

10 dicas sobre Firebird que todo desenvolvedor deve saber



Fernando Sarturi Prass

(fernando@dotbr.com.br)

é Mestre em Ciência da Computação pela UFSC. Professor da Universidade Luterana do Brasil (ULBRA) nos campus de Santa Maria e Cachoeira do Sul. Sócio-diretor da dotBR Soluções em TI (www.dotbr.com.br), empresa que presta serviços de desenvolvimento de sistemas e de consultoria em Bancos de Dados e Metodologias de Desenvolvimento.



Fábio Sarturi Prass

(fabio@dotbr.com.br)

é Bacharel em Sistemas de Informação pelo Centro Universitário Franciscano e Analista de Sistema da Di Uno Informática.

O Firebird tornou-se um banco de dados de grande aceitação, principalmente pela comunidade Delphi. Grande parte desse sucesso deve-se a sua facilidade de uso e de configuração. Entretanto, apesar de ser bastante simples, o Firebird exige alguns cuidados por parte dos desenvolvedores, do contrário corre-se o risco de que seu desempenho seja reduzido significativamente.

Este artigo mostrará 10 dicas que tendem a garantir um bom desempenho ao banco de dados. Algumas são baseadas em textos que foram publicadas por outros autores (logo as fontes estão citadas, nos devidos lugares), outras surgiram da necessidade de resolver problemas encontrados pelo autor em consultorias realizadas em empresas de todo o país.

1. Backup e Restore

É provável que não exista uma dica melhor e mais simples sobre Firebird do que

essa: sempre que possível faça um *backup* e *restore*. Essas operações trazem uma série de benefícios, os principais são:

- As páginas de dados e índices são alocadas de forma contínua e aquelas que não são usadas são eliminadas;
- A árvore de índices é reconstruída e a seletividade dos índices recalculada;
- Registros eliminados (através do DELETE) são excluídos fisicamente (*Garbage Collection*).

Uma dica: caso o principal uso do banco de dados seja a leitura e não a inserção ou atualização de dados, pode-se usar o parâmetro USE_ALL_SPACE ao restaurar o *backup*.

Devido à forma como o Firebird trabalha com as versões de um mesmo registro, utiliza um modelo chamado *Versioning*, as páginas de dados do banco armazenam múltiplas versões dos registros nelas contidos. Quando um BD é restaurado, o banco reserva um espaço de aproximadamente 20% da página para

armazenar as novas versões dos registros casos os mesmos sejam alterados. Se o banco de dados será lido e não atualizado (ou pouco atualizado), não há necessidade de que esse espaço seja reservado (KARWIN, 1998). O comando completo é mostrado a seguir:

```
GBAK -c -use_all_space -user sysdba -pas
masterkey <arquivo_backup.fbk> <novo_banco.
fdb>
```

2. Análise das estatísticas do cabeçalho do banco

As estatísticas do cabeçalho do banco de dados fornecem uma série de informações importantes para avaliar a real situação do servidor. Elas podem ser obtidas através do utilitário de linha de comando GSTAT, presente na pasta *bin* do Firebird.

Parâmetro	Função
-a	Analisa os dados e as páginas de índice
-d	Analisa as páginas de dados
-h	Analisa a página de cabeçalho (header page)
-i	Analisa as "folhas" das páginas de índices (leaf pages)
-l	Analisa a página de log
-u	Nome do usuário para conectar no banco
-p	Senha para conexão
-r	Analisa os registros de versões
-t	Especifica o nome da tabela a ser analisada (para o caso de ser uma única tabela)

Tabela 1. Principais parâmetros disponíveis para o utilitário GSTAT

O GSTAT retorna uma série de informações sobre o banco solicitado, essas informações dependem dos parâmetros passados. O comando "gstat -help" apresenta as opções disponíveis, as principais delas estão na Tabela 1.

