Universidade Estadual de Maringá

Centro de Tecnologia - Departamento de Informática

Curso de Bacharelado em Informática

Tipos de Sistemas e Arquiteturas de Banco de Dados Distribuídos

MySQL - *Workbench*

Acadêmicos	**RA**
Jonatas Alves da Silva	78076
Luiz Flávio Pereira	91706
William Rodrigues da Silva	99514

Maringá - PR

2018

Universidade Estadual de Maringá

Centro de Tecnologia - Departamento de Informática

Curso de Bacharelado em Informática

Tipos de Sistemas e Arquiteturas de Banco de Dados Distribuídos

MySQL - *Workbench*

Trabalho solicitado aos alunos da disciplina de Banco de Dados II, pela professora Drª. Maria Madalena Dias, como requisito de obtenção de nota da terceira avaliação semestral.

Maringá-PR

2018

Sumário

1. Introdução

Esta documentação é divida em duas partes, sendo elas a parte prática e teórica sobre o estudo do MySQL no ambiente de desenvolvimento do *Workbench*, onde será realizado algumas tarefas e testes a fim de saber se o Sistema de Gerenciamento de Banco de Dados faz tratativas e quais são elas no que diz respeito ao controle de concorrência.

O estudo prático visa realizar a criação de uma data base, criação de tabelas, importação de dados, realização de testes para controle de concorrência, levantamento dos controles de concorrência realizados pelo MySQL e algumas consultas na data base. Já a parte teórica foca em trazer um estudo sobre os tipos de sistemas de bancos de dados distribuídos disponíveis no mercado além de hardwares.

1.1. Glossário: Termos, Siglas e Abreviações

BD: Banco de dados

SGDB: Sistema gerencial de banco de dados

MR: Modelo relacional

IDE: Ambiente de desenvolvimento

DBA: Analista de banco de dados

PK: Primary key ou chave primária

FK: Foreign key ou chave estrangeira

Tupla: registro de uma tabela

Schema: Coleção de objetos de um banco de dados

2. Parte I – Trabalho Prático

O trabalho prático foi todo desenvolvido no ambiente do *Workbench*, esta IDE trás uma interface simples e prática na manipulação de *schemas*, tabelas e comandos em SQL com isso, é possível otimizar e ganhar muito tempo. Para um DBA, a otimização das atividades e ganho de tempo são cruciais principalmente em casos de falhas.

2.1. Criação de *Schema*

A criação de um schema é um tanto quanto simples, segue abaixo o comando realizado em MySQL na interface do *Workbench*:

```
1    CREATE SCHEMA `trab_bd2` ;
```

Figura 1: Criação de schema.

2.2. Criação de Tabelas

A criação de tabelas em um banco de dados consiste em dar um nome a uma coluna/campo, atribuir um tipo específico que atenda a necessidade daquela coluna e também especificar o tamanho do dado que alí será armazenado.

Vale ressaltar, que algumas colunas possuem uma caracteristica denominadas PK e FK. Essas colunas possuem um papel especial em relação as demais, as chaves primárias são únicas e referenciam um único registro dentro da tabela, já as chaves estrangeiras são um campo da tabela que apontam para uma chave primária de uma outra tabela, com isso, é possível relacionar as tabelas e uní-las quando necessário.

Segue abaixo os comandos necessários para a criação de tabelas e suas respectivas relações:

2.2.1. Criação da Tabela de Pacientes

```
CREATE TABLE `trab_bd2`.`paciente` (
  `numpaciente` INT NOT NULL,
  `codgrupo` VARCHAR(10) NULL,
  `codseguradora` VARCHAR(10) NULL,
  `datanasc` VARCHAR(20) NULL,
  `sexo` INT NULL,
  `bairro` VARCHAR(35) NULL,
  `cep` VARCHAR(10) NULL,
  `cidade` VARCHAR(45) NULL,
  `uf` VARCHAR(5) NULL,
  `dataultimareq` VARCHAR(20) NULL,
  `flagpaciente` VARCHAR(4) NULL,
  `nomepaciente` VARCHAR(60) NULL,
  PRIMARY KEY (`numpaciente`));
```

Figura 2: Criação da tabela paciente.

2.2.2. Criação da Tabela de Requisições

```
CREATE TABLE `trab_bd2`.`requisicao` (
  `numprotocolo` INT NOT NULL,
  `numpaciente` INT NOT NULL,
  `datarequisicao` VARCHAR(20) NULL,
  `sexo` INT NULL,
  `idade` INT NULL,
  `idadeinformada` VARCHAR(10) NULL,
  `codgrupoconvenio` VARCHAR(10) NULL,
  `codseguradoraconvenio` VARCHAR(15) NULL,
  `total` VARCHAR(15) NULL,
  `desconto` VARCHAR(15) NULL,
  `valorpago` VARCHAR(15) NULL,
  `idposto` VARCHAR(5) NULL,
  `orcamento` INT NULL,
  `codmedico` VARCHAR(10) NULL,
  `campovariavel2` VARCHAR(10) NULL,
  `descontopercent` INT NULL,
  PRIMARY KEY (`numprotocolo`),
  INDEX `req-pacientes_idx` (`numpaciente` ASC) VISIBLE,
  CONSTRAINT `req-pacientes`
    FOREIGN KEY (`numpaciente`)
    REFERENCES `trab_bd2`.`paciente` (`numpaciente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

Figura 3: Criação da tabela requisicao.

2.2.3. Criação da Tabela de Resultados de Exames

```
CREATE TABLE `trab_bd2`.`resultadoexame` (
  `numitem` INT NOT NULL,
  `numprotocolo` INT NOT NULL,
  `codexame` VARCHAR(10) NULL,
  `numatributo` INT NULL,
  `nomeatributo` VARCHAR(40) NULL,
  `tipo` VARCHAR(5) NULL,
  `visivel` INT NULL,
  `informado` INT NULL,
  `calculado` INT NULL,
  `formula` VARCHAR(10) NULL,
  `textovalornormal` VARCHAR(150) NULL,
  `resultadopadrao` VARCHAR(20) NULL,
  `unidade` VARCHAR(20) NULL,
  `normalidade` VARCHAR(20) NULL,
  `resultado` VARCHAR(100) NULL,
  `numexame` VARCHAR(20) NULL,
  `impressora` VARCHAR(20) NULL,
  PRIMARY KEY (`numitem`),
  INDEX `res.ex-req_idx` (`numprotocolo` ASC) VISIBLE,
  CONSTRAINT `res.ex-req`
    FOREIGN KEY (`numprotocolo`)
    REFERENCES `trab_bd2`.`requisicao` (`numprotocolo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

Figura 4: Criação da tabela resultadoexame.

2.3. Diagrama de Entidades e Relacionamentos (DER)

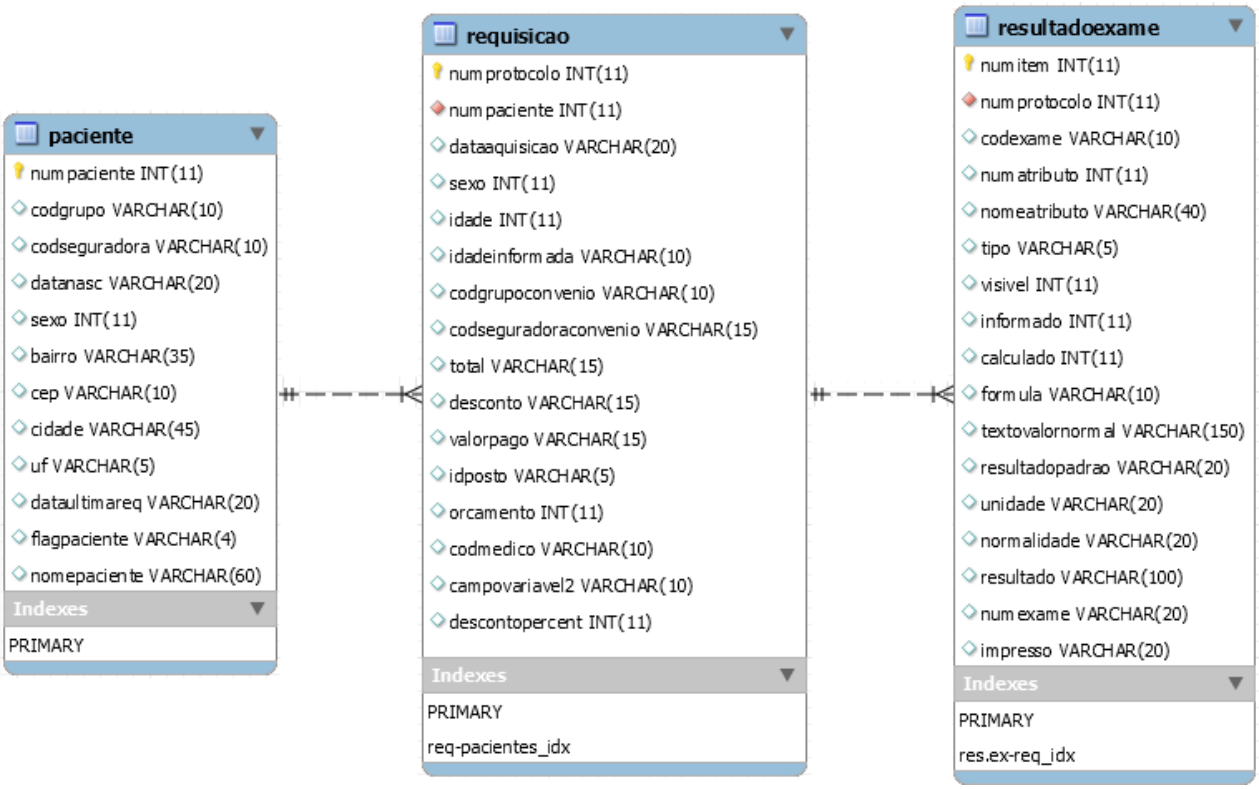


Figura 5: Diagrama de entidades e relacionamentos.

