Alexandre Malavasi Follow

Jan 2, 2018 · 10 min read

25 dicas e boas práticas de banco de dados para desenvolvedores

Dicas e boas práticas

Quase todos os sistemas que desenvolvemos envolvem banco de dados e, embora como desenvolvedores adoremos a parte do código-fonte de nossos sistemas, muitas vezes a base de dados, a forma como a projetamos e lidamos com ela representa a “alma” do sistema em relação ao valor agregado ao negócio, por questões de segurança, performance, políticas organizacionais, dentre outros fatores que tornam esta “camada” de nossas aplicações extremamente relevante, merecedora de uma atenção especial por nós desenvolvedores.

Sempre fui a favor e adepto da opinião de que é recomendável ao desenvolvedor entender um pouco de tudo: front-end, back-end, banco de dados, UX, etc. Logicamente que, cada um de nós acaba se especializando em uma destas vertentes, tornando uma delas prevalente em relação às outras por uma questão de tempo, otimização e conciliação entre vida pessoal e carreira profissional, que torna a arte de estudar e se aprimorar em relação à conhecimento aprofundado de muitas coisas algo sempre desafiador.

Considerando a quantidade enorme de recomendações de boas práticas e dicas que podem ser dadas em relação à banco de dados, esta singela lista constante neste artigo representa apenas uma pequena porção de tudo o que

pode ser abordado, sendo uma humilde lista que pode ser aumentada e aprimorada em valores múltiplos.

Quando decidi escrever este artigo ontem, no primeiro dia de 2018, estas 25 práticas são as que me vieram mais naturalmente à mente, com a maioria destas recomendações sendo aplicáveis aos principais bancos de dados relacionais utilizados no mercado: SQL Server, Oracle, MySql, etc.

Como estamos todos sempre em fase de constante aprendizado, caso não concorde com algum item ou tenha alguma sugestão de melhoria no artigo, ela será muito bem-vinda.

Mas, agora sem mais “enrolar”, vamos à lista em si.

1. Utilização de EXISTS

Caso precise retornar em uma consulta registros de uma tabela que satisfaçam uma determinada condição segundo referências de uma segunda tabela, ao invés de utilizar uma subconsulta na cláusula WHERE para um operador IN prefira a utilização de **EXISTS**:

```
1   SELECT *
2     FROM MINHA_TABELA M
3    WHERE EXISTS (SELECT *
```

```
4           FROM TABELA_LOG L
5          WHERE L.ID_MINHA_TABELA = M.ID
6            AND TO_CHAR(L.DATA, 'YYYY') = '2017')
7
```

exemplo_exists.sql hosted with ❤ by GitHub view raw

Na maior parte dos cenários, esta forma tem um desempenho muito superior, em diversos bancos dados, do que a utilização da cláusula IN com subconsultas:

```
1  SELECT *
2    FROM MINHA_TABELA M
3   WHERE M.ID IN (SELECT L.ID_MINHA_TABELA
4           FROM TABELA_LOG L
5          WHERE TO_CHAR(L.DATA, 'YYYY') = '2017')
```

clausula_in.sql hosted with ❤ by GitHub view raw

2. Utilize flags de tipo booleano ou inteiro

Caso tenha necessidade de possuir colunas em uma tabela cujo valor se refira à valores verdadeiros e falsos, como por exemplo, especificar se um registro está ativo ou não na sua tabela, opte por criar as colunas com o tipo booleano nativo da base dados específico que você está utilizando, ao invés de utilizar informações ‘S’ ou ’N’ em um campo do tipo texto. Em tabelas com

muitos registros, o filtros por estes campos torna-se desnecessariamente lento.

O Oracle infelizmente não possui um tipo booleano como o SQL Server e outros bancos relacionais. Então a saída neste caso poderia ser utilizar um campo do tipo NUMERIC(1) para armazenar 1 ou 0. Pelo menos é mais performático do que 'S' ou 'N'.

3. Conversões com UPPER, TO_CHAR e etc em cláusulas WHERE

Evite fazer conversões de tipo e formato em colunas na cláusula WHERE para realizar filtro de dados. Esta operação faz com que o banco de dados naturalmente ignore a utilização dos índices automáticos criados para estas colunas, que tornariam a consulta bem mais rápida. Estude sempre a possibilidade de já armazenar os dados no formato correto ou que tenha uma predominância na forma de visualização na aplicação.