A Listagem 1, cujo comando para obtenção está logo a seguir, mostra parte do resultado da análise do cabeçalho do banco de dados EMPLOYEE que acompanha o Firebird e será usado em todos os exemplos deste artigo (a Tabela 2 explica as principais variáveis):

```
gstat -h -u sysdba -p masterkey
c:\firebird\examples\employee.fdb
```

Conforme mostra CANTU (2005) a análise das variáveis de transações podem apontar dois problemas importantes:

- Se a diferença entre *Oldest active* e *Next transaction* for muito grande é provável que as transações fiquem abertas por muito tempo. A solução é realizar um *sweep* manual através do utilitário GFIX (também presente na pasta *bin*):

```
gfix -sweep -user sysdba -password masterkey
<banco de dados>
```

- Uma diferença maior que 20.000 entre *Next Transaction* e *Oldest Transaction* pode indicar que o *sweep* automático está desligado ou configurado para um valor muito alto. Pode-se alterar o valor através do GFIX (parâmetro *housekeeping*):

```
gfix -housekeeping 15000 -user sysdba -password
<banco de dados>
```

Informação	Descrição
Page Size	Tamanho definido para as páginas do banco de dados.
ODS Version	<i>On Disk Structure Version</i> , versão da estrutura do arquivo do banco de dados, informa em qual versão do Firebird o banco foi criado. É atualizado sempre que um <i>backup</i> de uma versão mais antiga é restaurado num servidor mais recente (CANTU, 2005).
Oldest Transaction	Transação mais antiga que ainda não recebeu um <i>commit</i> .
Oldest Active	ID da transação mais antiga ainda ativa.
Next Transaction	ID da próxima transação a ser criada.
Implementation ID	Sistema operacional onde o banco de dados foi criado, 16 representa o Windows de 32-bit e 19 o Linux.
Shadow count	Número de arquivos de <i>shadow</i> (sombra) do banco de dados.
Page buffers	Tamanho do <i>cache</i> em número de páginas. Se estiver zero, indica que o valor utilizado é o padrão.
Database dialect	Dialeto atual do banco de dados.
Attributes	Atributos do banco de dados: <i>force write</i> : modo de escrita síncrona (<i>write-through</i>), sem <i>cache</i> . <i>no_reserve</i> : nenhum é espaço reservado para os registros de versão. <i>shutdown</i> : banco indisponível para conexões (exceto para o usuário SYSDBA).
Sweep Interval	Intervalo para realização do <i>sweep</i> automático.

Tabela 2. Principais informações obtidas no cabeçalho (header) do banco

3. Análise das estatísticas das páginas de dados e de índices

As estatísticas do banco fornecem também informações sobre as páginas de dados, as páginas de índices e registros das tabelas. Veja a Listagem 2 o resultado da execução do comando:

```
gstat -r -u sysdba -p masterkey <banco de dados>
```

Primeiro as informações obtidas sobre os registros de versões: média em *bytes* do tamanho dos registros armazenados (*Average Record Length*); total de registros na tabela, incluindo ativos e inativos (*Total Records*); média do tamanho dos registros temporário (*Average Version Length*); total de registros temporários existentes (*Total Versions*) e número máximo de versões que um registro possui (*Max Versions*). Quanto maior o valor de *Total Versions*, maior será o espaço que será liberado com um *backup/restore*.

Sobre páginas de dados e de índices tem-se: número total de páginas de dados já alocadas para a tabela (*Data Pages*); número de ponteiros atualmente alocados para páginas de dados da tabela (*Data Page Slots*, deve ter obrigatoriamente o mesmo valor de *Data Pages*); média geral de ocupação das páginas alocadas (*Average Fill*); e média por faixas da ocupação das páginas (*Fill Distribution*).

Se a *Average Fill* for menor que 60% ou se a maior parte das páginas de dados não estiverem nas faixas 60-79% e 80-99%, então um *backup/restore* se faz necessário. Por último, nas estatísticas dos índices tem-se: número de páginas entre a primeira página do índice e as páginas *leaf* (*Depth*); número de páginas *leafs* (*Leaf Buckets*); total de páginas de índices que compõem a árvore (*Nodes*); total de linhas que apresentam chaves duplicadas/repetidas (*Total Dup*); e média por faixas da ocupação das páginas (*Fill Distribution*).

Se a distribuição da maior parte das páginas de índices não estiver na faixa 80-99%, é necessário que o índice seja reconstruído (veja como na Dica 9). Se o valor do *depth* for maior do que três, talvez seja preciso aumentar do tamanho das páginas de dados (ver Dica 5).

Nota: páginas *leafs* são aquelas que contêm informações que apontam para onde estão as linhas (registros) da tabela;

4. Tamanho da página (Page size)

O tamanho da página padrão utilizada pelo InterBase era de 1 KB, no Firebird esse valor passou para 4 KB, esse aumento por si só já gerou um ganho automático de desempenho.

