Banco de Dados Firebird

Rodrigo Gonçalves - Manager Systems

7 de dezembro de 2006

Capítulo 1

Introdução

Neste documento pretende-se fazer explicação o que é e como funcionar o banco de dados *Firebird*. Inicia-se com uma revisão sobre aspectos ligados a bancos de dados relacionais (à qual classe o *Firebird* pertence), introduzindo-se em seguida o Firebird. Faz-se uma revisão histórica e apresenta-se conceitos de estrutura e funcionamento do mesmo. Em seguida, passa-se as operações de manutenção do banco de dados, inciando pelo backup/restore e em seguida sobre o processo de identificação, análise e correção de eventuais problemas com o banco de dados.

Este manual é um documento *vivo*,sempre em constante renovação e melhoramentos, contando com a colaboração dos leitores para sua melhoria.

Uma cópia eletrônica (HTML) deste manual está disponível nos endereços: `http://www.managersystems.com.br/firebird/` e `http://sisclinica.hostmidia.com.br/firebird/`.

Versões

`Versão 1.0 - Versão inicial para o seminário sobre manutenção em bancos de dados Firebird`

Capítulo 2

Bancos de dados relacionais

Bancos de dados são sistemas destinados a armazenar grandes volumes de dados, provendo um acesso rápido e fácil aos dados. Existem diversas categorias de bancos de dados, das quais a mais importante e conhecida são os bancos de dados relacionais. Estes caracterizam-se basicamente por organizar as informações em tabelas (onde cada linha da tabela é um *registro*), estabelecendo ligações entre os dados destas tabelas.

Dos conceitos de banco de dados existentes, três são importantes para entender como funciona e também compreender a razão de ocorrerem determinados problemas. Estes conceitos são: *chave primária, chave estrangeira e índices*. Outros conceitos também serão apresentados.

Chave primária: Toda tabela em um banco de dados relacional deve conter uma *chave primária* (*Primary Key)*. Esta tem a função de identificar de forma única cada linha de uma tabela. Ou seja, pela chave primária pode-se saber exatamente de qual linha da tabela está-se falando.

Por exemplo, em uma tabela que represente os estados do Brasil, uma chave primária poderia ser a sigla do estado. Como cada linha seria um estado, um dado que poderia identificar unicamente cada linha da tabela seria a sigla do estado.

Uma das atribuições mais importantes de um banco de dados relacional é garantir que nunca existam duas linhas de uma tabela com um mesmo valor para a chave primária (por exemplo, dois estados com a mesma sigla).

Em nosso sistema (Sisclínica) estabelecemos uma chave primária padrão (adotada por quase todas as tabelas do banco de dados): a coluna `HANDLE`.

Chave estrangeira: Uma chave estrangeira (*Foreign Key*) estabeleca uma relação entre duas tabelas. Um exemplo de chave estrangeira que temos em nosso sistema é o relacionamento entre as tabelas de atendimentos (`CL_ATENDIMENTOS`) e a tabela de pacientes (`CL_PACIENTES`).

Na tabela de atendimentos, temos o campo `PACIENTE` que é uma *chave estrangeira* para a tabela de pacientes. Isso significa que o dado contido no campo `PACIENTE` é uma chave primária da tabela de pacientes. Através do valor contido no campo `PACIENTE` o banco de dados consegue identificar uma linha da tabela de pacientes, que é a linha sendo referenciada pelo atendimento.

Uma das funções básicas de um banco de dados é garantir que o valor de um campo que é chave estrangeira seja sempre válido, ou seja, seja sempre um valor correspondente a uma chave primária da tabela de pacientes.

Índices: Um índice em um banco de dados tem duas funções básicas: melhorar a performance do acesso aos dados, provendo uma maneira rápida de encontrar a informação desejada; e

também garantir a integridade, ou seja, a coerência dos dados no banco de dados. Para garantir que os conceitos de chave primária e chave estrangeira sejam respeitados, o banco de dado estabelece índices para cada chave estrangeira e chave primária criados.

Um exemplo de índices em nosso sistema é o índice criado em cima do nome do paciente na tabela de pacientes (`CL_PACIENTES`). Este índice garante uma melhor performance na pesquisa de pacientes no banco de dados pelo nome. Em geral os indices tem a sigla "`IDX`" em seu nome.

Stored Procedures: São rotinas criadas para executar determinadas tarefas de forma mais rápida, já que executam dentro do próprio banco de dados. Porém nem toda rotina pode ser transportada para dentro do banco de dados, pois este é limitado no que pode-se fazer dentro das *stored procedures*.

Generators: São elementos do banco de dados que geram números em seqüência de forma única (dois usuários não conseguem gerar o mesmo número). Estes são usados no nosso sistema por exemplo para gerar a chave primária de atendimentos e pacientes (o valor do campo HANDLE).

Triggers: São rotinas executadas quando um determinado evento ocorre com um registro de uma tabela. Este evento pode ser uma inclusão, exclusão ou alteração. Por exemplo, podemos ter uma trigger na tabela de países onde quando um país é incluído, a trigger gera um registro em outra tabela indicando qual usuário gerou aquele registro.

Capítulo 3

História

O Firebird surgiu em 1984, sob o nome de Interbase, fruto do trabalho de Jim Starkey e Ann Harrison (os *pais* do Firebird). O nome da empresa de Jim onde o Interbase surgiu era G*roton* D*atabase* S*ystems*, daí a clássica sigla *gds*. Em 1991 a companhia foi comprada pela Ashton Tate, que por sua vez foi comprada pela Borland logo em seguida.

De 1991 até 1999 a Borlando responsabilizou-se pelo desenvolvimento do Interbase. Em 1999, a Borland resolveu abrir o código fonte do Interbase de forma que desenvolvedores em todo o mundo pudessem trabalhar no seu desenvolvimento. Esta ficou conhecida como Firebrid. Ao mesmo tempo, criou uma versão própria do Interbase, com recursos adicionais desenvolvidos por ela e incorporando as melhorias acrescentadas ao Firebird.

Em resumo, o Firebird, banco de dados que utilizamos em nosso sistema, é um produto que já existe há quase 22 anos, tendo evoluído muito nesse período e que ainda continua a evoluir, tendo previsto o lançamento da versão 2.0 ao final do ano, que contará com consideráveis melhorias em relação à performance em múltiplos processados e na execução de consultas em geral.

Maiores informações sobre o Firebird podem ser encontradas nos sites `http://firebird.sf.net` e `http://www.firebase.com.br`.

Capítulo 4

Estrutura do sistema

O Firebird possui duas versões: a versão *SuperServer* a versão *Classic*. Entre as duas, a diferença está na forma como está estruturada o gerenciamento das conexões dos clientes. Na versão *SuperServer*, os recursos (valores recuperados do disco, consultas feitas) são compartilhados por todos os clientes conectados. Já na versão *Classic*, cada cliente tem sua própria *"instância do banco de dados"* e comunicam-se entre si através de um gerenciador de transações (`fb_lock_mgr`).

Nenhuma das duas versões é melhor ou pior, cada uma adequa-se melhor a cada caso. Para determinadas situações a versão *SuperServer* é melhor (pouca memória no servidor, número médio ou pequeno de clientes conectados, computador uniprocessado).

Em outras situações a versão *Classic* sai-se melhor (muita memória disponível, computados multiprocessados). A vantagem da versão Classic é tratar os usuários de forma independente, podendo melhor balancear os usuários entre os recursos do servidor. Uma desvantagem da mesma é gerar um "stress" maior para determinados pontos do banco de dados compatilhados entre diversos usuários (como por exemplo a tabela *CL_PREFERENCIAS*.

Em sistemas Windows, o Firebird pode ser instalado tanto como um aplicativo (ficando *ao lado do relógio*) ou como um serviço, não visível ao usuário. No Linux, ele sempre ocupa a pasta `/opt/firebird/` e pode ser executando tanto como um serviço (`/etc/init.d/firebird` como no caso da versão *SuperServer* como também como um processo de sistema `xinetd` (`/etc/xinet.d/firebird`. Entre as formas SuperServer e Classic muda a forma como para-se ou inicia-se o banco de dados.

