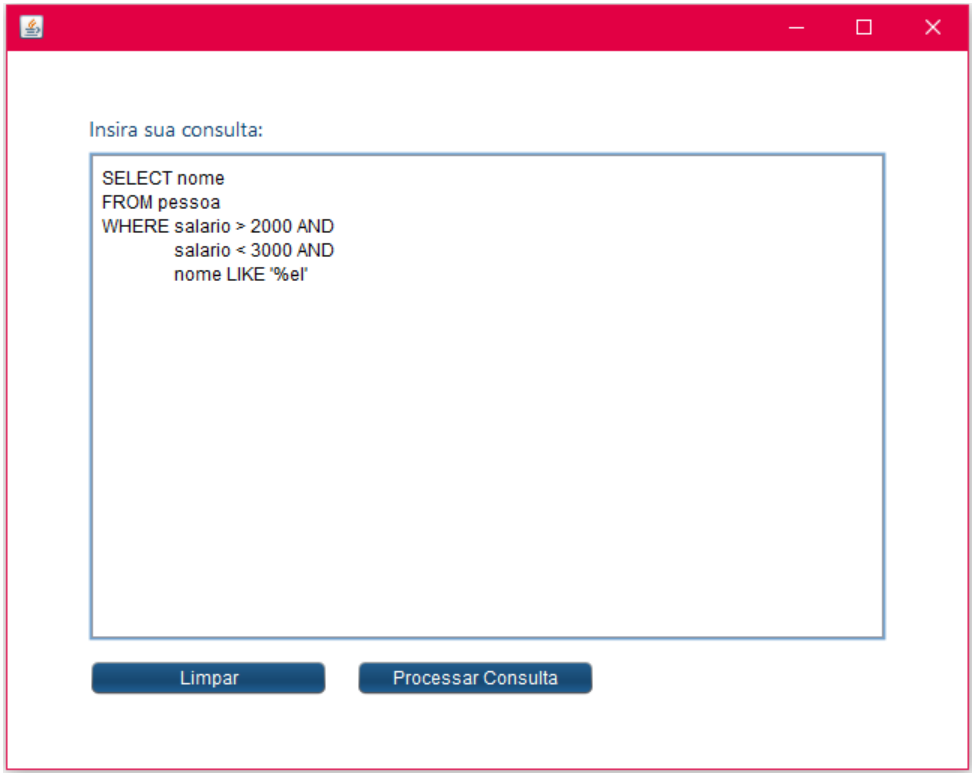
Ou seja, existe uma gama muito extensa de possibilidades, e não foi possível suportar à todas. Apesar da limitação imposta devido à complexidade do problema, a geração de dados se mostrou útil para um conjunto de consultas utilizadas em um cenário real, como será mencionado durante o capítulo 4.

3.5 INTERFACE GRÁFICA

Para facilitar a utilização por parte dos usuários, foi planejada uma interface gráfica simples e intuitiva, que possibilita a realização da atividade proposta pelo trabalho, que é a inserção de uma consulta SQL e a visualização de um esquema gerado pela ferramenta a partir dessa consulta. Para o desenvolvimento dessa interface gráfica, foi utilizada a biblioteca Java 'Swing' que provê diversos componentes de interface gráfica de usuário, como botões, caixas de seleção, campos de texto, barras de rolagem, tabelas, painéis, entre outros.

Com as telas da interface gráfica, também foram implementadas algumas formas de controle, com alertas para o usuário, em algumas situações. O usuário é avisado caso tente gerar um esquema sem informar uma consulta, caso a sua consulta tenha algum problema de sintaxe ou em casos em que foi inserida uma consulta que a ferramenta não consegue suportar, devido à sua limitação. A Figura 6 mostra a tela inicial do programa, na qual deve ser inserida a consulta a ser processada.

