2 – Introdução

- Um dos principais objetivos de um sistema gerenciador de banco de dados é garantir que as informações ali armazenadas estejam seguras e totalmente íntegras.

- O banco de dados de uma organização precisa ser cuidadosamente projetado, visto que as informações estarão de alguma forma interligadas entre si no que diz respeito ao processo do negócio como um todo.

- Por exemplo, um cadastro de fornecedores estará de alguma forma se comunicando com o cadastro de produtos, afinal os produtos pertencem a um fornecedor. Caso um fornecedor que possua ligação com um produto seja excluído, teremos um produto que não estará integro. Assim, a integridade de uma entidade pode ter impacto diretamente em outra entidade.

- Os sistemas gerenciadores de banco de dados possuem uma série de recursos que permitem aos profissionais garantir a integridade, segurança e eficiência do banco de dados.

2.1 – Índices

- Quando o usuário insere um registro, o banco de dados não tenta colocar os dados em um local específico dentro da tabela. O servidor insere e posiciona o registro na primeiro localização disponível.

- Como exemplo, caso um registro seja inserido na tabela de clientes, o servidor não irá posicionar a linha inserida em ordem numérica por meio da coluna “id_cliente”, e nem em ordem alfabética por meio da coluna “nome”.

- Desta forma, como o registro é inserido no primeiro espaço disponível, toda vez que uma consulta for feita em uma tabela, o servidor precisará inspecionar toda a tabela, deixando o processo mais lento e consumindo mais recursos.

2.1 – Índices

- Supondo que o usuário execute um comando SELECT buscando por todos os clientes que possuem o nome iniciando com a letra “A”. Como podem existir clientes com essa característica em qualquer posição da tabela, o servidor terá que analisar todas as linhas da tabela.

- Para uma tabela com poucos registro, o método de varrer toda a tabela (conhecido também como **table scan**) funciona bem, mas em tabelas com maior quantidade de registros, o consumo computacional será muito grande, possuindo um alto tempo de resposta.

- Para que o banco de dados possa ter mais agilidade em operações de consulta, principalmente em tabelas de maior tamanho, um recurso chamado “índice” pode ser utilizado.

- A ideia de índice no banco de dados é similar ao de índice de um livro. Supondo que um leitor deseja achar um capítulo específico do livro, ao invés de passar folha por folha, ele pode buscar no índice do livro (que se encontra em ordenado) a página que lhe interessa, é rapidamente encontrar o conteúdo.

2.1 – Índices

- Da mesma forma que uma pessoa usa o índice do livro para buscar conteúdos em específico, o servidor de banco de dados utiliza os índices para encontrar linhas em uma tabela.

- Índices de banco de dados são tabelas especiais que, diferentemente de tabelas normais, são mantidas em uma ordem específicas.

- No entanto, ao invés de armazenar todos os dados de uma tabela, os índices armazenam de forma ordenada, apenas os dados de uma coluna (ou colunas) que será utilizada para buscar registros da tabela, juntamente com informações que descrevem onde as linhas estão localizadas fisicamente.

- Desta forma, caso seja criado um índice para a coluna “nome” da tabela de clientes, o servidor irá manter de forma ordenada todos os nomes dos clientes cadastrados, bem como a localização física de cada registro da tabela.

2.1 – Índices

- Quando uma busca na tabela de clientes for feita, ao invés de procurar na tabela de clientes, o servidor irá efetuar esta busca no índice, que está ordenado, localizando o registro de forma muito mais rápida.

- O índice é uma recurso extremamente importante na busca de dados, sendo fundamental em tabelas com grande quantidade de dados.

- Como os índices são ordenados, não existe mais a necessidade do servidor efetuar uma varredura em toda a tabela, economizando recursos e apresentando respostas mais rápidas ao usuário.

- Deve-se criar índices apenas nas colunas que são usadas com maior frequência para busca de registros na tabela. Não se deve criar índices sem necessidade, pois eles também consomem recursos computacionais.

2.1 – Índices

- Apesar de inúmeras vantagens, os índices devem ser usados com cautela, pois seus custo computacional é alto, e se não utilizado corretamente pode trazer mais desvantagens do que vantagens.

- A desvantagem mais evidente consiste no gasto de espaço em disco, tendo em vista que as tabelas de índices ocupam um espaço de armazenamento considerável para manter os dados ordenados.