Capítulo 5

Estruturas do banco de dados

Nesta seção vamos conhecer como o Firebird armazena a estrutura do banco de dados. Para isto, ele utiliza as *tabelas de sistemas*, que podem ser reconhecidas por iniciarem com a sigla `RDB`. Dentre as tabelas existentes, convém destacar:

- *[RDB$RELATIONS]* Esta tabela contém os dados básicos de todas as tabelas do banco de dados. Por exemplo, para obter o nome de todas a tabelas do banco de dados, podemos executar o SQL: `select rdb$relation_name from rdb$relations`.
- *[RDB$RELATION_FIELDS]* Esta tabela contém os campos que uma determinada tabela possui. Por exemplo, para obter o nome de todas os campos de uma tabela, podemos executar o SQL: `select rdb$field_name from rdb$relation_fields where rdb$relation_name=''CL_USUARIO''`.
- *[RDB$FIELDS]* Esta tabela contém a definição dos campos da tabelas do banco de dados. A tabela `RDB$RELATION_NAMES` contém apenas o nome dos campos e uma referência ao tipo do campo, um registro da tabela `RDB$FIELDS`. Por exemplo, é através desta tabela que o *PowerConnection* consegue determinar como deve gerar as triggers que fazem a replicação dos dados (identificando o tipo de dados de cada campo corretamente).
- *[RDB$INDICES]* Aqui ficam armazenadas as informações sobre os índices do banco de dados. Através desta tabela pode-se determinar os índices de uma dada tabela (`select rdb$index_name from rdb$indices where rdb$relation_name=''CL_USUARIO''`) assim como determinar também se um dado índice de uma tabela está ativo ou não (`select rdb$index_name from rdb$indices where rdb$relation_name=''CL_USUARIO'' and rdb$index_inactive=1`).
- *[RDB$INDEX_SEGMENTS]* Aqui ficam armazenados os campos que compõe um índice. Para obter os campos que compõe um dado índice, podemos usar o seguinte sql: `select rdb$field_name from rdb$index_segments where rdb$index_name=''PK_CENTROCUSTOS''`.
- *[RDB$TRIGGERS]* Nesta tabela ficam armazenas as informações sobre as triggers do sistema. Para determinar por exemplo as triggers de uma tabela, podemos usar o sql `select rdb$trigger_name from rdb$triggers where rdb$relation_name=''CL_USUARIO''`.
- *[RDB$GENERATORS]* Nesta tabela ficam armazenas as informações sobre os *generators* que existem no banco de dados. Para saber o nome dos generators que existem

no banco de dados,podemos usar o seguinte sql: `select rdb$generator_name from rdb$generators where rdb$system_flag=0`.

- *[RDB$RELATION_CONSTRAINTS]* Aqui estão contidas as informações sobre as restrições de integridades (*Foreign Keys* e *Primary Keys*) da tabelas do banco de dados. Para determinar por exemplo as *Foreign Keys* de uma tabela, podemos usar o comando sql: `select rdb$constraint_name from rdb$relation_constraints where rdb$relation_name=''cl_usuario'' and rdb$constraint_type=''FOREIGN KEY''`. Para obter a chave primária, bastaria trocar o "`FOREIGN KEY`" por "`PRIMARY KEY`".

- *[RDB$REF_CONSTRAINTS]* Esta tabela armazena as informações sobre as *Foreign Keys* do banco de dados. Ela indica qual campo contém a chave estangeira e qual campo ele referencia. Para isso, ela armazena os índices da chave primária e do campo que contém a chave estrangeira. Por exemplo, para obter qual tabela referência uma determinada *Foreign Key*, podemos usar o seguinte sql:
```
select a.rdb$constraint_name,
a.rdb$const_name_uq,b.rdb$relation_name,c.rdb$field_name
from rdb$ref_constraints a inner join rdb$indices b
on ( rdb$index_name=a.rdb$const_name_uq ) inner join
rdb$index_segments c on (c.rdb$index_name=b.rdb$index_name) where
rdb$constraint_name=''FK_MOV_PLANOS''.
```

- *[RDB$PROCEDURES]* Nesta tabela ficam armazenas as informações sobre as *procedures* que existem no banco de dados. Para saber o nome das procedures que existem no banco de dados,podemos usar o seguinte sql: `select rdb$procedure_name from rdb$procedures where rdb$system_flag=0`.

Capítulo 6

Acessando as estruturas do banco de dados

Acessar as informações do banco de dados pelas tabelas de sistema é uma forma válida (e necessária em alguns casos) porém na maioria das vezes, pode-se usar alguns comandos mais simples para ver a estrutura do banco de dados, utilizando o programa ISQL que acompanha o Firebird.

No Windows, para iniciar o mesmo, basta ir para a pasta `C:\Arquivos de Programas\Firebird\Firebird_1_5\bin` e executar:

```
ISQL caminhoDoBanco -user sysdba -pass masterkey.
```

No Linux basta executar (de qualquer lugar) o comando:

```
/opt/firebird/bin/isql caminhoDaBase -user sysdba -pass masterkey.
```

Um vez dentro do ISQL, os seguintes comandos são úteis (todos os comandos no ISQL devem ser terminados com ; (ponto e vírgula):

- `show tables` Mostra as tabelas do banco de dados;
- `show table nomeDaTabela` Mostra a estrutura de uma tabela (mostra os campos, chaves estrangeiras e chave primária, além das triggers);
- `show triggers` Mostra as triggers do banco de dados;
- `show trigger nomeDaTrigger` Mostra o conteúdo de uma trigger;
- `show generators` Mostra os generators do banco de dados, exibindo o valor atual de cada um;
- `show procedures` Mostra as procedures do banco de dados;
- `show procedure nomeDaProcedure` Mostra o código de uma procedure.

No decorrer deste manual, os comandos SQLs poderão ser executados no ISQL. Para sair de uma sessão do ISQL basta executar o comando:

```
QUIT;
```

Capítulo 7

Ferramentas do Firebird

O Firebird tem três principais ferramentas de gerenciamento: o ISQL (já apresentado anteriormente), o GBAK e o GFIX. Estes dois últimos serão apresentados agora.

7.1 GBAK

Esta ferramenta é responsável por duas operações no banco de dados: efetuar o *backup* do banco de dados e restaurar o banco de dados a partir de um backup (*restore*).

7.1.1 Backup

A sintaxe padrão do comando de backup no Firebird é:

```
/opt/firebird/bin/gbak -B -V -USER sysdba -PASS masterkey
servidor:/caminhoDaBase nomeDoArquivoFBK
```

Onde nomeDoArquivoFBK é o arquivo de backup a ser gerado. Caso queira-se gerar mais de um arquivo de backup (no caso de bancos de dados muito grandes - mais de 1 giga), basta especificar o tamanho de cada arquivo gerado em gigas, segundo a sintaxe:

```
/opt/firebird/bin/gbak -B -V -USER sysdba -PASS masterkey
servidor:/caminhoDaBase nomeDoArquivoFBK 1G nomeDoArquivoFBK2
1G nomeDoArquivoFBK3
```

O último arquivo não deve ter o tamanho especificado e não há problema caso sejam especificados mais arquivos que o necessário.

Alguns parâmetros úteis no backup de banco de dados com problemas:

-G Este parâmetro especifica que o banco não deve fazer uma otimização dos dados durante a operação de backup. Isto possibilita o banco de alguns bancos de dados com problema em dados não mais utilizados que seriam consultados pelo backup. Porém este parâmetro leva a um banco de dados com dados velhos na hora do restore, necessitando em seguida de um novo backup/restore (ou pelo menos um gfix -sweep) para levar o banco de dados a um estado ótimo.

-L Este parâmetro é semelhante ao parâmetro -G, porém têm um risco de perda de dados (extremamente reduzido). Por isso deve ser usado apenas em último caso.