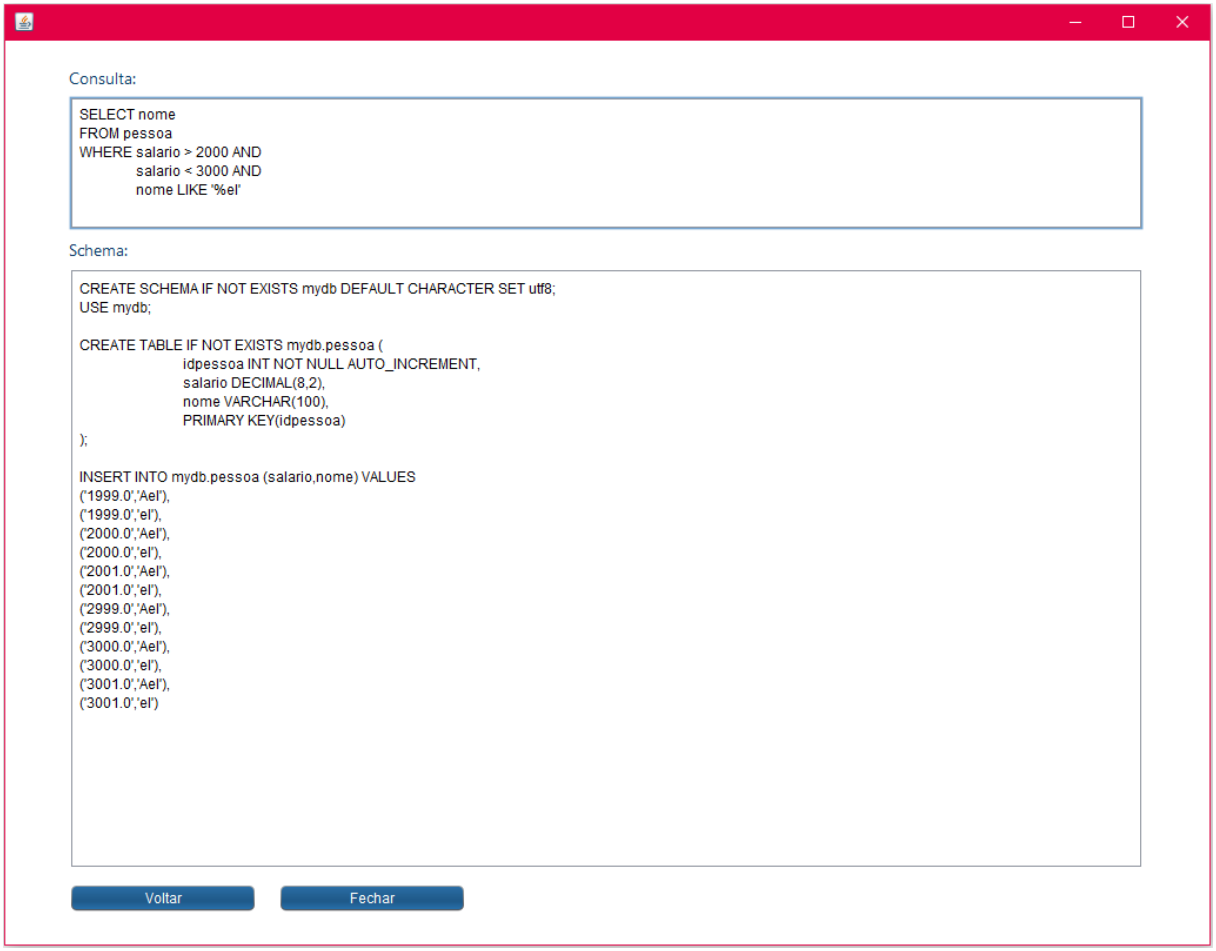
Figura 6 – Tela de inserção de consulta



Fonte: Próprio autor.

A Figura 7 mostra a tela de exibição de resultados, em que é exibida em forma de visualização a consulta inserida anteriormente pelo usuário e o esquema gerado pela ferramenta, em campo texto que possibilita alguma edição caso seja necessário.

