Clonar tabelas existentes

Antes de trabalhar com a alteração ou manipulação de tabelas existentes, existe uma operação que é extremamente útil e simples: clonar tabelas. Para clonar uma tabela, é preciso fazer apenas um comando:

Copiar

```
-- Sintaxe:
CREATE TABLE nome_para_nova_tabela LIKE tabela_a_ser_clonada;

-- Exemplo:
CREATE TABLE actor_clone LIKE sakila.actor;
```

Ao fazer isso, você terá criado uma tabela com a estrutura exatamente igual (chave primária, chave estrangeira, tipos, restrições etc.) usando apenas uma linha de código!
Pontos de Atenção

- Esse comando não copia os dados, apenas a estrutura;
- Caso não especifique qual banco de dados utilizar, a nova tabela será inserida no banco que estiver ativo no momento da execução. Sendo assim, sempre especifique o banco de dados antes.

Copiar

```
USE nome_do_banco_de_dados;
CREATE TABLE nome_para_nova_tabela LIKE tabela_a_ser_clonada;
```

Pequena pausa

Faça o seguinte: Clone alguma tabela do banco de dados `sakila` para ver na prática o resultado do comando acima.

O que é e como lidar com uma `VIEW`

Agora que você já sabe como clonar a estrutura de uma tabela, imagine ter que ficar montando a mesma query repetidamente. Deve haver uma maneira mais ágil de fazer isso, não é?
Claro, pode-se criar uma `VIEW` .

O que é uma `VIEW` ?

Uma `VIEW` é nada mais nada menos que uma tabela temporária no seu banco de dados, que pode ser consultada como qualquer outra. Porém, por ser uma tabela temporária, ela é criada a partir de uma query que você definir.
Uma `VIEW` te permite:

- Ter uma tabela que pode ser usada em relatórios;
- Ter uma tabela para usar como base para montar novas queries;
- Reduzir a necessidade de recriar queries utilizadas com frequência.

Anatomia de uma `VIEW`

Copiar

```
-- Defina em qual banco a view será criada
USE nome_do_banco_de_dados;
-- Comando para criar a view
CREATE VIEW nome_da_view AS query;
```

Um exemplo de uso

Suponha que a gerência quer ter uma maneira simples para sempre saber quem são os top 10 clientes que mais compram com a empresa. Pode-se criar uma view para resolver isso!

Copiar

```
CREATE VIEW top_10_customers AS
    SELECT c.customer_id, c.first_name, SUM(p.amount) AS
total_amount_spent
    FROM sakila.payment p
    INNER JOIN sakila.customer c ON p.customer_id = c.customer_id
    GROUP BY customer_id
    ORDER BY total_amount_spent DESC
    LIMIT 10;
```

Agora, caso alguém precise ter acesso a essa informação, você pode consultar a tabela temporária (`VIEW`) diretamente, sem a necessidade de montar uma nova query.

Copiar

```
SELECT * FROM top_10_customers;
```

customer_id	first_name	total_amount_spent
526	KARL	221.55
148	ELEANOR	216.54
144	CLARA	195.58
137	RHONDA	194.61
178	MARION	194.61
459	TOMMY	186.62
469	WESLEY	177.60
468	TIM	175.61
236	MARCIA	175.58
181	ANA	174.66

Resultado de `SELECT * FROM top_10_customers;`

Para excluir uma `VIEW` , use o seguinte comando:

Copiar

```
DROP VIEW nome_da_view;
```

Tudo que você deve saber sobre o ALTER TABLE

Algo extremamente comum durante o ciclo de desenvolvimento de software é a necessidade constante de fazer melhorias na estrutura do banco de dados. As tabelas são uma dessas estruturas que podem sofrer alterações.

Ao executar o bloco de código abaixo, a tabela `noticia` será criada. Essa tabela será utilizada como exemplo para testar modificações em sua estrutura.

Copiar

```
USE sakila;
CREATE TABLE noticia(
    noticia_id INT PRIMARY KEY,
    titulo VARCHAR(100),
    historia VARCHAR(300)
) engine = InnoDB;
```

Abaixo, algumas das alterações que podem ser feitas em uma tabela.