O DER é a principal ferramenta para representar graficamente as tabelas, suas colunas, seus tipos de dados e também as relações criadas entre as tabelas. Nesse modelo também é possível ver aquilo que chamamos de cardinalidade, ou seja, o grau de relação entre as entidades ou tabelas.

2.4. Inserção de Dados nas Tabelas do *Schema*

Para a inserção dos dados foi utilizado uma das técnicas mais tradicionais, ou seja, linhas de comando *insert into.* Foi escrito 136.243 linhas de comandos para a inserção de todos os registros de todas as tabelas sendo, 30.537 registros na tabela paciente, 7.117 registros na tabela requisicao e 98.589 registros na tabela resultadoexame.

A inserção dos dados por arquivo excel não funciounou na versão 8.0 da IDE *Workbench*, portanto, adotou-se a técnica mais comum para inserção de dados nas tabelas. Segue abaixo um exemplo de como os dados foram inseridos:

```
1  --Exemplo de inserção da tabela paciente
2  INSERT INTO `trab_bd2`.`paciente` (`numpaciente`, ... , `nomepaciente`) VALUES ('1', ... , 'João');
3
4  --Exemplo de inserção da tabela requisicao
5  INSERT INTO `trab_bd2`.`requisicao` (`numprotocolo`, ... , `descontopercent`) VALUES ('1', ... , '10');
6
7  --Exemplo de inserção da tabela resultadoexame
8  INSERT INTO `trab_bd2`.`resultadoexame` (`numitem`, ... , `impresso`) VALUES ('1', ... , '1');
```

Figura 6: Exemplo do método utilizado para inserir dados nas tabelas.

3. Aplicação Java com Conexão JDBC - MySQL

Para realizar testes de controles de concorrênciase suas formas de bloqueios, criamos uma aplicação em Java para simular duas transações sendo executadas simultâneamente, ou seja, uma transação seria realização na aplicação Java e a outra transação será executada na IDE do Workbench.

O controle de concorrência serve para evitar que o banco tenha inconsistências de dados e que por consequência apresente dados incorretos ao usuário. Essas inconsistências podem ocorrer quando duas transações distintas estão acessando os

mesmo campos e/ou tuplas e fazem alterações nesses campos, porém, apenas uma das transações têm seus dadosreplicados e gravados em disco no banco de dados.

A aplicação desenvolvida possui interface simples e prática de modo a facilitar os testes pertinentes ao controle de concorrência, trazer informações básicas e dados armazenados no banco, além das consultas otimizadas. Veja abaixo a tela inicial do programa de testes:

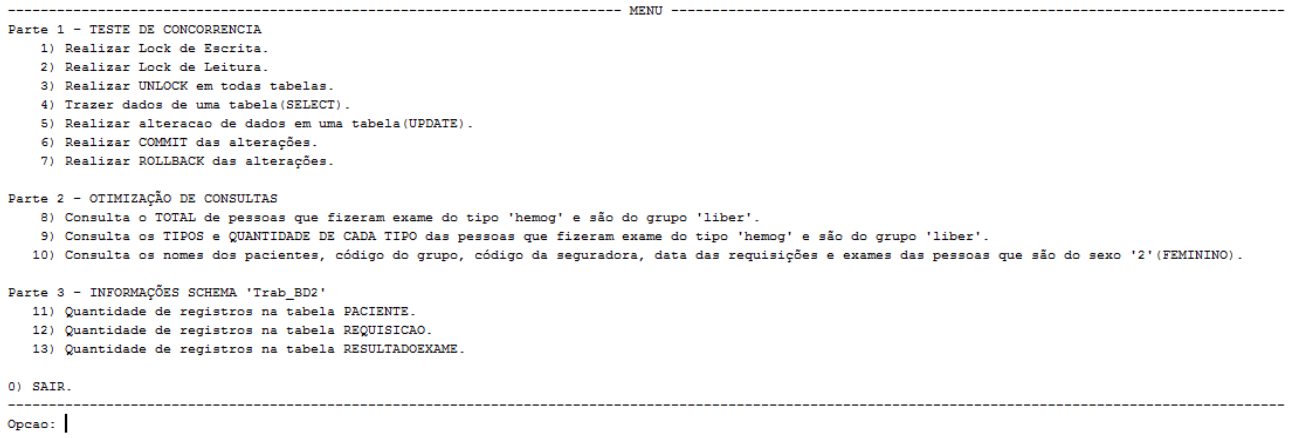


Figura 7: Menu da aplicação Java

O menu possui 13 opções e cada uma dessas opções executam um comando em particular e possui um retorno apresentado em tela ao usuário. Dizendo se o processo foi finalizado com sucesso ou não.

3.1. Funcionamento, Comandos e Resultados

Neste tópico mostraremos como funciona a interação do usuário com a aplicação.

3.1.1. Opção 1 : Lock de Escrita

Esta opção ao ser executada solicitará ao usuário o nome da tabela que ele deseja realizar o bloqueio de escrita, caso o usuário digite um nome de tabela inválido uma mensagem de erro é apresentada e o processo é repetido.

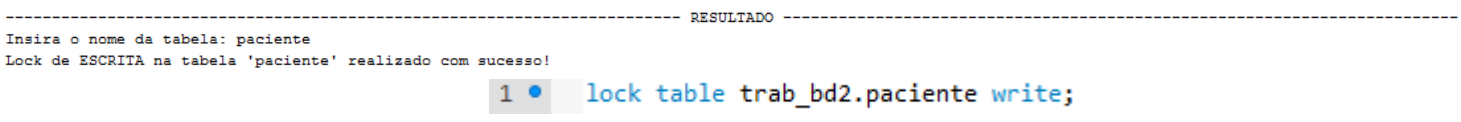


Figura 8: Resultado e comando executado ao escolher a opção 1.

3.1.2. Opção 2 : Lock de Leitura

Esta opção ao ser executada solicitará ao usuário o nome da tabela que ele deseja realizar o bloqueio de leitura, caso o usuário digite um nome de tabela inválido uma mensagem de erro é apresentada e o processo é repetido.

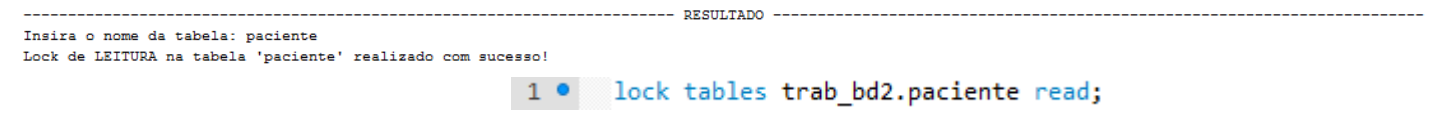


Figura 9: Resultado e comando executado ao escolher a opção 2.

3.1.3. Opção 3 : Unlock em Todas as Tabelas

Ao escolher a opção 3 no menu inicial o programa realizar o unlock de todas as tabelas que anteriormente pudessem estar com bloqueios.

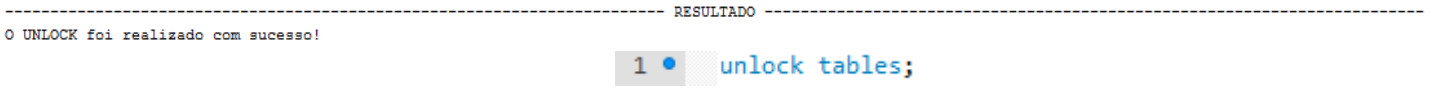


Figura 10: Resultado e comando executado ao escolher a opção 3.

3.1.4. Opção 4 : Realizar Consulta (Comando Select)

A opção 4 nos trará o dado que deserjamos desde que preenchamos corretamente as informações solicitadas, são elas: nome da tabela, nome da comuna e condição do where. Obs: Se a condição do where foi deixada em branco trará todos os dados da coluna escolhida.