Figura 7 – Tela de exibição após processamento da consulta



Fonte: Próprio autor.

4 EXPERIMENTOS E RESULTADOS

Este capítulo tem o objetivo de expor testes realizados utilizando a ferramenta desenvolvida, em diversos cenários, para apresentar os resultados obtidos e exemplificar o seu funcionamento nos casos propostos para abordagem.

4.1 CONSULTAS SIMPLES

Nessa seção, serão mostrados diferentes exemplos de uso da ferramenta, por meio da inserção de consultas simples, que são feitas sobre uma única tabela e contendo apenas um filtro específico dentre os diferentes filtros tratados pela ferramenta, para que fique claro e bem exemplificado o funcionamento isolado de cada tipo de filtro.

4.1.1 Consulta com filtro numérico

A primeira consulta, indicada no **Exemplo 12**, demonstra o funcionamento da ferramenta recebendo como entrada uma consulta simples na tabela 'pessoa' com filtro numérico no campo 'salario'.

Exemplo 12: SELECT nome, salario FROM pessoa WHERE salario > 2000

Como o filtro aplicado foi do tipo numérico, para a geração de dados sintéticos foram considerados três valores: um número abaixo, o próprio número e um número acima. Assim para o valor 2000, foram considerados os valores 1999, 2000 e 2001. Dessa forma, o resultado apresentado pela ferramenta é exibido na Figura 8.

Figura 8 – Resultado de consulta com filtro numérico

```
CREATE SCHEMA IF NOT EXISTS mydb DEFAULT CHARACTER SET utf8;
USE mydb;

CREATE TABLE IF NOT EXISTS mydb.pessoa (
                idpessoa INT NOT NULL AUTO_INCREMENT,
                salario DECIMAL(8,2),
                nome VARCHAR(100),
                PRIMARY KEY(idpessoa)
);

INSERT INTO mydb.pessoa (salario) VALUES
('1999.0'),
('2000.0'),
('2001.0')
```

Fonte: Próprio autor.

Na comparação de resultados, uma alteração do comparador de (salario > 2000) para (salario < 2000) já seria suficiente para produzir um resultado diferente do esperado.

4.1.2 Consulta com filtro de data

A segunda consulta, indicada no **Exemplo 13**, demonstra a ferramenta recebendo como entrada uma consulta simples na tabela ’pessoa’ com filtro de data no campo ’datanasc’.

Exemplo 13: SELECT nome, datanasc FROM pessoa WHERE datanasc >= ’20231105’

Como o filtro aplicado foi em um campo de data, para a geração de dados sintéticos foram considerados três valores: um dia antes da data do filtro, a própria data e um dia após a data do filtro. Assim para o valor ’20231105’, que representa o dia 05/11/2023, foram consideradas as datas ’2023-11-04’, ’2023-11-05’ e ’2023-11-06’, respeitando o padrão para inserção em banco de dados. Dessa forma, o resultado apresentado pela ferramenta é exibido na Figura 9.

Figura 9 – Resultado de consulta com filtro de data

```
CREATE SCHEMA IF NOT EXISTS mydb DEFAULT CHARACTER SET utf8;
USE mydb;

CREATE TABLE IF NOT EXISTS mydb.pessoa (
                idpessoa INT NOT NULL AUTO_INCREMENT,
                datanasc DATE,
                nome VARCHAR(100),
                PRIMARY KEY(idpessoa)
);

INSERT INTO mydb.pessoa (datanasc) VALUES
('2023-11-4'),
('2023-11-05'),
('2023-11-6')
```

Fonte: Próprio autor.

Na comparação de resultados, uma alteração do comparador de (datanasc >= ’20231105’) para (datanasc > ’20231104’) já seria suficiente para produzir um resultado diferente do esperado.

