

**INSTITUTO
FEDERAL**
Santa Catarina

Banco de Dados 2



- 1) Transações
- 2) Otimização de consultas
- 3) Administração de banco de dados



1/3

Transações



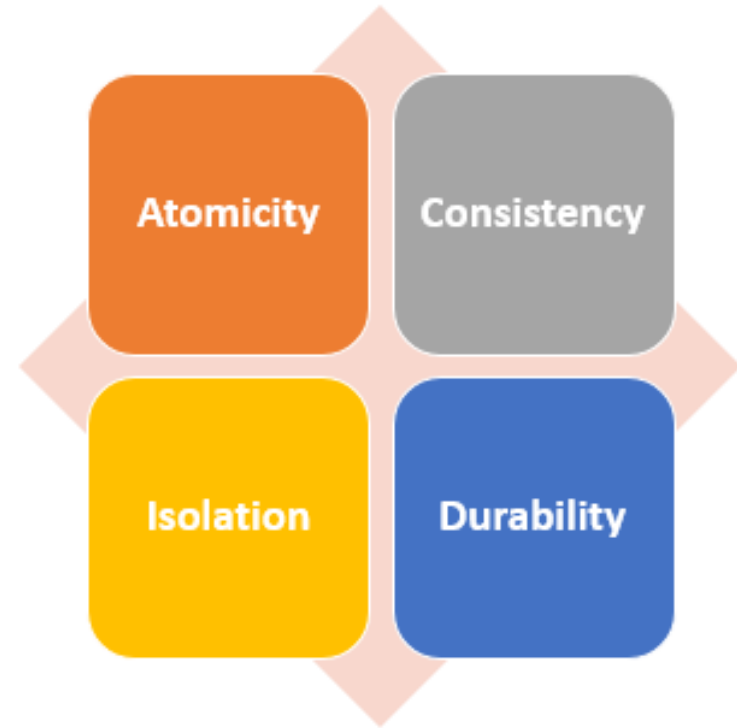
Transações

- Imagine uma transferência de dinheiro entre uma conta bancária e outra, as operações são
 - 1) Alterar o saldo da conta do debitado, retirando o valor solicitado.
 - 2) Alterar o saldo da conta do creditado, adicionando o valor solicitado.
- E se entre a primeira operação der algum problema no update e o SGBD não conseguir fazer o passo 2?

Transações

- Algumas definições:
 - Transação é uma unidade lógica de trabalho, envolvendo diversas operações de bancos dados.
 - É toda operação (ou conjunto de operações) cujo resultado final altera o banco de dados.
 - Uma transação SQL (transação) é uma sequência de execuções de instruções SQL que é atômica em relação à recuperação. Ou seja: ou o resultado da execução é completamente bem-sucedido ou não tem efeito em nenhum esquema SQL ou dados SQL.

- Os bancos de dados precisam ter confiabilidade em suas transações. ACID é um conceito que se refere às quatro propriedades de transação de um sistema de banco de dados: Atomicidade, Consistência, Isolamento e Durabilidade.



- **Atomicidade:** A transação deve ter todas as suas operações executadas em caso de sucesso ou, em caso de falha, nenhum resultado de alguma operação refletido sobre a base de dados.
- **Consistência:** A execução de uma transação deve levar o banco de dados de um estado consistente a um outro estado consistente, ou seja, uma transação deve respeitar as regras de integridade dos dados (como unicidade de chaves, restrições de integridade lógica, etc.).
- **Isolamento:** O isolamento é um conjunto de técnicas que tentam evitar que transações paralelas interfiram umas nas outras.
- **Durabilidade:** Os efeitos de uma transação em caso de sucesso devem persistir no banco de dados mesmo em casos de quedas de energia, travamentos ou erros.



Transações

- No MySQL uma transação básica possui a seguinte estrutura:

```
START TRANSACTION | BEGIN  
--instruções  
COMMIT | ROLLBACK
```


Transações

- Utilize o comando abaixo para informar que uma nova transação está sendo iniciada
- Adicione a sequência de instruções que deseja executar.

```
START TRANSACTION
```

```
-- instruções
```

Transações

- COMMIT é uma operação que confirma que ocorreu tudo bem
- Indica o término de uma transação bem-sucedida

```
START TRANSACTION | BEGIN  
--instruções  
COMMIT
```

Transações

- Caso algum problema aconteça, o ROLLBACK cancelará todas instruções da transação e retornará o BD ao estado anterior.
- Ou seja, assinala uma transação mal sucedida.