Hoje os sistemas ocupam cada vez mais espaço em disco, dessa forma o ideal é que se use 8 KB como tamanho da página de dados. As razões para o aumento do tamanho das páginas são explicadas por KARWIN (1998):

- *Menos fragmentos de registros divididos:* é comum existir registros que ocupem mais de 4 KB, o que faz com que sejam armazenados em múltiplas páginas forçando o servidor a ler várias páginas para recuperar a informação;

- *Melhor utilização dos índices Btrees:* um número menor de páginas será usado para armazenar os índices, quanto menor o número de páginas mais rápido o

processamento;

- *Operações de I/O mais contínuas:* o servidor armazena os registros na primeira página livre, o que pode significar que registros sucessivos não sejam armazenados em páginas próximas (uma página só armazena registros de uma mesma tabela), páginas maiores implicam em mais dados da mesma tabela numa mesma página;

- *Buffer maior é igual a mais memória de cache:* o cache do Firebird é alocado em número de páginas e não por uma quantidade fixa de bytes. Quanto maior o cache maior a chance dos dados serem encontrados nele.

A leitura das vantagens apresentadas pode levar o leitor a pensar que quanto maior o tamanho da página, melhor o desempenho, mas isso nem sempre é verdade. O Firebird suporta páginas de dados com até 16 KB, entretanto para a maioria dos bancos, 8 KB é o tamanho

ideal. Páginas maiores que 8 KB só devem ser usadas por bancos de dados muito grandes ou que contenham tabelas que possuem elevado acesso e cujos registros são maiores do que 8 KB.

O tamanho das páginas de dados pode ser alterado a qualquer momento através de um *backup/restore*. De toda forma, seja qual for o tamanho do banco de dados e o tamanho de página escolhido, é aconselhável que se façam testes com os outros valores para determinar qual o melhor para cada situação.

5. Cache de leitura

O Firebird mantém um *cache* em memória com as últimas páginas de dados e de índices utilizadas. Como em qualquer *cache*, o ganho de performance depende basicamente da repetição de acessos aos mesmos dados.

O tamanho do *cache* é definido em número de páginas de dados. Por padrão, o tamanho alocado é de 2048 páginas, valor que pode ser verificado no arquivo *firebird.conf* que se encontra na pasta onde o banco foi instalado (procure por *DefaultDbCachePages*). Se o tamanho de cada página estiver definido em 1 KB, então 2 MB de memória serão usados. Se o tamanho da página for 4 KB, então 8 MB de memória será usado (baseado no exemplo apresentado em PRENOSIL, 2002).

No InterBase o valor padrão do *cache* era de 256, pois na época as máquinas possuíam pouca memória. Nos servidores de hoje, onde há grande quantidade de memória disponível, é altamente recomendável que aumente o tamanho do *cache* para obter uma melhor performance. É possível fazer isso através do utilitário GFIX:

```
gfix -buffers <valor> <banco de dados>
```

O tamanho máximo permitido é 65535, entretanto não se deve usar *cache* maior do que 10000, pois quanto maior o valor maior a necessidade de processamento, o que pode causar perda de desempenho (PRENOSIL, 2002).

Além disso, não se deve usar um valor muito alto ao ponto do servidor utilizar a memória virtual (SWAP), pois isso anularia a vantagem de usar um *cache* visto que o acesso ao disco é muito mais lento do que o acesso à memória.

O valor também não deve ultrapassar o

Listagem 1. Resultado da execução do GSTAT no banco de dados EMPLOYEE

```
Database header page information:
Flags                0
Checksum             12345
Generation           160
Page size            4096
ODS version          10.1
Oldest transaction   156
Oldest active        157
Oldest snapshot      157
Next transaction     159
Bumped transaction   1
Sequence number      0
Next attachment ID   0
Implementation ID    16
Shadow count         0
Page buffers         0
Next header page     0
Database dialect     3
Creation date        Jan 17, 2006 12:07:17
Attributes           force write

Variable header data:
Sweep interval:     20000
*END*
```

Listagem 2. Estatísticas da tabela SALES do banco de dados EMPLOYEE

```
(...)
SALES (138)
Primary pointer page: 218, Index root page: 219
Average record length: 67.30, total records: 33
Average version length: 0.00
Total versions: 0, max versions: 0
Data pages: 1, data page slots: 1,
average fill: 68%
Fill distribution:
 0 - 19% = 0
20 - 39% = 0
40 - 59% = 0
60 - 79% = 1
80 - 99% = 0
Index NEEDX (3)
Depth: 1, leaf buckets: 1, nodes: 33
Average data length: 2.00, total dup: 11,
max dup: 6
Fill distribution:
 0 - 19% = 1
20 - 39% = 0
40 - 59% = 0
60 - 79% = 0
80 - 99% = 0
(...)
```

número de páginas do banco, pois uma página do disco ocupa uma página na *cache*, que nunca é duplicada. É preciso experimentar os valores do *cache* e analisar o resultado de cada caso. O ganho pode chegar 30% (KARWIN, 1998).