Copiar

```
-- Adicionar uma nova coluna
ALTER TABLE noticia ADD COLUMN data_postagem date NOT NULL;

-- Modificar o tipo e propriedades de uma coluna
```

```sql
ALTER TABLE noticia MODIFY noticia_id BIGINT;

-- Adicionar incremento automático a uma coluna
-- (especifique o tipo da coluna + auto_increment)
ALTER TABLE noticia MODIFY noticia_id BIGINT auto_increment;

-- Alterar o tipo e nome de uma coluna
ALTER TABLE noticia CHANGE historia conteudo_postagem VARCHAR(1000)
NOT NULL;

-- Dropar/Excluir uma coluna
ALTER TABLE noticia DROP COLUMN data_postagem;

-- Adicionar uma nova coluna após outra
ALTER TABLE noticia ADD COLUMN data_postagem DATETIME NOT NULL AFTER
titulo;
```

Com os comandos acima, foram cobertas as operações mais comuns que você deve saber para alterar uma tabela. Para confirmar se a estrutura da sua tabela foi alterada corretamente, você pode usar o comando `SHOW COLUMNS FROM nome_da_tabela;` . Veja o exemplo abaixo:

Copiar

```sql
SHOW COLUMNS FROM sakila.noticia;
```

Field	Type	Null	Key	Default	Extra
noticia_id	bigint(20)	NO	PRI	NULL	auto_increment
titulo	varchar(100)	YES		NULL	
data_postagem	datetime	NO		NULL	
conteudo_postag...	varchar(1000)	NO		NULL	

Visualizando a estrutura da tabela `noticia`

DROPando uma tabela

Para excluir uma tabela existente, você pode utilizar o comando `DROP TABLE` .

Copiar

```sql
DROP TABLE nome_da_tabela;
```

Ponto Importante

Você não conseguirá dropar (excluir) uma tabela que é referenciada por uma restrição de chave estrangeira. A chave estrangeira ou a tabela que a contém deve ser excluída antes.
Por exemplo, tente dropar a tabela `sakila.language` com o comando abaixo:

Copiar

```
DROP TABLE sakila.language;
```

Ao executar o comando, você verá que ele não funciona, retornando a seguinte mensagem de erro:
Error Code: 3730. Cannot drop table 'language' referenced by a foreign key constraint 'fk_film_language' on table 'film'
Isso acontece em função de as informações da tabela `language` serem utilizadas na tabela `film` . Caso tente dropar a tabela `film` , você perceberá que ela também possui restrições. Essas restrições estão relacionadas ao conceito de integridade referencial , que deve ser considerado quando se cria um banco de dados. Elas têm o intuito de evitar que tabelas sejam excluídas acidentalmente.
Integridade referencial : Propriedade que afirma que todas as referências de chaves estrangeiras devem ser válidas.
Então, lembre-se: nem todas as tabelas podem (ou devem) ser dropadas diretamente. É necessário avaliar as restrições existentes naquela tabela para entender o que pode ser feito e como deve ser feito, caso precise excluí-la.

Como usar um INDEX

Quando se fala em otimização de queries, o termo índice (ou `INDEX`) pode vir a ser mencionado como solução para problemas de performance. Mas o que são índices, e quando se deve usá-los?

```
-- Criando um índice em uma coluna
CREATE [INDEX | FULLTEXT INDEX | UNIQUE INDEX] nome_indice
ON tabela (coluna);

-- Criando um índice composto, em duas ou mais colunas
CREATE [INDEX | FULLTEXT INDEX | UNIQUE INDEX] nome_indice
ON tabela (coluna1, coluna2);

-- Excluindo índices
DROP INDEX nome_do_indice ON tabela;
```