Estes dois parâmetros devem ser usados somente quando ao tentar-se executar um procedimento de backup normal o mesmo aborta com algum erro.

7.1.2 Restore

A sintaxe padrão do comando de restore no Firebird é:

```
/opt/firebird/bin/gbak -R -V -P 4096 -USER sysdba -PASS masterkey
nomeDoArquivoFBK caminhoDaBase
```

Onde nomeDoArquivoFBK é o arquivo de backup gerado. Caso tenha-se gerar mais de um arquivo de backup (no caso de bancos de dados muito grandes - mais de 1 giga), basta especificar o nome de cada arquivo gerado, segundo a sintaxe:

```
/opt/firebird/bin/gbak -R -V -P 4096 -USER sysdba -PASS masterkey
nomeDoArquivoFBK nomeDoArquivoFBK2 nomeDoArquivoFBK3  caminhoDaBase
```

Caso tenha-se um banco de dados muito grande, pode-se particioná-lo segundo a sintaxe:

```
/opt/firebird/bin/gbak -R -V -P 4096 -USER sysdba -PASS masterkey
nomeDoArquivoFBK nomeDoArquivoFBK2 nomeDoArquivoFBK3  caminhoDaBase1
250000 caminhoDaBase2 250000 caminhoDaBase3
```

Onde o último arquivo da base não deve ter o tamanho especificado. O tamanho aqui refere-se ao *número de páginas* que cada arquivo vai ter. Para transformar em bytes, basta multiplicar o valor pelo número 4096 (o valor especificado no parâmetro -P). Neste caso, 250000 * 4096 = 1024000000, que é aproximadamente 1 Giga.

Alguns parâmetros úteis no restore de um banco de dados com problemas:

`-I` Este parâmetro desabilita os índices do banco de dados. Em casos onde existem dados com problema (chaves estrangeiras referenciando registros que não existem mais, chaves primárias duplicadas, etc) que não podem ser resolvidos antes do backup do banco de dados (um UPDATE não tem sucesso para tentar corrigir este problema), esta opção permite restaurar o banco de dados integralmente e fazer as correções devidas. Em seguida pode-se reativar os índices do banco de dados manualmente (isto será exemplificando nas seções seguintes).

7.2 GFIX

O GFix é a ferramenta de manutenção de bancos de dados Firebird. Através dela é possível executar uma série de operações de manutenção no banco de dados visando corrigir o banco tanto para continuar a usá-lo, quanto para executar um posterior backup. Operações de manutenção de performance também são feitas através desta ferramenta.

A sintaxe básica do GFIX é:

```
/opt/firebird/bin/gfix comando servidor:bancoDeDados
```

Os comandos que merecem destaque no GFIX são:

`-buffers` Este item especifica o tamanho do buffer de dados que o banco de dados deve utilizar, impactando na performance do banco de dados. Valores até 10.000 são adequados. Em geral não é necessário mexer neste parâmetro a menos que busque-se um melhor ajuste de performance.

`-validate -full` Este comando analisa o banco de dados, identificando e reportando problemas encontrados. Ele *não* faz a correção dos problemas.

`-mend -full` Este comando analisa e inutiliza as partes danificadas do banco de dados. Dependendo da extensão do problema, uma operação de backup/restore após ele é obrigatória. Não há como analisar se deve-se ou não fazer o backup/restore a não ser pelo acompanhamento do banco de dados após o procedimento de gfix -mend -full.

`-shut -force 0` Este comando derruba todos os usuários conectados ao banco de dados, permitindo que acesse-se o mesmo de forma exclusiva. Em qualquer operação de manutenção ou backup/restore este comando deve sempre ser executado antes.

`-online` Este comando deve sempre acompanhar o comando de -shut -force 0. Ele reativa o banco de dados para conexões de usuários. Sem este comando os usuários não conseguirão utilizar o banco de dados.

`-sweep` Este comando faz um procedimento de manutenção no banco de dados, reorganizando estruturas e corrigindo erros de menor importância. É algo semelhante ao que uma ferramenta de desfragmentação de disco faz.

`-housekeeping valor` Este comando ativa o sistema de execução automática de sweep do banco de dados. Em geral esta execução automática impacta em performance do sistema em momentos aleatórios. Portanto deixá-la desligada (valor 0) é recomendado.

`-ignore` Em procedimentos de -validate e -mend, este comando adicional pode ser incluído para indicar que o banco de dados deve ignorar validações adicionais dos dados. Em alguns casos estas validações impedem a manutenção e posterior backup do banco de dados. Porém nestes caso pode ocorrer a perda de dados. Portanto esta opção deve ser usada apenas quando realmente necessário.

`-write sync` Este comando indica que o banco não deve fazer *caching* dos dados a serem gravados em disco. Em servidor onde pode ocorrer a falta repentina de energia (um cabo de força puxado, um servidor sem no-break) é recomendado que este comando seja executado. Recomenda-e na prática sempre trabalhar com este comando ativo.

`-z` Com este comando consegue-se a versão do Firebird.

Capítulo 8

Problemas

8.1 Introdução

O Firebird é um banco de dados que já está no mercado há muito tempo e têm uma reconhecida estabilidade e confiança. Porém, como todo sistema computacional complexo, não pode ser considerado infalível. Fatores relacionados ao hardware, sistema operacional ou mesmo ao próprio Firebird podem vir eventualmente a falhar, gerando problemas no banco de dados.

Do ponto de vista do hardware, diversos elementos podem contribuir para uma falha: uma memória danificada, uma controladora de disco que corrompeu determinado segmento de dados, um hd danificado, um hd que corrompeu determinado dado gravado em disco, etc. Devido à intensa utilização que um banco de dados faz do hardware, qualquer menor falhar pode vir a ocasionar problemas. Diversos algorítimos já estão implementados no Firebird e no próprio sistema operacional visando contornar estes problemas, porém estes também não são infalíveis.

Do ponto de vista do sistema operacional, um buffer de disco não gravado (por causa de uma falta de energia), um erro na gravação dos dados em disco devido a um problema anterior com o sistema de arquivo devido à alguma falha de hardware, um *kernel* com alguma falha nos elementos utilizados pelo banco de dados (semáforos, memória, etc.) também podem ocasionar eventuais problemas ao banco de dados.

Um exemplo muito conhecido é o `scandisk`, existente na versão 9x do Windows - este é conhecido por ocasionar problemas graves no banco de dados ao "recuperar" clusters perdidos. Para um banco, uma falha na estrutura dos seus dados em disco pode significar desde um banco que não pode ser utilizado até que seja feito um procedimento de manutenção até um banco de dados que nunca mais poderá ser utilizado.

Já em relação próprio Firebird, ele pode ter (e provavelmente contém) problemas que manifestam-se apenas em situações muito específicas e de difícil identificação e posterior correção. Com a grande base de usuários e desenvolvedores que tem, todos os problemas conhecidos vêm sido sanados e os problema identificados em geral são muito específicos, não havendo falhas de grande porte.

8.2 Causas

Identificar a causa de um problema com o banco de dados é algo complexo e em geral não conclusivo. Em alguns casos, pode-se identificar claramente o problema, quando por exemplo o servidor foi desligado indevidamente (buffers de disco não gravados que perderam-se), quando um HD está danificado (erros no log do sistema operacional). Porém em outros casos é muito difícil detectar a causa do erro, como por exemplo no caso de uma memória danificada.

Através da análise de diversos elementos do banco de dados e do sistema operacional, podemos tentar identificar a causa do erro, como será visto na seção seguinte, relacionada a análise de problemas com o banco de dados.

8.3 Análise

O ponto de partida para analisar um problema com o banco de dados é verificar a mensagem de erro que o mesmo está retornado. Por esta mensagem já podemos ter um diagnóstico base do problema. Em geral, ao usuário sempre chegará uma mensagem similar a esta:

```
internal gds software consistency check (can't continue after bugcheck)
```

Esta mensagem é resultado dos mecanismos de proteção do Firebird, que ao identificar que houve um problema com o banco de dados, impede que o mesmo continue a operar. Assim evita que o problema já existente aumente ainda mais.