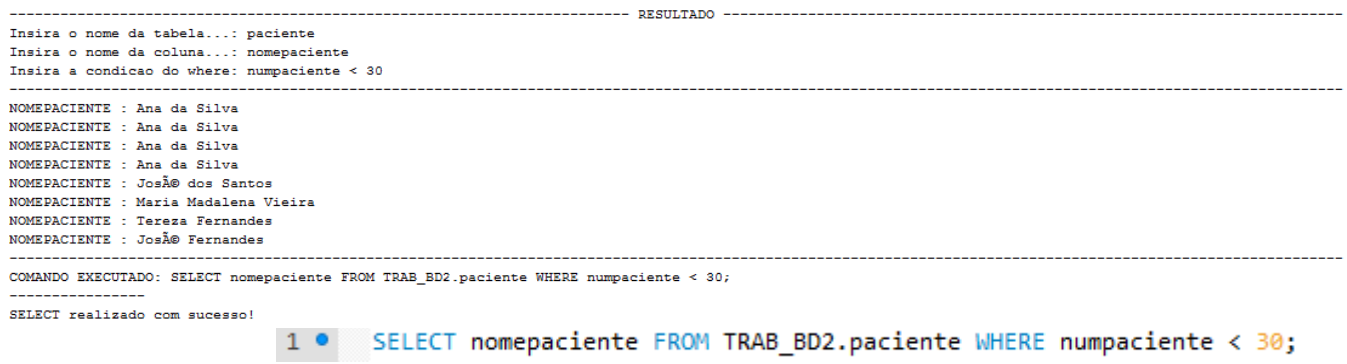


Figura 11: Resultado e comando executado ao escolher a opção 4.

3.1.5. Opção 5 : Realizar Alteração (Comando Update)

A opção 5 fara uptade desde que preenchamos corretamente as informações solicitadas, são elas: nome da tabela, nome da comuna, novo valor a ser setado e condição do where. Obs: Se a condição do where foi deixada em branco, ou ainda, preenchida incorretamente poderá fazer alteração indesejada em toda a tabela ou o programa abortará. Caso ocorra o alterações indesejadas em toda a tabela poderemos usar a Opção 7 e dar Rollback.

```
------------------------------------------------------------------------ RESULTADO ------------------------------------------------------------------------
Insira o nome da tabela...: paciente
Insira o nome da coluna...: nomepaciente
Valor a ser setado.......: Luiz Flavio Pereira
Insira a condicao do where: numpaciente = 100
-----------------------------------------------------------------------------------------------------------------------------------------------------------
COMANDO EXECUTADO: UPDATE TRAB_BD2.paciente SET nomepaciente = 'Luiz Flavio Pereira' WHERE numpaciente = 100;
----------------
UPDATE realizado com sucesso!
```

```
1 • start transaction;
2 • lock tables paciente write, paciente as pac read;
3 • UPDATE TRAB_BD2.paciente SET nomepaciente = 'Luiz Flavio Pereira' WHERE numpaciente = 100;
```

Figura 12: Resultado e comando executado ao escolher a opção 5.

3.1.6. Opção 6 : Commit

O commit é uma das opções mais importantes no que diz respeito à consistência de dados, visto que as alterações só serão replicadas ao banco após executá-lo. Vale ressaltar que ao fazer o commit na aplciação as tabelas já são desbloqueadas instantaneamente e um novo savepoint é registrado nos logs.

```
------------------------------------------------------------------------ RESULTADO ------------------------------------------------------------------------
COMMIT realizado com sucesso!
```

```
1 • commit;
2 • savepoint transaction;
```

Figura 13: Resultado e comando executado ao escolher a opção 6.

3.1.7. Opção 7 : Rollback

Caso ocorra alguma inconsistência e/ou alterações indesejadas durante a execução da opção 5, podemos então utilizar o comando rollback da opção 7 para revertê-los e um novo savepoint é registrado nos logs.

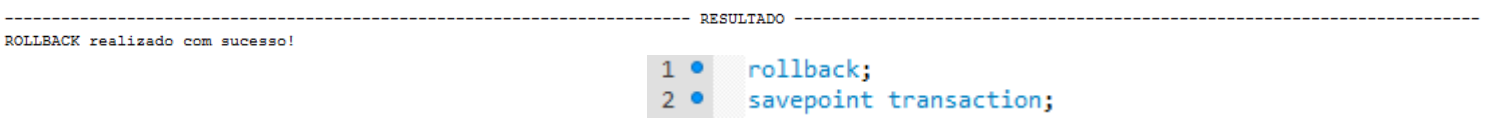


Figura 14: Resultado e comando executado ao escolher a opção 7.

3.1.8. Opção 8 : Consulta Total de Pessoas do Grupo ‘liber’ que Fizeram Exame ‘hemog’

A opção 8 é a primeira opção que trás os resultados das otimizações, como resultado é apresentado o tempo sem otimização, o tempo com otimização, o ganho de tempo e também o retorno do select.

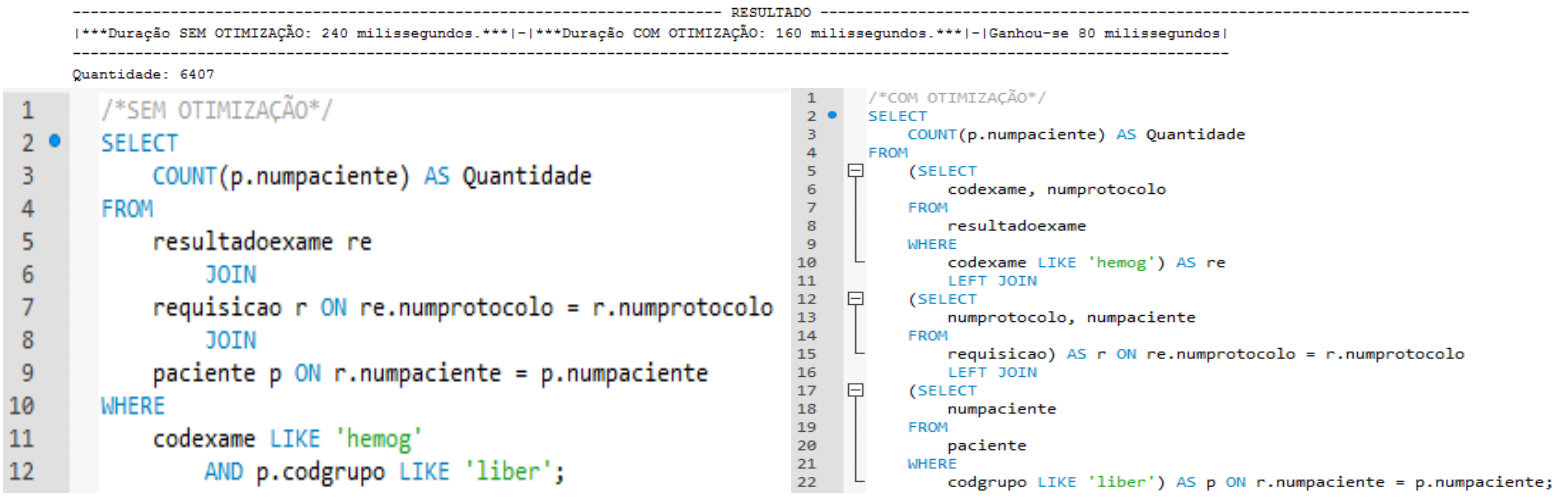


Figura 15: Resultado e comando executado ao escolher a opção 8.

3.1.9. Opção 9 : Consulta os Tipos de Exames e Suas Quantidades Às Pessoas do Grupo 'liber' que Fizeram Exame 'hemog'

A opção 9 separa os tipos e suas respectivas quantidades agrupados.

```
------------------------------------------------------------------------ RESULTADO ------------------------------------------------------------------------
|***Duração SEM OTIMIZAÇÃO: 172 milissegundos.***|-|***Duração COM OTIMIZAÇÃO: 132 milissegundos.***|-|Ganhou-se 40 milissegundos|
-----------------------------------------------------------------------------------------------------------------------------
Tipo: NULL | Quantidade: 678
Tipo: N | Quantidade: 5274
Tipo: C | Quantidade: 455
```

```sql
/*SEM OTIMIZAÇÃO*/
SELECT
    re.tipo AS Tipo, COUNT(*) AS Quantidade
FROM
    resultadoexame re
        LEFT JOIN
    requisicao r ON re.numprotocolo = r.numprotocolo
        LEFT JOIN
    paciente p ON r.numpaciente = p.numpaciente
WHERE
    re.codexame LIKE 'hemog'
        AND p.codgrupo LIKE 'liber'
GROUP BY re.tipo;
```

```sql
/*COM OTIMIZAÇÃO*/
SELECT
    re.tipo AS Tipo, COUNT(*) AS Quantidade
FROM
    (SELECT
        numprotocolo, tipo
    FROM
        resultadoexame
    WHERE
        codexame LIKE 'hemog') AS re
        JOIN
    requisicao r ON re.numprotocolo = r.numprotocolo
        JOIN
    (SELECT
        numpaciente, codgrupo
    FROM
        paciente
    WHERE
        codgrupo LIKE 'liber') AS p ON r.numpaciente = p.numpaciente
GROUP BY re.tipo;
```

Figura 16: Resultado e comando executado ao escolher a opção 9.

