Apostila de Firebird 1.0

Acesso Nativo com o InterBase Express “IBX”

Autor : **Anderson Haertel Rodrigues**
Colaboração : Marcus Boi

Título Original: Apostila de Firebird 1.0
Adaptação de Interbase para Firebird por um colaborador da CFLP

O Que Veremos :

Uma Visão Geral do Firebird.

- ✓ **O que é o Firebird ?**
- ✓ **IBConsole**
- ✓ **SQL – Structured Query Language**
- ✓ **Comandos e Funções**
- ✓ **O que é Dialect ?**
- ✓ **Segurança com os usuários**
- ✓ **Tipos de Dados do Firebird**
- ✓ **Join´s**
- ✓ **Transação**

Acesso nativo ao Firebird, através dos componentes “IBX”.

- ✓ **O que é acesso nativo.**
- ✓ **O que é o InterBase® Express “IBX”.**
- ✓ **Explicação dos 12 componentes da palheta InterBase®.**
- ✓ **Explicação dos Componentes da palheta IB ADmin**

O que é Firebird ?

O Firebird é um poderoso banco de dados Cliente/Servidor relacional que é compatível com SQL-ANSI-92, e foi desenvolvido para ser um banco de dados independente de plataformas e de sistemas operacionais.

Este banco de dados, dispensa maiores estruturas dentro da empresa, (DBA / Preparação), onde basta instalar o software e usar-lo, sem a interferência freqüente de profissionais, especializados na manutenção do banco de dados de produção.

Acompanhando, isso tudo ele ainda dispensa o uso de super-servidores, usando pouco espaço em disco para sua instalação e utilizando pouca memória em situações normais de uso. Por isso a plataforma necessária para a sua instalação e utilização pode ser reduzida diminuindo consideravelmente os custos do projeto.

Seu desenvolvimento iniciou em meados de 1985 por uma equipe de engenheiros da DEC (Digital Equipament Corporation). Tendo como nome inicial de Groton, esse produto veio sofrendo varias alterações até finalmente em 1986 receber o nome de Interbase® iniciando na versão 2.0. Nesta época, a idéia era produzir um SGBDR (Sistema Gerenciador de Bancos de Dados Relacionais) que oferecesse benefícios não encontrados em outros da época.

Ao longo do desenvolvimento, foi introduzido muitas características, dentre elas :

Acesso nativo a driver JDBC	Commit Automático de Duas Fases
Sombreamento do Banco de Dados	Replicação
Tratamento de Blob´s	Sistema de Eventos

Mas então, se o Firebird é tão bom, porque ele não é tão reconhecido como o Oracle, o Microsoft SQL server e outros servidores SQL ? Aparentemente, o maior problema enfrentado pelo Firebird durante todos os anos de sua existência foi a falta de marketing e divulgação por parte da Borland/Inprise/ISC nos meios especializados (revistas, livros, etc...). No entanto, com os últimos acontecimentos, essa imagem vai tender à mudar rapidamente, pois o Firebird é uma base de dados Open Source, construído com base no código do Interbase Open Source, sendo que as licenças de utilização e distribuição agora são totalmente FREE ! Isso mesmo, custo 0, de graça !!! Isso quer dizer que você não precisará mais utilizar as famosas (e ja mais do que ultrapassadas) base de dados padrão xBase ou Paradox para diminuir o custo do seu cliente. Você vai poder contar com um Banco de Dados poderoso, eficiente e seguro e seu cliente não vai precisar pagar nada a mais por isso !

Outra grande vantagem do Firebird é que ele é múlti plataforma ou seja funciona em vários Sistemas Operacionais, dentre eles destacamos:

Windows 9x®	Windows NT®
Linux	Solaris®

IBConsole

O que é ? e pra que serve ?

O IBConsole é o gerenciador de Dados que acompanhava o InterBase, e que pode ser utilizado com o Firebird 1. A grande vantagem dele é o fato, de não ser uma ferramenta de criação de Tabelas.
No IBConsole, você realmente aprende a linguagem SQL, pois, toda e qualquer criação, relacionamento, manutenção, é feito no ISQL, tudo via Linha de Comando. Existem outras ferramentas no mercado, como Quick Desk, IB Admin, Maratho, IBExpert´s, todas muito boas e de fácil aprendizado e, com criação automática da grande maioria dos Comandos de DML "Linguagem de manipulação de Dados".

No IBConsole o usuário máster é "SYSDBA" e a sua senha é "masterkey", o seu uso é bastante simples, inicialmente você precisa se "logar" no IBConsole para isso clique com o botão direito sobre a opção "Local Server" e escolha "login" em USERNAME você informa o usuário máster e a sua senha acima descrita. Na opção Databases você pode registrar ou criar um novo banco de dados, para fazer o registro o banco já deve existir, clique com o botão direito sobre a opção "Register", em Files você informa o nome do banco e o seu caminho se preferir pode procurar com o botão de atalho que esta localizado a sua direita, o Alias Name é o nome do Alias para esse banco, User Name / Password você deve informar o usuário e a sua senha, clicando no botão de OK para finalizar o registro.

Para criarmos um Banco de Dados em ambiente Client/Server via IBConsole deve escolher a opção "interactive SQL" no menu Tools ou clicar no ícone SQL. Com ela, podemos enviar comandos SQL para o servidor Firebird administrar nossos dados. Ao carregarmos o Windows ISQL, veremos uma tela dividida em duas partes, a parte superior aceita comandos SQL e os resultados aparecerão na parte inferior. Devemos observar que o ISQL não enviará nenhum comando SQL até logarmos com um usuário e senha correta e nos conectarmos a um banco de dados.