- Outra desvantagem consiste no comprometimento das operações de escrita, que tendem a ficar mais lentas em virtude da necessidade de reorganização do índice após algum INSERT, DELETE ou UPDATE.

- Desta forma, é importante que apenas os campos que serão usados como parâmetros de operações SELECT sejam indexados.

2.1.1 – Criação de Índices

- O exemplo abaixo mostrará o código necessário para para criar um índice na coluna “nome” da tabela de clientes. Desta forma, o servidor irá criar uma tabela (índice) contendo de forma ordenada todos nomes dos clientes, bem como a localização física de cada registro:

Sintaxe

ALTER TABLE <nome da tabela> **ADD INDEX** <nome do índice> **(<**nome da coluna>**)**

Exemplo da Tabela de Cliente

ALTER TABLE cliente **ADD INDEX** idx_nome_cliente **(**nome**)**

Observação: A sintaxe acima é para o MYSQL (que considera o índice como parte da tabela). Em outros banco de dados (como Oracle e SQL SERVER), os índices são independentes, por isso são criados com o comando (A partir da versão 5.0, o MYSQL também aceita este comando, embora mapeie para o ALTER TABLE):

CREATE INDEX <nome do índice> **ON** <nome da tabela> (<nome da coluna>)

2.1.1 – Criação de Índices

- Mesmo criando um índice para uma coluna, existe a possibilidade deste recurso não ser usado em todas as operações de consulta. O otimizador de consultas irá analisar se a utilização do índice tornará o tempo de resposta menor.

- Os índices possuem um custo inicial, desta forma, em tabelas com poucos registros, pode ser mais viável efetuar buscas linha por linha, ao invés de utilizar a tabela de índices.Pode acontecer de uma tabela possuir mais de um índice, nesse caso o otimizador de consultas do servidor de banco de dados irá analisar qual será a melhor alternativa.

- Quando se define a chave primária de uma tabela, automaticamente um índice é criado para a coluna. Este procedimento é feito pelo banco, para agilizar a busca por chaves repetidas, principalmente na inserção, visto que o banco deve garantir que a chave seja única.

- Na definição de chave estrangeira, o banco de dados também cria um índice, visando agilizar busca e junções nas tabelas relacionadas.

2.1.2 – Consulta de Índices

- O comando **SHOW INDEX** retorna os índices da tabela, que no caso da tabela de clientes existe na coluna “nome”, na chave primária “id_cliente” (a chave primária é automaticamente indexada) e na chave estrangeira (as chaves estrangeiras, quando relacionadas, criam automaticamente um índice).