3.1.10. Opção 10 : Recuperar Dados de Exames Feitos Por Mulheres

A opção 10 busca especialmente por pessoas do sexo feminino e que já fizeram algum tipo de exame.

```
------------------------------------------------------------------------ RESULTADO ------------------------------------------------------------------------
|***Duração SEM OTIMIZAÇÃO: 423 milissegundos.***|-|***Duração COM OTIMIZAÇÃO: 393 milissegundos.***|-|Ganhou-se 30 milissegundos|
-----------------------------------------------------------------------------------------------------------------------------
Deseja IMPRIMIR [0 = NÃO / 1 = SIM]
R: 1
-----------------------------------------------------------------------------------------------------------------------------------------------------------
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: ambc | Data Requisicao: 03/01/2004 00:00:00 | Exame: cpk
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: ambc | Data Requisicao: 03/01/2004 00:00:00 | Exame: ferrC
Nome: Maria da Silva | Cod.Grupo: cedlab | Cod.Seguradora: pass | Data Requisicao: 05/01/2004 00:00:00 | Exame: estra
Nome: Maria da Silva | Cod.Grupo: cedlab | Cod.Seguradora: pass | Data Requisicao: 05/01/2004 00:00:00 | Exame: estra
Nome: Maria da Silva | Cod.Grupo: cedlab | Cod.Seguradora: pass | Data Requisicao: 05/01/2004 00:00:00 | Exame: ferr2
Nome: Maria da Silva | Cod.Grupo: cedlab | Cod.Seguradora: pass | Data Requisicao: 05/01/2004 00:00:00 | Exame: estro
Nome: Maria da Silva | Cod.Grupo: cedlab | Cod.Seguradora: pass | Data Requisicao: 05/01/2004 00:00:00 | Exame: sdC
Nome: Maria Luiza Pereira | Cod.Grupo: cedlab | Cod.Seguradora: amb3 | Data Requisicao: 06/01/2004 00:00:00 | Exame: t4li
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: unimed | Data Requisicao: 06/01/2004 00:00:00 | Exame: bence
Nome: Maria Helena Pereira | Cod.Grupo: cedlab | Cod.Seguradora: amb3 | Data Requisicao: 07/01/2004 00:00:00 | Exame: IgEt
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: uni2 | Data Requisicao: 08/01/2004 00:00:00 | Exame: ferrC
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: amb1 | Data Requisicao: 09/01/2004 00:00:00 | Exame: prola
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: amb1 | Data Requisicao: 09/01/2004 00:00:00 | Exame: prola
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: amb1 | Data Requisicao: 09/01/2004 00:00:00 | Exame: lhAlv
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: bradesco | Data Requisicao: 09/01/2004 00:00:00 | Exame: dheas
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: bradesco | Data Requisicao: 09/01/2004 00:00:00 | Exame: ferrC
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: bradesco | Data Requisicao: 09/01/2004 00:00:00 | Exame: est
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: bradesco | Data Requisicao: 09/01/2004 00:00:00 | Exame: est
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: bradesco | Data Requisicao: 09/01/2004 00:00:00 | Exame: est
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: bradesco | Data Requisicao: 09/01/2004 00:00:00 | Exame: est
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: bradesco | Data Requisicao: 09/01/2004 00:00:00 | Exame: t4li
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: clor
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: cpk
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: hemog
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: hemog
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: hemog
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: hemog
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: hemog
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: hemog
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: hemog
Nome: JoÃ£o Ferreira | Cod.Grupo: cedlab | Cod.Seguradora: dra.let | Data Requisicao: 10/01/2004 00:00:00 | Exame: hemog
```

```sql
/*SEM OTIMIZAÇÃO*/
SELECT
    pac.nomepaciente,
    pac.codgrupo,
    pac.codseguradora,
    req.dataaquisicao,
    result.codexame
FROM
    resultadoexame result
        LEFT JOIN
    requisicao req ON result.numprotocolo = req.numprotocolo
        LEFT JOIN
    paciente pac ON req.numpaciente = pac.numpaciente
WHERE
    pac.sexo = 2
LIMIT 99999;/*trará no max 99.999 registros*/
```

```sql
/*COM OTIMIZAÇÃO*/
SELECT
    pac.nomepaciente,
    pac.codgrupo,
    pac.codseguradora,
    req.dataaquisicao,
    result.codexame
FROM
    (SELECT
        codexame, numprotocolo
    FROM
        resultadoexame) AS result
        JOIN
    (SELECT
        dataaquisicao, numprotocolo, numpaciente
    FROM
        requisicao) AS req ON result.numprotocolo = req.numprotocolo
        JOIN
    (SELECT
        nomepaciente, codgrupo, codseguradora, numpaciente
    FROM
        paciente
    WHERE
        sexo = 2) pac ON req.numpaciente = pac.numpaciente
LIMIT 99999;/*trará no max 99.999 registros*/
```

Figura 17: Resultado e comando executado ao escolher a opção 10.

3.1.11. Opções 11, 12 e 13 : Informações das Tabelas

As últimas opções do menu são responsáveis por trazer quantos registros há em cada tabela, veja abaixo os comandos realizados.

```
---------- RESULTADO ----------
Qtde. Pacientes: 30537
---------- RESULTADO ----------
Qtde. Requisições: 7117
---------- RESULTADO ----------
Qtde. Resultados de Exames: 98589
```

```sql
SELECT
    COUNT(numpaciente) AS QuantidadePacientes
FROM
    paciente;
```

```sql
SELECT
    COUNT(numprotocolo) AS QuantidadeRequisicoes
FROM
    requisicao;
```

```sql
SELECT
    COUNT(numitem) AS QuantidadeResultados
FROM
    resultadoexame;
```

Figura 18: Resultados e comandos executados ao escolher as opções 11, 12 e 13.

3.2. Controle de Concorrência e Consistência de Dados

Por convenção chamaremos as transações realizadas dentro da aplicação java de T1 e as transações realizadas dentro do Workbench de T2.

- Para obter o um teste de controle de concorrência, executaremos os seguintes passos:

1. Lock de Escrita ou Leitura (T1)
2. Alterar ou Consultar (T2)
3. Observar Comportamento da T2
4. Realizar Unlock (T1)
5. Oberservar o Novo Status da T2

- Para obter uma correta consistência de dados durante alterações, seguiremos os seguintes passos:
    1. Consultar Dado a Ser Alterado (T1)
    2. Realizar Lock de Escrita (T1)
    3. Alterar Dado com Comando Update (T1)
    4. Consultar Dado Que Foi Alterado (T1)
    5. Consultar Dado Que Foi Alterado (T2)
    6. Observar Dados Inconsistentes Em T1 e T2

    -Se alterações realizadas corretamente no passo 3, então:

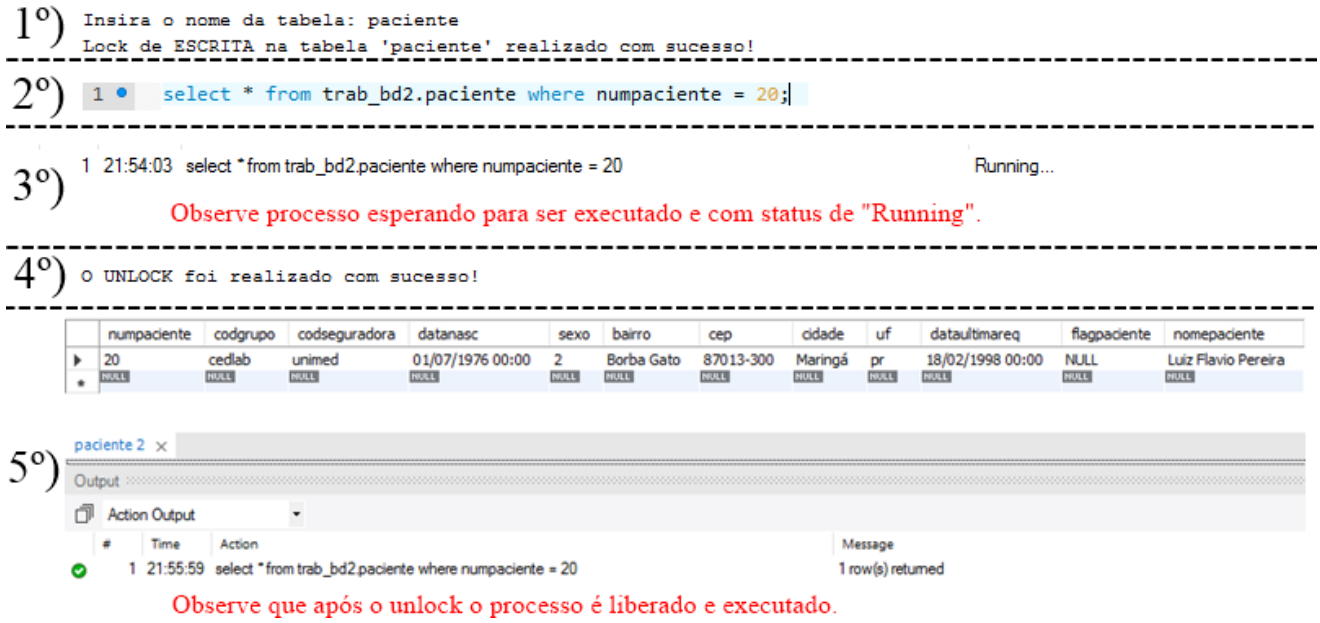
    7. Commitar(T1)

    -Se alterações realizadas incorretamente no passo 3, então:

    8. Rollback(T1)
    9. Observar Dados Consistentes Em T1 e T2

3.2.1. Teste de Controle de Concorrência

Aqui executaremos os passos citados anteriormente, veja abaixo os comportamentos de T1 e T2.