START TRANSACTION | BEGIN

--*instruções*

ROLLBACK

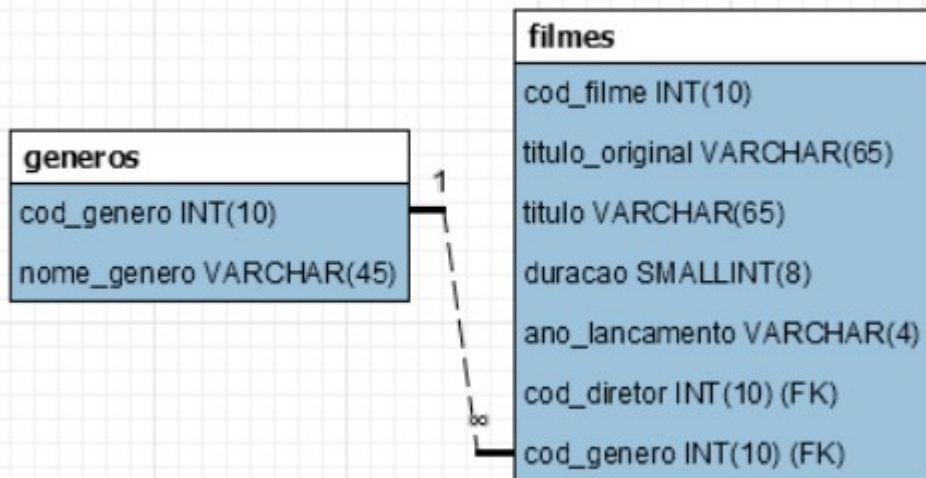
Transações implícitas

- Normalmente, os SGBDs possuem a propriedade ativa **AUTOCOMMIT = 1**
- Isso quer dizer que qualquer comando **SELECT**, **INSERT**, **UPDATE** e **DELETE** fará um **COMMIT** automaticamente.
- Se desativar este parâmetro, a transação passa a ser explícita, ou seja, o SGBD ficará esperando um comando **COMMIT** ou **ROLLBACK** para encerrar a transação.

Transações implícitas

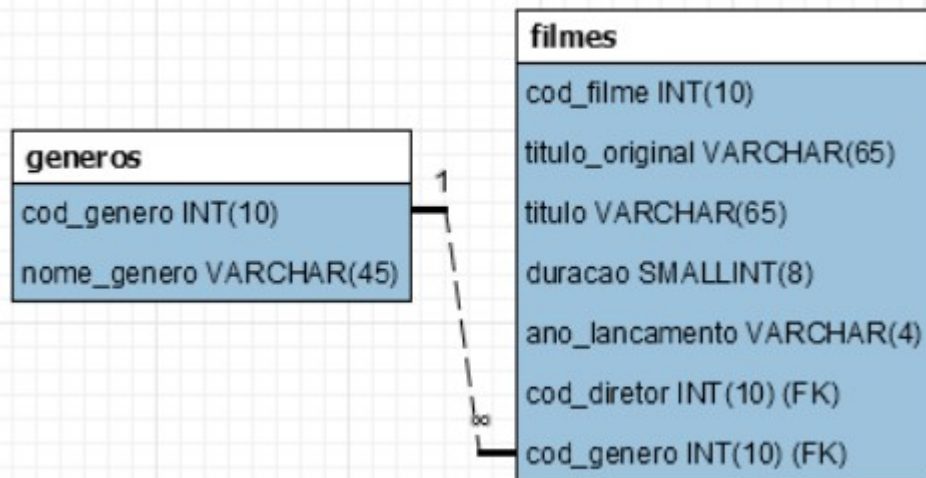
- Porém, quando você inicia uma transação explicitamente com `START TRANSACTION`, o SGBD aguardará o `COMMIT` ou `ROLLBACK` indiferente do `AUTOCOMMIT`.
- As operações `COMMIT` ou `ROLLBACK` são geralmente definidas através de códigos de programação em softwares ou no SGBD.
- Mas também são úteis em operações com alta concorrência.

