
MySQL®

JOINS

- Uma cláusula JOIN em SQL, correspondente a uma operação de junção em álgebra relacional, combina colunas de uma ou mais tabelas em um banco de dados relacional

- Um JOIN é um meio de combinar colunas de uma ou mais tabelas, usando valores comuns a cada uma delas

- O SQL padrão ANSI especifica cinco tipos de JOIN: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN e CROSS JOIN.

Marcio Michelluzzi – marciomichelluzzi@gmail.com

JOINS

- Para exemplificar vamos tomar por base estas duas tabelas:

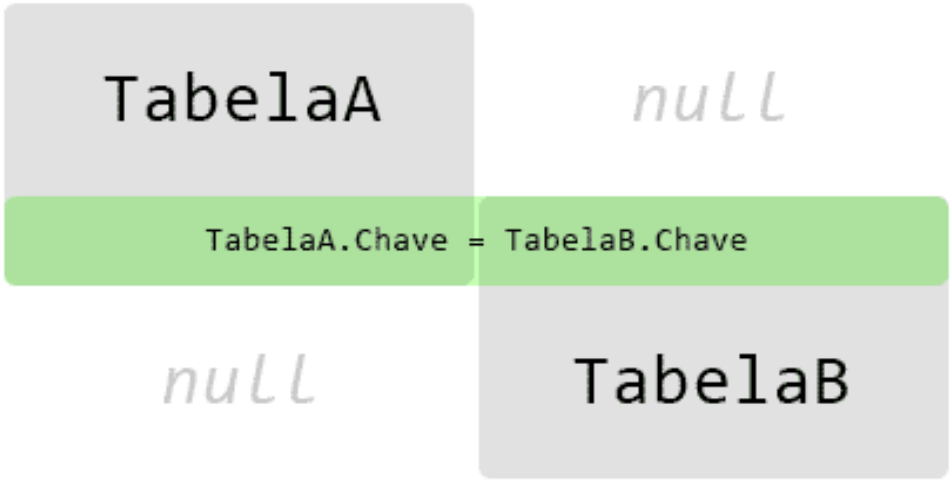
```
TabelaA

Nome         |Chave
-------------+-----
José         |    1
Maria        |    3
Renata       |    4
Roberto      |    8

TabelaB

Profissao    |Chave
-------------+-----
Estivador    |    1
Gerente      |    3
Porteiro     |    5
Engenheiro   |    8
```

INNER JOIN

- Esse é um formato comum de join, que retorna dados apenas quando as duas tabelas tem chaves correspondentes na cláusula ON do join.

- Query:
  - SELECT TabelaA.*, TabelaB.* FROM TabelaA INNER JOIN TabelaB ON TabelaA.Chave = TabelaB.Chave

INNER JOIN



- Resultado:

TabelaA.Nome	TabelaA.Chave	TabelaB.Profissao	TabelaB.Chave
José	1	Estivador	1
Maria	3	Gerente	3
Roberto	8	Engenheiro	8

LEFT JOIN

- É um dos formatos mais usados de join, que retorna a Tabela A inteira e apenas os registros que coincidirem com a igualdade do join na TabelaB (ou campos nulos para os campos sem correspondência).

- Query:
  - SELECT TabelaA.*, TabelaB.* FROM TabelaA LEFT JOIN TabelaB ON TabelaA.Chave = TabelaB.Chave

LEFT JOIN

- Resultado:

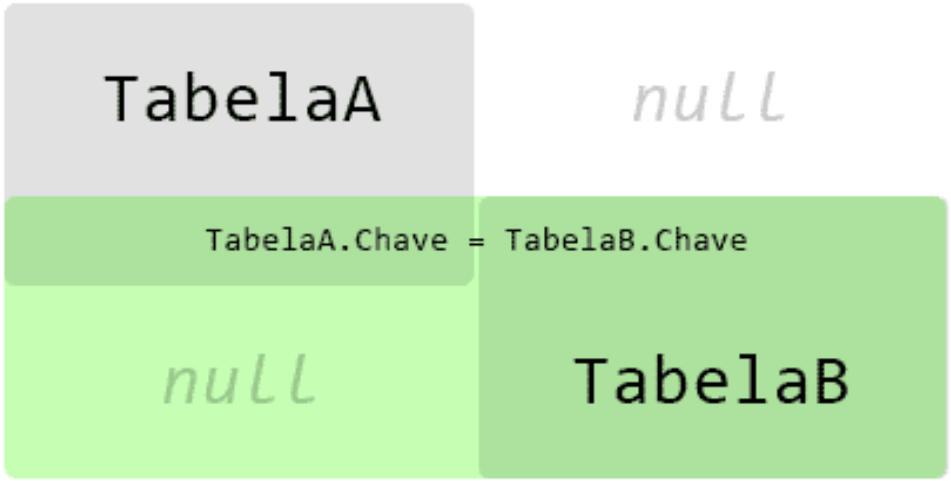
```
TabelaA.Nome|TabelaA.Chave|TabelaB.Profissao|TabelaB.Chave
------------+-------------+-----------------+-------------
José        |            1|Estivador        |            1
Maria       |            3|Gerente          |            3
Renata      |            4|NULL             |NULL
Roberto     |            8|Engenheiro       |            8
```