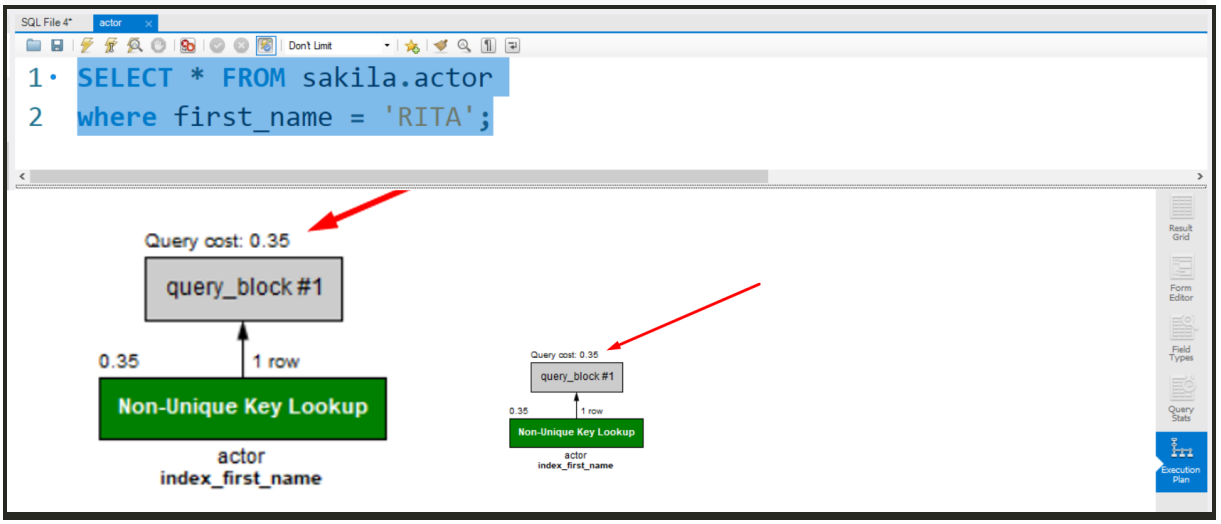
Entenda o impacto do `INDEX`

Para entender o impacto de um `INDEX` , hora de comparar o antes e o depois da adição de um `INDEX` à coluna `first_name` da tabela `sakila.actor` e verificar seu impacto no custo de uma query.
Execute o comando abaixo para criar um índice na coluna `first_name` dentro da tabela `actor` .

Copiar
```
CREATE INDEX index_first_name ON sakila.actor(first_name);
```

Execute a query abaixo e verique seu custo através do execution plan.

Copiar
```
SELECT *
FROM sakila.actor
WHERE first_name = 'RITA';
```



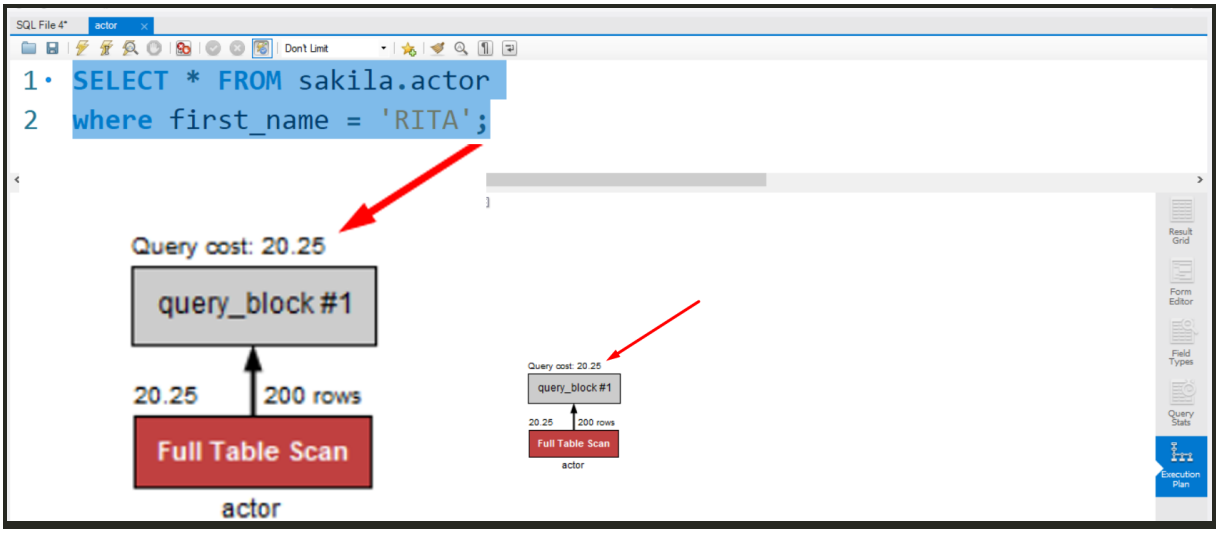
Custo da query `SELECT * FROM sakila.actor WHERE first_name = 'RITA'` com índice