Fugira 19: Exemplo de controle de concorrência efetuado com sucesso.

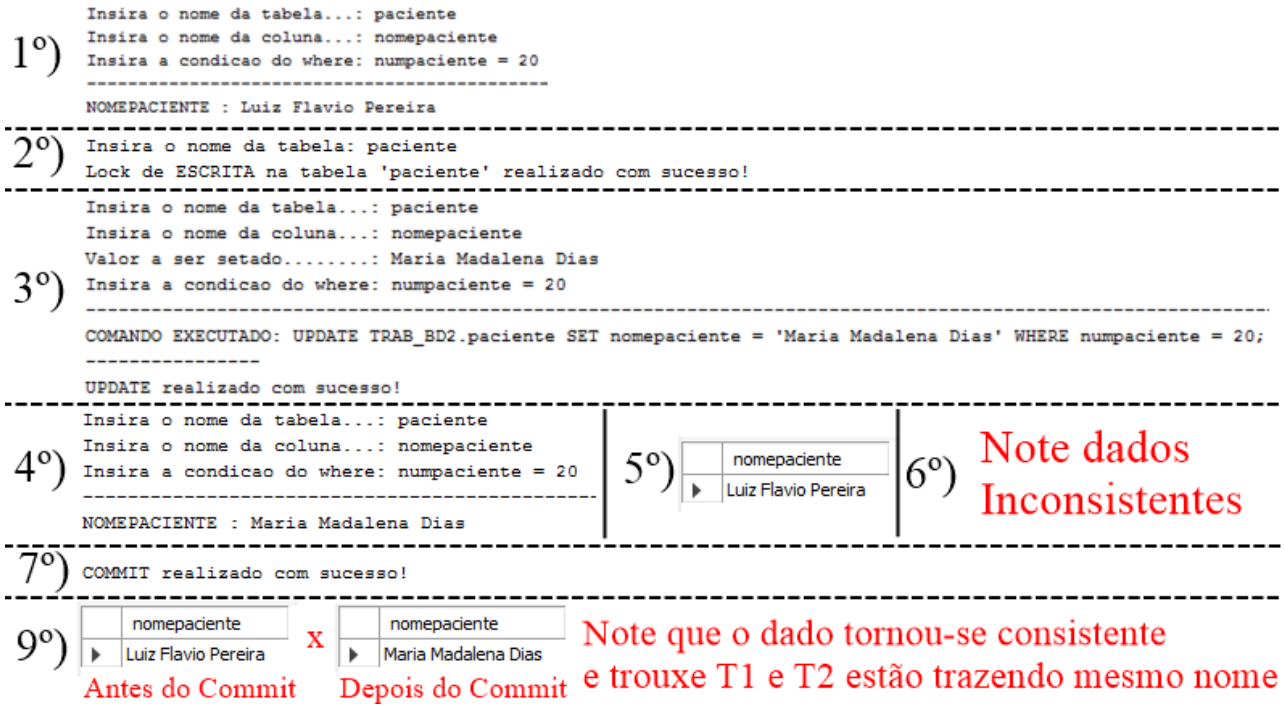
Podemos notar que na etapa 1 foi realiado Lock de Escrita, e depois tentamos realizar uma leitura e mesmo assim é bloqueado no passo 3. Então isso mostra que o Lock de Escrita numa tabela toda resulta que não seja possível nem mesmo realizar a consulta de uma tupla em específico.

Outro ponto a ser ressaltado é os passos 3 e 5, eles possuem status diferentes e isso prova que os comandos lock e unlock estão sendo executados corretamente e as transações T2 estão realmente aguardando o lock de T1. Caso demorássemos para realizar o unlock, então o passo 5 mostraria mensagem de erro (perda de conexão com o MySQL durante a execução da Query).

3.2.2. Teste de Consistência de Dados

Aqui mostraremos como é realizado a consistencia de dados ao efetuar a troca de um paciente. O dado é replicado para o banco após o comando de commit, portanto, é essencial este comando para evitar erros.

Veja exemplo abaixo:



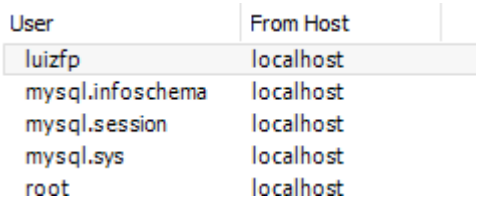
Fugira 19: Exemplo de controle de concorrência efetuado com sucesso.

Neste teste podemos notar que inicialmente em T1 tinhamos o nome “Luiz Flavio Pereira” e após o update do passo 3, trocamos para o nome “Maria Madalena Dias”, porém, no passo 5 oberservamos que a alteração não foi refletida no banco Workbench em T2. Isso ocorreu pelo simples fato de que uma transaction deve conter os comandos commit ou rollback ao ser finalizada. Por isso, após o passo 7 onde foi feito o commit nós podemos oberservar que o novo nome foi refletido no banco em T2 no passo 9.

4. Controle de Acessos e Permissões de Usuários

O controle de acessos é extremamente importante, principalmente para evidar perda, exclusão ou trocas indesejadas de informações. Informações de uma empresa podem vazar a qualquer momento e gerar prejuízos milhonários.

Para minimizar a possibilidade desses problemas acontecem, os Sistemas de Gerenciamento de Banco de Dados costumam dar a opção de criar usuários diferentes com perfils de acessos diferentes.

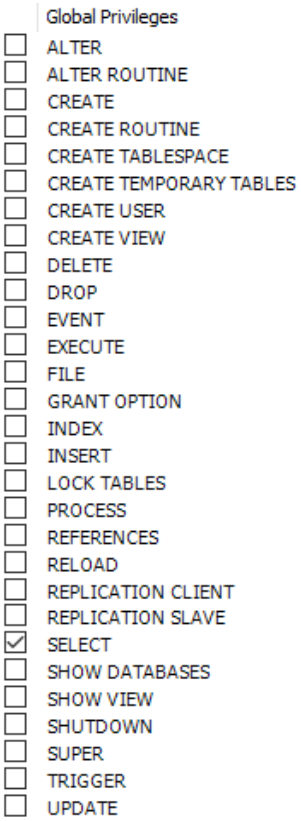


User	From Host
luizfp	localhost
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

Fugira 20: Usuários do banco de dados.

Para o banco em questão temos 2 usuários principais, são eles: root e luizfp.

O usuário root tem todas as permissões de um DBA, já o usuário luizfp foi criado com intuito de mostrar como funcionam os bloqueio. Para o usuário luizfp demos uma única permissão, a de realizar consultas/selects.



Global Privileges

- [ ] ALTER
- [ ] ALTER ROUTINE
- [ ] CREATE
- [ ] CREATE ROUTINE
- [ ] CREATE TABLESPACE
- [ ] CREATE TEMPORARY TABLES
- [ ] CREATE USER
- [ ] CREATE VIEW
- [ ] DELETE
- [ ] DROP
- [ ] EVENT
- [ ] EXECUTE
- [ ] FILE
- [ ] GRANT OPTION
- [ ] INDEX
- [ ] INSERT
- [ ] LOCK TABLES
- [ ] PROCESS
- [ ] REFERENCES
- [ ] RELOAD
- [ ] REPLICATION CLIENT
- [ ] REPLICATION SLAVE
- [x] SELECT
- [ ] SHOW DATABASES
- [ ] SHOW VIEW
- [ ] SHUTDOWN
- [ ] SUPER
- [ ] TRIGGER
- [ ] UPDATE

Isso significa dizer que se o usuário ‘luizfp’ ao tentar realizar qualquer outro comando que não seja de consulta, acarretará em um erro e desconexão automática com o banco de dados por *acess violation/denied.*

Fugira 21: Permissões do usuário ‘luizfp’.

Ao realizar o select, funcionou corretamente e trouxe as informações desejadas. Porém, ao fazer um update, deu erro. Veja abaixo as imagens:

```
------------------------------------------------
Insira o nome da tabela...: paciente
Insira o nome da coluna...: nomepaciente
Insira a condicao do where: numpaciente = 20
------------------------------------------------
NOMEPACIENTE : Ana da Silva
```

Fugira 22: Sucesso ao realizar consultas com o usuário ‘luizfp’.

```
------------------------------------------------------------------------ RESULTADO ------------------------------------------------------------------------
Insira o nome da tabela...: paciente
Insira o nome da coluna...: nomepaciente
Valor a ser setado........: Luiz
Insira a condicao do where: numpaciente = 20
-----------------------------------------------------------------------------------------------------------------------------------------------------------
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: UPDATE command denied to user 'luizfp'@'localhost' for table 'paciente'
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
        at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
        at com.mysql.jdbc.Util.handleNewInstance(Util.java:425)
        at com.mysql.jdbc.Util.getInstance(Util.java:408)
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:944)
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3976)
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3912)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:2530)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:2683)
        at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2482)
        at com.mysql.jdbc.StatementImpl.executeUpdateInternal(StatementImpl.java:1552)
        at com.mysql.jdbc.StatementImpl.executeLargeUpdate(StatementImpl.java:2607)
        at com.mysql.jdbc.StatementImpl.executeUpdate(StatementImpl.java:1480)
        at trab_bd2.Aplicacao.main(Aplicacao.java:59)
```

Fugira 22: Erro ao realizar alterações com o usuário ‘luizfp’.