- O exemplo abaixo mostra o código para visualizar os índices de uma tabela no servidor de banco de dados MYSQL:

```
SHOW INDEX FROM cliente
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
cliente	0	PRIMARY	1	id_cliente	A	13	*NULL*	*NULL*		BTREE		
cliente	1	id_cidade	1	id_cidade	A	13	*NULL*	*NULL*	YES	BTREE		
cliente	1	idx_cliente_nome	1	nome	A	13	*NULL*	*NULL*		BTREE		

2.1.3 – Exclusão de Índices

- Caso seja observado que o índice criado para determinada coluna não esta sendo eficiente, é possível efetuar exclusão do mesmo, para isso utiliza-se o comando abaixo:

Sintaxe

ALTER TABLE <nome da tabela> **DROP INDEX** <nome do índice>

Exemplo da Tabela de Cliente

ALTER TABLE cliente **DROP INDEX** idx_nome_cliente

Observação: Para outros banco de dados, como Oracle e SQL SERVER (O MYSQL a partir de sua versão 5.0 também aceita), o comando para exclusão de índices é:

DROP INDEX idx_cliente_nome; (Oracle e MySQL)

DROP INDEX idx_cliente_nome **ON** cliente; (SQL Server)

2.2 – Índices Exclusivos (UNIQUE)

- As restrições **UNIQUE** impõem exclusividade dos valores de uma coluna, ou seja, duas linhas de uma tabela não podem ter o mesmo valor nas colunas. A chave primária também impõe a exclusividade, porém as chaves primárias não aceitam valor nulo.

- Uma restrição UNIQUE precisa ser bem definida para não causar ineficiência nos registros armazenados, visto que é importante ter em mente que a restrição de PRIMARY KEY pode já ter sido implementada na tabela e não seria necessária mais uma restrição de UNIQUE.

- Um bom exemplo de uso dessa restrição seria uma tabela de cadastro de clientes, onde existe o campo CPF, que mesmo não sendo definido como chave primária, possui valor único.

- Desta forma, se for inserida uma restrição UNIQUE na coluna CPF, dois clientes jamais poderão ter o mesmo número, garantindo que um cliente não seja cadastrado em duplicidade e, principalmente, que um cliente não use CPF de outro cliente.

2.2 – Índices Exclusivos (UNIQUE)

- Um índice exclusivo, além de possuir todas as vantagens de um índice comum, ele também serve como um mecanismo de desativação de valores duplicados na coluna indexada. Sempre que um linha é inserida ou a coluna indexada é alterada, o servidor verifica o índice exclusivo para analisar se o valor já existe em outra linha da tabela. A sintaxe para índices exclusivos é:

Sintaxe

ALTER TABLE <nome da tabela> **ADD UNIQUE** <nome do índice>(<campo>)

Exemplo da Tabela de Cliente

ALTER TABLE cliente **ADD UNIQUE** idx_cpf_cliente(cpf)

Observação: Para outros banco de dados, como Oracle e SQL SERVER (O MYSQL a partir de sua versão 5.0 também aceita), o comando para criar o índice exclusivo é:

CREATE UNIQUE INDEX idx_cpf_cliente **ON** cliente(cpf);

2.3 – Índices em Múltiplas Colunas

- Além dos índices de coluna única, é possível a implementação de índices que se estendam por múltiplas colunas. Desta forma, as duas colunas ficam armazenadas em um único índice.

- Nos índices múltiplos, deve-se selecionar muito bem o campo que será colocado na primeira posição, pois este será o índice principal. Os demais serão apenas índices complementares.

- No caso do exemplo da tabela de clientes, quando uma consulta for feita, o otimizador de consultas primeiro irá efetuar a busca pelo índice da coluna “nome” (índice principal). Entretanto, como diversos clientes podem ter o mesmo nome, então o índice da coluna “telefone” pode ser usada para otimizar o tempo de resposta.

- O exemplo abaixo ilustra o código para a criação de um índice duplo das colunas nome e telefone da tabela de clientes:

```
ALTER TABLE cliente ADD INDEX  idx_nome_telefone_cliente(nome, telefone)
```

2.4 – Tipos de Índices

- A indexação é uma ferramenta poderosa, mas como existem diversos tipos de dados, uma única estratégia de indexação pode nem sempre ser eficiente. Existem diversos algoritmos e tipos para indexar as tabelas, veremos os principais:

1. B-tree: Também conhecido como *índice de árvore balanceada*, o índice B-tree é o padrão de indexação do banco de dados MySQL, Oracle e SQL Server, ou seja, se não for especificado o método no comando SQL, o servidor irá utilizar índices B-tree.

Os índices B-tree são organizados como árvores, com um ou mais níveis de nós-galhos (estruturas de localização) e nós-folhas (dados). Os nós galhos são usados para navegar pela árvore, e os nós-folhas armazenam os dados.

2.4 – Tipos de Índices

- A figura abaixo mostra um exemplo de índice B-tree para a coluna nome de uma tabela de clientes. De acordo com os nomes buscados nas consultas, o servidor navega pela árvore.

- Conforme os registros vão sendo atualizados, o servidor tenta manter a árvore equilibrada, para que não haja mais galhos/folhas de um lado da raiz do que do outro.

2.4 – Tipos de Índices

- Os índices B-tree são padrão do MySQL, logo se não for especificado nenhum outro tipo de indexação, o B-tree será usado automaticamente.

- O comando SQL abaixo ilustra uma forma de criar o índice especificando que o servidor use a estrutura B-tree:

```
ALTER TABLE cliente ADD INDEX idx_cliente_nome USING BTREE(nome)
```

2.4 – Tipos de Índices

2.Hash: É um tipo de índice menos usado, por ser considerado mais lento. Para que o hashing funcione adequadamente, uma série de configurações mais complexas precisam ser feitas. O comando abaixo ilustra uma forma de criar um índice do tipo hash:

```
ALTER TABLE cliente ADD INDEX idx_cliente_nome USING HASH(nome)
```

2.4 – Tipos de Índices

- O sistema gerenciador de banco de dados MySQL possui diversas storage engines, e não são todos os tipos de indexação que são aceitos pelas principais engines (MyISAM e InnoDB).

- A tabela abaixo mostra os tipos de índices permitidos para cada engine:

Storage Engine	Permissible Index Types
MyISAM	BTREE
InnoBD	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	HASH, BTREE

2.5 – Índices Fulltext

- A partir da versão 3 do MySQL, foi desenvolvido um mecanismo de índices chamado **fulltext**, capaz de efetuar buscas textuais com maior precisão. Inicialmente, este tipo de indexação era possível apenas para tabelas MyISAM. A partir da versão 5.6 do MySQL, a engine InnoDB implementou este recurso.

- A indexação **fulltext** é um recurso mais poderoso que o LIKE, pois, além de ordenar o resultado pela similaridade semântica, oferece mais opções para filtragem na consulta.

- O comando LIKE apenas traz resultados iguais ao parâmetro passado na cosulta, sendo que uma consulta fullText analisa a semântica completa dos registros. Aplicações com grande massa de texto que precisam efetuar pesquisas baseadas na relevância são candidatas ao uso de índices fulltext.

- O exemplo muito comum pode ser de uma biblioteca virtual, onde o usuário busca por palavras chaves, e não por um termo exato existente nos registros.

2.5 – Índices Fulltext

- Os índices fulltext podem ser usados em colunas do tipo **CHAR, VARCHAR e TEXT**. A sintaxe para criação de uma ou mais colunas com indexação fulltext é mostrada no quadro abaixo:

```
ALTER TABLE produto ADD FULLTEXT idx_produto_descricao (descricao)

ou

CREATE FULLTEXT INDEX idx_produto_descricao ON cliente (descricao) ;
```

2.5 – Índices Fulltext

- Para efetuar a pesquisa através de um índice fulltext utilizamos as funções **MATCH** e **AGAINST**, que recebem o nome dos campos e o valor a ser pesquisado, respectivamente.

- Os índices fulltext podem ser usados em colunas do tipo **CHAR, VARCHAR e TEXT**.

- Para cada registro, o MySQL atribui um valor de relevância, que representa a similaridade da string de pesquisa com a linha em questão. Um valor de relevância 0 (zero) significa nenhuma semelhança, fazendo com que o registro não seja exibido.

- O cálculo de relevância é feito através de um algoritmo projetado para pesquisa em grandes massas de texto, tornando a busca inadequada para pequenas tabelas.

- Entre as variáveis que são levadas em consideração nesse cálculo, o MySQL considera o número de palavras encontradas em cada campo do índice, o número de palavras encontradas por linha, o número de ocorrências da mesma palavra em todas as linhas, entre outros.

2.5 – Índices Fulltext

- Como é possível observar na consulta abaixo, uma seleção é feita na tabela de produto, onde é utilizado um índice fulltext para localizar registros que tenham sua descrição similar à string 'Monitor SANSUNG '.

- Como o índice fulltext faz uma análise semântica dos registros, palavras similares também são inseridas no conjunto resposta. A sintaxe para criação de uma ou mais colunas com indexação fulltext é mostrada no quadro abaixo:

```
SELECT id_produto, descricao FROM produto
WHERE MATCH (descricao) AGAINST ('Monitor SAMSUNG');
```

id_produto	descricao
4	Monitor LG
3	Monitor Sansung

2.5 – Índices Fulltext

- O servidor MySQL atribui um valor de relevância para cada linha, sendo possível averiguar este valor inserindo comandos MACTH e AGAINST on local de declaração dos campos.

- No caso do exemplo abaixo, podemos observar um valor maior de relevância para a string que mais se parece com a que foi passada como parâmetro na consulta:

```
SELECT id_produto, descricao, MATCH (descricao) AGAINST ('Monitor SAMSUNG') AS
Relevância FROM produto
WHERE MATCH (descricao) AGAINST ('Monitor SAMSUNG');
```

id_produto	descricao	Relevância
3	Monitor Samsung	2.2508163452148438
4	Monitor LG	0.6852666139602661

2.6 – Índices Fulltext em Modo Booleano

- A partir da versão 4.0.1, o MySQL disponibiliza o recurso de pesquisa fulltext com parâmetros booleanos, aumentando significativamente o poder na construção de filtragens de texto.

- A pesquisa booleana tem como base a manipulação de strings de acordo com alguns operadores. Veja a lista dos operadores disponíveis:

+: A string deve estar presente em todos os registros retornados;

-: A string não deve estar presente nos registros retornados;

*: Trabalha com parte da palavra a ser procurada;

“ ”: Retorna a string entre aspas duplas exatamente da maneira como foi digitada;

( ): Agrupa palavras em sub-expressões;

< >: Muda a contribuição da string no cálculo da relevância. O operador < decrementa a relevância e o operador > aumenta a relevância;

~: Age como operador de negação. A contribuição de relevância da string se torna negativa.

2.6 – Índices Fulltext em Modo Booleano

- O exemplo abaixo ilustra uma situação em que se deseja selecionar registros que possuem a palavra 'Monitor", mas não possua a palavra Samsung:

```
SELECT id_produto, descricao FROM produto
WHERE MATCH (descricao) AGAINST('+Monitor -Samsung' IN BOOLEAN MODE);
```

id_produto	descricao
4	Monitor LG

2.7 – Como os Índices são Usados

- Os índices costumam ser usados pelo servidor para rapidamente localizar linhas em uma tabela específica. Feito isso, o servidor visita a tabela associada para buscar os dados adicionais solicitados pelo usuário.

- Considerando que uma tabela pode ter vários índices, e que o índice tem um custo inicial para ser executado, não sendo vantajoso para tabelas pequenas, o servidor de banco de dados possui um otimizador de consultas, que analisa a melhor forma de execução.

- Para verificar a forma de execução de uma consulta, deve-se utilizar o comando **EXPLAIN** na frente do SELECT, A coluna **KEY** no quadro abaixo mostra o nome do índice usado na consulta, que no caso se chama “nom”:

```
EXPLAIN SELECT * FROM `cliente` WHERE nome LIKE 'P%' ORDER BY nome;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	cliente	range	nome	nome	52	*NULL*	3	Using where

2.8 – Desvantagem dos Índices

- É um erro muito comum de profissionais sem muita experiência criar índices em diversas colunas da mesma tabela. Entretanto, índices em excesso certamente vão mais atrapalhar do que ajudar.

- Para cada índice criado, uma tabela auxiliar (tabela de índice) que manterá armazenado de forma ordenada todos os registros da coluna. Desta forma, além do espaço em disco ocupado, para cada operação de escrita na tabela (INSERT, DELETE e UPDATE), o servidor precisará atualizar todos os índices, de forma a mantê-lo ordenado.

- Manter os índices atualizados é uma operação custosa para o servidor de banco de dados. Desta forma, quando uma tabela possui índices demais, a tendência é que operações de escrita fiquem mais lentas.

- Os índices devem ser criados apenas nas colunas que serão usadas com frequência em operações de **busca, junção, agrupamento, agregação e ordenação**.

2.9 – Restrições

- Uma restrição (constraint) é simplesmente uma limitação colocada em uma ou mais colunas de uma tabela. Existem vários tipos diferentes de restrições:

1. Restrições de Chave Primária: Garantem a exclusividade dos valores da chave primária.

2. Restrições de Chave Estrangeira: Garante que uma chave estrangeira tenha um vlor correspondente na tabela de origem. Além disso permitem outras regras, como restrição de exclusão e atualização em cascata.

3. Restrições Exclusivas (UNIQUE): Restringe uma ou mais colunas para que tenham valores exclusivos.

4. Restrições de Verificação (CHECK): Restringem os valores aceitos por uma coluna.

2.9.1 – Restrições de Chave Primária

- Alguns sistemas de banco de dados permitem que tabelas sejam criadas sem chave primária, mas esta pratica deve ser evitada. Toda tabela deve possuir um campo definido como chave primária, desta forma o servidor garantirá que não existam valores repetidos.

- A chave primária pode ser declarada na criação da tabela ou após a sua criação, com os comandos abaixo:

```
CREATE TABLE `cliente` (
`id_cliente` int(11) NOT NULL,
`nome` varchar(50) NOT NULL,
`endereco` varchar(50),
`id_cidade` int(11) NOT NULL,
CONSTRAINT PK_Cliente PRIMARY KEY (id_cliente));
```

```
ALTER TABLE cliente ADD (CONSTRAINT PK_Cliente PRIMARY KEY (id_cliente));
```

2.9.2 – Restrições de Chave Estrangeira

- As restrições de chave estrangeira garantem a consistência entre os relacionamentos das tabelas do banco de dados. Alguns desenvolvedores possuem o péssimo hábito de não relacionar as tabelas, deixando para fazer isso no código da aplicação.