Agora, exclua o índice para fazer a comparação:

Copiar
```
DROP INDEX index_first_name ON sakila.actor;
```

Veja o custo da mesma query, quando executada sem um índice na coluna `first_name` :

Copiar
```
SELECT *
FROM sakila.actor
WHERE first_name = 'RITA';
```



Custo da query `SELECT * FROM sakila.actor WHERE first_name = 'RITA'` sem índice

Como se vê, neste caso o índice possui uma melhor performance.

Entenda o impacto do `FULLTEXT INDEX`

Hora de fazer outro exemplo para analisar o impacto que um `FULLTEXT INDEX` , em conjunto com uma full-text search, , possui na performance de uma query. Para esse exemplo será alterada a coluna `address` da tabela `sakila.address` . Veja a criação do índice logo abaixo:

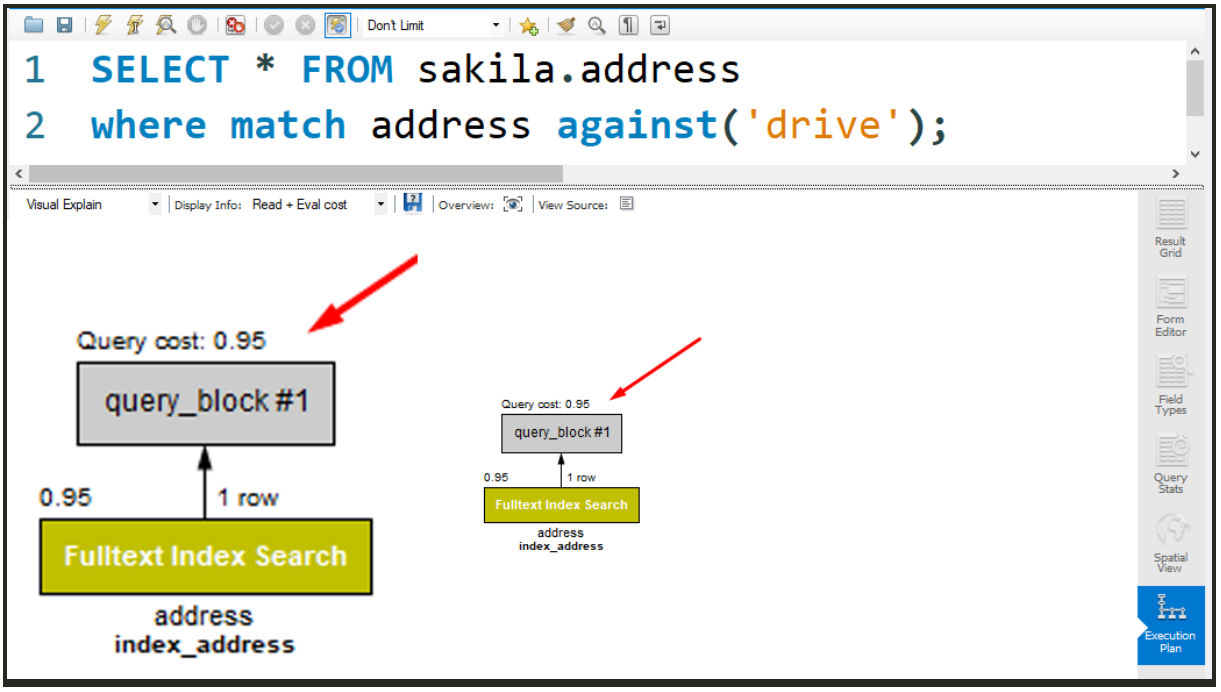
Copiar

```
CREATE FULLTEXT INDEX index_address ON sakila.address(address);
```