Outras mensagens que pode aparecer ao usuário são:

```
internal gds software consistency check (Too many savepoints (287))
```

E também:

```
internal gds software consistency check (error during savepoint backout
(290)).
```

Aqui temos um problema interno do Firebird, que devido a um volume muito grande de alterações em um ponto específico do banco de dados, acabou ocorrendo um problema de controle destas alterações e para evitar problemas, o sistema achou melhor suspender o processo. Na versão 2.0 do Firebird alguns aspectos do banco foram melhorados buscando evitar a ocorrência deste problema.

Já para as seguintes mensagens:

```
Internal gds software consistency check (cannot find tip page (165))

Database file appears corrupt.  Wrong page type.  Page NNN is of wrong type
(expected X, found Y)

Unknown database I/O error for file "*.gdb".  Error while trying to read
from file.

Internal gds software consistency check (decompression overran buffer (179))

Internal gds software consistency check.  Wrong record length

Database file appears corrupt.  Bad checksum.  Checksum error on database
page XX.

Internal gds software consistency check (cannot find record back version
(291))

Corrupted header page (at least)
```

```
internal gds software consistency check (partner index description not found
(175));
```

Temos problemas mais graves, relacionados ao corrompimento da estrutura do banco de dados. Aqui as origens dos problemas são diversas, podendo ser tanto de hardware quanto software.

Já com a mensagem :

```
INET/inet_error:  read errno
```

temos um erro de rede que não afeta o banco de dados, mas se estiver repetindo-se com uma freqüência muito alta pode indicar um problema com a rede onde o servidor está instalado (rede de dados - ethernet).

8.4 Identificando o problema

Temos algumas maneira para identificar um problema com o banco de dados. A maneira mais simples e direta é analisar o arquivo de log do Firebird. No linux o nome do arquivo é `/opt/firebird/firebird.log`. No Windows o arquivo de log é `C:\Program Files\Firebird\Firebird_1_5`.

Algumas mensagens comuns no arquivo de log do Firebird e suas explicações:

```
DEVGONCALVES (Client) Fri May 19 15:50:45 2006
INET/inet_error: connect errno = xxxxx
```

```
rot.local.ecomax-ubr.com.br Thu Jun  1 16:49:25 2006
INET/INET_connect: gethostbyname failed, error code = 1
```

Esta mensagem de erro indica algum problema com a conexão de rede de algum cliente. A menos que exista um volume muito grande deste tipo de mensagens em um curto espaço de tempo, é uma mensagem que pode seguramente ser ignorada.

```
rot.local.ecomax-ubr.com.br Thu May 25 10:55:37 2006
SERVER/process_packet: broken port, server exiting
```

Esta mensagem de erro indica algum problema com a conexão de rede de algum cliente. A menos que exista um volume muito grande deste tipo de mensagens em um curto espaço de tempo, é uma mensagem que pode seguramente ser ignorada. Ela pode ser gerada também pelo fato do servidor estar sobrecarregado e descartando algumas conexões.

```
DEVGONCALVES (Client) Fri May 26 16:30:54 2006
REMOTE INTERFACE/gds_detach: Unsuccesful detach from database.
Uncommitted work may have been lost
```

Esta mensagem indica que um cliente iniciou um processo no banco e desconectou sem concluir o processo. Neste caso o banco descartou as alterações feitas pelo cliente sem prejuízo a integridade dos dados. Este tipo de erro pode impactar um pouco a performance do banco mas não a ponto de preocupar. Se o número destas mensagens for muito grande em um curto espaço de tempo, existe algum problema com o servidor ou a rede.

```
linuxserver (Client)    Thu May 18 14:42:47 2006
        /opt/firebird/bin/fbguard: guardian starting bin/fbserver
```

Esta mensagem indica que o banco de dados foi reiniciado. Ela ocorre tanto quando o Firebird é reiniciado quando também quando ocorre algum problema grave com o banco de dados e ele é finalizado. Neste caso o guardião (fbguard) reinicia o processo do banco de dados. No caso do Firebird Classic, isto não ocorre, pois a cada conexão de um usuário uma nova instância do banco de dados é inicializada.

```
rot.local.ecomax-ubr.com.br Wed May 24 19:54:39 2006
ISC_kill: process 6023 couldn't deliver signal 16 to process 6869: permission denied
```

Esta mensagem indica que ocorreu um erro na comunicação entre os processos do Firebird. Não é uma mensagem que pode ser interpretada como normal mas a menos que repita-se com certa freqüência não merece uma análise mais detalhada. Em resumo ela indica que o gerenciamento do Firebird ordenou que algum dos processos seja finalizado porém isso não foi possível no momento.

```
rot.local.ecomax-ubr.com.br Wed May 24 19:54:43 2006
Database: /opt/firebird/security.fdb
internal gds software consistency check (page in use during flush (210))
```

Este erro indica que o banco de dados tentou gravar em disco alguma alteração porém os dados estavam sendo usados por outro processo (isto pode ocorrer por exemplo devido a um processo que não foi morto quando deveria - ver a mensagem de erro anterior). Neste caso, é recomendável parar o banco e executar um procedimento de manutenção simples (`gfix`) para evitar futuros problemas.

```
rot.local.ecomax-ubr.com.br Wed May 24 19:54:43 2006
Fatal lock manager error: invalid lock id (730336), errno: xx
```

Esta mensagem relaciona-se a algum erro com o controle de concorrência do Firebird - está relacionada ao clássico *deadlock*. A menos que repita-se com uma freqüência alta, não é algo alarmante.

```
rot.local.ecomax-ubr.com.br Wed May 24 20:09:58 2006
Database: /manager/gdb/antEcomaxUBR2000.GDB
Page 19579 wrong type (expected 5 encountered 7)

rot.local.ecomax-ubr.com.br Wed May 24 20:09:58 2006
Database: /manager/gdb/antEcomaxUBR2000.GDB
Data page 19579 (sequence 0) is confused in table SISSINCENVIO (401)

rot.local.ecomax-ubr.com.br Wed May 24 20:10:00 2006
Database: /manager/gdb/antEcomaxUBR2000.GDB
Page 19594 is use but marked free

rot.local.ecomax-ubr.com.br Wed May 24 20:10:00 2006
Database: /manager/gdb/antEcomaxUBR2000.GDB
Page 19843 is an orphan

rot.local.ecomax-ubr.com.br Fri Jun  2 07:53:26 2006
Database: /manager/gdb/_EcomaxUBR2000.GDB
Chain for record 246 is broken in table CL_PREFERENCIAS (143)
```

Aqui temos uma mensagem de erro grave no banco de dados. Algum problema ocorreu com os dados do banco e onde este esperava um determinado tipo de informação encontrou outra. Recomenda-se aqui urgentemente parar o banco de dados e executar um procedimento de `gfix`, acompanhado de um backup/restore.

Se a mensagem apareceu apenas uma vez, o procedimento de backup/restore pode ser descartado desde que acompanhe-se a o banco de dados por algum tempo após a operação de `gfix`. Porém se alguma das mensagens aparece sucessivas vezes, o procedimento de backup/restore é mandatório para evitar futuros danos mais graves ao banco de dados.

```
rot.local.ecomax-ubr.com.br Thu May 25 14:49:27 2006
Fatal lock manager error: semaphores are exhausted, errno: 4
```

Esta mensagem indica que está havendo alguma sobrecarga no servidor ao ponto em que este não está conseguindo gerenciar os processos do banco de dados. Dois parâmetros podem ser mudados no arquivo `/opt/firebird/firebird.conf` de forma a evitar este problema: o parâmetro `LockSemCount` e `LockMemSize`. Aumentando os valores deste parâmetros aumenta a quantidade de recursos do servidor disponibilizada para o gerenciamento dos processos do Firebird.

```
rot.local.ecomax-ubr.com.br Fri Jun  2 07:07:05 2006
Database: /manager/gdb/EcomaxUBR2000.GDB
internal gds software consistency check (cannot find record back version (291))

rot.local.ecomax-ubr.com.br Fri Jun  2 07:53:25 2006
Database: /manager/gdb/_EcomaxUBR2000.GDB
Relation has 55 orphan backversions (3 in use) in table Z_TABELAS (130)
```

Aqui temos uma mensagem que indica um problema no gerenciamento das alterações de dados no banco de dados. Em geral este tipo de erro ocorre devido a uma sobrecarga de um determinado componente do banco de dados (como o registro de uma tabela) e para evitar danos mais graves (como o retorno de dados inconsistentes), o banco de dados aborta o processo que está tentando acessar este registro.