Nota: O tamanho do *cache* também pode ser definido durante o *restore* através do parâmetro `BUFFERS: gbak -c -buffers 1234 <demais parâmetros>`

Para finalizar, a forma como *cache* é alocado difere da versão *Classic* para a *SuperServer*. Na versão *Classic* cada usuário possui uma instância do servidor, logo cada usuário possui seu próprio *cache*.

Na *SuperServer* um único *cache* é compartilhado entre todos os usuários, o que é mais eficaz uma vez que se dois ou mais usuários requisitarem a mesma informação, quando o segundo o fizer, os dados já estarão em memória.

Usuários que utilizam a versão *Classic* precisam tomar um cuidado maior ao definir o tamanho do *cache* já que um *cache* é alocado para cliente conectado ao banco. Por exemplo: se o tamanho do *cache* é 5000 e o *page size* do banco é 4 KB serão alocados 20 MB de memória para cada cliente conectado, caso existam 10 clientes simultâneos 200 MB de memória serão alocados. Existindo dois bancos no servidor serão 400 MB, existindo 3 serão 600 MB e assim por diante.

6. Cache de escrita

O Firebird possui dois modos de escrita: com *cache* (chamado de escrita síncrona ou *write-through*) e sem *cache* (chamado de escrita assíncrona ou *write-back*). O primeiro modo é o padrão na plataforma Windows, o segundo na plataforma Linux.

Na escrita síncrona cada operação de escrita é imediatamente passada pelo servidor do banco para o Sistema Operacional (observe a linha contínua na **Figura 1**). Na escrita assíncrona o banco armazena os dados em memória e atrasa a gravação dos mesmos, ou seja, múltiplas operações de escrita para uma página do *cache* são executadas na memória para depois serem gravadas no disco, o que resulta num melhor tempo de resposta para as operações de escrita (observe a

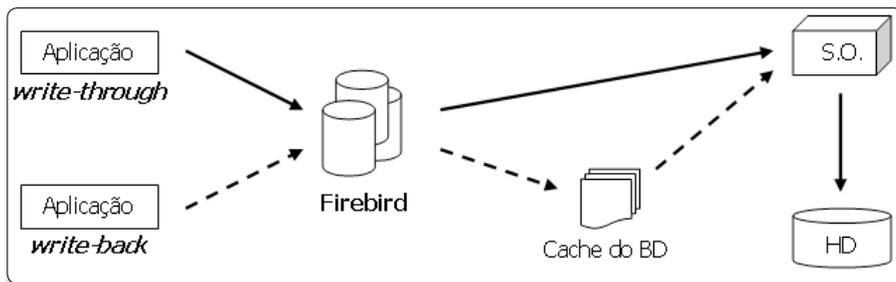


Figura 1. Representação dos modos de gravação *write-through* e *write-back*

linha tracejada na **Figura 1**).

Na escrita assíncrona a gravação física ocorre quando uma determinada quantidade de informação é coletada, um evento associado ocorre ou quando um intervalo de tempo tenha passado. Como se pode deduzir, o modo utilizado no Linux é mais eficiente do que o utilizado no Windows.

A configuração do modo como o *cache* deve se comportar pode ser alterada manualmente através do comando:

```
gfix -write async <banco de dados>
```

KARWIN (1998), diz que o uso de escrita assíncrona pode melhorar em cerca de 4 vezes o desempenho de aplicações padrões e em até 20 vezes aplicações que executam milhares de operações de escrita (INSERT, UPDATE, DELETE) em seqüência.

O autor alerta que o modo *write-back* somente deve ser usado em servidores estáveis e que contam com *nobreaks*, pois dados podem ser perdidos no caso do servidor sofrer uma queda de energia ou se for finalizado de maneira anormal. No modo *write-through*, isso não acontece.