RIGHT JOIN

- Segue o mesmo raciocínio do Left Join, mas se aplicando à tabela B em vez da A

- Query:
  - SELECT TabelaA.*, TabelaB.* FROM TabelaA RIGHT JOIN TabelaB ON TabelaA.Chave = TabelaB.Chave

RIGHTJOIN



- Resultado:

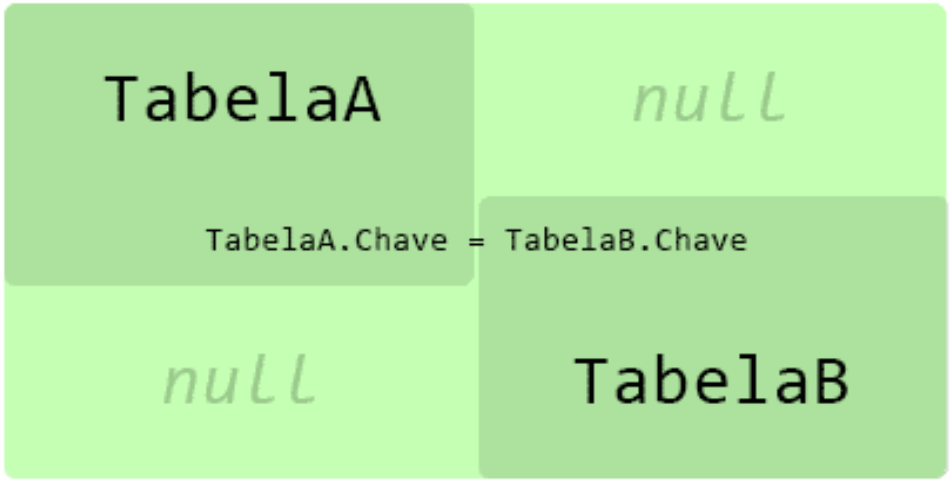
TabelaA.Nome	TabelaA.Chave	TabelaB.Profissao	TabelaB.Chave
José	1	Estivador	1
Maria	3	Gerente	3
NULL	NULL	Porteiro	5
Roberto	8	Engenheiro	8

FULL OUTER JOIN

- Conhecida como OUTER JOIN ou simplesmente FULL JOIN, este retorna todos os registros de ambas as tabelas.

- Query:
  - SELECT TabelaA.*, TabelaB.* FROM TabelaA FULL OUTER JOIN TabelaB ON TabelaA.Chave = TabelaB.Chave

FULL OUTER JOIN



- Resultado:

```
TabelaA.Nome|TabelaA.Chave|TabelaB.Profissao|TabelaB.Chave
------------+-------------+-----------------+-------------
José        |            1|Estivador        |            1
Maria       |            3|Gerente          |            3
Renata      |            4|NULL             |NULL
NULL        |NULL          |Porteiro         |            5
Roberto     |            8|Engenheiro       |            8
```

CROSS JOIN

- Basicamente é o produto cartesiano entre as duas tabelas. Para cada linha de TabelaA, são retornadas todas as linhas de TabelaB.