Deste modo notamos que um DBA deve saber exatamente a quem ele dará tais permissões, uma decisão incorreta pode gerar inúmeros erros.

	Role	Description
☐	DBA	grants the rights to perform all tasks
☐	MaintenanceAdmin	grants rights needed to maintain server
☐	ProcessAdmin	rights needed to assess, monitor, and kill any user proce...
☐	UserAdmin	grants rights to create users logins and reset passwords
☐	SecurityAdmin	rights to manage logins and grant and revoke server an...
☐	MonitorAdmin	minimum set of rights needed to monitor server
☐	DBManager	grants full rights on all databases
☐	DBDesigner	rights to create and reverse engineer any database sche...
☐	ReplicationAdmin	rights needed to setup and manage replication
☐	BackupAdmin	minimal rights needed to backup any database
☑	Custom	custom role

Fugira 23: Categorias de permissões pré-definidas pelo SGDB.

5. Parte II – Pesquisa Teórica

5.1. Tipos de Sistemas de Banco de Dados Distribuídos

BDDS – Banco de Dados Distribuídos

A tecnologia de BDD surgiu com uma fusão de outras duas tecnologias:

1º Tecnologia de banco de dados.

2º Tecnologia de rede e comunicação de dados.

Enquanto os primeiros bancos de dados se voltaram para a centralização e resultaram em bancos de dados monolíticos gigantescos nos anos 70, a tendência se inverteu para mais descentralização e autonomia de processamento no final dos anos 80.

Os bancos de dados distribuídos trazem as vantagens da computação distribuída para o domínio do gerenciamento de banco de dados. Podemos definir um banco de dados distribuído (BDD) com uma coleção de múltiplos bancos de dados logicamente inter-relacionados distribuídos por uma rede de computadores. E um sistema de gerenciamento de banco de dados distribuídos (SGBDD) como um sistema de software que gerencia um banco de dados distribuído enquanto torna a distribuição transparente para o usuário.

A característica principal que todos os sistemas desse tipo possuem em comum é o fato de que os dados e o software são distribuídos por múltiplos sites conectados por alguma forma de rede de comunicação.

Existem dois tipos de banco de dados distribuídos, os homogêneos e os heterogêneos.

Homogêneo: Todos os nós rodando o mesmo SGBD.

Heterogêneo: Diferentes nós rodando diferentes SGBDs, relacionais ou mesmo não relacionais.

5.2. Formas de Armazenamentos:

Os arquivos do banco de dados distribuídos são divididos em três formas de armazenamento: Replicação, Fragmentação e fragmentação e replicação.

5.2.1. Replicação:

O sistema mantém réplicas idênticas da relação, onde cada réplica é armazenada em locais diferentes, resultando na replicação dos dados. Existem pontos de destaques na replicação:

Os dados estão mais disponíveis neste esquema.

O paralelismo é aumentado quando a solicitação de leitura é atendida. Aumenta a sobrecarga nas operações de atualização, pois cada site que contém a réplica precisava ser atualizado para manter a consistência.

5.2.2. Replicação Síncrona:

Todas as cópias de uma relação modificada (fragmentos) devem ser atualizadas antes da transação modificante fazer commit;

A distribuição de dados fica transparente para o usuário.

5.2.3. Replicação Assíncrona:

As cópias da relação modificada só são atualizadas periodicamente; réplicas podem ficar inconsistentes por algum tempo;

Os usuários devem estar cientes da distribuição e replicação.

5.2.4. Fragmentação

A relação é particionada em vários fragmentos, onde cada fragmento é armazenado em um local diferente. Existem duas formas de fazer a fragmentação: Fragmentação Horizontal e Fragmentação Vertical, onde:

Fragmentação Horizontal: os fragmentos são definidos por seleção de tuplas.

Fragmentação Vertical: os fragmentos são definidos por projeção de atributos.

5.2.5. Replicação e Fragmentação

A relação é particionada em vários segmentos, e o sistema mantém diversas réplicas de cada fragmento.

5.3. Regras de um SGBDD

Em 1987, Christopher J. Date, um dos primeiros projetistas de bancos de dados relacionais, junto com o Dr. Edgar Frank Codd, autor da teoria relacional, propôs 12 regras que um SGBDD completo deveria seguir.

1. Autonomia local: Cada nó participante de um sistema distribuído deve ser independente dos outros nós. Cada nó deve prover mecanismos de segurança, bloqueio, acesso, integridade e recuperação após falha.

2. Não dependência de um nó central: Um sistema de banco de dados distribuído não deve depender de um nó central, pois isso acarretaria um único ponto de falha, afetando todos os outros nós. Um nó central também poderia ficar sobrecarregado, resultando em perda de desempenho do sistema.

3. Operação contínua: Um sistema de banco de dados distribuído nunca deve precisar ser desativado. As operações de backup e recuperação devem ser suportadas on-line. Essas operações devem ainda ser rápidas o bastante para não afetarem o funcionamento do sistema (backup incremental, por exemplo).

4. Transparência/independência de localização: Os usuários do sistema não devem precisar saber o local onde estão localizados os dados; devem se comportar como se os dados estivessem armazenados localmente. A transparência de localização pode ser alcançada pela utilização de sinônimos estendidos e pelo extenso uso do dicionário de dados. A transparência de localização permite que aplicações sejam portadas de um nó da rede para outro sem a necessidade de modificações.

5. Independência de fragmentação: As tabelas que fazem parte de um sistema de banco de dados distribuído podem estar divididas em fragmentos, localizados fisicamente em diferentes nós, de forma transparente para o usuário.

6. Independência de replicação: Dados podem estar replicados em vários nós da rede de forma transparente. As réplicas de dados devem ser mantidas sincronizadas automaticamente pelo SGBDD.

7. Processamento de consultas distribuído: O desempenho de uma consulta deve ser independente do local onde a mesma é submetida. Um SGBDD deve possuir um otimizador capaz de selecionar não apenas o melhor caminho para o acesso a um determinado nó da rede, mas também otimizar o desempenho de uma consulta distribuída, levando em conta a localização dos dados, utilização de CPU, I/O e o tráfego na rede.

8. Gerenciamento de transações distribuídas: Um SGBDD deve suportar transações atômicas. As propriedades ACID (Atomicidade, Consistência, Independência e Durabilidade) das transações e a serialização devem ser suportadas não apenas para transações locais, mas também para transações distribuídas.

9. Independência de hardware: Um SGBDD deve poder operar e acessar dados em uma variedade de plataformas de hardware. Um SGBDD verdadeiro não deve depender de uma determinada plataforma.

10. Independência de sistema operacional: Um SGBDD deve poder executar em sistemas operacionais diferentes. Assim como na regra anterior, um SGBDD não deve depender de um sistema operacional em especial.

11. Independência de rede: Um SGBDD deve ser projetado para executar independentemente do protocolo de comunicação e da topologia de rede usada para interligar os vários nós que fazem parte da rede.

12. Independência de SGBD*:** Um SGBDD ideal deve ser capaz de se comunicar com outros sistemas de gerenciamento de banco de dados, executando em diferentes nós mesmo sendo sistemas de banco de dados diferentes (heterogêneos). Todos estes sistemas devem usar APIs.

5.4. Vantagens de SGBDD em relação ao SGBD centralizado:

Compartilhamento de dados e controle distribuído: Existe um administrador global responsável pelo sistema como um todo, mas parte das responsabilidades são delegadas à administradores locais que gozam de certa autonomia.

Maior confiabilidade: O sistema funciona conforme o projeto.

Maior disponibilidade: O sistema está disponível por maior percentual de tempo.

Melhor desempenho no processamento de consultas: Sub-consultas podem ser executadas em paralelo.

Maior escalabilidade: É mais fácil acrescentar um nó desde que, os mesmos sejam autônomos, do que substituir um sistema centralizado existente por um maior.

5.5. Desvantagens de SGBDD em relação a SGBD centralizado:

Custo de desenvolvimento do software: A alta complexidade torna mais difícil implementar um SGBDD, tornando-o mais caro.

Grande potencial para bugs: Ocorrência de erros muito sutis na colaboração entre os nós do SGBDD.

Aumento do overhead de processamento: Devido a troca de mensagens e a computação adicional para obter a coordenação entre os nós.

Questões de projeto específicas: Por exemplo, replicação e fragmentação de dados.

Dificuldades para obter conhecimento global: Por exemplo, controle de concorrência entre transações distribuídas e detecção de deadlock.