- As restrições de chave estrangeira devem ser feitas no banco de dados, pois desta forma, além de agilizar operações de JOIN, o servidor também garantirá que valores relacionados não sejam apagados, ou sejam apagados em cascata se for o caso.

- Feito a restrição de chave estrangeira, o servidor validará se os valores inseridos em campos relacionados existem na tabela pai. Não existindo, o banco de dados não permitirá a inserção.

- Por todos os motivos apresentados acima, tabelas que possuem chave estrangeiras devem ser relacionadas, para que o banco possa garantir a integridade das informações com uma boa eficiência.

2.9.2 – Restrições de Chave Estrangeira

- As restrições de chave estrangeira podem ser feitas no momento de criação da tabela ou após sua criação. No caso do exemplo abaixo, é feito a restrição de chave estrangeira entre as tabelas de cliente e cidade por meio do campo “id_cidade”. Neste caso, foi usado o parâmetro **ON DELETE RESTRICT,** de forma que se um usuário tentar excluir uma cidade que esteja sendo usada na tabela de cliente, o banco não permitirá (deleção restrita):

```
CREATE TABLE `cliente` (
 `id_cliente` int(11) PRIMARY KEY,
 `nome` varchar(50) NOT NULL,
 `endereco` varchar(50),
 `id_cidade` int(11) NOT NULL,
  CONSTRAINT FK_Cidade FOREIGN KEY(id_cidade) REFERENCES cidade(id_cidade));
```

```
ALTER TABLE cliente ADD (CONSTRAINT FK_Cidade
FOREIGN KEY(id_cidade) REFERENCES cidade(id_cidade));
```

2.9.2 – Restrições de Chave Estrangeira

- No exemplo abaixo foi feita uma restrição de chave estrangeira com o parâmetro **ON DELETE CASCADE** (deleção em cascata), desta forma, caso uma cidade seja excluída, todos os clientes que estiverem relacionados a ela serão automaticamente excluídos:

```
CREATE TABLE `cliente` (
  `id_cliente` int(11) PRIMARY KEY,
  `nome` varchar(50) NOT NULL,
  `endereco` varchar(50),
  `id_cidade` int(11) NOT NULL,
   CONSTRAINT FK_Cidade FOREIGN KEY(id_cidade)
   REFERENCES cidade(id_cidade) ON DELETE CASCADE);
```

```
ALTER TABLE cliente ADD (CONSTRAINT FK_Cidade
FOREIGN KEY(id_cidade) REFERENCES cidade(id_cidade) ON DELETE CASCADE);
```

2.9.2 – Restrições de Chave Estrangeira

- Em uma restrição de chave estrangeira, não será permitido que seja alterada a chave primária da tabela pai, pois isto afetaria o relacionamento com as tabelas filhas. Entretanto, é possível utilizar o parâmetro **ON UPDATE CASCADE** (atualização em cascata), que irá atualizar todos os registros das tabelas filhas:

```
CREATE TABLE `cliente` (
  `id_cliente` int(11) PRIMARY KEY,
  `nome` varchar(50) NOT NULL,
  `endereco` varchar(50),
  `id_cidade` int(11) NOT NULL,
   CONSTRAINT FK_Cidade FOREIGN KEY(id_cidade)
   REFERENCES cidade(id_cidade) ON UPDATE CASCADE);
```

```
ALTER TABLE cliente ADD (CONSTRAINT FK_Cidade
FOREIGN KEY(id_cidade) REFERENCES cidade(id_cidade) ON UPDATE CASCADE);
```

2.10 – Restrições e Índices

- A criação de restrições as vezes envolve a criação de índices, com o objetivo de melhorar o desempenho.

- Por exemplo, se não houver um índice para a coluna que é chave primária da tabela, toda vez que uma inserção for feita, o servidor precisará varrer toda a tabela para averiguar se o valor inserido na chave primária é único.

- Os servidores de banco de dados se comportam de maneira distinta na criação de índices para as restrições, como mostra a tabela abaixo:

Tipo de Restrição	MySQL	SQL Server	Oracle
Chave Primária	Gera Índice	Gera Índice	Gera Índice
Chave Estrangeira	Gera Índice	Não Gera Índice	Não Gera Índice
Exclusivas (Unique)	Gera Índice	Gera Índice	Gera Índice