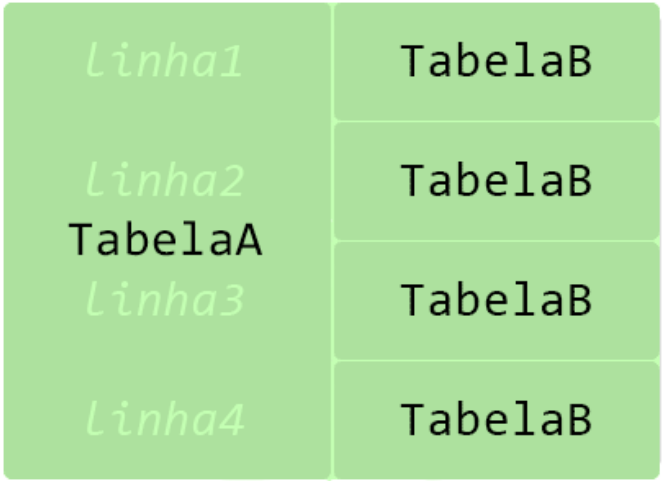
- É mais fácil entender o Cross Join como um "Join sem cláusula ON", ou seja, todas as combinações de linhas de A e B são devolvidas.

CROSS JOIN

- Inclusive, se você fizer um Cross Join com cláusla ON, ele "vira" um mero Inner Join.

- Query:
  - SELECT TabelaA.*, TabelaB.* FROM TabelaA CROSS JOIN TabelaB

CROSS JOIN


linha1
TabelaB
linha2
TabelaB
TabelaA
linha3
TabelaB
linha4
TabelaB

CROSS JOIN

- Resultado:

TabelaA.Nome	TabelaA.Chave	TabelaB.Profissao	TabelaB.Chave
José	1	Estivador	1
José	1	Gerente	3
José	1	Porteiro	5
José	1	Engenheiro	8
Maria	3	Estivador	1
Maria	3	Gerente	3
Maria	3	Porteiro	5
Maria	3	Engenheiro	8
Renata	4	Estivador	1
Renata	4	Gerente	3
Renata	4	Porteiro	5
Renata	4	Engenheiro	8
Roberto	8	Estivador	1
Roberto	8	Gerente	3
Roberto	8	Porteiro	5
Roberto	8	Engenheiro	8

CONSIDERAÇÕES

- Notar que todos os campos pedidos no select sempre retornam (desde que existam na tabela, obviamente), independente de existirem para aquela linha específica.

- O que acontece no caso de uma linha ser retornada para apenas uma das tabelas é que os campos da outra vêm com conteúdo null.

CONSIDERAÇÕES

- Usualmente, caso você precise diferenciar um nulo que realmente exista na tabela de um nulo por falta de correspondência, basta ver se os campos usados na condição do ON não retornaram null também.

ALIASES/APELIDOS

- No SQL um alias ou apelido é um nome que pode ser dado para uma tabela ou coluna no contexto de uma pesquisa. Isto pode ser bom se você tem uma consulta muito grande ou nomes de tabelas e colunas complexos.

ALIASES/APELIDOS

- No SQL um alias ou apelido é um nome que pode ser dado para uma tabela ou coluna no contexto de uma pesquisa. Isto pode ser bom se você tem uma consulta muito grande ou nomes de tabelas e colunas complexos.

- Query:
  - SELECT nome_coluna(s) FROM nome_tabela AS nome_alias

EXERCÍCIOS

- Tomando como base o banco de dados do projeto modelo, vamos fazer alguns exercícios relacionados a JOINS

1. Listar as mesas juntamente com o nome dos atendentes responsável pela mesa;

ATIVIDADE 1

1. Listar o código das mesas juntamente com o nome dos atendentes responsáveis por cada mesa (apenas as mesas que contém atendente);
2. Listar as comandas com seus produtos mostrando o código da comanda e o nome do produto;
3. Listar as comandas mostrando o valor total de cada comanda;
4. Buscar a comanda que teve a maior valor, mostrando o código da comanda, o valor da comanda e o nome do atendente responsável por aquela comanda;

