Escondendo o código da metadata do banco de dados Interbase ou Firebird

Receita de bolo para proteger meta-dados do seu banco de dados Firebird dos olheiros de plantão.

Este artigo tem particular importância à aqueles que desenvolvem softwares para terceiros, pois permitirá à estes que possam distribuir livremente seus softwares baseados em banco de dados Firebird sem preocupar-se com aquelas pessoas que aproveitam código de terceiros e promovem seus sistemas baseando-se nos códigos obtidos, isto é o que chamamos de pirataria indústrial.

Entendendo o Problema

O administrador de um servidor de banco de dados FireBird, o tal "**sysdba"** tem privilégios especiais para acessar todos os objetos deste servidor, inclusive obter o código fonte de cada um deles, esse código fonte é chamado de script DML ou **Metadados**.

O típico administrador de sistemas, recorre ao script meta-dados para muitas utilidades : alterar regras estabelecidas, criar novos objetos a partir de outro objeto, ver furos de segurança dentre muitas outras cosas.

Esse é um ótimo recurso do Firebird, porém cria um problema para aqueles que desenvolvem programas para terceiros. Uma empresa pode investir muito tempo e esforço de desenvolvimento usando o Firebird e quando for distribui-lo à seus clientes, estará distribuindo também o codigo fonte de seu banco de dados, então todo o "know-how" tecnológico por trás de sua base de dados podem ser recuperados por quem tiver acesso à ele.

Imagine a situação de seu banco de dados cair na mão de seu concorrente, já pensou ?
Seu concorrente absorveria todo o seu "know-how", descobriria todos os segredos de sua base de dados, a saber :

- **as triggers** que harmoniozamente disparam procedimentos automaticamente, como por exemplo requisições de compra de material, cancelamento de débitos, etc... ;
- **as views** que juntam tabelas diferentes produzindo uma visão única de um ou vários processos ;
- **as stored procedures** que guardam procedimentos para serem executados no lado do servidor, aumentando assustadoramente a velocidade em operações como calcular folha de pagamento, controlar níveis de estoque, etc...

Diante dessa situação, surge a seguinte pergunta : **Como proteger o código fonte (Metadados) dos objetos criados no Firebird ?**
Antes de partirmos para a resposta, entendamos primeiro a forma em que os objetos do Firebird são armazenados.

Os objetos do Firebird observados de um outro ângulo

Quando voce cria um objeto no Firebird através de comandos SQL, o seu objeto é registrado em tabelas de sistema. Tabelas de Sistema são tabelas que ficam escondidas do usuário normal e só podem ser acessadas pelo administrador (SYSDBA).

Essas tabelas possuem a tarefa de registrar todas as informações lógicas sobre o banco de dados e o relacionamento de cada objeto criado com os demais objetos. Analisada as informações nessas tabelas é então formado o nexo entre todas as atividades do sistema, o grau de parentesco dos dados, relacionamento de chaves, tipagem, permissões, etc....

Tome por exemplo o seguinte código de ***stored procedure*** :

```
CREATE PROCEDURE "SP_VERSION"
RETURNS( "VERSION" VARCHAR(30) )
AS
BEGIN
  VERSION='1.0BR';
  SUSPEND;
END
```

As tabelas de sistema irão registrar esse objeto **"stored procedure"** na tabela **RDB$PROCEDURES** com informações relevantes tal como : nome, quantidade de parametros de entrada e saida, proprietário, permissões, dependências com outros objetos, etc...
Além da tabela de sistema **RDB$PROCEDURES** que guardam as "stored procedures", temos também **RDB$TRIGGERS** guardando as "triggers" e **RDB$RELATIONS** guardando as "views".

Essas 3 tabelas em sí guardam o ovo de ouro dos desenvolvedores, pois aquelas **VIEWS** com JOINS enormes, aquelas regras tão bem estruturadas com **TRIGGERS**, ou todos os procedimentos internos tão bem entesourados nas **PROCEDURES** são quase que todo o projeto do desenvolvedor, o resto foi apenas a criação de aplicativos com telas para acrescentar, acessar e processar tais objetos à partir da mesa do usuário.

Enfim, como proteger o Metadados dos objetos criados no Firebird ?

Acontece que as 3 tabelas supracitadas : **RDB$PROCEDURES**, **RDB$TRIGGERS**e **RDB$RELATIONS** também guardam dentro de sí script's DML's que geraram cada objeto dentro do banco de dados. Essas tabelas possuem dois campos que nos interessam : Um campo do tipo **BLOB subtype 1**, isto é, **um MEMO onde fica o script DML cujo conteúdo é puro texto**. E outro campo, desta vez no formato **binário (Blob subtype 2)** também conhecido como **BLR (Binary Language Representation) que contém o mesmo objeto só que pré-compilado**.