Exemplo



- Considere o banco descrito ao lado, com alta concorrência.
- Você quer em sequência gravar um novo gênero, pegar o id gerado pelo **AUTO_INCREMENT** e utilizar na chave estrangeira **cod_genero** em filmes

Exemplo



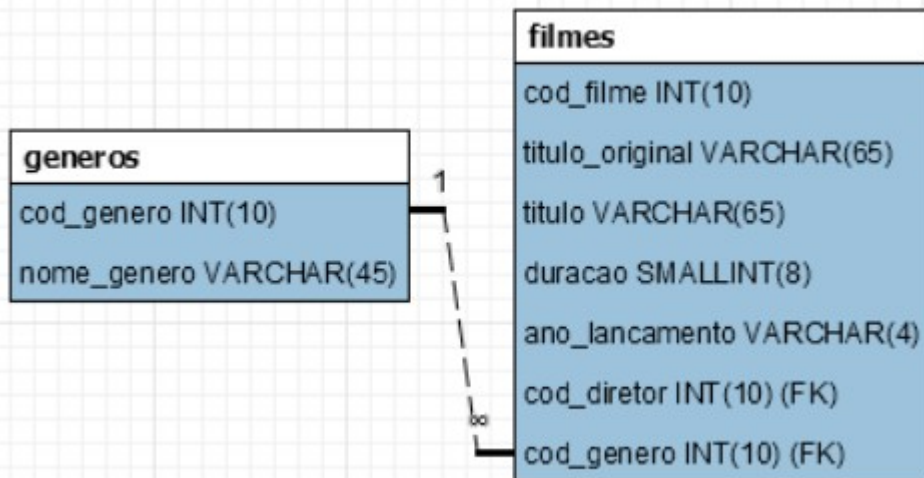
-- 1. inicia uma transação

START TRANSACTION;

-- 2. grava o novo gênero

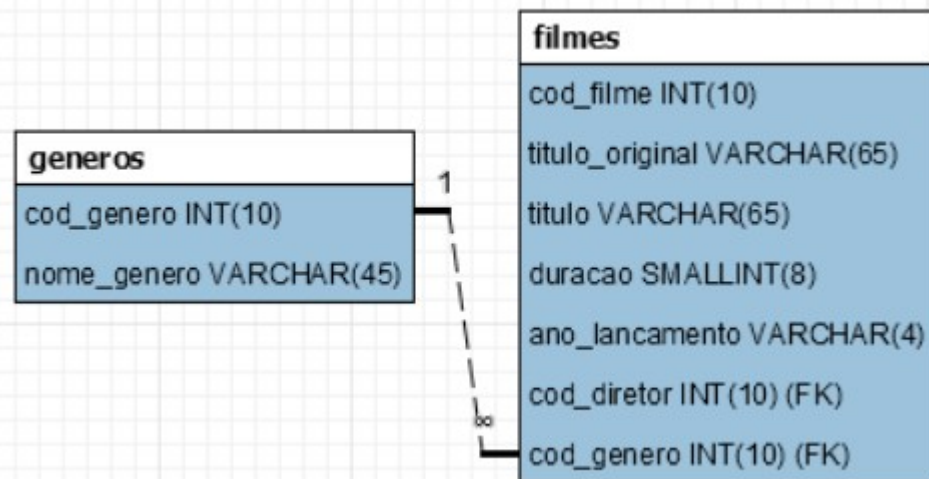
INSERT INTO generos (nome_genero)
VALUES ('Cultura Nerd');

Exemplo



```
-- 3. Seleciona o último id gerado  
SELECT  
    @genero_id:=LAST_INSERT_ID()
```