4. Não utilize HAVING para filtrar dados

Caso necessite filtrar dados em um agrupamento de informações, prefira sempre realizar esta operação na cláusula WHERE ao invés do HAVING, por questões de performance, a não ser que seja necessário realizar algum filtro utilizando realmente as operações de agregação:

```sql
--NÃO RECOMENDÁVEL

SELECT NOME, TIPO
FROM MINHA_TABELA A
GROUP BY NOME, TIPO
HAVING TIPO = 2

--RECOMENDÁVEL

SELECT NOME, TIPO
FROM MINHA_TABELA A
WHERE TIPO=2
GROUP BY NOME, TIPO
```

exemplo_having_1.sql hosted with ❤ by GitHub view raw

5. Evite atribuir valores em variáveis com DUAL

Quem utiliza o banco de dados Oracle no seu dia a dia está muito habituado a utilizar o DUAL em muitas situações, mas eu recomendaria não utilizá-lo em atribuições de variáveis em procedures, pois isto representa perda de performance em sistemas críticos:

```sql
declare
x number;
  begin
  -- NÃO RECOMENDÁVEL
 select 1 into x from dual;

```

```
7     -- RECOMENDÁVEL
8       x := 1;
9     end;
```

exemplo_dual.sql hosted with ❤ by GitHub view raw

6. Tome cuidado com MaxValue das SEQUENCES

No banco de dados Oracle, ao criar um objeto do tipo SEQUENCE , coloque na propriedade MAXVALUE um valor extremamente alto para que não ocorra de, depois de algum tempo, a sua aplicação parar de funcionar por ter atingido este valor máximo. A atribuição de uma valor alto não acarreta em alocação de espaço desnecessário, pois é apenas uma informação de configuração do objeto do tipo SEQUENCE.

Para quem não está habituado com SEQUENCE ou não sabe o que é, esta é uma exclusividade do banco de dados Oracle, que não possui um campo do tipo auto-incremento para geração de números sequenciais em tabelas. E, por este motivo, é necessário criar este tipo de objeto adicional no Oracle para versões anteriores à 12c em que eles implementaram um recurso semelhante ao auto-incremento.

7. Influência da orientação a objetos na modelagem de base de dados

Caso o programador não tenha muita experiência em modelagem de banco de dados, ficando restrito ao código da aplicação, é comum haver uma tendência de ser influenciado a “pensar” em orientação a objetos ao modelar uma base de dados. Embora a utilização de ORM tenha facilitado muito a vida dos desenvolvedores, é sempre importante frisar que os bancos de dados relacionais não são orientados a objetos, mesmo podendo haver semelhanças entre tabela x objetos, colunas x propriedades e por aí vai.

A base de dados deve ser projetada e modelada segundo boas práticas de banco de dados e não boas práticas de orientação a objetos.

8. Utilize procedures e views

Quando não utilizamos procedures e views, toda vez que executamos uma instrução SQL é necessário que o SGBD analise se a sintaxe do comando esta correta, se os objetos referenciados realmente existentes, dentre outras análises igualmente necessárias.

Quando o código a ser executado encontra-se em uma procedure ou view, o banco de dados não precisa fazer estas verificações e validações, pois as mesmas já foram feitas ao ser criar as procedures e views. Portanto, com o banco de dados poupando este trabalho, logicamente a perfomance das

execuções de SQL enviados pela aplicação aumenta consideravelmente em sistemas críticos.

9. Tipos são extremamente importantes

Se preocupe sempre com os tipos das colunas das tabelas do sistema que você está criando para verificar se os mesmos correspondem fielmente ao tipo de informação que será armazenada. Por exemplo, se a coluna irá armazenar uma data, crie um campo do tipo DATE. Se vai armazenar um número inteiro, crie uma coluna do tipo INTEGER e por aí vai. Isto parece óbvio, mas é muito comum encontrarmos este tipo de situação.