4.1.3 Consulta com filtro textual

A terceira consulta, indicada no **Exemplo 14**, demonstra a ferramenta recebendo como entrada uma consulta simples na tabela ’pessoa’ com filtro textual no campo ’nome’.

Exemplo 14: SELECT nome FROM pessoa WHERE nome LIKE ’%el’

Como o filtro aplicado foi em um campo textual e foi utilizado o caractere porcentagem (%), representando qualquer substring antes do texto especificado ’el’, para a geração de dados sintéticos foram considerados dois valores: um sendo a string especificada ’el’ e o outro sendo essa mesma string com uma substring antes. Como substring anterior ao texto foi utilizado o caractere ’A’, sendo gerada a string ’Ael’. Dessa forma, o resultado apresentado pela ferramenta é exibido na Figura 10.

Figura 10 – Resultado de consulta com filtro textual

```
CREATE SCHEMA IF NOT EXISTS mydb DEFAULT CHARACTER SET utf8;
USE mydb;

CREATE TABLE IF NOT EXISTS mydb.pessoa (
                idpessoa INT NOT NULL AUTO_INCREMENT,
                nome VARCHAR(100),
                PRIMARY KEY(idpessoa)
);

INSERT INTO mydb.pessoa (nome) VALUES
('Ael'),
('el')
```

Fonte: Próprio autor.

Na comparação de resultados, uma alteração do comparador de (nome LIKE ’%el’) para (nome LIKE ’el%’) já seria suficiente para produzir um resultado diferente do esperado.

4.2 CONSULTA CONTENDO DIVERSOS FILTROS

Nessa seção, será visualizado um teste realizado na ferramenta utilizando uma consulta mais completa, sobre uma única tabela e contendo diversos filtros de diferentes tipos. O objetivo é exemplificar o comportamento da ferramenta e como é realizada a geração de dados sintéticos quando vários filtros estão envolvidos.

A consulta usada está descrita no **Exemplo 15**. Foram aplicados dois filtros numéricos no campo ’salario’, um filtro de data no campo ’datanasc’ e um filtro textual no campo ’nome’.

Exemplo 15: SELECT nome FROM pessoa WHERE salario > 2000 AND salario < 3000 AND datanasc >= ’20231105’ AND nome LIKE ’Samuel’

A Figura 11 mostra o esquema de banco de dados gerado a partir da consulta inserida. Os dados sintéticos que foram gerados como resultado pela ferramenta foram ocultados da imagem devido ao seu volume.

Figura 11 – Esquema gerado para a consulta com diversos filtros

```
CREATE SCHEMA IF NOT EXISTS mydb DEFAULT CHARACTER SET utf8;
USE mydb;

CREATE TABLE IF NOT EXISTS mydb.pessoa (
                idpessoa INT NOT NULL AUTO_INCREMENT,
                datanasc DATE,
                salario DECIMAL(8,2),
                nome VARCHAR(100),
                PRIMARY KEY(idpessoa)
);
```

Fonte: Próprio autor.

Para o filtro com valor numérico 2000 são geradas três variações de valores para geração de dados sintéticos (1999, 2000 e 2001). Para o valor numérico 3000 é utilizada a mesma lógica (2999, 3000 e 3001). Para o filtro de data com valor ’20231105’ que representa o dia 05/11/2023, são geradas três variações de valores para a geração de dados (’2023-11-04’, ’2023-11-05’ e ’2023-11-06’). Para o filtro textual foram geradas duas variações de valores para a geração de dados (’Samuel’ e ’SamuelA’).

Como já mencionado, a geração de dados sintéticos gera todas as combinações possíveis para os dados encontrados. Considerando que para os campos ’salario’, ’datanasc’ e ’nome’ possuem seis, três e duas variações, respectivamente, a quantidade de registros gerados é dada pelo cálculo 6x3x2. Ou seja, para essa consulta são gerados 36 registros.

Com essa quantidade de registros, seria possível analisar diferenças entre a consulta apresentada como modelo e alguma outra consulta que tente representar o mesmo enunciado.