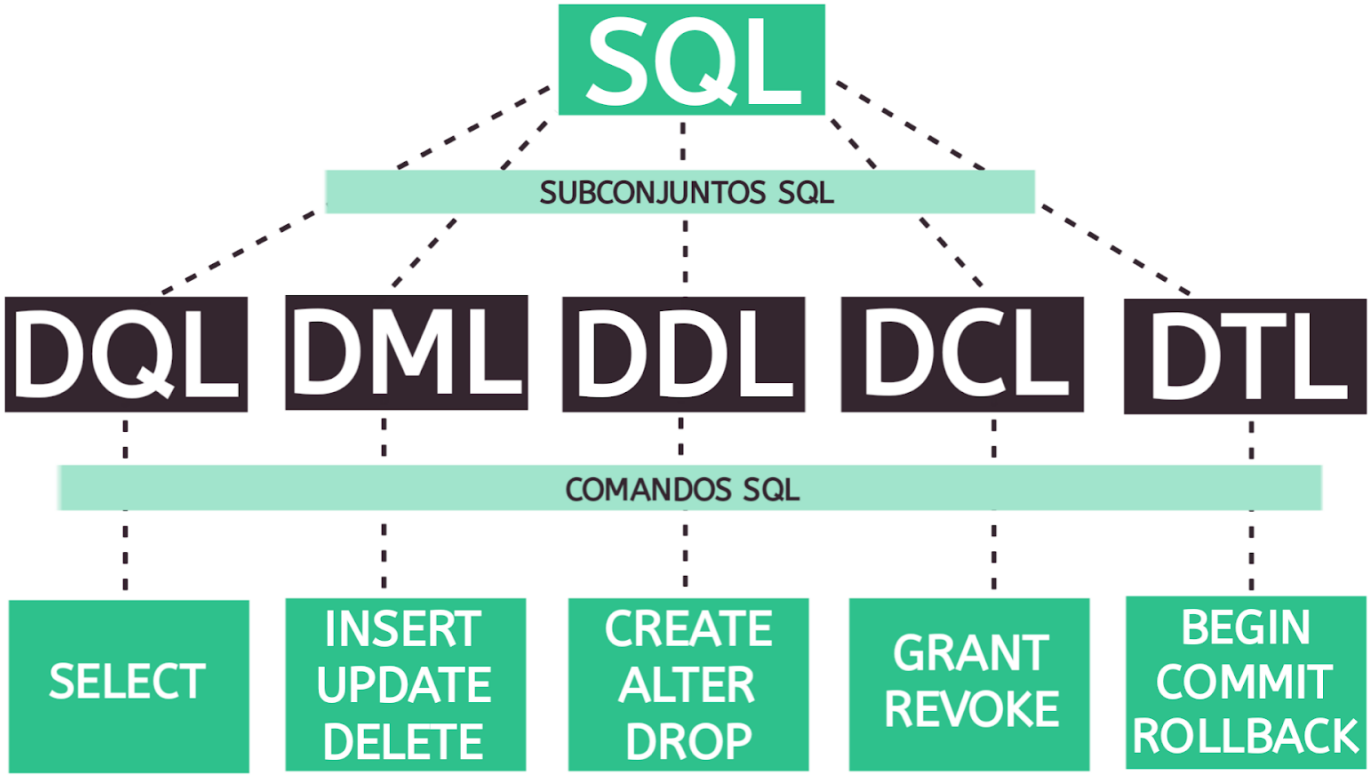
Marcio Michelluzzi – marciomichelluzzi@gmail.com

ATIVIDADE 1

5. Listar todas as comandas com todos os itens de cada comanda;

Marcio Michelluzzi – marciomichelluzzi@gmail.com

TIPOS DE COMANDOS SQL


SQL
SUBCONJUNTOS SQL
DQL
DML
DDL
DCL
DTL
COMANDOS SQL
SELECT
INSERT
UPDATE
DELETE
CREATE
ALTER
DROP
GRANT
REVOKE
BEGIN
COMMIT
ROLLBACK

DCL

DATA CONTROL LANGUAGE

DCL

- Esse subconjunto do SQL envolve comandos relacionados à segurança do banco de dados. A DCL (Data Control Language) controla o acesso aos dados, tanto concedendo privilégio de acesso, quanto retirando a permissão do usuário ou usuária.

GRANT

- Fornece a determinada pessoa o privilégio de acesso dentro do banco de dados. No exemplo a seguir, estamos permitindo a Luiz consultar os dados da tabela estudantes.

- Query:
  - GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

REVOKE

- Esse comando retira os privilégios de acesso. Ou seja, faz a operação inversa ao GRANT, negando a permissão. A seguir, vamos desfazer o que fizemos com GRANT. Para isso, utilizaremos o FROM.

- Query:
  - REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

TCL

TRASANCIONAL CONTROL LANGUAGE

TCL

- TCL (Data Transaction Language) é um subconjunto do SQL para transação de dados. A DTL envolve gerenciamento e controle de transações.

TRANSAÇÃO ????

- Uma transação simboliza uma unidade de trabalho executada dentro de um sistema de gerenciamento de banco de dados (ou sistema similar), sobre um banco de dados , e tratada de maneira coerente e confiável, independente de outras transações.