Recomenda-se aqui urgentemente parar o banco de dados e executar um procedimento de `gfix`, possivelmente acompanhado de um backup/restore. Porém se a mensagem continuar a aparecer, o procedimento de backup/restore é mandatório para evitar futuros danos mais graves ao banco de dados.

```
rot.local.ecomax-ubr.com.br Wed May 24 20:09:58 2006
Database: /manager/gdb/antEcomaxUBR2000.GDB
Index 1 is corrupt on page 744056 in table SISSINCENVIO (401)

rot.local.ecomax-ubr.com.br Fri Jun  2 07:55:05 2006
Database: /manager/gdb/_EcomaxUBR2000.GDB
Index 9 is corrupt (missing entries) in table CL_CONSULTAS (267)
```

Aqui temos uma mensagem de erro grave no banco de dados. Algum problema ocorreu com os dados do banco e os índices estão corrompidos. Deve-se parar o banco de dados e executar um procedimento de gfix, acompanhado de um backup/restore.

Um ponto importante aqui é salientar que mesmo que o arquivo de log apresente um número muito grande de problemas, isto não significa que o banco de dados está *perdido*. Em 99% dos casos é possível recuperar o banco de dados.

8.4.1 Identificando problemas com o sistema operacional e hardware

Para identificar problemas com o sistema operacional, podemos analisar o arquivo de log do sistema. Vamos considerar aqui somente como analisar no caso do Linux. Em Windows pode-se obter informações a respeito de problemas com o sistema operacional ou hardware no Log de Eventos (em Ferramentas Administrativas).

No caso do Linux, o arquivo que contém o log do sistema é /var/log/messages. As seguintes mensagens indicam problemas com o disco rígido:

```
Dec 11 15:32:02 lan-2 kernel: Buffer I/O error on device sda2,
logical block 45058

Dec 11 15:32:02 lan-2 kernel: lost page write due to I/O
error on sda2

Dec 11 15:32:02 lan-2 kernel: Buffer I/O error on device sda2,
logical block 45059

hdb: drive_cmd: status=0x51 { DriveReady SeekComplete Error }

hdb: drive_cmd: error=0x04 { DriveStatusError }
```

Já esta mensagem em 95% dos casos indica um problema com o hardware da máquina, provavelmente a memória.

```
CPU : 0
EIP : 0018:[<c01139bb2>] Not tainted
EFLAGS : 00253046
eat: c1186070 ebx: c11860707 ecx: 00001400 edx: c03e4d00
esi: 0000ffa0 edi: 000001f0 ebp: c04a8d80 esp: c1199ed4
ed: 0018 es: 0013 ss: 0018
Process swapper (pid: 1,stackpage = c1199000)
Stack : c04a8d80 0000ffa0 c03f8bf0 c04a8d80 c02d1992 c1186070 000001f0
 c1190400
00000800 c04a9198 c04a8d80 c02d1d76 c04a8d80 c1190400 c04434cd c04a8d80
0000ffa0 00000008 c02d8118 c04a8d80 0000ffa0 00070400 c04a8d80 c03f8bf0
Call Trace: [<c02d1992>] [<c02d1076>] [<c02d0118>] [<c0200381>] [<c0105000>]
[<c02d0585>] [<c02bc0c0>]
[<c0105088>] [<c0105000>] [<c0197556>] [<c0105060>]
Code: 3001 0f 84 06 ff ff ff 0f 0b d6 04 05 04 39 c0 e9 f9 fe ff
<0> Kernel panic:
```

8.5 Corrigindo problemas

Vai-se agora analisar os procedimentos básicos padrões para a manutenção do banco de dados. Inicialmente apresenta-se os procedimentos padrões de manutenção, backup e restore do banco de dados, já identificando opções e parâmetros úteis para a correção de problemas com o banco de dados. Vai usar-se aqui a sintate do Linux para exemplificar os processos.

8.5.1 Manutenção

Uma vez identificado um problema com o banco de dados, podemos estabeler um conjunto de passos visando corrigí-lo. Vamos analisar agora os passos um a um. Depois vamos analisar como tratar alguns casos especiais de danos em bancos de dados. Vamos assumir aqui que o banco de dados está no arquivo /manager/gdb/base.gdb. Convém destacar que em bancos com múltiplos arquivos, nunca será necessário especificar todos os nomes dos arquivos do banco de dados. Basta sempre especificar apenas o primeiro arquivo.

8.5.2 Procedimento padrão de manutenção

Inicialmente devemos derrubar o banco de dados, evitando que outro usuário tenha acesso ao mesmo. Para isto executamos o comando:

```
/opt/firebird/bin/gfix -shut -force 0
/manager/gdb/base.gdb -user sysdba -pass masterkey
```

Uma vez derrubado o banco de dados, devemos garantir que nenhum processo está usando o banco de dados. Para tal, usamos o seguinte comando:

```
fuser -v caminhoDaBase
```

Se existe algum processo usando o arquivo, podemos terminá-los através do comando:

```
fuser -k caminhoDaBase
```

Uma vez derrubado o banco de dados, devemos em seguida renomeá-lo para evitar novas conexões ao mesmo durante o processo de manutenção:

```
mv /manager/gdb/base.gdb /manager/gdb/_base.gdb
```

Derrubamos novamente o banco para garantir que não temos nenhuma conexão pendente:

```
/opt/firebird/bin/gfix -shut -force 0 /manager/gdb/_base.gdb
-user sysdba -pass masterkey
```

Deve-se então fazer uma cópia dos arquivos do banco de dados.

```
mkdir /manager/gdb/bkp
cp /manager/gdb/base*.GDB /manager/gdb/bkp/
```

Com o banco derrubado e copiado, executamos o comando do gfix para analisar os problemas com o banco de dados:

```
/opt/firebird/bin/gfix -validate -full /manager/gdb/_base.gdb
-user sysdba -pass masterkey
```

Ao final ele irá indicar os problemas encontrados. Podemos então solicitar ao banco de dados que tente corrigir os problemas:

```
/opt/firebird/bin/gfix -mend -full /manager/gdb/_base.gdb
-user sysdba -pass masterkey
```

Terminado o processo, executamos novamente o comando de análise do banco de dados:

```
/opt/firebird/bin/gfix -validate -full /manager/gdb/_base.gdb
-user sysdba -pass masterkey
```

No final, caso não seja apresentado nenhum erro, pode-se considerar o banco de dados como corrigido. Caso ainda existam erros, deve-se executar o procedimento de backup/restore do banco de dados para que o mesmo fique em um estado íntegro. Porém a menos que seja crítico colocar o banco no ar o quanto antes, recomenda-se *sempre* fazer o backup/restore do banco.

Não pode-se esquecer de, ao final do processo, reativar o banco de dados para que os usuários possam utilizá-lo:

```
/opt/firebird/bin/gfix -online /manager/gdb/base.gdb
-user sysdba -pass masterkey
```

Em situações normais esta seqüência de procedimentos é capaz de recuperar um banco de dados com problemas. Porém têm-se situações onde alguns passos adicionais serão necessários para que consiga-se fazer o backup/restore do banco de dados.

Pode-se dividir os problemas em dois grandes grupos: problemas que ocorrem durante o backup do banco de dados e problemas que ocorrem durante o restore.