7. Analise os Planos de Execução

A análise do PLAN de execução é o primeiro fator a ser levado em consideração para garantir o bom desempenho de um *query*, pois ele mostra como *engine* do banco de dados chega até os dados solicitados. Para analisar os planos de acesso será usado neste artigo o IBO-Console (www.mengoni.it/Downloads/IBOConsole.zip).

Como exemplo, a *query* que lista todos os funcionários que receberam um aumento maior ou igual a 7% (sete por cento):

```
select e.full_name, s.percent_change
from employee e, salary_history s
where
e.emp_no = s.emp_no and
s.percent_change >= 7
```

Ao analisar as estatísticas (**Figura 2**) pode-

se verificar que a tabela SALARY_HISTORY foi lida em sua totalidade (*full scan*).

Isso é indicado em PLAN pela palavra NATURAL após o "S" (alias que foi dado à tabela):

```
PLAN SORT (JOIN(S NATURAL, E
INDEX(RDB$PRIMARY7)))
```

O plano de execução também mostra que o acesso à tabela EMPLOYEE (alias "E") se deu através do índice da chave primária (RDB\$PRIMARY7). Outros dados importantes das estatísticas são *Execution Time* que mostra quanto tempo a *query* levou para ser executada, *Reads* e *Writes* quantas leituras e escritas foram feitas (respectivamente) e *Rows Affected* que mostra quantas linhas retornaram.

Se um índice for criado para o campo PERCENT_CHANGE, o plano de execução será modificado:

```
PLAN SORT (JOIN(S INDEX (IDX_SALARY_HISTORY),
E INDEX (RDB$PRIMARY7)))
```

Ou seja, agora o acesso à tabela SALARY_HISTORY também se deu através do índice. Se o leitor verificar o tempo que as duas *query* levaram para ser executadas (*Execution Time*) notará que praticamente não houve diferença, isso ocorreu apenas porque a base de dados é limitada.

E se a tabela guardasse o histórico de aumento de salário ao longo de uma década de uma empresa com centenas de funcionários? Provavelmente a tabela de histórico teria milhares de linhas e aí o tempo ganho seria significativo.

8. Sempre que possível use Joins e não SubSelects

É inegável que o *subselect* fornece ao programador uma alternativa eficiente para a solução de diversos problemas, porém essa alternativa muitas vezes faz com que o desempenho da *query* caia significativamente.

A explicação para a queda de desempenho é que um *subselect* é na verdade um *loop* aninhado. Se a *query* possui dois *subselects*, são três *loops* aninhados e assim sucessivamente.

A solução é, sempre que possível, usar um *joins*. Veja o exemplo da **Listagem 3**.

As duas consultas retornam o mesmo resultado: a lista de pessoas que trabalham no Canadá. Se o leitor analisar os planos de execução de ambas verificará que a primeira é mais lenta, pois precisa “varrer” toda a tabela EMPLOYEE para encontrar o resultado, enquanto a segunda utiliza os índices.

9. Use índices corretamente

Índices, quando utilizados corretamente, podem aumentar consideravelmente a performance de um SELECT. A seguir algumas dicas sobre índices (RODRIGUES, 2006):

- Crie índices para as colunas mais usadas em JOINS, exceto para aquelas que fazem parte de uma chave estrangeira (*Foreign Key*), pois essas já possuem índices;
- Crie também índices para as colunas mais usadas em ORDER BY, mas lembre-se que índices são ordenados de forma ascendente e ou descendente, logo só serão usados se a ordenação do comando

for a mesma do índice;

- Não crie índices em tabelas que possuem poucos registros (menos de 50);
- Reconstrua os índices periodicamente, pois as tabelas que sofrem muitas atualizações tendem a ficar com índices desatualizados. Para tanto, basta usar os comandos:

```
alter index <nome do índice> inactive;  
alter index <nome do índice> active;  
set statistics index <nome do índice>;
```

• Índices prejudicam o desempenho em operações de atualização de dados (INSERT, UPDATE, DELETE) porque são atualizados pelo banco durante essas operações. Pode-se evitar esse transtorno desativando os índices antes de realizar atualizações em lote e ativando-os logo após a operação;

• Se for preciso criar um índice para um campo com baixa seletividade (por exemplo, um campo cujas opções são SIM/NÃO), a melhor opção é criar um índice composto pelo campo desejado seguido de outro campo usado na pesquisa.