Ex: Vamos mostrar todos os dados da tabela Employee, esta tabela é uma tabela de exemplo que é instalada juntamente com o Firebird, não esqueça de se conectar a essa banco antes.

Select * from employee

Depois disso pressionamos o botão Execute Query ou Ctrl + Enter e teremos o resultado mostrado na parte inferior da tela.

SQL – Structured Query Language

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas várias linguagens destinadas à sua manipulação. O Departamento de Pesquisas da IBM desenvolveu a SQL como forma de interface para o sistema de Banco de Dados relacional denominado SYSTEM R, no inicio dos anos 70. Em 1986 o American National Standard Institute (ANSI), publicou um padrão SQL e ela se estabeleceu como linguagem padrão de Banco de Dados Relacional.

A SQL apresenta uma série de comandos que permitem a definição dos dados, chamada de DDL (Data Definition Language – Definição de Dados Declarados), composta entre outros pelos comandos Create, que é destinado à criação do Banco de Dados, das tabelas que o compõe, além das relações existentes entre as tabelas. Como exemplo da classe DDL temos os comandos Create, Alter, Drop e Rename.

Os comandos da série DML (Data Manipulation Language – Manipulação de Dados Declarados), destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML temos os comandos Select, Insert, UpDate, Delete, Commit e Rollback.

Uma subclasse de comandos DML, é a DCL (Data Control Language – Controle de dados Declarados), que dispõe de comandos de controle como Grant, Revoke e Lock.

A Linguagem SQL tem como grande virtude a sua capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro. Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de visualizarmos os dados, como listagens independentes das tabelas e organização lógica dados dados.

Outra característica interessante na linguagem SQL é a capacidade que dispomos de cancelar uma série de atualizações ou de as gravarmos, depois de iniciarmos uma seqüência de atualizações. Os comandos Commit e Rollback são responsáveis por estas facilidades.

Devemos notar que a linguagem SQL consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices.

Comandos e Funções:

A seguir serão listados alguns comandos e funções mais utilizadas do Firebird, com parâmetros mais comuns. Não serão abordados todos os comandos, mas o essencial para se obter um bom conhecimento e conseguir usufruir do potencial desse banco de dados.

ALTER DATABASE	CREATE EXCEPTION	DROP PROCEDURE
ALTER DOMAIN	CREATE GENERATOR	DROP TABLE
ALTER EXCEPTION	CREATE INDEX	DROP TRIGGER
ALTER INDEX	CREATE PROCEDURE	DROP VIEW
ALTER PROCEDURE	CREATE TABLE	EXECUTE PROCEDURE
ALTER TABLE	CREATE TRIGGER	GEN_ID( )
ALTER TRIGGER	CREATE VIEW	INSERT
AVG( )	DECLARE EXTERNAL FUNCTION	MAX( ) / MIN( )
CAST( )	DELETE	ROLLBACK
CLOSE	DROP DATABASE	SELECT
COMMIT	DROP DOMAIN	SET GENERATOR
COUNT( )	DROP EXCEPTION	SUM( )
CREATE DATABASE	DROP EXTERNAL FUNCTION	UPDATE
CREATE DOMAIN	DROP INDEX	UPPER( )

ALTER DATABASE

Adiciona arquivos secundários ao Banco de Dados. Isso significa que poderemos ter um banco de Dados com vários arquivos dentro do mesmo GDB. Para a alteração da Base de Dados, o usuário SYSDBA, precisa ter acesso exclusivo ao Banco de Dados Firebird. Este comando, auxilia na repartição do Banco, deixando em algumas vezes o acesso aos Dados mais rápido.

Sintaxe :

```
ALTER [DATABASE | SCHEMA ]
ADD FILE 'nome' [LENGHT = PAGES |        STARTING AT PAGE]
```

Ex :
```
ALTER DATABASE
ADD FILE 'FATURAMENTO.GD1' STARTING AT PAGE 10001 LENGHT 10000 ADD FILE
'ESTOQUE.GD1' LENGHT 10000;
```

ALTER DOMAIN

Altera a definição de um domínio que já tenha sido criado. Pode-se alterar qualquer elemento de domínio, exceto os domínio de NOT NULL e a troca do tipo de Dado. Para redefinir o tipo de domínio e ou alterar o NOT NULL, deve apagar o domínio e criá-lo novamente. Atento para que se alguma tabela estiver usando o Domínio no qual você quer alterar os itens citados acima, você precisará deletar a coluna da tabela para ter sucesso no processo de troca. Aliás, o Firebird não deixará você trocar o tipo e ou a constraint NOT NULL, enquanto encontrar referências para este domínio. A criação de domínio, requer uma certa analise, para não encontrar este tipo de referência.

Sintaxe :

```
ALTER DOMAIN name { [SET DEFAULT { literal | NULL | USER}]| [DROP DEFAULT]| [ADD
[CONSTRAINT] CHECK ( <dom_search_condition>)]
| [DROP CONSTRAINT] | new_col_name| TYPE data_type};
```

Ex :
```
CREATE DOMAIN D_MES  AS SMALLINT CHECK(VALUE BETWEEN 1 AND 12 );
ALTER DOMAIN D_MES SET DEFAULT 1;
```

ALTER EXCEPTION

Altera a mensagem associada a uma exceção.