Exemplo

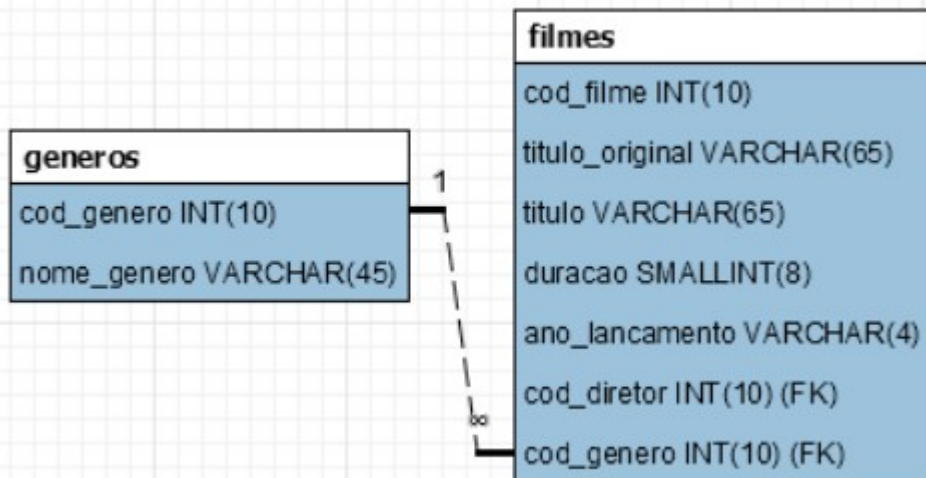


-- 4. Insere o filme

```
INSERT INTO filmes (titulo_original, titulo, ano_lancamento, cod_genero)
VALUES ('Jobs', 'Jobs', 2013, @genero_id);
```



Exemplo



```
-- 5. commit as mudanças  
COMMIT;
```

Qual a vantagem do uso de transações nessa sequência de operações?

-- 1. inicia uma transação

START TRANSACTION;

-- 2. grava o novo gênero

INSERT INTO generos (nome_genero)
VALUES ('Cultura Nerd');

-- 3. Salva o último id gerado

SELECT
@genero_id:=LAST_INSERT_ID()

-- 4. Insere o filme

INSERT INTO filmes (titulo_original, titulo,
ano_lancamento,cod_genero)
VALUES ('Jobs', 'Jobs', 2013, @genero_id);

-- 5. commit as mudanças

COMMIT;

2/3

Otimização de consultas



Materiais desta aula

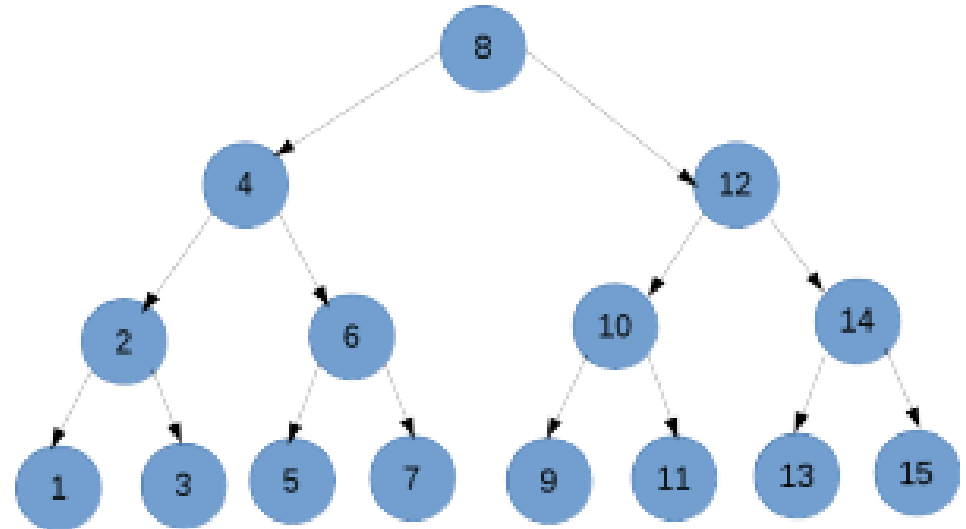
Para esta aula precisaremos do BD “world”

Otimização de Consultas

- A otimização de consultas ajuda a diminuir o gargalo do banco de dados em aplicações com muitos acessos ou em bancos com tabelas gigantes (muitos dados).
- Os principais métodos estão relacionados a:
 - Indexação de colunas
 - Melhores práticas do uso do SQL
 - Configurações no servidor

Indexação de colunas

- Este provavelmente é o método mais importante pois consegue obter os melhores resultados de forma simples e em pouco tempo
- Significa manter armazenada externamente a uma tabela uma de suas colunas ordenadas
- Não é a mesma coisa que manter a tabela ordenada por um campo, como a chave primária
- Permite ter várias ordenações na mesma tabela



Indexação de colunas

- Vamos conhecer o comando **EXPLAIN**
- Utilizando o banco de dados world, quantas linhas o SGBD precisou pesquisar para encontrar as cidades de Santa Catarina?