Na **Listagem 4** temos alguns exemplos para verificar o desempenho dos índices na prática.

Analisando os planos de execução verifica-se que na primeira e na segunda consulta o índice composto já existente na tabela (NAMEX, formado pelos campos LAST_NAME e FIRST_NAME, nessa ordem) foi usado, mas na última não.

A situação muda se um índice for criado para o campo FIRST_NAME, com o código da **Listagem 5**.

O índice criado é usado na primeira consulta, mas não na segunda. E se forem usadas as funções UPPER e CONTAINING? (**Listagem 6**).

Mais uma vez o índice não é utilizado, de onde se conclui: um índice composto só é usado se todos os campos que fazem parte dele forem usados na *query* ou se o(s) primeiro(s) campo(s) do índice estiver(em) nela.

O comando LIKE utiliza índice desde que não se utilize o “%” no início do parâmetro; UPPER e CONTAINING não utilizam índices. Para finalizar execute e análise os planos de execução.

Uma pergunta: como explicar o fato Firebird ter usado o índice na segunda consulta, se no parágrafo anterior foi dito

Listagem 3. Utilizando Join ao invés de SubSelects

```
select first_name, last_name  
from employee  
where  
  job_code in (select job_code  
              from job j  
              where j.job_country = 'Canada')  
  
select first_name, last_name  
from employee e, job j  
where  
  e.job_code = j.job_code and  
  j.job_country = 'Canada'
```

Listagem 4. Exemplos para verificar desempenho de índices

```
select first_name, last_name  
from employee  
where first_name like 'K%' and last_name like 'B%'  
  
select first_name, last_name  
from employee  
where last_name like 'B%'  
  
select first_name, last_name  
from employee  
where first_name like 'K%'
```

Listagem 5. Criando um índice para o campo FIRST_NAME

```
create index idx_first_name on employee(first_name)  
  
select first_name, last_name  
from employee  
where first_name like 'K%'  
  
select first_name, last_name  
from employee  
where first_name like '%e%'
```

Listagem 6. Usando as funções UPPER e CONTAINING

```
select first_name, last_name  
from employee  
where upper(first_name) like 'K%'  
  
select first_name, last_name  
from employee  
where first_name containing 'nn'
```

Listagem 7. Utilizando CONTAINING e LIKE

```
select first_name, last_name  
from employee  
where first_name like 'B%' or first_name  
  containing 'nn'  
  
select first_name, last_name  
from employee  
where first_name like 'B%' and first_name  
  containing 'nn'
```

que a função CONTAINING não utiliza-va índice? Veja a **Listagem 7**.

A presença do operador AND na segunda consulta faz com que um registro só seja mostrado se FIRST_NAME iniciar com 'B' e se contiver (em qualquer parte dele) 'nn'. Ciente disso, o otimizador do banco de dados executa primeiro a parte da consulta que utiliza o LIKE, fazendo uso do índice existente para o campo.

Isso reduz significativamente a quantidade de registros, e é somente nessa quantidade reduzida de registros que a segunda parte da consulta (que utiliza o CONTAINING) é executada.

Na verdade, embora o PLAN mostre que o índice foi utilizado, é preciso deixar claro que isso ocorreu apenas na primeira parte da busca, na segunda não houve uso de nenhum índice.

10. Não use chave-primária composta

Imagine o seguinte modelo de dados: aluno, curso e turmas (relação M:N entre aluno e curso). Para que o sistema esteja na terceira forma normal a tabela TURMA deve possuir uma chave primária composta pelas chaves primárias das tabelas ALUNO e CURSO ("Modelo

Normalizado" na **Figura 2**).

Entretanto, em termos de desempenho, essa não é a melhor opção. Encontrar um registro em uma tabela com chave composta é mais lento do que encontrar um registro numa tabela com chave primária simples, a explicação é óbvia: na primeira é preciso comparar dois valores, na segunda apenas um.

Nesse caso, o ideal é que se quebre a forma normal e crie-se uma chave primária simples (número inteiro seqüencial) e um índice composto único contendo as chaves primárias das demais tabelas ("Modelo Ideal" na **Figura 2**).

Conclusão

Este artigo mostrou que mesmo usando um banco de dados de fácil configuração, o Firebird exige por parte do seu administrador pequenos cuidados em relação às configurações como: tamanho da página de dados e de cache.