- Uma transação geralmente representa qualquer alteração em um banco de dados. As transações em um ambiente de banco de dados têm dois propósitos principais:

PROPÓSITOS DAS TRANSAÇÕES

1. Fornecer unidades de trabalho confiáveis que permitam a recuperação correta de falhas e manter um banco de dados consistente mesmo em casos de falha do sistema, quando a execução é interrompida (completamente ou parcialmente) e muitas operações em um banco de dados permanecem incompletas, com estado pouco claro.

PROPÓSITOS DAS TRANSAÇÕES

2. Fornecer isolamento entre programas que acessam um banco de dados simultaneamente. Se esse isolamento não for fornecido, os resultados dos programas possivelmente serão errôneos.

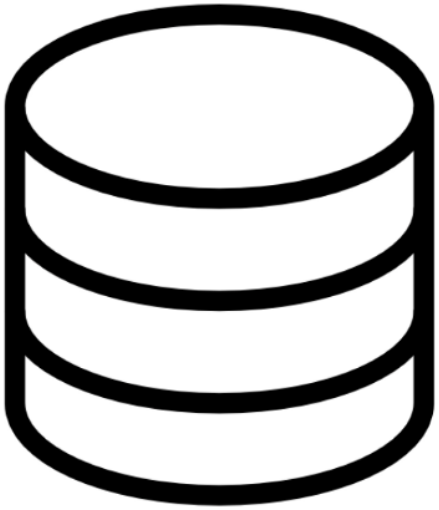
TRANSAÇÕES

- Uma transação de banco de dados, por definição, deve ser atômica , consistente , isolada e durável .
- Profissionais de banco de dados geralmente se referem a essas propriedades de transações de bancos de dados usando o acrônimo ACID

TRANSAÇÕES

- Uma transação de banco de dados, por definição, deve ser atômica , consistente , isolada e durável .
- Profissionais de banco de dados geralmente se referem a essas propriedades de transações de bancos de dados usando o acrônimo ACID

TRANSAÇÕES



A	Atomicidade
C	Consistência
I	Isolamento
D	Durabilidade

ATOMICIDADE

- Todas as ações que compõem a unidade de trabalho da transação devem ser concluídas com sucesso, para que seja efetivada.

- Se durante a transação qualquer ação que constitui unidade de trabalho falhar, a transação inteira deve ser desfeita (rollback).

- Quando todas as ações são efetuadas com sucesso, a transação pode ser efetivada e persistida em banco (commit).

Consistencia

- Todas as regras e restrições definidas no banco de dados devem ser obedecidas.
- Relacionamentos por chaves estrangeiras, checagem de valores para campos restritos ou únicos devem ser obedecidos para que uma transação possa ser completada com sucesso.

ISOLAMENTO

- Cada transação funciona completamente à parte de outras estações. Todas as operações são parte de uma transação única.

- O principio é que nenhuma outra transação, operando no mesmo sistema, possa interferir no funcionamento da transação corrente(é um mecanismo de controle).

- Outras transações não podem visualizar os resultados parciais das operações de uma transação em andamento (ainda em respeito à propriedade da atomicidade).

Durabilidade

- Significa que os resultados de uma transação são permanentes e podem ser desfeitos somente por uma transação subsequente.

- Por exemplo: todos os dados e status relativos a uma transação devem ser armazenados num repositório permanente, não sendo passíveis de falha por uma falha de hardware.

BEGIN/SET TRANSACTION

- Tanto o comando BEGIN TRANSACTION quanto o SET TRANSACTION indicam o início de uma transação.

- Devem ser usados imediatamente no começo do código, registrando que tudo que vem abaixo faz parte da mesma transação.

BEGIN/SET TRANSACTION

- A diferença entre BEGIN TRANSACTION e SET TRANSACTION está que na segunda pode-se atribuir especificações a respeito daquela transação, como, por exemplo, se será apenas para leitura.

- Query:
  - BEGIN TRANSACTION
    - DELETE FROM estudantes WHERE matricula = 1776;
  - COMMIT;

ROLLBACK

- O comando Rollback reverte **uma transação**. Na prática, restaura o banco de dados desde a última vez que o comando COMMIT foi aplicado, garantindo apenas até onde as alterações já foram salvas.