Sintaxe :

```
ALTER EXCEPTION nome_da_excecao 'Novo Texto';
```

Ex :
```
CREATE EXCEPTION E_VALOR_INVALIDO 'Valor Inválido';
ALTER EXCEPTION E_VALOR_INVALIDO 'O Valor informado não é válido';
```

ALTER INDEX

Torna um índice ativo e ou inativo. Este comando está relacionado diretamente na performance do índice no seu Banco de Dados. Em certos momentos, o índice no Firebird pode ficar desbalanceado, desta forma, este comando recria o índice do Firebird.

Sintaxe :

```
ALTER INDEX name {ACTIVE | INACTIVE};
```

Ex :
```
CREATE INDEX INDEX_NOME_TABELAX ON CLIENTES(NOME);
ALTER INDEX INDEX_NOME_TABELAX INACTIVE;
ALTER INDEX INDEX_NOME_TABELEX ACTIVE;
```

ALTER PROCEDURE

Altera uma Stored Procedure existente no Firebird. A sintaxe é a mesma da Create Procedure, se muda apenas o CREATE por ALTER. A sintaxe ALTER PROCEDURE, só não pode ser usada para alterar o nome da Stored Procedure. Mas, todos os seus itens, Parâmetros e o Corpo da Procedure pode ser alterada.

Sintaxe :
```
ALTER PROCEDURE name[( var datatype [, var datatype …])]
[RETURNS ( var datatype [, var datatype …])]
AS
Begin
   //Linguagem da Procedure.
end;
```
Ex :
```
ALTER PROCEDURE SOMA_VENDAS_NO_MES  ( PMES SMALLINT ) RETURNS ( SOMA
NUMERIC(18,02) ) AS
BEGIN
        SELECT SUM(VALOR_VENDA) FROM VENDAS WHERE MES :PMES  INTO SOMA
END
```

ALTER TABLE

Altera a estrutura de uma tabela e ou a integridade da mesma.

Sintaxe :

```
ALTER TABLE table ADD <col_def> <col_def> = col { <datatype> | [COMPUTED [BY] (<expr>)
| domain}[DEFAULT { literal | NULL | USER}][NOT NULL] [ <col_constraint>]
[COLLATE collation]<col_constraint> = [CONSTRAINT constraint] <constraint_def>
[ <col_constraint>]
```

Ex :
```
ALTER TABLE FORNECEDORES ADD CGC CHAR(14), DROP TIPOFORNECEDOR, ADD
CONSTRAINT E_MAIL CHECK ( E_MAIL CONTAINING '@' OR E_MAIL IS NULL )
```

ALTER TRIGGER

Altera a definição de uma Trigger. Caso algum argumento for omitido, é assumido o valor definido no CREATE TRIGGER, ou no ALTER TRIGGER anteriormente usado.

Sintaxe :

```
ALTER TRIGGER name
[ACTIVE | INACTIVE] [{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
[POSITION number]
AS < trigger_body>;
```

Ex :
```
ALTER TRIGGER "BAIXA_ESTOQUE" INACTIVE;
ALTER TRIGGER "EXCLUI_ITENS" FOR "TABELAVENDAS" BEFORE DELETE AS
BEGIN
        DELETE FROM ITENS: WHERE NUM_VENDA = OLD.NUM_VENDA
END
```

AVG()

Retorna a média de valores de uma coluna.

Ex: SELECT MES, AVG(VALOR_DA_VENDA) FROM VENDAS ORDER BY MES

CAST()

Usado em colunas, onde há a necessidade de se converter tipos de dados para outro formato.

Ex : SELECT CODIGO FROM FORNECEDORES WHERE DATA >= CAST(: DATA AS DATE)

SELECT CAST(CODIGO AS CHAR(10)) || ' – ' || NOME FROM VENDEDORES.

Neste exemplo foi usado para fazer a formatação e a "soma" entre dois campos. As barras verticais, foram usadas para concatenação, isto é, no Firebird e concatenação de campos, é através das barras ||, e não com o sinal de "+" como em outros SGDB´s.

Este são os "Type Casting" aceitos pelo Firebird :

Caracter em Numérica ou Data
Numérico em Caracter ou Data
Data em Caracter ou Numérico

COMMIT

Grava as alterações de uma transação permanente no Banco de Dados.

Sintaxe :

COMMIT

Ex : COMMIT

COUNT()

Retorna a quantidade de registros para uma condição em um SELECT

Sintaxe :

COUNT(* | ALL | valor | DISTINCT valor)

Ex : SELECT COUNT(*) FROM CLIENTES

CREATE DATABASE

Cria um novo Banco de Dados ".GDB". Nele pode especificar as suas características, como :

Nome do Arquivo;
Tamanho da página de dados (PAGE SIZE);

Sintaxe :

CREATE {DATABASE | SCHEMA} ' filespec'
[USER ' username' [PASSWORD ' password']]
[PAGE_SIZE [=] int]
[LENGTH [=] int [PAGE[S]]]
[DEFAULT CHARACTER SET charset]
[<secondary_file>];
<secondary_file> =FILE 'filespec' [<fileinfo>][<secondary_file>]
<fileinfo> = LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]] int
[<fileinfo>]

Ex :

CREATE DATABASE 'C:\DB\TESTE.GDB' DEFAULT CHARACTER SET ISO8859_1 FILE 'C:\DB\TESTE.GD1' STARTING AT PAGE 10001 LENGHT 10000 PAGES

CREATE DOMAIN

Cria uma definição de um “novo tipo de dado”. Onde pode ser feito também, checagem de valor, isto é, regras para o dado ser gravado neste “novo tipo de dado”. Quando falo em novo tipo de dado, não é possível no Firebird “sem alterar os fontes do mesmo”, criar um novo tipo de dado, mas, com DOMAINS, nós usamos dados já existentes, mas, especificando o tamanho e a regra a ser seguida.