-- lista as cidades

```
SELECT * FROM city WHERE District='Santa Catarina'
```

-- etapas envolvidas na pesquisa

```
EXPLAIN SELECT * FROM city WHERE District='Santa Catarina'
```


- **id**
 - select_type
 - table
 - type
 - possible_keys
 - key
 - key_len
 - ref
 - rows
 - Extra
- O campo id serve apenas para identificar cada eventual linha do resultado

- id
- **select_type**
- table
- type
- possible_keys
- key
- key_len
- ref
- rows
- Extra

SIMPLE	Seleção simples
PRIMARY	Seleção primária
UNION	Segunda seleção de uma união
DEPENDENT_UNION	Segunda seleção de uma união secundária
UNION RESULT	Resultado de uma união
SUBQUERY	Primeira seleção de uma subconsulta
DEPENDENT SUBQUERY	Primeira seleção de uma subconsulta de uma consulta secundária
DERIVED	Consulta com origem em uma subconsulta

- id
- select_type
- **table**
- type
- possible_keys
- key
- key_len
- ref
- rows
- Extra

- O campo table exibe a tabela (ou tabelas) utilizadas

- id
- select_type
- table
- **type**
- possible_keys
- key
- key_len
- ref
- rows
- Extra

- Se o resultado for ALL, indica que nenhuma otimização aconteceu e todas linhas tiveram de ser lidas.

- id
 - select_type
 - table
 - type
 - possible_keys
 - key
 - key_len
 - ref
 - rows
 - Extra
- possible_keys mostra as colunas e índices candidatos à utilização na consulta analisada
 - Key mostra, entre as candidatas, qual foi a coluna utilizada na otimização
 - key_len indica o tamanho da coluna-chave utilizada (quantos bytes), a seguir, ref a referência que é usada como valor-chave.

- id
- select_type
- table
- type
- possible_keys
- key
- key_len
- ref
- **rows**
- **Extra**

- Rows é um dos parâmetros mais importantes e indica quantas linhas possivelmente esta consulta precisará processar.
- Por fim, extra reportará informações extras de forma simplificada

Para saber mais detalhes:

[https://mariadb.com/kb/en/explain /](https://mariadb.com/kb/en/explain/)

Indexação de colunas

- Vamos supor que este banco pertence a um site onde é normal exibir a lista de cidades pelo seu distrito.
- Agora já sabemos que cada vez que a página é carregada, a tabela inteira deve ser lida.

-- lista as cidades

```
SELECT * FROM city WHERE District='Santa Catarina'
```

-- etapas envolvidas na pesquisa

```
EXPLAIN SELECT * FROM city WHERE District='Santa Catarina'
```

Indexação de colunas

- Agora vamos adicionar um índice nesta coluna.
- A seguir, rode novamente a query de pesquisa e verifique quantas linhas foram utilizadas.
- Teste o SELECT com outros estados.

```
-- criar um indice  
CREATE INDEX indice_01 ON city(District);  
  
-- etapas envolvidas na pesquisa  
EXPLAIN SELECT * FROM city WHERE District='Santa Catarina'
```


Indexação de colunas

- Embora seja útil, a criação de índices deve ser feita com cautela.
- Índices aumentam muito o custo de operações de INSERT, UPDATE e DELETE pois precisam reorganizar toda informação a cada mudança.
- Abaixo está o comando par apagar um índice.

```
-- apagar um indice  
DROP INDEX index_name ON tbl_name
```

Indexação de colunas

- Uma das operações mas custosas em um BD é pesquisar palavras ou substrings em colunas de texto utilizando wildcards

```
SELECT * FROM city WHERE city.Name LIKE 'sao%' OR city.District LIKE 'sao%'
```