4.3 CONSULTA UTILIZADAS EM DISCIPLINA DE BANCO DE DADOS

Nesta seção serão vistos exemplos de consultas obtidas com um professor da disciplina de banco de dados da Universidade Federal de Santa Maria (UFSM). Foram fornecidas diversas listas, mas para realização dos testes, foi utilizada uma das listas com o total de 16 consultas variadas. Desse total, em 25% dos casos foi possível gerar tanto o esquema quanto os dados. Em outros 25% dos casos foi possível gerar apenas o esquema (isso ocorre quando a consulta não possui nenhum filtro, uma vez que são os filtros que regem à geração de valores para teste).

Consultas que geraram esquema e dados:

- SELECT titulo FROM filme WHERE titulo LIKE ’batman%’.
- SELECT titulo, ano FROM filme WHERE ano BETWEEN 2010 AND 2015

ORDER BY ano.

- SELECT titulo FROM filme WHERE titulo LIKE 'A%'.
- SELECT nome, altura FROM ator WHERE sexo = 'F'.

Consultas que geraram apenas esquema:

- SELECT DISTINCT(ano) FROM filme ORDER BY ano.
- SELECT COUNT(DISTINCT ano) FROM filme.
- SELECT ano, COUNT(*) AS 'Qtd Filmes' FROM filme GROUP BY ano.
- SELECT AVG(YEAR(CURDATE()) - YEAR(nasc)) FROM ator.

Nos casos restantes, que representam 50% das consultas, foram enfrentados problemas principalmente devido às consultas possuírem relacionamento entre tabelas.

Além disso, foi percebido que a ferramenta enfrenta problemas quando a consulta utiliza como nome do campo de junção o mesmo nome que a ferramente utiliza para criar a chave primária automática da tabela. Esse caso se repetiu em cinco das consultas que apresentaram problemas, como no exemplo **Exemplo 16**, indicado abaixo:

Exemplo 16: SELECT nome, titulo FROM filme, diretor WHERE filme.idDiretor = diretor.idDiretor ORDER BY nome

Vemos que na cláusula 'WHERE' está sendo referenciado um campo chamado 'idDiretor' da tabela 'diretor'. Esse nome de campo é o mesmo gerado automaticamente pela ferramenta como chave primária, seguindo o padrão id concatenado com o nome da tabela. Assim, como resultado final para essa consulta, é gerado um script que não pode ser executado com sucesso devido à duplicidade de campos em uma mesma tabela, o que pode ser resolvido removendo um dos campos do resultado.

Outro cenário em que a ferramenta encontrou dificuldades é quando acontecem operações na cláusula 'SELECT", como no caso da consulta no **Exemplo 17**.

Exemplo 17: SELECT titulo, ano, (bilheteria-custo)/1000000 AS lucro FROM filme ORDER BY lucro DESC

No exemplo, está sendo aplicada uma operação de divisão pelo valor 1000000, o que acarretou em um erro. Ao realizar um teste removendo essa divisão, a ferramenta conseguiu gerar o esquema do banco de dados.

Como visto, a ferramenta ainda precisa de aprimoramentos. No entanto, o fato de ter funcionado em metade dos casos testados mostra um potencial significativo, indicando que ela pode ser considerada uma solução viável para o problema, desde que haja melhorias adicionais. Este resultado positivo sugere que a ferramenta possui bases promissoras para oferecer uma solução adequada.

5 CONCLUSÃO

O presente trabalho teve como objetivo principal o desenvolvimento de um dos módulos de uma ferramenta educacional inovadora, implementada na linguagem de programação Java, para auxiliar no ensino e aprendizado da disciplina de Banco de Dados, com um enfoque particular na escrita e análise de consultas SQL. O módulo referido foi projetado para gerar automaticamente um banco de dados relacional a partir de consultas SQL fornecidas, e complementar este processo com a criação de dados sintéticos.