Sintaxe :

```
CREATE DOMAIN domain [AS] <datatype>
[DEFAULT { literal | NULL | USER}]
[NOT NULL] [CHECK ( <dom_search_condition>)]
[COLLATE collation];
```

Ex : CREATE DOMAIN STRINGNOME AS VARCHAR(45);

CREATE DOMAIN IDCODIGO AS INTEGER DEFAULT 1000 CHECK (VALUE > 1000) NOT NULL

CREATE TABLE “FORNECEDOR” (ID INTEGER NOT NULL PRIMARY KEY,
NOME STRINGNOME, ---- Usamos o Domínio criado acima ----);

CREATE EXCEPTION

Cria uma mensagem de erro, armazenada no servidor, e só pode ser usado em Stored Procedure e ou Trigger

Sintaxe :

CREATE EXCEPTION “NOME_DA_EXCEPTION” ‘MENSAGEM’

Ex: CREATE EXCEPTION “NOME_INVALIDO” ‘O Valor informado para o campo, é inválido’

Este trecho de código, está contigo dentro de uma Trigger.

```
IF (NEW.NOME = ‘’) THEN
  EXCEPTION NOME_INVALIDO;
```

CREATE GENERATOR

Cria um Generator de número inteiros e seqüenciais. O Generator em conjunto com Trigger e ou Stored Procedure, é usado para simular o campo auto-incremento dos Bancos Desktop´s. Serve também para evitar chaves duplicadas em campos numéricos.

O Valor inicial é Zero, mas, é atualizado toda vez que é chamado a função GEN_ID(). O Generator pode ser usado para incrementar ou decrementar valores. Para se saber o código atual do Generator você passa o valor Zero “0” para o Gen_ID().

Sintaxe :

CREATE GENERATOR “NOME_DO_GENERATOR”

Ex : Vamos simular neste exemplo, um auto numérico seqüencial da tabela de fornecedores da coluna ID.

```
CREATE GENERATOR COD_FORNECEDOR;
CREATE TRIGGER “COD_AUTO_FORNECEDOR” FOR “FORNECEDORES” BEFORE
INSERT POSITION 0 AS
BEGIN
        NEW.ID := Gen_ID(“COD_FORNECEDOR”,1);
END
```

Para excluir um GENERATOR, ele tem que ser excluído direto da tabela de sistema do Firebird “RDB$GENERATORS”.

Ex : DELETE FROM RDB$GENERATORS WHERE RDB$GENARATOR_NAME = ‘NOME_DO_SEU_GENERATOR’;

CREATE INDEX

Cria um índice para uma ou mais colunas especificas da tabela. O índice está ligado diretamente a performance do seu banco de dados. O conceito de índices em ambientes Desktop´s “xBase, Access, Paradox” é muito diferente do conceito de índices em ambiente Client/Server. Um índice em ambiente Client/Server “Firebird, Oracle, DB2”, não tem a função de organizar a tabela, pois, você tem o mesmo efeito com ORDER BY. A função de um índice em ambiente Client/Server, é de performance em primeiro lugar, caso o índice seja um PK “Primary Key”, tem a função de manter a integridade da tabela, caso o índice seja um FK “Foreign Key”, tem a função de relacionamento e integridade da tabela, caso o índice seja UNIQUE, tem a função de não deixar valores iguais serem incluídos.

As cláusulas ASCENDING e DESCENDING, tem a função de organizar da maneira desejada o índice. O valor default é ASC

Sintaxe :

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX index ON table ( col [, col ...]);
```

Ex : CREATE INDEX IND_DATA_VENDA ON “VENDAS” (DATA_VENDA);
CREATE DESC INDEX IND_SALARIOS ON “FUNCIONARIOS” (SALARIO);
CREATE UNIQUE INDEX IND_COD_PRODUTO ON “PRODUTOS” (ID);

CREATE PROCEDURE

Cria uma Stored Procedure “SP” e, define seus parâmetros de entrada e saída e o corpo da procedure. Uma SP é escrita em “linguagem” Firebird, e armazenada no próprio Banco de Dados. Stored Procedure´s aceita todas as sentenças de manipulação de Dados e algumas extensões avançadas do Firebird, como :

IF..THEN..ELSE, WHILE...DO, FOR SELECT..DO, EXCEPTIONS ...

Existem dois tipos de SP :

Select Procedure : Utilizada na cláusula FROM do comando SELECT, como se fosse uma tabela e ou uma View. Este tipo de SP, deve obrigatoriamente retornar uma ou mais linhas de dados, caso contrário, ocorre um erro. Para retornar uma linha, use SUSPEND, para retornar uma ou mais linhas, use FOR SELECT.... DO e SUSPEND. Isto fará com que os parâmetros de retorno sejam preenchidos com a linha de retorno.

Executable Procedure : Utilizada no comando EXECUTE PROCEDURE, para realizar uma ou mais tarefas, mas, não retorna dados.

Sintaxe :