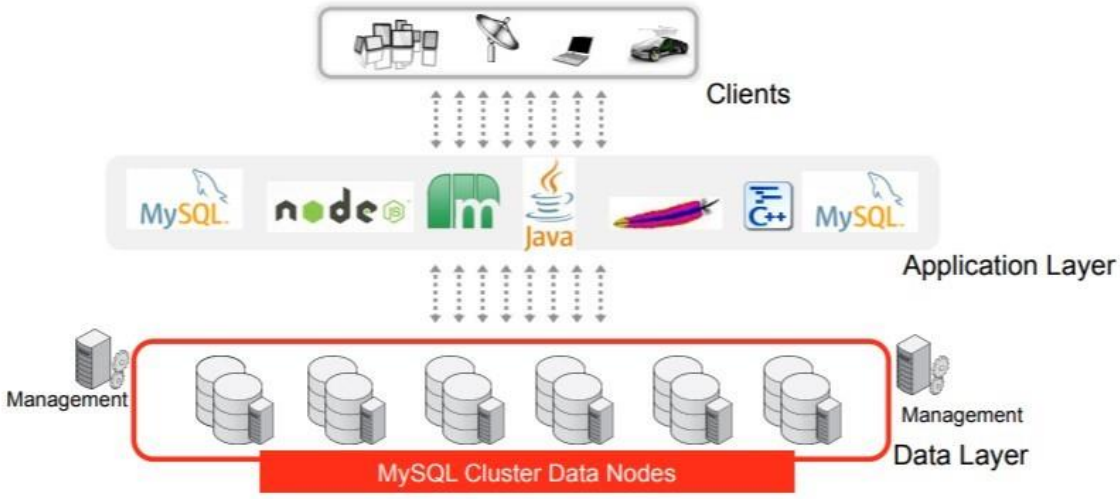


Figura 24: Exemplos de SGBDD.

5.6. Arquiteturas de Bancos de Dados Distribuídos

Inicialmente mostraremos brevemente as diferenças entre a arquitetura de banco de dados distribuídos e a arquitetura de banco de dados paralela.

5.6.1. Arquitetura paralela versus distribuída

Existem dois tipos de arquiteturas de sistemas multiprocessador que são mais utilizados:

• **Arquitetura de memória compartilhada (altamente acoplada):** Vários processadores compartilham o disco (memória secundária), assim como também compartilham a memória principal.

• **Arquitetura de disco compartilhado (livremente acoplada):** Vários processadores compartilham o disco (memória secundária), mas neste caso, cada um tem sua própria memória principal.

Os sistemas de gerenciamento de banco de dados que utilizam estas arquiteturas são chamados de **sistemas de gerenciamento de banco de dados paralelos**, pois usam a tecnologia de processadores paralelos.

Outro tipo de arquitetura multiprocessador é a **arquitetura nada compartilhada**, nela cada processador tem seu próprio disco e memória principal, não existindo memória comum.

A comunicação dos processadores é feita por uma rede de interconexão de alta velocidade (barramento ou switch). Vejamos a seguir um exemplo de arquitetura nada compartilhada.

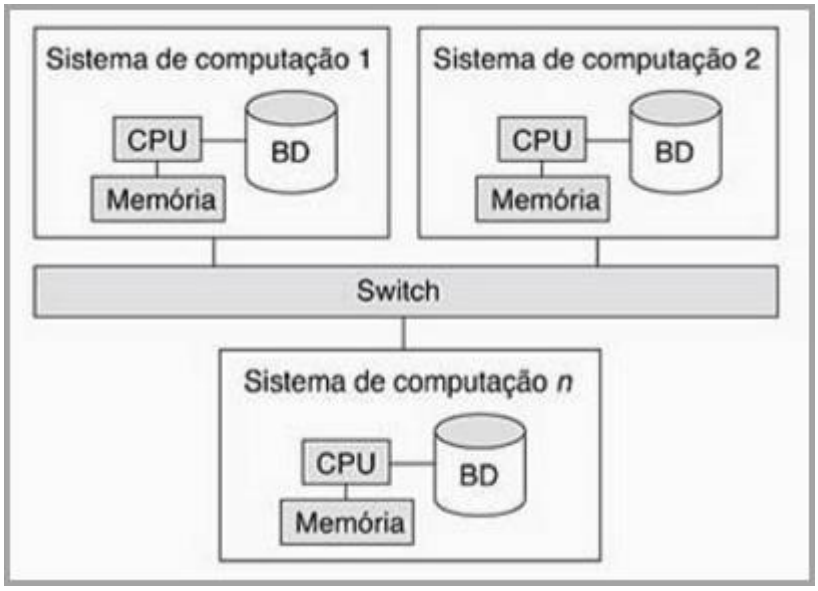


Figura 25: Exemplo de arquitetura nada compartilhado.

Embora a arquitetura nada compartilhado seja parecida com a arquitetura de banco de dados distribuídos existem diferenças no modo de operação. Na arquitetura nada compartilhado, existe simetria e homogeneidade de nós, algo que não ocorre na arquitetura de banco de dados distribuído, onde a heterogeneidade do hardware e do S.O. em cada nó é muito presente. Portanto a arquitetura nada compartilhada também é considerada um ambiente de banco de dados paralelos.

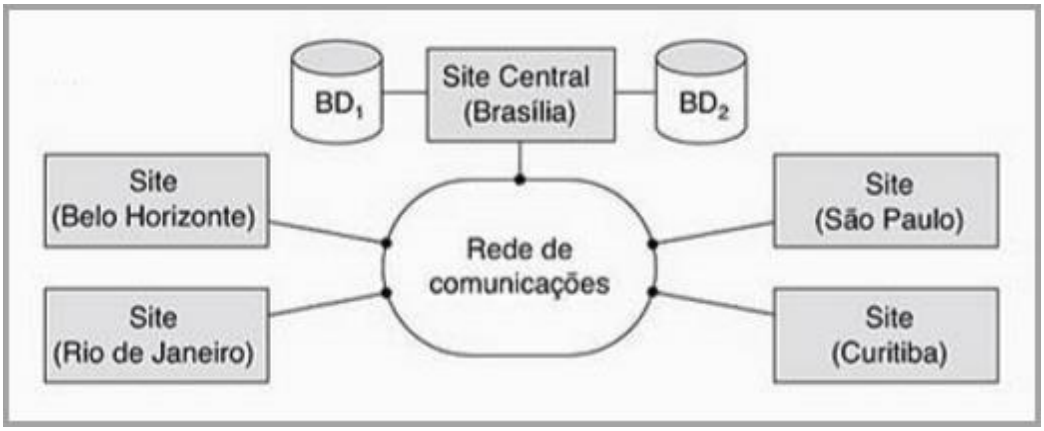


Figura 26: Banco de dados centralizado com acesso distribuído.

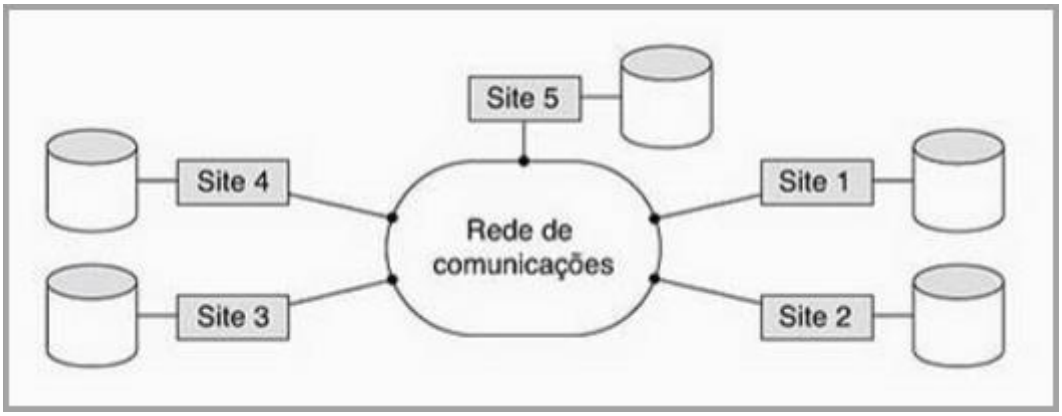


Figura 27: Banco de dados distribuído puro.

5.6.2. Arquitetura geral de bancos de dados distribuídos

A figura a seguir ilustra a **arquitetura geral de esquem**a de um banco de dados distribuído.