Esta boa prática vai dar segurança por não permitir que seja inserida informação com tipo não compatível ou inconsistente, e vai melhorar a performance de consultas futuramente por não haver necessidade de se fazer conversões de tipo

Esta é a dica mais clichê de todas: evite utilizar SELECT * FROM. É muito comum fazermos consultas com JOINS em diversas tabelas. Especificar no SELECT somente as colunas que vai realmente utilizar melhorará é uma boa prática quase obrigatória para nós desenvolvedoras. Um outro benefício que

também considero importante é facilitar a leitura do SQL em manutenções também.

11. Utilização de Cache

Caso se utilize na aplicação consultas de informações que não são frequentemente atualizadas, considere a possibilidade de colocar estes dados em cache para poupar o banco de dados deste trabalho. Mas, esta opção deve ser sempre analisada considerando o cenário de cada projeto e seus pré-requisitos.

12. Tipos das variáveis e parâmetros

Procure utilizar nas variáveis e parâmetros de procedures e functions exatamente os mesmos tipos das colunas da tabela para evitar ter que fazer conversões desnecessárias.

13. Só utilize ORDER BY e DISTINCT se for realmente necessário

Muitas vezes, para determinadas funcionalidades do sistema, ordenação seja algo que não importe ou não haja necessidade, devendo ser evitado. Ás vezes queremos, por exemplo, apenas listar conteúdo em tela e resolvemos por conta própria, sem haver especificações explícitas pra isto, fazer ordenação por data, registro mais recente e por aí vai, sendo que às vezes isto não

agregará valor ao usuário final na aplicação. É estranho algum retorno de consulta sem ordenação? Pode até parecer, mas deve ser utilizado conscientemente por questões de performance, assim como o DISTINCT.

14. Configuração de linguagem, idioma e cultura

O ideal é que o banco de dados utilizado pela aplicação esteja configurado em relação à idioma/cultura compatível com as regras de negócio ou contexto do sistema para evitar ter que realizar conversões explícitas nas consultas de banco de dados, gerando prejuízo à performance. Tais configurações estão relacionadas a aspectos de globalização. No caso do Oracle diz respeito ao **National Language Support.**

15. Utilização de “VALUES multi-row”

Caso necessite realizar múltiplas inserções em uma mesma tabela de forma sequencial, é preferível utilizar a sintaxe conforme abaixo, pois gerará ganho de performance em sistemas críticos e te economizará algumas linhas de digitação também:

```
1  --RECOMENDÁVEL
2
3  INSERT INTO MINHA_TABELA (CAMPO1, CAMPO2)
4  VALUES (1, 'Teste1'),
5      (2, 'Teste2'),
```

```
6       (3, 'Teste3')
7
8   --NÃO RECOMENDÁVEL
9
10  INSERT INTO MINHA_TABELA (CAMPO1, CAMPO2) VALUES (1,'Teste1');
11  INSERT INTO MINHA_TABELA (CAMPO1, CAMPO2) VALUES (2,'Teste2');
12  INSERT INTO MINHA_TABELA (CAMPO1, CAMPO2) VALUES (3,'Teste3');
```

multple_row.sql hosted with ❤ by GitHub view raw

16. Monitoramento de querys

Mesmo após o sistema já esteja em produção ou ainda em ambiente de homologação, você como parte integrante do time de desenvolvimento, procure participar ativamente com o time de infraestrutura e banco de dados, para auxiliar no monitoramento e análises das operações do banco de dados relacionados ao sistemas dos projetos que você participa, a fim de poder identificar possíveis pontos de melhoria e identificar de forma antecipada possíveis problemas. Estimular contato constante entre os times é algo que está ficando cada vez mais comum e é extremamente recomendável, sendo esta uma premissa básica para DevOps.

17. Utilização de índices em colunas muito acessadas

Caso seja identificado que é necessária a criação de algum índice que vise melhorar a performance das consultas à base de dados de uma aplicação,

procure fazer as seguintes perguntas para determinar o critério de criação, exatamente nesta ordem:

> *Qual coluna é acessada ou requisitada com mais frequência, sendo chave-primária ou não?*
>
> *Será que não é conveniente a modificação ou remodelagem da estrutura para fins de performance, considerando a criticidade desta minha consulta?*

18. Cuidado ao utilizar índices em colunas que são atualizadas com muita frequência

A utilização de índices nem sempre é uma boa alternativa em determinados cenários. Um deles é quando o índice é criado em colunas que são atualizadas com uma frequência absurda. Mesmo tendo boas intenções, o cumprimento do objetivo de melhorar a performance das consultas pode acarretar em perdas de performance em operações de INSERT, DELETE e UPDATE nesta tabela.