- **É um comando crucial, que pode ser usado em situações de erro.**

- Query:
  - ROLLBACK TO nome_savepoint;
- Query:
  - ROLLBACK;

SAVEPOINT

- O SAVEPOINT define um ponto de salvamento dentro de uma transação, funciona como um "ponto de segurança". Tudo que é anterior a ele não pode ser descartado com o comando ROLLBACK, apenas o que vem após.

- Query:
  - SAVEPOINT nome_savepoint;

OTIMIZAÇÃO DE QUERIES

O PROBLEMA

- O que acontece quando executamos um select no banco de dados?

- Resposta: O SGBD de dados executa aquele comanda e busca a informação necessária. Em cada consulta que fazemos estamos dizendo o que queremos que o SGBD faça.

O PROBLEMA

- E como o SGBD executa esse comando?

- Resposta: Ele faz uma varredura na tabela que especificamos no nosso select buscando os valores que atendam nossa clausula WHERE

O PROBLEMA

- E se tivermos uma tabela “pessoa” e nessa tabela tivermos 10 milhão de registros e executarmos o seguinte comando:
  - SELECT * FROM pessoa WHERE pessoa.cpf = “40336988095”

O PROBLEMA

- O SGBD fará uma varredura de todas as linhas da tabela e retornará os registros que atendam a clausula WHERE

- O tempo de retorno do banco de dados será satisfatório?

O PROBLEMA

O PROBLEMA

- A resposta para essa pergunta é DEPENDE ...
- Dependendo de como nós estruturamos nosso banco de dados e de como estruturamos nossas queries, um comando SQL pode demorar 1 min ou 1 hora (ou até mais)
- Depende de fatores relacionados a massa de dados e a quantidade de acessos que faremos a tabela em questão

OU SEJA ...

- O SGBD tem mecanismos para ajudar a otimização de queries, porém nós desenvolvedores devemos garantir que nossas queries estejam sempre construídas de forma a garantir que se tenha o melhor desempenho ...

- Vamos ver um pouco como o SGBD nos auxilia fazendo otimizações nas queries que executamos ...

ENTENDENDO O SQL

- Os SGBD's possuem estruturas internas que ajudam a otimizar a execução de queries.
- Basicamente quando uma query for executada existem 3 etapas pelas quais a query passa:

PARSER

- O analisador de consulta garante que ela esteja sintaticamente correta (por exemplo, verifica se há vírgulas fora do lugar) e semanticamente correta (por exemplo, verifica se a tabela existe) e retorna erros em caso negativo.

- Se ela estiver correta, então ele a transforma em uma expressão algébrica e a passa para a próxima etapa.

OPTIMIZE

- A etapa de planejamento e otimização da consulta realiza o trabalho duro de pensar.

- Primeiro, ela realiza otimizações simples (melhorias que sempre resultam em uma performance melhor, como simplificar 5*10 em 50).

- Então, ela considera diferentes "planos de consulta" que podem ter otimizações diferentes, estima o custo (de processamento e de tempo) de cada plano de consulta, com base no número de linhas nas tabelas relevantes, e então escolhe o melhor plano e o passa para a próxima etapa.

EXECUTE

1. A etapa de execução da consulta recebe o plano e o transforma em operações para o banco de dados, retornando os resultados caso eles existam.

FAZENDO NOSSA PARTE

- Agora que compreendemos um pouco sobre como o SGBD otimiza as queries que executamos vamos ver um pouco como nós podemos garantir que nossa query esteja estruturada da melhor forma ...

INDICES

- Índice, no contexto da estrutura de dados, é uma referência associada a uma chave, que é utilizada para fins de otimização, permitindo uma localização mais rápida de um registro quando efetuada uma consulta

- Por padrão, os SGBDs criam índices automaticamente para campos de:
  - Chave Primária
  - Chave Estrangeira
  - Constraint UNIQUE

INDICES

- Índice, no contexto da estrutura de dados, é uma referência associada a uma chave, que é utilizada para fins de otimização, permitindo uma localização mais rápida de um registro quando efetuada uma consulta

- Por padrão, os SGBDs criam índices automaticamente para campos de:
  - Chave Primária
  - Chave Estrangeira
  - Constraint UNIQUE

INDICES

- Se suas consultas realizam buscas em tabelas grandes, uma boa recomendação é o uso de índices.
- Um índice é uma estrutura em disco associada a uma tabela ou exibição que agiliza a recuperação de linhas.
- Usá-los é uma tomada de decisão adequada para resolver a maior parte dos problemas de consultas com tempo demorado.
- Todas as chaves primárias precisam de índices para realizar as junções com outras tabelas de forma mais eficaz. Isso faz com que todas as tabelas precisem de uma chave primária.