8.5.3 Erros durante o backup

No caso de problemas que ocorrem durante o backup, sugere-se executar novamente o comando de gfix :

```
/opt/firebird/bin/gfix -mend -full /manager/gdb/\_base.gdb
-user sysdba -pass masterkey
```

E tentar novamente o backup do banco de dados. Se o backup falhar novamente, podemos especificar as opções `-G` e `-L` no comando do `gbak`.

Caso mesmo assim ainda não seja possível realizar o backup do banco de dados. teremos que realizar procedimentos para remover a parte do banco de dados danificada - o que pode levar a alguma perda de dados.

Em geral isto irá ocorrer principalmente quando houve alguma pane do disco rígido que impede o acesso a determinadas áres do banco de dados. Porém também pode acontecer (mas com uma probabilidade praticamente nula) de algum problema interno no Firebird gerar um dano grave à estrutura do banco.

Quando não for possível realizar o backup do banco de dados, deve-se primeiro verificar em qual tabela o processo está parando. O procedimento pode parar tanto retornando um erro quanto simplesmente não avançar (não retornando nenhum erro).

Neste caso, deve-se analisar o log do sistema operacional (`/var/log/messages` para detectar se não há algum erro de disco ocorrendo. Se este não for o caso, recomenda-se reiniciar o servidor e tentar novamente o procedimento de backup.

Uma vez determinada a tabela onde o backup está parando, deve-se analisar se a tabela tabela não é crítica ao sistema (por exemplo , a tabela de `CL_PREFERENCIAS`), podendo ser refeita. Se assim o for podemos primeiramente tentar eliminar os dados da tabela:

```
DELETE FROM nomeDaTabela
```

Tenta-se então o backup do banco de dados novamente. Caso ainda assim não seja possível fazer o backup, devemos tentar remover a tabela do banco de dados:

```
DROP TABLE nomeDaTabela
```

Novamente faz-se o backup, que não deverá mais parar devido a tabela danificada.

O problema surge quando a tabela com problemas é uma tabela importante, com dados do cliente. Neste caso vamos precisar elaborar um sistema que faça a cópia dos dados que podem ser salvos da tabela para recuperá-los no banco de dados restaurado porém sem estes dados.

Aqui há a necessidade de programação. Porém nunca chegou-se a enfrentar este tipo de caso exceto em servidores onde houve dano no disco rígido. Neste caso precisa-se apenas levantar os passos feitos e a tabela com problema para que seja possível desenvolver o programa.

Para acelerar o processo, pode-se já levantar a faixa de dados da tabela com problemas através do comando:

```
SELECT FIRST(xx) FROM nomeDaTabela
```

onde xx é o número de registros que deseja-se buscar do ínicio da tabela. Deve-se ir aumentando os poucos os valores até ocorrer um erro de acesso aos dados. Neste momento sabe-se onde começa a faixa de registros da tabela com problema.

Se mesmo removendo a tabela (ou se não for possível remover a tabela) ainda não for possível fazer o backup do banco de dados, deve-se restaurar um backup anterior ou gerar um novo banco de dados limpo e transportar todos os dados possíveis do banco antigo para este novo banco. Porém aqui novamente têm-se a necessidade de elaborar um software que faça isto.

8.5.4 Erros durante o restore

No procedimento de restore, existem quatro tipos possíveis de erro: erros de chave primária, de chave estrangeira, de valores de campos e erros de sistema/hardware.

Em caso de erros de sistema/hardware, o restore pode falhar devido a máquina apresentar problemas. Para verificar se é esta a questão, basta acompanhar o arquivo de log do servidor (`/var/log/messages`) e o arquivo de log do Firebird (`/opt/firebird/firebird.log`) em busca de mensagens de erro que indiquem um problema desse gênero.

Por exemplo, a mensagem:

```
GBAK error : database might be in use
```

E mensagens similiar a:

```
gbak: cannot commit index RDB$FOREIGN18
gbak: ERROR: unsuccessful metadata update
gbak: ERROR:     object BUYER is in use
gbak: ERROR: action cancelled by trigger (3) to preserve data integrity
gbak: ERROR:     Cannot deactivate primary index
gbak: Exiting before completion due to errors
```

Indica que algum usuário conectou no banco de dados durante o restore. Neste caso o restore deve ser feito novamente.

O erro:

```
 Done with volume #1, "c:\TIM\Traxback.gbk"
         Press return to reopen that file, or type a new
         name followed by return to open a different file.
```

Indica que o backup não foi concluído e o GBK/FBK gerado está incompleto.

Para os erros de chaves primárias e chaves estrangeiras, têm-se dois caminhos para a correção deles. No primeiro, corrige-se os problemas na base original e faz-se novamente o backup/restore do banco de dados. No segundo caminho, restaura-se o banco de dados sem os índices (o que permite que o banco tenha chaves primárias duplicadas e chaves estrangeiras inválidas) e corrige-se os dados no banco novo, reativando os índices posteriormente.

A segunda alternativa em geral é mais rápida e simples. Para restaurar um banco de dados sem os índices basta especificar a opção `-I` no `gbak`.

Restore - Chaves primárias duplicadas

Para a correção de chaves primárias duplicadas deve-se remover os registros com a chave duplicada, deixando apenas um registro com a chave que apresentou duplicidade. O erro apresentado será similar a:

```
gbak:cannot commit index RDB$PRIMARY131
gbak: ERROR:attempt to store duplicate value
(visible to active transactions) in
 unique index "RDB$PRIMARY131"
gbak: ERROR:action cancelled by trigger (2) to
preserve data integrity
gbak: ERROR:    Cannot deactivate index used by
an integrity constraint
gbak:Exiting before completion due to errors
```

Primeiro devemos descobrir de qual tabela é esta chave primária. Para isto usamos o seguinte SQL:

```
SELECT
RDB$RELATION_NAME
FROM
RDB$RELATION_CONSTRAINTS
WHERE
RDB$CONSTRAINT_TYPE="PRIMARY KEY"
AND
RDB$INDEX_NAME="RDB$PRIMARY131"
```

Qual das duplicatas excluir vai de acordo com a análise dos dados, buscando eliminar o registro menos completo. Um ponto a salientar é que se as duas duplicatas forem exatamente iguais, não será possível remover apenas uma. Será necessário remover as duas e recriar uma delas.

Para detectar os registros com chave duplicada em uma tabela, deve-se executar o comando:

```
SELECT HANDLE FROM nomeDaTabela GROUP BY HANDLE
HAVING COUNT(HANDLE) > 1
```

Uma vez detectados os registros com problemas, deve-se recuperar os dados deles para decidir qual será mantido e qual será excluído:

```
SELECT * FROM nomeDaTabela WHERE HANDLE=numeroDoHandle
```

Para fazer a correção da chave primária, pode-se remover tanto na base original quanto na base restaurada sem os índices.

Para remover da base original, basta apagar as duplicatas e executar o backup/restore novamente. Se não for possível fazer isto devido a alguma falha no banco (o sistema retornar um erro de estrutura - gds - ao tentar remover as duplicatas) a solução será remover da base restaurada sem os índices.