Saindo do lado do servidor e entrando no desenvolvimento propriamente dito, cabe ao programador tomar alguns cuidados para que o desempenho não seja afetado pela falta ou pelo mau uso dos índices. Para isso, cabe sempre analisar os planos de execução. ●

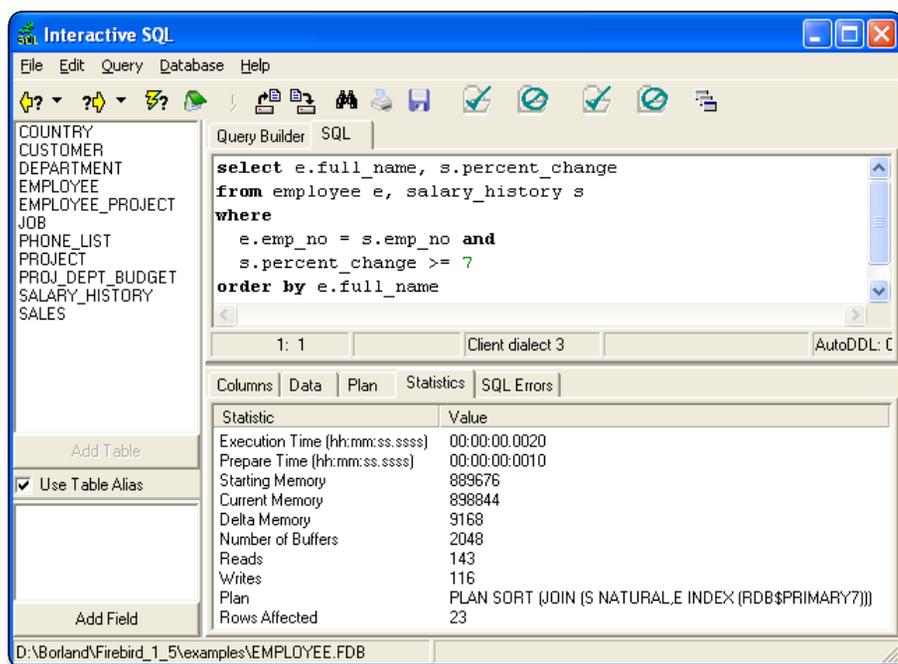


Figura 2. Estatísticas do plano de execução da query no IBOConsole

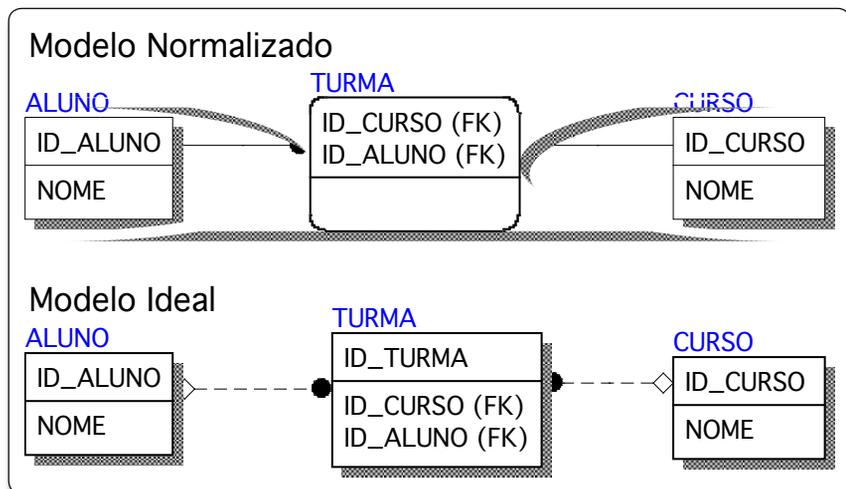


Figura 3. Modelo de dados Normalizado e Modelo de Dados Ideal

Referências Bibliográficas

KARWIN, Bill.
Ten Things You Can Do To Make InterBase Scream. Inprise Conference, Agosto de 1998.

CANTU, Carlos Henrique.
Entendendo o GSTAT – Parte I. DB Free Magazine, volume 1, Abril de 2005.

PRENOSIL, Ivan.
Cache settings. Disponível em www.volny.cz/iprenosil/interbase. Fevereiro de 2002

RODRIGUES, Anderson Haertel.
Índices, Performance e Estatísticas no Firebird. Disponível em www.comunidade-firebird.org. Acessado em Setembro de 2006.