Indexação de colunas

- Uma forma de trazer alta eficiência em pesquisa textual no seu banco é utilizar índices FULLTEXT, fazendo com que cada palavra de uma ou mais colunas sejam indexadas

```
ALTER TABLE city ADD FULLTEXT(Name, District);
```

Indexação de colunas

- Utilize o match against para fazer uma pesquisa de cidades e estados com “Sao”:

```
SELECT * FROM city  
WHERE MATCH (NAME, District) AGAINST ('sao');
```

Indexação de colunas

- Desta forma não funciona com substrings porém é mais personalizável, como no exemplo abaixo, que testa palavras chaves para comparar com as informações do BD

```
SELECT * FROM city  
WHERE MATCH (NAME, District)  
AGAINST ('tem são paulo no nome?' IN NATURAL LANGUAGE MODE);
```

Indexação de colunas

- No modo boolean você pode personalizar palavras chave tornando algumas obrigatórias ou excludentes

```
SELECT * FROM city
WHERE
  MATCH (NAME, District)
  AGAINST ('proibido -são mas deve ter +paulo no nome' IN BOOLEAN MODE);
```

Indexação de colunas

- A pesquisa abaixo permite procurar palavras que começam com “sant” mas não tenham “catarina” na informação.
- Ou seja, é possível trabalhar com substring utilizando o operador “*”

```
SELECT * FROM city
WHERE
  MATCH (NAME, District)
  AGAINST ('sant* -catarina' IN BOOLEAN MODE);
```

Dicas finais de otimização

- Índices
 - Não abuse
 - Recomendado em colunas envolvidas em:
 - WHERE ou JOIN
 - GROUP BY, ORDER BY
 - DISTINCT
 - MIN() e MAX()
- Evite junções externas quando não for necessário
- Utilize cache em bancos com pouca atualização.

```
SHOW variables LIKE 'have_query_cache';
```

```
SHOW variables LIKE '%query_cache%';
```


Dicas finais de otimização

- Analise o custo das consultas da seção com profile (no MariaDB)

```
-- ativa o profile
SET profiling = 1;

-- faça algumas consultas de teste...
SELECT * FROM city ORDER BY Name LIMIT 3;

-- verifique as queries
SHOW PROFILES;

-- analise uma especificamente
SHOW PROFILE FOR QUERY 1;
```

3/3

Administração de banco de dados



Privilégios de acesso

- Um recurso importante num programa de gerenciamento de banco de dados é a capacidade de criação e a manutenção das contas dos usuários.
- O usuário root é criado automaticamente quando o SGBD é instalado mas deve ser evitado seu uso.
- Os SGBD permitem a configuração de privilégios diferentes para usuários diferentes.



Privilégios de acesso

- O Sistema de Privilégios do MySQL utilizam tabelas, responsáveis por diferentes itens de segurança.
- É possível controlar não só quais usuários terão acesso ao servidor mas também a quais bancos e tabelas, e quais operações poderão realizar em cada.



Criando um usuário

- Um usuário recém criado não terá **nenhum** privilégio.
- Os comandos iniciais de manipulação de usuários são:

-- utilizando o BD mysql, para verificar todos usuários:

```
SELECT User, Host FROM mysql.user;
```

-- criar um novo usuário

```
CREATE USER miguel IDENTIFIED BY 'abc123';
```

-- apagar um usuário

```
DROP USER miguel
```

Criando um usuário

- Os nomes de conta têm tanto o nome de usuário quanto o nome do host, e são especificados como 'user_name'@'host_name'. Se o nome do host não for fornecido, assume-se que é '%'.
 - Os nomes de host podem corresponder tanto nomes de domínio quanto endereços IP. Use 'localhost' como o nome do host para permitir somente conexões do cliente local.