Todo nosso problema é com o campo memo-textual onde fica guardado o script DML, pois este quando lido descobre o código fonte de nosso objeto. É lendo este campo que aplicativos como IBConsole, IBExpert,... conseguem fazer uma engenharia reversa nos objetos e obter o codigo fonte. O campo contendo objetos no formato **BLR** não devem ser mexidos, pois isso faria com que o codigo pré-compilado não

correspondesse ao respectivo script DML, o que acabaria sendo um desastre pois apenas o código pré-compilado **BLR** é executado no Firebird.

Logo, se pararmos para pensar um pouco.... e se eu substitui-se esse campo do tipo MEMO com um texto qualquer e não sem mecher no campo **BLR** ? EUREKA! Os programas que recuperam scripts DML recuperariam apenas o texto que eu desejasse e como é o código **BLR** que é sempre executado então continuaria com a mesma funcionalidade. Como já diria o Chaves : isso! isso! isso!... Assim voce pode dar adeus a espionagem indústrial.

Os exemplos que irei passar a seguir, irão substituir o meta-dados de **stored procedures**,**views** e **triggers** por **NULL** (nulos), isto é, sem nenhum código fonte. Porém nada impede que voce possa substituir a palavra **NULL** por algo mais elegante como **"Protegido por lei de copyright"**.
O exemplo abaixo também funciona para o Interbase, porém se voce for substituir a palavra NULL por outro texto, terá de tomar o seguinte cuidado : o Firebird aceita um simples update de um campo Memo (blob subtype 1) por uma string sem nenhuma restrição, no entanto, para fazer isso no Interbase seria preciso uma UDF para converter string para blob-memo (algo do tipo StringToBlob). Então se voce usa o Interbase é melhor usar a palavra **NULL.**
Aviso importante para usuários que usam versões anteriores ao IB6 (4.x,5.x) : Se voce substituir o script DML por **NULL**, o Interbase irá disparar uma trigger automática que apaga o conteúdo **BLR** de qualquer forma, então se esse for seu caso, terá obrigatóriamente que utilizar uma UDF para converter uma string num 'blob'.

Mãos à obra :

Aviso aos despreucupados : Como está escrito na Bíblia dos programadores : "Felizes os pessímistas, pois estes fazem backup".
Ao executar os procedimentos logo adiante atente-se para o fato de que voce perderá todos os scripts DML de sua base de dados (**stored procedures**, **triggers** e **views**), então não esqueça da palavra mágica que pode salvar seu emprego : **backup** !.

- ***Escondendo o script DML de stored procedures (procedimentos armazenados)***

Como já foi dito antes, as stored procedures ficam armazenadas na tabela **RDB$PROCEDURES**, essa tabela possui um registro para cada **stored procedure** existente e contém informações como números de parametros de entrada e saida, proprietário da tabela, codigo fonte, codigo binário pré-compilado,etc...
O campo que nesta tabela nos interessa é o campo **RDB$PROCEDURE_SOURCE** que contém o codigo fonte DML da **stored procedure**. Não toque e não mexa no campo **RDB$PROCEDURE_BLR** que contém o código pré-compilado, senão a funcionalidade da **stored procedure** vai pro espaço, isto é, deixaria de funcionarar.
Então vamos ao comando SQL :

```
UPDATE rdb$procedures
SET rdb$procedure_source = NULL,
    rdb$description ='Protegido por lei de copyright'
WHERE ((rdb$system_flag = 0) OR (rdb$system_flag IS NULL))
```

Segundo consta nos manuais do Interbase (o qual o Firebird é derivado), a coluna rdb$system_flag igual a 0 (zero) indica ao Firebird que o objeto, isto é, a **stored procedure** foi criada pelo usuário com comandos normais usando SQL, isto é,

determina a existência de código fonte. Imagino que todos os rdb$system_flag seriam igual a zero, a menos que a uma stored procedure tenha sido criada apenas para uso interno do próprio Firebird.