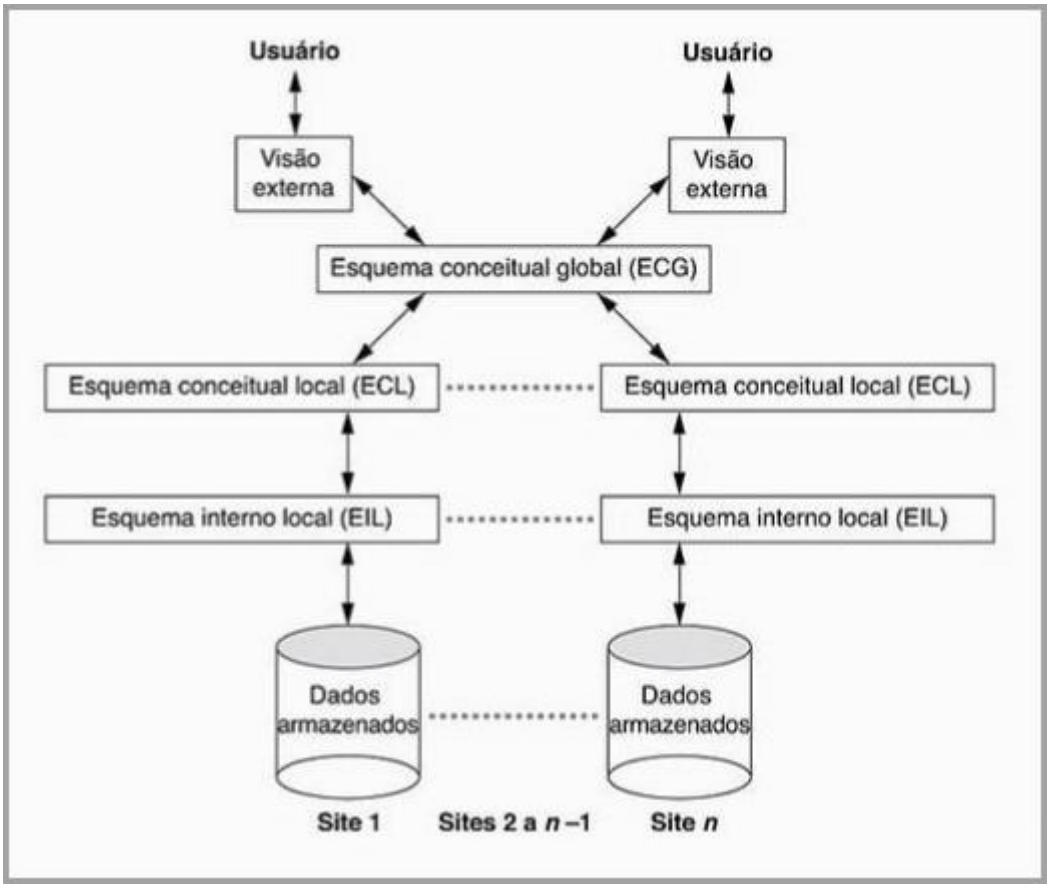


Figura 28: Arquitetura do esquema para BDD.

A visão é representada pelo **esquema conceitual global** (ECG), que oferece transparência de rede.

Já a organização lógica dos dados em cada nó é especificada pelo **esquema conceitual local** (ECL).

Para dar suporte a heterogeneidade em potencial no BDD, cada nó tem seu próprio **esquema interno local** (EIL) com base nos detalhes da organização local nesse nó em particular.

A seguir é apresentada uma figura que representa a **arquitetura de componentes** de um banco de dados distribuído.

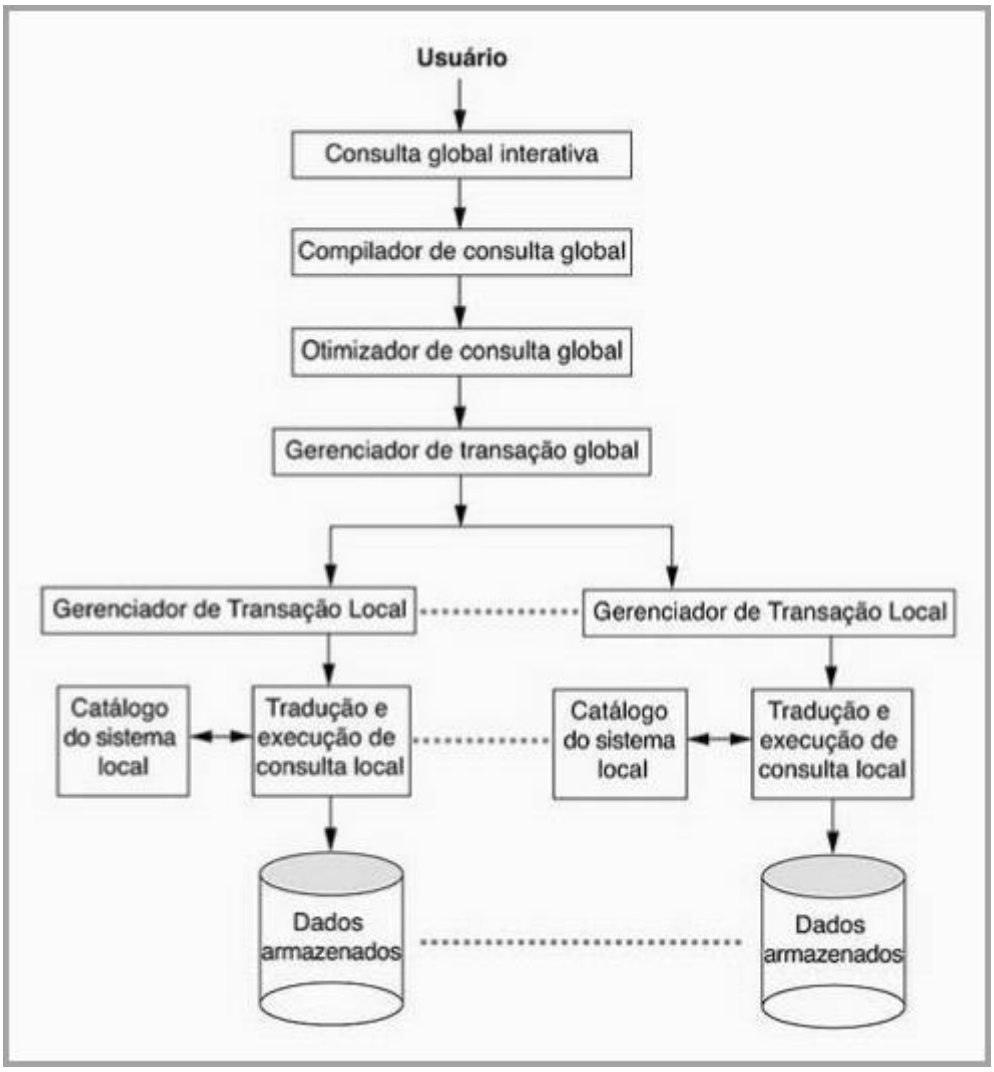


Figura 29: Arquitetura de componentes para BDD.

O **compilador de consulta global** faz referência ao esquema conceitual global, com base no catálogo global, verificando e impondo as restrições definidas.

O **otimizador de consulta global** faz referência aos esquemas conceituais globais e locais, com o objetivo de gerar consultas locais otimizadas tendo como base as consultas globais. Ele verifica todas as possíveis estratégias usando uma função de custo, estimando o custo com base no tempo de resposta e nos tamanhos estimados de resultados intermediários. Após calcular o custo para cada estratégia, o otimizador escolhe a estratégia com o menor custo para execução.

O **gerenciador de transação global** tem a função de coordenar a execução por vários sites em conjunto com o gerenciador de transação local dos mesmos.

5.6.3. Visão geral a arquitetura cliente-servidor de três camadas

A seguir é apresentada uma figura que ilustra a arquitetura cliente-servidor de três camadas, uma arquitetura comumente utilizada em aplicações web.

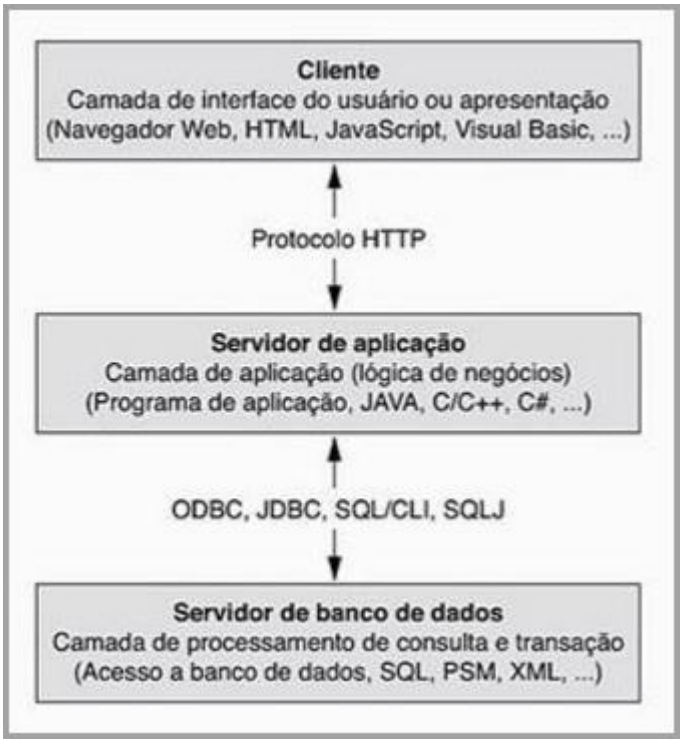


Figura 30: Arquitetura cliente-servidor de três camadas.

Camada de apresentação (cliente)

Camada de interface e interação com o usuário. Interfaces web ou formulários para o cliente com o objetivo de realizar a ligação com a aplicação. Para tal funcionalidade geralmente são utilizados navegadores web, e as linguagens e especificações incluem HTML, XHTML, CSS, entre outras.

Essa camada trata as entradas e saídas do usuário, aceitando comandos e exibindo informações. Quando uma interface web é utilizada para representar esta camada, geralmente utiliza o protocolo HTTP para comunicar-se.

Camada de aplicação (regras de negócio)

Camada responsável por programar a estrutura lógica da aplicação. As verificações de segurança, verificação de identidade e demais são realizadas nesta camada.

Esta camada pode interagir com um ou mais banco de dados conforme necessário.

Servidor de banco de dados

Camada responsável por tratar solicitações de consulta e atualização da camada de aplicação. Normalmente a SQL é usada para acesso ao banco de dados.

6. Referências Bibliográficas

ELMASRI, RAMEZ; NAVATHE, SHAMKANT. **Sistemas de banco de dados:** 6ª ed. São Paulo: Editora Pearson, 2011.