```
CREATE PROCEDURE name
[( param datatype [, param datatype …])]
[RETURNS ( param datatype [, param datatype …])]
AS
<procedure_body>;
< procedure_body>=[<variable_declaration_list>]
< block>
< variable_declaration_list>=
DECLARE VARIABLE var datatype;
[DECLARE VARIABLE var datatype; …]
<block> =
BEGIN
< compound_statement>
[< compound_statement>…]
END
< compound_statement>={<block> | statement;}
```

Ex :

```
CREATE PROCEDURE RESUMO_VENDAS ( VENDEDOR INTEGER) RETURNS (
VALOR_TOTAL DOUBLE PRECISION, MEDIA DOUBLE PRECISION VALOR_MIN
DOUBLE PRECISION, VALOR_MAX DOUBLE PRECISION ) AS
BEGIN
        SELECT SUM(VALOR), AVG(VALOR), MIN(VALOR), MAX(VALOR) FROM
        VENDAS WHERE IDVENDEDOR = :VENDEDOR INTO :VALOR_TOTAL, :MEDIA,
        :VALOR_MIN, :VALOR_MAX;
        EXIT;
END
SELECT * FROM RESUMO_VENDAS 10;
```

“10, Neste caso é o código do vendedor”.

CREATE TABLE

Cria uma nova tabela no seu banco de dados Firebird.

Sintaxe :

```
CREATE TABLE table [EXTERNAL [FILE] 'filespec']
(<col_def> [, <col_def> | <tconstraint> …]);
```

<col_def> = col {<datatype> | COMPUTED [BY] (<expr>) | domain}
[DEFAULT {literal | NULL | USER}]
[NOT NULL]
[<col_constraint>]
[COLLATE collation]
<datatype> =
{SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION}[<array_dim>]
| (DATE | TIME | TIMESTAMP} [<array_dim>]
| {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(int)]
[<array_dim>] [CHARACTER SET charname]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
[VARYING] [(int)] [<array_dim>]
| BLOB [SUB_TYPE {int | subtype_name}] [SEGMENT SIZE int]
[CHARACTER SET charname]
| BLOB [(seglen [, subtype])]<array_dim> = [[x:]y [, [x:]y …]]
<expr> = A valid SQL expression that results in a single value.
<col_constraint> = [CONSTRAINT constraint]
{ UNIQUE
| PRIMARY KEY
| REFERENCES other_table [(other_col [, other_col …])]
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
| CHECK (<search_condition>)}
<tconstraint> = [CONSTRAINT constraint]
{{PRIMARY KEY | UNIQUE} (col [, col …])
| FOREIGN KEY (col [, col …]) REFERENCES other_table
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
| CHECK (<search_condition>)}
<search_condition> = <val> <operator> {<val> | (<select_one>)}
| <val> [NOT] BETWEEN <val> AND <val>
| <val> [NOT] LIKE <val> [ESCAPE <val>]
| <val> [NOT] IN (<val> [, <val> …] | <select_list>)
| <val> IS [NOT] NULL
| <val> {>= | <=}
| <val> [NOT] {= | < | >}
| {ALL | SOME | ANY} (<select_list>)
| EXISTS (<select_expr>)
| SINGULAR (<select_expr>)
| <val> [NOT] CONTAINING <val>
| <val> [NOT] STARTING [WITH] <val>
| (<search_condition>)
| NOT <search_condition>
| <search_condition> OR <search_condition>
| <search_condition> AND <search_condition>
<val> = { col [<array_dim>] | :variable
| <constant> | <expr> | <function>
| udf ([<val> [, <val> …]])
| NULL | USER | RDB$DB_KEY | ? }
[COLLATE collation]

<constant> = num | 'string' | charsetname 'string'
<function> = COUNT (* | [ALL] <val> | DISTINCT <val>)
| SUM ([ALL] <val> | DISTINCT <val>)
| AVG ([ALL] <val> | DISTINCT <val>)
| MAX ([ALL] <val> | DISTINCT <val>)
| MIN ([ALL] <val> | DISTINCT <val>)
| CAST (<val> AS <datatype>)
| UPPER (<val>)
| GEN_ID (generator, <val>)

Ex :
```
CREATE TABLE PRODUTOS (
ID          INTEGER         NOT NULL,
NOME        VARCHAR(50)     NOT NULL,
DATA        DATE            DEFAULT CURRENTE DATE NOT NULL,
PRECO       DOUBLE          PRECISION ( CHECK PRECO > 0),
ESTOQUE     INTEGER         ( CHECK ESTOQUE > 0),
VALOR       COMPUTED BY     ( PRECO * ESTOQUE ),
CONSTRAINT PK_PRODUTOS PRIMARY KEY(ID));
```

CREATE TRIGGER

Cria uma Trigger “Gatilho” para a tabela especificada. Trigger é um gatilho disparado após alguma ação ocorrida na tabela, isto é, podem existir Trigger de Insert, Update e Delete. As Trigger pode ser definidas como “Before-Antes” e “After-Depois”. Também pode ser definido um número onde indica qual a seqüência de tirareis a ser seguida. A Trigger só é disparada pela ação na tabela, não podendo ser disparada pela aplicação. Dentro da Trigger há duas formas de se referenciar as colunas das tabelas :

OLD.Coluna e NEW.Coluna. Onde OLD. referencia o valor anterior da coluna e NEW. referencia o novo valor da coluna.

Sintaxe :

CREATE TRIGGER name FOR table
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE}
[POSITION number]
AS <trigger_body> terminator
<trigger_body> = [<variable_declaration_list>] <block>
<variable_declaration_list> =
DECLARE VARIABLE variable <datatype>;
[DECLARE VARIABLE variable <datatype>; …]
<block> =
BEGIN
<compound_statement>
[<compound_statement> …]
END
<datatype> = SMALLINT
| INTEGER
| FLOAT
| DOUBLE PRECISION
| {DECIMAL | NUMERIC} [(precision [, scale])]

| {DATE | TIME | TIMESTAMP)
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}
[(int)] [CHARACTER SET charname]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(int)]
<compound_statement> = {<block> | statement;}

Ex:

```
SET TERM !! ;
CREATE TRIGGER  “TRG_VERIFICA_ESTOQUE” FOR “ITENS_NOTA” BEFORE INSERT
POSITION 0 AS
BEGIN
        IF ( NOT EXISTS ( SELECT * FROM PRODUTOS WHERE ID = NEW.ID AND
        ESTOQUE >= NEW.QTDE_VENDIDA ) ) THEN
        EXCEPTION ACABOU_ESTOQUE;
END !!
SET TERM ; !!
```

CREATE VIEW

Cria uma nova visão dos dados já existentes em uma tabela. Uma VIEW é uma tabela normal, pode ser fazer o mesmo que uma tabela normal. Um dos impedimentos referentes a VIEW é o ORDER BY.

VIEW são utilizadas para realizar tarefas como :

Restringir o acesso dos usuários;
Mostras apenas as colunas “x”;
Filtragem de dados já pré formatados;
E de certa forma, serve também como Data WareHousing;

A VIEW fica armazenada no Banco de Dados, mas, é armazenado apenas a definição da VIEW.

Existem dois tipos de VIEW´s :

VIEW Read-Only : Quando o resultado não pode ser editado.
VIEW Update : Views que podem ser editadas.

Sintaxe :

CREATE VIEW name [(view_col [, view_col …])]
AS <select> [WITH CHECK OPTION];

Ex : CREATE VIEW TELEFONES (NOME, TELEFONE) AS SELECT NOME, TELEFONE FROM CLIENTES;

DECLARE EXTERNAL FUNCTION

Declara uma função externa ao Banco de Dados Firebird. A função, mais conhecida como UDF. O principio de criação de UDF, no Windows, é de construir a função em DLL “Delphi, C, C++, VB”, e após isto, declarar a função no Firebird.

Sintaxe :

```
DECLARE EXTERNAL FUNCTION name [datatype | CSTRING (int)
[, datatype | CSTRING (int) …]]
RETURNS {datatype [BY VALUE] | CSTRING (int)} [FREE_IT]
ENTRY_POINT 'entryname'
MODULE_NAME 'modulename';
```

Ex : DECLARE EXTERNAL FUNCTION ABREVIAR_NOME CHAR(60) RETURNS CHAR(60) BY VALUE ENTRY_POINT “MINHAUDF” MODULE_NAME “MINHADLL.DLL”

DELETE

Apaga um ou mais registros de uma tabela Firebird. Se não for utilizado a cláusula WHERE, será apagado todos os registros da tabela.

Sintaxe :

```
DELETE [TRANSACTION transaciona] FROM table
{[WHERE <search_condition>] | WHERE CURRENT OF cursor};
```

Ex : DELETE FROM VENDAS WHERE DATA_VENDA <= ‘1-JAN-1999’;

DROP DATABASE

Apaga o Banco de Dados “.GDB” Firebird. O Banco de Dados, só pode ser deletado pelo seu criador “Owner” e ou pelo SYSDBA do Banco de Dados.

Sintaxe :

```
DROP DATABASE;
```

Ex :

```
DROP DATABASE;
```

DROP DOMAIN

Deleta um domínio previamente criado no Firebird. Se o Domínio estiver em uso por alguma tabela, para solucionar este problema, o campo tem que ser excluído e após isto apagar o Domínio.

Sintaxe :