Para verificar a diferença na performance, deve-se utilizar os comandos `MATCH` e `AGAINST` , conforme foi visto anteriormente no texto sobre full-text search. . Execute a query abaixo e verifique seu custo através do execution plan:

Copiar

```
SELECT *
FROM sakila.address
WHERE MATCH(address) AGAINST('drive');
```



Custo da query `SELECT * FROM sakila.address WHERE MATCH(address) AGAINST('drive')` com índice em `address`

Agora, exclua o índice para fazer a comparação:

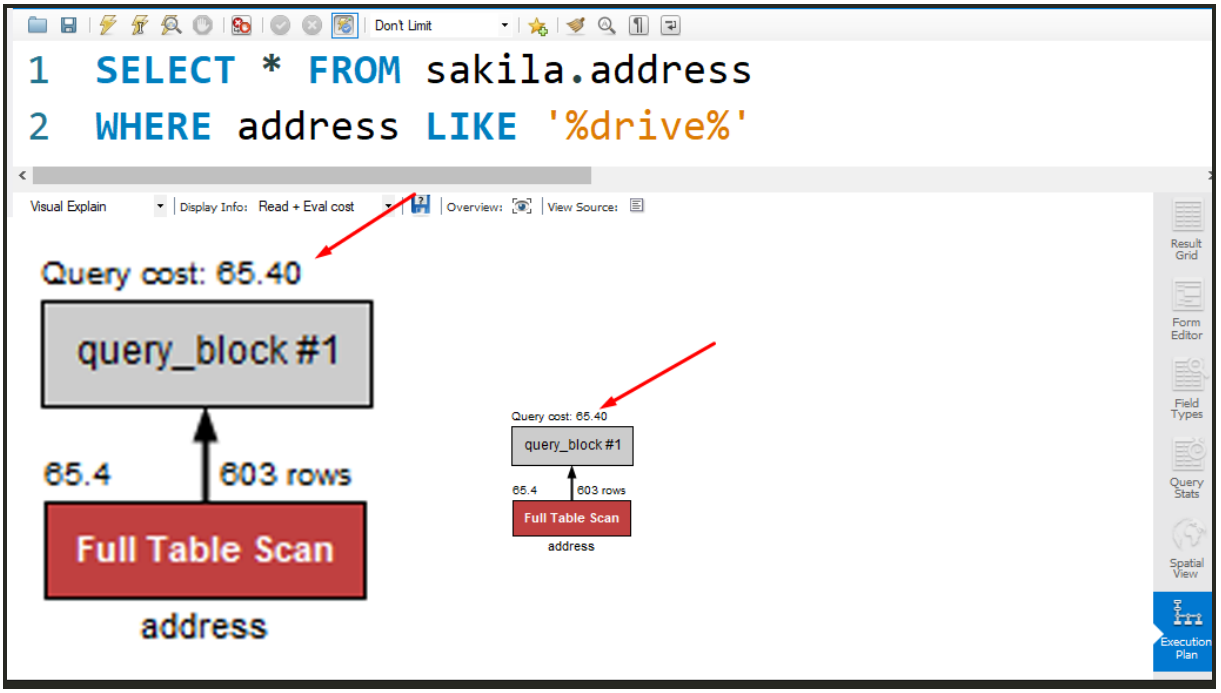
Copiar

```
DROP INDEX index_address ON sakila.address;
```

Veja o custo da query, quando executada sem um índice na coluna `address`:

Copiar

```
SELECT *
FROM sakila.address
WHERE address LIKE '%drive%';
```



Custo da query `SELECT * FROM sakila.address WHERE address LIKE '%drive%'` sem índice

Novamente, houve uma melhoria na performance.