A metodologia adotada neste projeto envolveu várias etapas cruciais, começando com o desenvolvimento de módulos específicos para o parsing de consultas SQL. O uso do JSQLParser para este fim provou ser uma escolha acertada, permitindo a transformação eficiente das consultas em uma estrutura de árvore, facilitando assim a extração de informações relevantes. Posteriormente, essas informações foram utilizadas para gerar esquemas de banco de dados e produzir dados sintéticos, elementos fundamentais para o teste e a validação das consultas SQL em um ambiente controlado, porém realista.

Os testes realizados com a ferramenta abrangeram uma ampla gama de cenários, desde consultas simples, envolvendo uma única tabela e um tipo de filtro, até consultas mais complexas com múltiplos filtros. Estes testes foram essenciais para demonstrar a versatilidade e a eficácia da ferramenta em diferentes contextos de uso. Eles confirmaram o potencial que a ferramenta apresenta para facilitar a compreensão das consultas SQL por parte dos estudantes e de oferecer aos professores um recurso valioso para a avaliação e correção de trabalhos relacionados a esta disciplina.

Durante o desenvolvimento e os testes da ferramenta, algumas limitações foram identificadas, especialmente em relação ao tratamento de consultas mais complexas que envolvem subconsultas e múltiplas tabelas. Essas limitações não diminuem o valor da ferramenta, mas destacam áreas que podem ser aprimoradas em trabalhos futuros. A identificação dessas limitações é um passo crucial para o desenvolvimento contínuo da ferramenta, sugerindo a necessidade de evolução e adaptação para abranger uma gama ainda mais ampla de cenários de consulta SQL.

Além disso, este trabalho contribuiu para o campo da educação em Ciência da Computação, fornecendo uma ferramenta prática que pode vir a ser utilizada em diferentes instituições de ensino para colaborar com o processo de aprendizagem em Banco de Dados. A ferramenta desenvolvida tem o potencial de auxiliar os estudantes na interação com os conceitos de Banco de Dados, tornando o aprendizado mais interativo e prático.

Em conclusão, este trabalho alcançou seus objetivos propostos e estabelece um ponto de partida para futuras pesquisas e desenvolvimentos na área de ferramentas educacionais para Banco de Dados, abrindo novos caminhos para a inovação no ensino de Ciência da Computação.

REFERÊNCIAS

BARBOSA, C. d. S. et al. **Compiladores**. Grupo A, 2021. ISBN 9786556902906. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9786556902906/>.

BINNIG, C.; KOSSMANN, D.; LO, E. Reverse query processing. **Technical Report** / **ETH Zurich, Department of Computer Science**, v. 515, p. 48, 2011.

BINNIG, C. et al. Qagen: Generating query-aware test databases. **Technical Report** / **ETH Zurich, Department of Computer Science**, v. 555, p. 41, 2011.

DATE, C. **Introdução a sistemas de bancos de dados**. 8. ed. Elsevier Editora, 2004. ISBN 9788535212730. Disponível em: <https://books.google.com.br-/books?id=xBeO9LSIK7UC>.

DTM soft. **Test Data Generator Overview**. DTM soft, 2023. Acesso em 04 dez. 2023. Disponível em: <https://sqledit.com/dg/>.

GONÇALVEZ, P. F. et al. **Testes de software e gerência de configuração**. Grupo A, 2019. ISBN 9788595029361. Disponível em: <https://integrada.minhabiblioteca.com.br/reader-/books/9788595029361/>.

IBM. **IBM Documentation**. IBM, 2023. Acesso em 04 dez. 2023. Disponível em: <https:/-/www.ibm.com/docs/en>.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistemas de gerenciamento de banco de dados - 3.ed.** McGraw Hill Brasil, 2008. ISBN 9788563308771. Disponível em: <https://books-.google.com.br/books?id=COUJpkH5v38C>.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 7. ed. Rio de Janeiro: GEN LTC, 2020. ISBN 9788595157330.