INDICES

- Índices são normalmente criados em colunas que são acessadas com maior frequência de modo que a informação possae ser recuperada mais rapidamente.

- Os índices podem ser criados em uma única coluna ou em um grupo delas.

- Quando um índice é criado, ele primeiro classifica os dados e, em seguida, atribui um Rowid (chave única e sequencial) para cada linha.

INDICES

- Índices são normalmente criados em colunas que são acessadas com maior frequência de modo que a informação possa ser recuperada mais rapidamente.

- Os índices podem ser criados em uma única coluna ou em um grupo delas.

- Quando um índice é criado, ele primeiro classifica os dados e, em seguida, atribui um Rowid (chave única e sequencial) para cada linha.

INDICES

- ATENÇÃO:

- Os índices são muito bons no sentido de performance do banco de dados, otimizam as buscas de dados, mas, por outro lado, consomem muito espaço em disco, o que pode se tornar concorrente do próprio banco se você o detém em um espaço generoso ou pode se tornar caro quando de detém o banco em um storage.

INDICES

- Considerações:

- Quando colunas indexadas são modificadas, o SGBD desloca recurso internamente para manter esses índices atualizados e associados;

- A manutenção de índices requer tempo e recursos, portanto, não crie índices que não serão usados efetivamente;

INDICES

- Considerações:

- Quando se contém grande quantidade de dados duplicados, índices apresentam mais custo que benefícios. Assim como usar índices com atributos de pouca variação, como “sexo” ou atributos do tipo flag.

INDICES

- Como criar um índice para campos das minhas tabelas:

- CREATE INDEX index_name ON table_name (column_name);

- CREATE INDEX user_id_and_org_id_idx ON users (user_id, org_id);

UNION

- O operador UNION combina os resultados de duas ou mais queries em um único result set, retornando todas as linhas pertencentes a todas as queries envolvidas na execução.

- Para utilizar o UNION, o número e a ordem das colunas precisam ser idênticos em todas as queries e os data types precisam ser compatíveis.

UNION

- Existem dois tipos de operador UNION, sendo eles UNION e UNION ALL.

- O operador UNION, por default, executa o equivalente a um SELECT DISTINCT no result set final.

- Em outras palavras, ele combina o resultado de execução das duas queries e então executa um SELECT DISTINCT a fim de eliminar as linhas duplicadas. Este processo é executado mesmo que não hajam registros duplicados.

UNION

- Query:
  - SELECT ShipName, ShipAddress from Orders WHERE CustomerID ="WARTH" UNION SELECT ShipName, ShipAddress from Orders WHERE CustomerID ="VINET"

UNION ALL

- O operador UNION ALL tem a mesma funcionalidade do UNION, porém, não executa o SELECT DISTINCT no result set final e apresenta todas as linhas, inclusive as linhas duplicadas.

UNION ALL

- Query:
  - SELECT ShipName, ShipAddress from Orders WHERE CustomerID ="WARTH“ UNION ALL SELECT ShipName, ShipAddress from Orders WHERE CustomerID ="VINET“
- Observação: UNION ALL X UNION
- Discussão sobre pode ser encontrada em:
  - https://stackoverflow.com/questions/35627923/union-versus-select-distinct-and-union-all-performance

Escolha o tipo de dados correto

- Escolha o tipo de dados correto. Se você vai criar uma tabela que possui uma coluna com a sigla do estado brasileiro (por exemplo), sabemos que este atributo tem duas posições, neste caso o data type mais adequado é o char(2).

- Usar o data type incorreto, pode consumir mais espaço e pode causar conversões implícitas e explicitas. Que consomem recursos desnecessariamente e podem levar o otimizador de consultas a escolher planos pouco eficientes

Use tipo de dados unicode somente quando necessário

- Os tipos de dados Unicode, como nchar e nvarchar, usam duas vezes mais espaço de armazenamento em comparação com tipos de dados ASCII como char e varchar.

Modele o seu banco de dados

- Parece besteira? Mas quando você cria uma representação visual fica mais fácil verificar se os dados terão consistência.