Se não for possível remover devido a alguma chave estrangeira que depende do registro, será preciso substituir nas chaves estrangeiras que dependem do registro o valor das mesmas por outro valor que não esteja com problemas. Para isto, precisamos recuperar as chaves estrangeiras que dependem do registro com problema:

```
SELECT
    A.RDB$CONSTRAINT_NAME CHAVEESTRANGEIRA,
    A.RDB$RELATION_NAME TABELAQUEDEPENDE,
    D.RDB$FIELD_NAME CAMPOQUEDEPENDE
FROM
    RDB$RELATION_CONSTRAINTS A
    INNER JOIN RDB$REF_CONSTRAINTS B ON
     (B.RDB$CONSTRAINT_NAME=a.RDB$CONSTRAINT_NAME)
    INNER JOIN RDB$RELATION_CONSTRAINTS C ON
     (B.RDB$CONST_NAME_UQ=C.RDB$CONSTRAINT_NAME AND
     C.RDB$RELATION_NAME='CL_USUARIO' AND
     C.RDB$CONSTRAINT_TYPE='PRIMARY KEY')
    INNER JOIN RDB$INDEX_SEGMENTS D ON
     (D.RDB$INDEX_NAME=A.RDB$INDEX_NAME)
WHERE
  A.RDB$CONSTRAINT_TYPE='FOREIGN KEY'
```

Com isto conseguimos descobrir que campos de quais tabelas dependem da tabela onde temos o registro com problemas. Precisamos então substituir os valores nos campos destas tabelas pelo valor de outro registro da tabela com problemas, de forma a podermos eliminar o registro com problema. Para isto, podemos montar os SQLs que trocam o valor dos campos com o seguinte SQL:

```
SELECT
    'UPDATE ' || A.RDB$RELATION_NAME || ' SET ' ||
    D.RDB$FIELD_NAME || ' = novoValor WHERE ' ||
    D.RDB$FIELD_NAME || ' = valorComProblema' SQL
FROM
    RDB$RELATION_CONSTRAINTS A
    INNER JOIN RDB$REF_CONSTRAINTS B ON
     (B.RDB$CONSTRAINT_NAME=a.RDB$CONSTRAINT_NAME)
    INNER JOIN RDB$RELATION_CONSTRAINTS C ON
     (B.RDB$CONST_NAME_UQ=C.RDB$CONSTRAINT_NAME AND
     C.RDB$RELATION_NAME='CL_USUARIO' AND
     C.RDB$CONSTRAINT_TYPE='PRIMARY KEY')
    INNER JOIN RDB$INDEX_SEGMENTS D ON (D.RDB$INDEX_NAME=A.RDB$INDEX_NAME)
WHERE
  A.RDB$CONSTRAINT_TYPE='FOREIGN KEY'
```

Este SQLs retorna um comando SQL por linha. Para poder executar este conjunto de registros SQL que ele gera, vamos usar o utilitário `ISQL`. Depois de abrir o `ISQL`, executamos

o comando:

```
OUTPUT nomeDoArquivo;
SELECT
    'UPDATE ' || A.RDB$RELATION_NAME ||
    ' SET ' || D.RDB$FIELD_NAME ||
    ' = novoValor WHERE ' || D.RDB$FIELD_NAME
    || ' = valorComProblema;' SQL
FROM
    RDB$RELATION_CONSTRAINTS A
    INNER JOIN RDB$REF_CONSTRAINTS B ON
     (B.RDB$CONSTRAINT_NAME=a.RDB$CONSTRAINT_NAME)
    INNER JOIN RDB$RELATION_CONSTRAINTS C ON
     (B.RDB$CONST_NAME_UQ=C.RDB$CONSTRAINT_NAME AND
     C.RDB$RELATION_NAME='CL_USUARIO' AND
     C.RDB$CONSTRAINT_TYPE='PRIMARY KEY')
    INNER JOIN RDB$INDEX_SEGMENTS D ON (D.RDB$INDEX_NAME=A.RDB$INDEX_NAME)
WHERE
  A.RDB$CONSTRAINT_TYPE='FOREIGN KEY;
OUTPUT;
```

Os comandos `OUTPUT` ligam e desligam a saída da consulta para um arquivo em disco. Com este arquivo criado, editamos o arquivo e removemos todas as linhas que não são comandos SQL. Salvamos o arquivo e voltamos ao `ISQL`. Dentro do ISQL, executamos o comando

```
INPUT nomeDoArquivo;
```

Este comando faz com que o `ISQL` execute todos os comandos executados no arquivo.

Vamos exemplificar este caso. Por exemplo, imaginando que o registro da tabela de usuário de handle 8008 está duplicado e é preciso remover este registro, podemos inserir um novo registro de usuário de handle `9999` e atualizar todos os campos no banco de estavam ligado ao registro `8008` para este novo registro `9999`. Assim podemos remover o registro `8888`, fazer o backup/restore do banco e depois reinserir o registro `8008` e voltar os campos que dependiam dele para ele, ao invés de deixá-los no registro `9999`. Os seguintes SQLs para este exemplo seriam utilizados.

Supondo que os comandos SQLs indicados serão todos executados dentro do ISQL, inserimos um usuário fantasma para quem vamos associar os registros que dependem do usuário com problema:

```
INSERT INTO CL_USUARIO(HANDLE,NOME,APELIDO)
VALUES (9999,'CORRECAO','CORRECAO');
```

Em seguida geramos os SQLs para alterar os registros que dependem do usuário com problema:

```
OUTPUT comandos.sql;

SELECT
    'UPDATE ' || A.RDB$RELATION_NAME ||
    ' SET ' || D.RDB$FIELD_NAME ||
    ' = 9999 WHERE ' || D.RDB$FIELD_NAME ||
```

```
    ' = 8008;'  SQL
FROM
    RDB$RELATION_CONSTRAINTS A
    INNER JOIN RDB$REF_CONSTRAINTS B ON
     (B.RDB$CONSTRAINT_NAME=a.RDB$CONSTRAINT_NAME)
    INNER JOIN RDB$RELATION_CONSTRAINTS C ON
     (B.RDB$CONST_NAME_UQ=C.RDB$CONSTRAINT_NAME AND
     C.RDB$RELATION_NAME='CL_USUARIO' AND
     C.RDB$CONSTRAINT_TYPE='PRIMARY KEY')
    INNER JOIN RDB$INDEX_SEGMENTS D ON (D.RDB$INDEX_NAME=A.RDB$INDEX_NAME)
WHERE
  A.RDB$CONSTRAINT_TYPE='FOREIGN KEY';

OUTPUT;

QUIT;
```

Editamos então o arquivo `comandos.sql`, que incialmente tem um conteúdo similar a:

```
SQL
====================================
UPDATE CL_GRUPOS SET USUARIO = 9999
WHERE USUARIO = 8008;
UPDATE CL_RECURSOINDISPONIBILIDADES SET USUARIOALTEROU = 9999
WHERE USUARIOALTEROU = 8008;
UPDATE CL_ATENDIMENTOSITUACOES SET USUARIO = 9999
WHERE USUARIO = 8008;

SQL
====================================
UPDATE CL_PAALTERACOES SET USUARIO = 9999
WHERE USUARIO = 8008;
UPDATE CL_ATENDIMENTOMATERIAIS SET USUARIOALTEROU = 9999
WHERE USUARIOALTEROU = 8008;
UPDATE CL_NOTAFISCALPRODUTOS SET USUARIOALTEROU = 9999
WHERE USUARIOALTEROU = 8008;
```

E no final deve ficar com um conteúdo similar a:

```
UPDATE CL_GRUPOS SET USUARIO = 9999
WHERE USUARIO = 8008;
UPDATE CL_RECURSOINDISPONIBILIDADES SET
USUARIOALTEROU = 9999 WHERE USUARIOALTEROU = 8008;
UPDATE CL_ATENDIMENTOSITUACOES SET
USUARIO = 9999 WHERE USUARIO = 8008;
UPDATE CL_PAALTERACOES SET USUARIO = 9999
WHERE USUARIO = 8008;
UPDATE CL_ATENDIMENTOMATERIAIS
SET USUARIOALTEROU = 9999 WHERE USUARIOALTEROU = 8008;
UPDATE CL_NOTAFISCALPRODUTOS
SET USUARIOALTEROU = 9999 WHERE USUARIOALTEROU = 8008;
```

Salvamos o arquivo e entramos novamente no ISQL. Executamos então o arquivo:

```
INPUT comandos.sql;

COMMIT;
```

Agora podemos remover o registro com problema:

```
DELETE FROM CL_USUARIO WHERE HANDLE=8008;

COMMIT;
```

Após o backup e restore, podemos voltar os registro que dependem deste usuário para o usuário correto, só trocando os registros que dependem do usuário `9999` para o usuário `8008`, seguindo a lógica acima. Não esquecendo que o usuário `8008` deve ser incluído novamente no banco.

Caso não seja possível fazer isto na base original, pode-se restaurar a base sem os índices e corrigir na base sem estes. Em seguida ligam-se os índices e o problema está resolvido. O procedimento é o mesmo apresentado acima.