```
DROP DOMAIN “name”
```

Ex : DROP DOMAIN “STRINGNOME”;

DROP EXCEPTION

Deleta uma exceção previamente criada no seu Banco de Dados Firebird. Se a exceção estiver sendo usada por alguma Stored Procedure e ou Trigger, a exclusão falhará. Desta forma, o usuário precisa retirar a EXCEPTION da SP e ou Trigger e após isto executar novamente a EXCEPTION.

Sintaxe :

DROP EXCEPTION “name”

Ex : DROP EXCEPTION “ACABOU_ESTOQUE”;

DROP EXTERNAL FUNCTION

Deleta do Banco de Dados Firebird a declaração do uso da UDF. Este comando não exclui da DLL, mas, a torna inacessível ao Banco de Dados Firebird. Se alguma SP ou Trigger estiver usando a UDF, ocorrerá um erro na execução dos comandos.

Sintaxe :

DROP EXTERNAL FUNCTION “name”

Ex : DROP EXTERNAL FUNCTION “ABREVIAR_NOME”;

DROP INDEX

Deleta o índice definido pelo usuário do Banco de Dados Firebird.

Sintaxe :

DROP INDEX “name”

Ex : DROP INDEX IND_NOME;

DROP PROCEDURE

Deleta uma SP previamente criada pelo usuário. As SP que estão sendo referenciadas em Trigger, VIEW, não poderão ser excluídas.

Sintaxe :

DROP PROCEDURE “name”

Ex : DROP PROCEDURE RESUMO_VENDAS;

DROP TABLE

Apaga uma tabela do Banco de Dados, e também os índices referenciados e trigger´s que a tabela faz referencia.

Sintaxe :

DROP TABLE “name”;

Ex : DROP TABLE “FORNECEDORES”;

DROP TRIGGER

Apaga uma Trigger do banco de dados.

Sintaxe :

DROP TRIGGER “name”;

Ex : DROP TRIGGER “TRG_VERIFICA_ESTOQUE”;

DROP VIEW

Deleta uma VIEW do Banco de Dados Firebird. Se a VIEW estiver sendo referenciada em outra VIEW, SP, Trigger, não poderá ser apagada. Apenas a definição da VIEW é excluída do Banco de Dados, os dados da VIEW permanecerão intactos na tabela original.

Sintaxe :

DROP VIEW “name”;

Ex : DROP VIEW “TABELA_PRECOS”;

EXECUTE PROCEDURE

Executa uma Stored Procedure, não “Seletável-Select”.

Sintaxe :

EXECUTE PROCEDURE [TRANSACTION transaction]
name [:param [[INDICATOR]:indicator]]
[, :param [[INDICATOR]:indicator] …]
[RETURNING_VALUES :param [[INDICATOR]:indicator]
[, :param [[INDICATOR]:indicator] …]];

Ex : EXECUTE PROCEDURE CUSTO_DEPTO 4 RETURNING_VALUES :SOMA;

GEN_ID()

Retorna o valor do GENERATOR, isto é, pode retornar o valor do GENERATOR e ou incrementar/decrementar. É informado do nome do Generator e o valor do retorno do GENERATOR.

Sintaxe :

GEN_ID(“generator”,ID);

Ex : SELECT GEN_ID(“GEN_FORNECEDOR,0) FROM RDB$DATABASE;

INSERT

Comando responsável para adicionar um mais registros na tabela de Banco de Dados Firebird. Os campos que forem omitidos recebem valores NULOS “NULL”.

Sintaxe :

INSERT [TRANSACTION transaction] INTO <object> [(col [, col …])]
{VALUES (<val> [, <val> …]) | <select_expr>};
<object> = tablename | viewname
<val> = {:variable | <constant> | <expr>
| <function> | udf ([<val> [, <val> …]])
| NULL | USER | RDB$DB_KEY | ?
} [COLLATE collation]
<constant> = num | 'string' | charsetname 'string'
<function> = CAST (<val> AS <datatype>)
| UPPER (<val>)
| GEN_ID (generator, <val>)

Ex : INSERT INTO CLIENTES (ID,NOME) VALUES (1,’Nome do Cliente’);
INSERT INTO VENDAS_OLD SELECT * FROM VENDAS WHERE DATA_VENDA = CURRENTE DATE;

MAX( )

Função que agrega e retorna o valor máximo de uma coluna.

Sintaxe :

MAX([ALL <col> | DISTINCT <col>)

Ex : SELECT MAX(SALARIO) FROM FUNCIONARIOS;

MIN( )

Função que agrega e retorna o valor mínimo de uma coluna.

Sintaxe :

MIN([ALL <col> | DISTINCT <col>)

Ex : SELECT MIN(SALARIO) FROM FUNCIONARIOS;

ROLLBACK

Desfaz as mudanças ocorridas até o exato momento no Banco de Dados Firebird, sem que o comando COMMIT tenha sido executado. Este comando e o Commit fecham a transação aberta pela aplicação e ou ferramenta de gerenciamento as tabelas.

Sintaxe :

ROLLBACK

Ex : ROLLBACK;

SELECT

Este é o comando responsável pela obtenção dos dados da tabela, view´s e ou Stored Procedures.

Sintaxe :