Portanto, criação de índices é algo que deve ser sempre analisada com muito cuidado.

19. Índices em colunas muito presentes em WHERE, JOIN, ORDER BY e TOP

Uma boa dica para verificar se seria conveniente a criação de um índice em determinada coluna é verificar a frequência de utilização delas em cláusulas WHERE, JOIN, ORDER BY e TOP. Esta é sempre uma boa pista de índices que poderiam ser criados.

Mas, como dito no tópico anterior, a criação de índices sempre deve ser analisada e aplicada com muito cuidado.

20. Não deixe as chaves estrangeiras para depois

Esta dica é quase que tão óbvia e trivial quanto a não utilização de SELECT * FROM. Mas, é muito comum vermos sistemas criados utilizando tabelas sem os devidos relacionados no banco de dados. Então, nunca deixe pra depois a criação das devidas referências de chave-primária e estrangeira. Crie-as no exato momento da criação da própria tabela.

21. Utilização otimizada das tabelas de log e históricos

Ás vezes temos em nossos sistemas tabelas de logs com milhões de registros que são muito pouco ou nunca acessadas. Estude a possibilidade de armazenar boa parte do histórico em outras tabelas (arquivo morto),

permanecendo nas tabelas de histórico e log somente os registros mais recentes. Mas, embora esteja recomendando assim friso que cada caso deve ser analisado de forma específica, pois muitas vezes os requisitos de negócio demandem que todo o histórico e logs estejam facilmente acessíveis nas tabelas originais e não em um "arquivo morto".

22. Insira comentários à vontade

Ao criar uma tabela ou coluna no banco de dados não economize nos comentários a respeito de seu significado, fazendo isto na própria base de dados, principalmente se o sistema for legado. Considero os comentários em base de dados até mesmo mais importantes que comentários na aplicação. Facilitam absurdamente a interpretação e manutenção pelos desenvolvedores que precisam lidar com a base de dados.

23. Tabelas sem chave-primária

Sim, infelizmente isto existe aos montes por aí. Se a sua tabela não possui chave primária, é recomendável que seja feita revisão na sua modelagem, pois em teoria uma tabela não deveria ficar "isolada" em modelo relacional.

24. Dedicação de tempo à modelagem

Como dito anteriormente, muitas vezes o banco de dados é a “alma” do sistema. Vale a pena investir tempo no correto planejamento e modelagem da base de dados, refletindo na estrutura a ser criada de cada tabela, coluna, relacionamentos e muitos outros aspectos.

Investir nesta etapa vale muito a pena.

25. UPDATE sem WHERE

E pra finalizar, já que você teve a paciência de chegar até aqui na vigésima e quinta dica, fica aqui uma descontraída recomendação: não execute um UPDATE sem WHERE, principalmente se você estiver em **PRODUÇÃO!**

HAHAHAHAHAHAHA

Obrigado por ter lido este artigo até o final. Como estamos iniciando o ano de 2018, desejo a você um ótimo ano, com muito sucesso profissional e desenvolvimento de sistemas performáticos.

Linkedin: https://www.linkedin.com/in/alexandremalavasi

Alexandre Malavasi (@alemalavasi) | Twitter

The latest Tweets from Alexandre Malavasi (@alemalavasi). Analista de Sistemas | MCP | MCTS | MCPD | ITIL v3. | MTAC ...

twitter.com

Referências

45 Database Performance Tips for Developers - Redgate Software

Download your free copy of 45 Database Performance Tips for Developers to see the SQL Server performance tips and...

www.red-gate.com

Meu Livro

Fico muito feliz em anunciar que tive a felicidade de escrever meu primeiro livro. É um conteúdo muito bem detalhado sobre Design Patterns, orientação a objeto, princípios SOLID, .NET 5 e C# no geral. Você pode adquirir o livro no link abaixo:

Implementing Design Patterns in C# and .NET 5: Build Scalable, Fast, and Reliable .NET Applications...

Buy Implementing Design Patterns in C# and .NET 5: Build Scalable, Fast, and Reliable .NET Applications Using the Most...

www.amazon.com

Sql Server Database Best Practices Banco De Dados