- ***Escondendo o script DML de triggers (gatilhos)***

Semelhante a stored procedures, as **triggers** também possuem sua própria tabela de sistema e esta se chama **RDB$TRIGGERS**, e o campo que guarda o codigo fonte e binário são respectivamente **RDB$TRIGGER_SOURCE** e **RDB$TRIGGER_BLR**. O campo que nos interessa é apenas o **RDB$TRIGGER_SOURCE** :

```
UPDATE rdb$triggers a
SET rdb$trigger_source = NULL,
    rdb$description ='Protegido por lei de copyright'
WHERE ((rdb$system_flag = 0)
   OR (rdb$system_flag IS NULL))
  AND NOT EXISTS(
    SELECT rdb$trigger_name FROM rdb$check_constraints
    WHERE rdb$trigger_name=a.rdb$trigger_name)
```

Novamente, usaremos o campo rdb$system_flag para apenas mexermos nas triggers que nós mesmos criamos, imagino que deva haver algumas que pertencem ao próprio Firebird que não devem ser tocadas.
A parte grifada em vermelho é para não mexer naquelas triggers que são auto-geradas pelo Firebird quando adicionamos uma restrição a um campo, o tal **CHECK CONSTRAINT**.
Esses "CONSTRAINT" vão gerar triggers internas, eu pessoalmente não apagaria o código fonte destes, mas se voce desejar faze-lo basta remover a parte em vermelho.

- ***Escondendo o script DML de views (visões)***

As **views** também possuem sua própria tabela de sistema e ela se chama **RDB$RELATIONS**, e como os seus companheiros de banco de dados : stored procedures e triggers, também possuem o campo **RDB$VIEW_SOURCE** para armazenar o codigo fonte, e o campo **RDB$VIEW_BLR** para armazenar o codigo pré-compilado do objeto. O processo para se eliminar o codigo fonte das **views** é praticamente o mesmo que fizemos com os outros objetos :

```
UPDATE rdb$relations
SET rdb$view_source = NULL,
    rdb$description ='Protegido por lei de copyright'
WHERE ((rdb$system_flag = 0) OR (rdb$system_flag IS NULL))
```

- ***Funciona ?***

Quando for executar os procedimentos descritos acima, irá notar que ainda será possivel recuperar os cabeçalhos dos scripts DML, no entanto, tente analisar o codigo até o final e perceberá que o conteúdo do script não estará lá. Por exemplo, o cabeçalho de uma stored procedure sempre estará disponível com os seus parametros de entrada/saida, porém o conteúdo após as declarações de entrada/saída não irá aparecer.
Talvez voce pergunte : "Mas seria possivel recupera-lo após os updates acima ?"

Resposta : definitivamente não ! O código fonte dos objetos acima foram definitivamente trocado por **nulos**, não existe donde recuperar, a menos que algum dia alguém invente um descompilador do codigo **BLR** que mantém sua funcionalidade. Mas isso é improvável...pelo menos por enquanto.

Cuidados com cópias de segurança (Backup e Restore)

Lembre-se de que quando seus clientes forem fazer uma cópia de segurança de suas bases de dados que eles não estarão fazendo cópias de segurança das **triggers, views e stored procedures**, apenas os dados em sí estarão sendo copiados, isto é: tabelas, indices, constraints, domains, relacionamentos e todos os outros objetos com excessão dos objetos já mencionados : **triggers, views e stored procedures**.

Suponhamos que um cliente ligue para você e diga que vai precisar restaurar uma cópia de segurança (backup), o que você fará então ?
Faça a restauração normal dos dados, apenas aperceba-se de que **trigers, views e stored procedures** não existem mais porque obviamente voce eliminou o código fonte das mesmas. O seu trabalho será re-criar novamente tais objetos após a restauração dos dados, isto poderá ser feito através de um aplicativo permite acessar remotamente a máquina do usuário como por exemplo o VNC (http://www.uk.research.att.com/vnc/index.html), ou então através do próprio IBConsole conectando à base de dados sobre a internet.

Para facilitar ainda mais sua vida é bom voce preparar um programa que crie ou atualize tais objetos e em seguida remova os código fonte de **triggers,views e stored procedures** novamente sem precisar de nenhuma intervenção humana.

Nota do Autor :

Este pequeno documento dá uma visão interessante sobre os objetos do Firebird. Elas não são uma maneira de hackear o banco de dados, pois as informações obtidas acima constam na documentação do próprio Interbase 6 (do qual o Firebird deriva-se), porém pode acontecer que fique desatualizado com as futuras versões do FiredBird (atualmente Firebird 1.0.826), portanto tenha sempre a mão o código fonte original de seu banco de dados.

Referencias:

- **Documentacão InterBase 6 Open Source**
- **Language Reference, página 241.**

Autor : Gladiston Santana
Data : 01-Outubro-2002
Home Page : www.gladisto.hpg.com.br