- E eu não estou bancando a AD super rigorosa. Mas pense no seguinte cenário, uma equipe com vários desenvolvedores criando um banco de dados, talvez fique impossível analisar se o fluxo da informação está correto e se as informações estão consistentes utilizando somente os scripts.

Utilize o recursos do SGBD

- Evite deixar para a sua aplicação validar algumas regras de qualidade que podem ser garantidas com recursos nativos e declarativos do SGBD. Por exemplo, se uma coluna aceita somente 3 valores crie uma check constraint para validar os valores.

Utilize o recursos do SGBD

- Se a coluna tem um valor default, use o objeto default. Digo isso porque se existem N aplicações desenvolvidas por equipes diferentes usando o mesmo BD, e uma das equipes não criou o código para garantir as regras de qualidade, o banco de dados poderá ficar inconsistente.

Pense bem nos seus índices

- Se a sua base de dados ainda está vazia, então tenha em mente que os testes de desempenho podem indicar a necessidade de criar índices. Contudo, vale a regrinha básica… Se a coluna tem muitas atualizações utilizá-la em um índice pode degradar o desempenho ao invés de melhorar.

- Na clausula WHERE evite negações, funções e conversões, porque podem ignorar os índices

WHERE <> HAVING !!!

- Use a cláusula HAVING para o que ela se propõe, filtrar operações de agregação.

SELECT * é do mal

- Se não é para testes, não use o SELECT * , pois o SGBD “gastará” um tempo de processamento para buscar quais são todos as colunas da(s) tabela(s) especificada(s) na cláusula FROM

Triggers podem ser fofas

- Evite ações longas em triggers, por que podem fazer com que os bloqueios sejam mantidos por mais tempo. Mantenha o código da trigger tão pequeno e eficiente quanto possível

Avalie o plano de execução da consulta

- No SQL Query Analyzer, ative a opção Exibir plano de execução e execute sua consulta contra uma carga significativa de dados para ver o plano criado pelo otimizador.

- Avalie este plano e depois identifique o que pode ser melhorado. Por exemplo, as operações mais demoradas. Compreender o plano de execução é um passo importante para otimizar uma consulta. Tal como acontece com a indexação, leva tempo e conhecimento do seu sistema para poder identificar o melhor plano.

Sempre prefira usar o namespace (owner / esquema) dos objetos (nome da tabela, procedures, etc.)

- No SQL Query Analyzer, ative a opção Exibir plano de execução e execute sua consulta contra uma carga significativa de dados para ver o plano criado pelo otimizador.

- Avalie este plano e depois identifique o que pode ser melhorado. Por exemplo, as operações mais demoradas. Compreender o plano de execução é um passo importante para otimizar uma consulta. Tal como acontece com a indexação, leva tempo e conhecimento do seu sistema para poder identificar o melhor plano.

REFERÊNCIAS

- https://pt.stackoverflow.com/questions/6441/qual-%C3%A9-a-diferen%C3%A7a-entre-inner-join-e-outer-join
- http://phpsqlbr.blogspot.com/2009/05/aliases-ou-apelidos-no-sql.html
- https://blog.betrybe.com/linguagem-de-programacao/subconjuntos-sql/
- https://blog.betrybe.com/linguagem-de-programacao/subconjuntos-sql/
- https://dotnettutorials.net/lesson/create-new-user-using-mysql-workbench/

REFERÊNCIAS

- https://www.javatpoint.com/types-of-sql-commands
- https://programadoresdepre.com.br/comandos-sql-ddl-dml-dcl-tcl-dql-e-clausulas/
- https://www.devmedia.com.br/otimizacao-de-consultas-sql/33485
- https://pt.khanacademy.org/computing/computer-programming/sql/relational-queries-in-sql/a/more-efficient-sql-with-query-planning-and-optimization
- https://www.devmedia.com.br/entendendo-e-usando-indices/6567
- https://pt.wikipedia.org/wiki/%C3%8Dndice_(estruturas_de_dados)

REFERÊNCIAS

- http://www.bosontreinamentos.com.br/bancos-de-dados/o-que-sao-indices-em-bancos-de-dados-indexacao-em-tabelas/
- https://www.w3schools.com/sql/sql_create_index.asp
- https://stackoverflow.com/questions/35627923/union-versus-select-distinct-and-union-all-performance
- https://www.devmedia.com.br/sql-utilizando-o-operador-union-e-union-all/37457
  http://www.linhadecodigo.com.br/artigo/3620/indices-mysql-otimizacao-de-consultas.aspx