Restore - Problemas com chaves estrangeiras

No outro problema que temos, relativo a chaves estrangeiras, podemos ter chaves apontando para registros que não existem mais ou que estão inacessíveis devido aos danos no banco de dados. Neste caso o erro será similar a este:

```
gbak: cannot commit index RDB$FOREIGN48
gbak: ERROR: violation of FOREIGN KEY
constraint "PK_INVCARDS"
on table "INVCARDS"
gbak: ERROR: action cancelled by
trigger (3) to preserve data integrity
gbak: ERROR:     Cannot deactivate primary index
gbak: Exiting before completion due to errors
```

Primeiro temos que obter o campo e a tabela da chave estrangeira com problema:

```
SELECT A.RDB$RELATION_NAME,
B.RDB$FIELD_NAME
FROM RDB$INDICES A
INNER JOIN RDB$INDEX_SEGMENTS B
ON (B.RDB$INDEX_NAME=A.RDB$INDEX_NAME)
WHERE
A.RDB$INDEX_NAME="RDB$FOREIGN48";
```

Neste segundo caso, não teremos problema no restore pois eles passaram a estar acessíveis. Já no primeiro caso, temos que corrigir esta situação. Novamente temos duas alternativas: corrigir os registros com problema na base original ou corrigí-los na base restaurada sem os índices.

Para corrigir os registros na base original, basta associar um novo valor da chave estrangeira aos registros com problemas. Sugere-se criar um registro fantasma, anotar o valor da chave estrangeira com problema e substituir este valor pelo valor do registro fantasma.

Após o backup/restore do banco, volta-se os registros que tinham a chave estrangeira com problema para o valor correto (incluino novamente, se preciso, o registro que não existe mais) e removendo o registro fantasma. Este é o mesmo procedimento aplicado acima para tratar chaves primárias duplicadas. O único passo adicional que precisamos é determinar quais valores de chaves estrangeiras estão referenciando valores inválidos.

Para isto podemos executar o seguinte comando para montar os SQLs que nos darão quais valores de chaves estrangeiras inválidos existem para uma determinada tabela:

```
SELECT
    'SELECT ' || D.RDB$FIELD_NAME || ' FROM ' || A.RDB$RELATION_NAME ||
    ' WHERE ' || D.RDB$FIELD_NAME || ' NOT IN (SELECT HANDLE ' ||
    ' FROM CL_USUARIO;'  SQL
FROM
    RDB$RELATION_CONSTRAINTS A
    INNER JOIN RDB$REF_CONSTRAINTS B ON
     (B.RDB$CONSTRAINT_NAME=a.RDB$CONSTRAINT_NAME)
    INNER JOIN RDB$RELATION_CONSTRAINTS C ON
     (B.RDB$CONST_NAME_UQ=C.RDB$CONSTRAINT_NAME AND
     C.RDB$RELATION_NAME='CL_USUARIO' AND
     C.RDB$CONSTRAINT_TYPE='PRIMARY KEY')
    INNER JOIN RDB$INDEX_SEGMENTS D ON (D.RDB$INDEX_NAME=A.RDB$INDEX_NAME)
WHERE
  A.RDB$CONSTRAINT_TYPE='FOREIGN KEY'
```

Com estes SQLs podemos verificar para quais registros devemos corrigir os valores das chaves estrangeiras.

Restore - Problemas com valores de campos

O último erro que podemos enfrentar é relativo a valores inválidos de campos. Neste caso, temos que corrigir estes valores na base original antes do backup do banco (não é possível restaurar o banco de dados com valores inválidos para os campos).

Neste caso, caso não seja possível alterar o valor dos registros com problemas, temos que eliminar o registro que contém o valor inválido, ou mesmo a tabela inteira (seguindo a lógica explicada anteriormente para fazer o backup de bancos de dados onde o backup não consegue passar de uma determinada tabela).

Por exemplo, a mensagem:

```
gbak: restoring data for table SIG
gbak: ERROR: validation error for column TAG, value "*** null ***"
gbak: ERROR:  warning -- record could not be restored
gbak: Exiting before completion due to errors
```

Indica que um campo possui um valor nulo (`NULL`) quando não pode possuir. Neste caso, temos que procurar os campos TAG na tabela SIG que possuem NULL como valor. Neste caso o SQL seria:

```
SELECT * FROM SIG WHERE TAG IS NULL;
```

E informar um valor para este campo nos registros achados;

```
UPDATE SIG SET TAG=xxxx WHERE condicao
```

Já a mensagem:

```
gbak: ERROR: string truncated
gbak: Exiting before completion due to errors
```

Indica que um campo um campo da tabela possui um valor maior que o permitido. Neste caso é necessário verificar o tamanho dos dados na base original e encontrar o registro com o campo com valor maior que o permitido. Para saber o tamanho permitido em um campo, basta usar o comando `show table nomeDaTabela` que irá mostrar o tamanho dos campos ao lado do nome deles.

O erro:

```
gbak: ERROR: arithmetic exception, numeric overflow, or string
truncation
gbak: ERROR:     Cannot transliterate character between character sets
```

Indica um campo com valor inválido ou muito grande. Como o item anterior, é necessário analisar os dados da tabela no banco de dados original.

Restore - Recriando os índices inativos

Quando restaura-se um banco de dados sem os índices para corrigir algum problema mais grave com o banco de dados, é necessário reativar os índices antes dos usuários utilizarem o banco de dados. Para tal devemos executar o seguinte SQL:

```
SELECT '/* 1 */','ALTER INDEX ' || RDB$INDEX_NAME || ' ACTIVE; ' SQL
FROM RDB$INDICES WHERE RDB$INDEX_INACTIVE=1
AND RDB$INDEX_NAME IN (SELECT RDB$INDEX_NAME FROM
    RDB$RELATION_CONSTRAINTS WHERE RDB$CONSTRAINT_TYPE='UNIQUE')

UNION

SELECT '/* 2 */','ALTER INDEX ' || RDB$INDEX_NAME || ' ACTIVE; ' SQL
FROM RDB$INDICES WHERE RDB$INDEX_INACTIVE=1
AND RDB$INDEX_NAME IN (SELECT RDB$INDEX_NAME FROM
RDB$RELATION_CONSTRAINTS WHERE RDB$CONSTRAINT_TYPE='NOT NULL')

UNION

SELECT '/* 3 */','ALTER INDEX ' || RDB$INDEX_NAME || ' ACTIVE; ' SQL
FROM RDB$INDICES WHERE RDB$INDEX_INACTIVE=1
AND RDB$INDEX_NAME IN (SELECT RDB$INDEX_NAME FROM RDB$RELATION_CONSTRAINTS
WHERE RDB$CONSTRAINT_TYPE='PRIMARY KEY')

UNION

SELECT '/* 4 */','ALTER INDEX ' || RDB$INDEX_NAME || ' ACTIVE; ' SQL
FROM RDB$INDICES WHERE RDB$INDEX_INACTIVE=1
AND RDB$INDEX_NAME NOT IN (SELECT RDB$INDEX_NAME FROM RDB$RELATION_CONSTRAINTS
    WHERE RDB$CONSTRAINT_TYPE='PRIMARY KEY'
        OR RDB$CONSTRAINT_TYPE='NOT NULL'  OR RDB$CONSTRAINT_TYPE='CHECK'
        OR RDB$CONSTRAINT_TYPE='UNIQUE')
```

```
ORDER BY 1
```

Capítulo 9

Conclusão

Buscou-se aqui dar uma visão geral sobre o banco de dados Firebird, apresentando suas características e funcionalidades. Mostrou-se como está estrutura internamente o banco de dados e como recuperar informações sobre o mesmo. Fez-se também uma explanação sobre as duas principais ferramentas utilizadas: gfix e gbak. Mostrou-se também como identificar e corrigir problemas com os bancos de dados. Buscou-se uma explicação de tal forma que possa ser seguida como um roteiro.

Capítulo 10

Referências

http://www.firebase.com.br/
http://firebird.sf.net/
Documentação do Firebird