-- utilizando o BD mysql, para verificar todos usuários:

```
SELECT User, Host FROM mysql.user;
```

-- criar um novo usuário

```
CREATE USER miguel@ifsc.edu.br IDENTIFIED BY 'abc123';
```

-- apagar um usuário

```
DROP USER miguel@ifsc.edu.br
```

Liborando acessos para um usuário

- Antes de começar, nunca forneça mais acesso que um usuário precisa.
- Ainda que seja possível manipular as permissões com insert, update, delete diretamente na tabela de permissão, o ideal é utilizar os comandos **GRANT** e **REVOKE**, exclusivos para estes casos.
- O comando abaixo exhibe o que um usuário pode fazer.

```
-- para ver os privilégios de um usuário:  
SHOW GRANTS FOR 'root'@'localhost';  
  
SHOW GRANTS FOR 'miguel'@'%';
```

Privilégios de um usuário

PRIVILÉGIO	DESCRIÇÃO
ALL/ ALL PRIVILEGES	Todos privilégios
ALTER	Permite alterar tabelas
ALTER ROUTINE	Permite alterar e excluir stored procedures / functions
CREATE	Permite criar bancos de dados e tabelas
CREATE ROUTINE	Permite criar stored procedures / functions
CREATE TEMPORARY TABLES	Permite criar tabelas temporárias
CREATE USER	Permite criar e gerenciar usuários
CREATE VIEW	Permite criar e gerenciar visões
DELETE	Permite excluir informações

Privilégios de um usuário

PRIVILÉGIO	DESCRIÇÃO
DROP	Permite excluir estruturas
EVENT	Permite criar event schedulers
FILE	Permite ler e escrever arquivos no sistema
GRANT	Permite conceder novos privilégios a usuários
INDEX	Permite o gerenciamento de índices
INSERT	Permite inserir informações na tabela
LOCK TABLES	Permite bloquear tabelas para uso
PROCESS	Permite visualizar e encerrar processos do Mysql
RELOAD	Permite recarregar bancos de dados

Privilégios de um usuário

PRIVILÉGIO	DESCRIÇÃO
REPLICATION CLIENT	Permite solicitar uma replicação
REPLICATION SLAVE	Permite replicar suas informações
SELECT	Permite realizar consultas SQL
SHOW DATABASES	Permite visualizar a estrutura de todos BDs
SHOW VIEW	Permite visualizar os códigos de criação de visões
SHUTDOWN	Permite desligar o servidor MySQL
SUPE	Permite configurar os dados dos servidor Master no caso de replicação
TRIGGER	Permite criar e gerenciar triggers
UPDATE	Permite alterar informações em tabelas
USAGE	Permite apenas utilizar comandos básicos (faz nada)

Utilizando comando GRANT

-- *sintaxe*

```
GRANT type_of_permission ON database_name.table_name TO 'username'@'localhost';
```

-- *atualiza os privilégios*

```
FLUSH PRIVILEGES;
```




OPÇÃO	DESCRIÇÃO
.	Define privilégio de acesso global
banco.*	Define privilégio de acesso a qualquer tabela deste banco
banco.tabela	Define privilégio de acesso a uma determinada tabela do banco

Mais informações e comandos: <https://mariadb.com/kb/pt-br/grant/>

Utilizando comando REVOKE

```
-- sintaxe  
REVOKE type_of_permission ON database_name.table_name FROM 'username'@'localhost';
```



FROM no lugar de TO

OPÇÃO	DESCRIÇÃO
,	Define privilégio de acesso global
banco.*	Define privilégio de acesso a qualquer tabela deste banco
banco.tabela	Define privilégio de acesso a uma determinada tabela do banco

Mais informações e comandos: <https://mariadb.com/kb/pt-br/grant/>

Exemplos de GRANT

- O comando GRANT também é capaz de criar um usuário já concedendo permissões.

```
-- cria o usuário1 no localhost e concede privilégios completo de uso com status GRANT OPTION
    permitindo que ele possa conceder novos privilégios a outros usuários
GRANT ALL PRIVILEGES ON *.* TO usuario1@localhost IDENTIFIED BY 'abc123' WITH GRANT OPTION;

-- cria o usuário2 para qualquer host (%) e concede privilégios completo de uso com status GRANT OPTION
GRANT ALL PRIVILEGES ON *.* TO usuario2 IDENTIFIED BY 'abc123' WITH GRANT OPTION;

-- cria um usuário chamado usuario3 no servidor localhost com senha de acesso abc123 e informa que esse
    usuário pode apenas acessar as colunas Name e District da tabela city do banco de dados world.
GRANT SELECT (Name, District) ON world.city TO usuario3@localhost IDENTIFIED BY 'abc123';
```