Entenda o impacto do `UNIQUE INDEX`

A Sintaxe para *criar* um `UNIQUE INDEX` é a seguinte:

Copiar

```
CREATE UNIQUE INDEX nome_do_indice ON nome_tabela(nome_coluna);
```

Para *dropar* (excluir), pode-se usar:

Copiar

```
DROP INDEX nome_do_indice ON nome_tabela;
```

O `UNIQUE INDEX` é utilizado em uma coluna para, principalmente, prevenir a duplicação de dados em uma tabela e, secundariamente, melhorar a performance de busca.

Colunas que fazem uso dessa restrição podem receber valores *nulos* . É importante lembrar também que a restrição `PRIMARY KEY` , quando aplicada a uma coluna, insere por padrão as restrições `UNIQUE INDEX` + `NOT NULL` naquela coluna.

Logo, pode-se entender que a `PRIMARY KEY` também é um `UNIQUE INDEX` que não permite valores *nulos* .

Isso pode ser confirmado usando o comando `SHOW INDEX` , que lista os detalhes sobre um índice em uma tabela. Veja abaixo um exemplo de uso do comando.

Copiar

```
SHOW INDEX FROM sakila.actor;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name
actor	0	PRIMARY	1	actor_id
actor	1	idx_actor_last_name	1	last_name

Coluna `actor_id`, que possui um `UNIQUE INDEX`

Como se pode ver acima, embora nenhum índice tenha sido criado ainda, todas as colunas do banco de dados que usam a restrição `PRIMARY KEY` possuem internamente um `UNIQUE INDEX` . Isso pode ser confirmado na coluna `actor_id` pelo `Non_Unique = 0` , que quer dizer que a coluna possui um índice único.

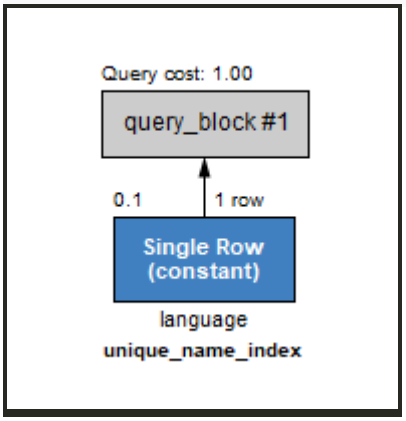
Um exemplo de uso do `UNIQUE INDEX`

Hora de verificar a performance de uma query antes de inserir um `UNIQUE INDEX` na coluna `name` da tabela `language` .

Copiar

```
CREATE UNIQUE INDEX unique_name_index ON sakila.language(name);
```

```
SELECT *
FROM sakila.language
WHERE name = 'Mandarin';
```

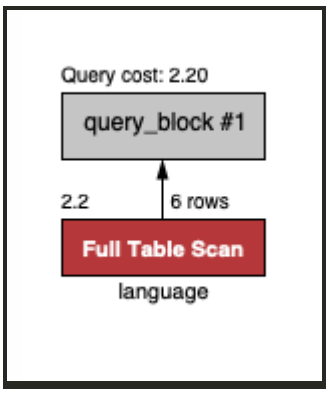


Resultado da busca COM uso do `UNIQUE INDEX`

Copiar

```
DROP INDEX unique_name_index ON sakila.language;
```

```
SELECT * FROM sakila.language
WHERE name = 'Mandarin';
```



Resultado da busca SEM uso do `UNIQUE INDEX`

Aqui, mais uma vez, teve-se uma melhoria na performance.

Quando não utilizar índices

Mesmo notando que os resultados foram favoráveis para o uso de índices nesses exemplos, é importante ressaltar que eles nem sempre devem ser utilizados. Abaixo, segue uma lista das situações em que o uso de índices deve ser evitado:

- Em tabelas pequenas, pois a diferença de performance será mínima, se houver;
- Em colunas que retornarão uma grande quantidade dados quando filtradas. Por exemplo, você não adicionaria os artigos "o" e "a" ao índice de um livro;
- Em tabelas que frequentemente têm atualizações em grande escala, uma vez que a performance dessas atualizações será afetada;
- Em colunas que são frequentemente manipuladas, haja vista que a manutenção do índice dessa coluna pode demandar muito tempo quando feita em excesso;
- Em colunas que possuem muitos valores nulos.