```
SELECT [TRANSACTION transaction]
[DISTINCT | ALL]
{* | <val> [, <val> …]}
[INTO :var [, :var …]]
FROM <tableref> [, <tableref> …]
[WHERE <search_condition>]
[GROUP BY col [COLLATE collation] [, col [COLLATE collation] …]
[HAVING <search_condition>]
[UNION <select_expr> [ALL]]
[PLAN <plan_expr>]
[ORDER BY <order_list>]
[FOR UPDATE [OF col [, col …]]];
<val> = {
col [<array_dim>] | :variable
| <constant> | <expr> | <function>
| udf ([<val> [, <val> …]])
| NULL | USER | RDB$DB_KEY | ?
} [COLLATE collation] [AS alias]
<array_dim> = [[x:]y [, [x:]y …]]
<constant> = num | 'string' | charsetname 'string'
<function> = COUNT (* | [ALL] <val> | DISTINCT <val>)
| SUM ([ALL] <val> | DISTINCT <val>)
| AVG ([ALL] <val> | DISTINCT <val>)
| MAX ([ALL] <val> | DISTINCT <val>)
| MIN ([ALL] <val> | DISTINCT <val>)
| CAST (<val> AS <datatype>)
| UPPER (<val>)
| GEN_ID (generator, <val>)
<tableref> = <joined_table> | table | view | procedure
[(<val> [, <val> …])] [alias]
<joined_table> = <tableref> <join_type> JOIN <tableref>
ON <search_condition> | (<joined_table>)
```

<join_type> = [INNER] JOIN
| {LEFT | RIGHT | FULL } [OUTER]} JOIN
<search_condition> = <val> <operator> {<val> | (<select_one>)}
| <val> [NOT] BETWEEN <val> AND <val>
| <val> [NOT] LIKE <val> [ESCAPE <val>]
| <val> [NOT] IN (<val> [, <val> …] | <select_list>)
| <val> IS [NOT] NULL
| <val> {>= | <=}
| <val> [NOT] {= | < | >}
| {ALL | SOME | ANY} (<select_list>)
| EXISTS (<select_expr>)
| SINGULAR (<select_expr>)
| <val> [NOT] CONTAINING <val>
| <val> [NOT] STARTING [WITH] <val>
| (<search_condition>)
| NOT <search_condition>
| <search_condition> OR <search_condition>
| <search_condition> AND <search_condition>
<operator> = {= | < | > | <= | >= | !< | !> | <> | !=}
<plan_expr> =
[JOIN | [SORT] [MERGE]] ({<plan_item> | <plan_expr>}
[, {<plan_item> | <plan_expr>} …])
<plan_item> = {table | alias}
{NATURAL | INDEX (<index> [, <index> …]) | ORDER <index>}
<order_list> =
{col | int} [COLLATE collation]
[ASC[ENDING] | DESC[ENDING]]
[, <order_list> …]

Começaremos explicando o que acontece nos bastidores do SELECT quando você executa o comando de pesquisa simples.

Ex. : SELECT codigo,nome FROM funcionarios

Após está linha de comando, veja o que acontece com o "motor" do Firebird.

Abre a sessão para troca de dados;
Abre a tabela informada;
Posiciona o cursor de leitura no inicio da tabela;
Repete até o fim da tabela;
Lê as linhas de dados;
Envia as linhas de dados;
Posiciona na próxima linha de dados;
Testa o laço de repetição;
Fecha a tabela;
Fecha sessão de troca de dados.

Operador e funções ligados ao SELECT, isto é, antes da cláusula WHERE

Literal e String - No Firebird, você pode preencher colunas com valores Literais e ou String´s para colunas virtuais e ou colunas fixas da tabela de dados. Mostraremos para este exemplo, o que seria o preenchimento de uma literal em uma coluna virtual ou coluna fixa por literal e ou String.

Ex. Literal :SELECT 1 FROM TabFuncionarios

SELECT 1 as ID TabFuncionarios

Ex. String : SELECT 'Firebird Brasil' FROM TabFuncionarios

SELECT 'Firebird Brasil' as Nome FROM TabFuncionarios

Expressão || – A expressão conhecida para concatenação no Firebird é ||. Desta forma, você pode usa-la para concatenar colunas virtuais e ou colunas fixas. O operador "+" para o Firebird, é usado para cálculos matemáticos, desta forma, se precisar concatenar colunas, use o ||.

DISTINCT – Prevê a exclusão de linhas semelhantes do Result Set.

INTO:var[,:var.]] – Cláusula que transfere o valor de uma coluna para a variável indicada ao SELECT. Se o Result Set retornar mais de uma linha de dados, ocorrerá um erro, indicando que está clausula apenas poderá retornar uma linha de dados.

UDF([<val> [, <val> ...]]) – O Firebird permite você criar funções definidas pelo usuário. E desta forma, você pode chamar está função em instruções SELECT

USER – Variável de ambiente do Firebird, indicando o ID do usuário de conexão com o Firebird.

Operadores que fazem parte da cláusula WHERE

BETWEEN – Este operador testa se o valor da coluna encontra-se no intervalo declarado.

IN – Verifica se valor está contido no Sub-Conjunto de dados na coluna declarada.

ALL – Verifica se uma valor é igual a todos os valores retornados em um SubQuery(*).

ANY e SOME – Verifica se um valor está contido em qualquer valor retornado num SubQuery(*).

EXISTS – Verifica se um valor existe e ou está presente em pelo menos uma linha no retorno do SubQuery(*). Está clausula pode conter também NOT EXISTS.

SINGULAR – Opera com semelhança ao EXISTS, com a diferença de que o valor tem que existir exatamente em uma ocorrência do SubQuery(*).

CONTAINING – Testa se o valor passado a coluna, contém em uma parte da string. Está clausula é CASE-SENSITIVE.

STARTING WITH- Testa se a coluna inicia exatamente como indicado pelo valor passado.

Clausula UNION

Executa a união de uma mais tabelas com o mesmo nome de colunas.

SUM( )

Função de Agregação que retorna a soma dos valores da coluna

Sintaxe :

SUM([ALL <val> | DISTINCT <val>)

Ex : SELECT SUM(VALOR) FROM VENDAS;

UPDATE

Comando responsável pela atualização da tabela no Banco de Dados Firebird. Update trabalha de forma semelhante ao DELETE “é claro, com sua enorme diferença”, se não passarmos a cláusula WHERE, toda a coluna da tabela será atualizada.

Sintaxe :

UPDATE [TRANSACTION transaction] {table | view}
SET col = <val> [, col = <val> …]
[WHERE <search_condition> | WHERE CURRENT OF cursor];

Ex : UPDATE CLIENTE SET DATA_INCLUSAO = CURRENT DATE;

UPPER( )

Função que retorna uma string com todos os caracteres em maiúsculo.

Sintaxe :

UPPER(<col>);

Ex : CREATE DOMAIM SEXO AS CHAR(01) (CHECK VALUE = UPPER(VALUE));

SELECT UPPER(NOME) FROM CLIENTES;

O Que é DIALECT´s

O Firebird introduz o conceito de dialetos para permitir aos usuários utilizar novos recursos que não são compatíveis com as versões anteriores do Firebird.

Como o Firebird procura cada vez mais ser compatível com os padrões definidos para o SQL, algumas novas características da versão 6.0 se tornam incompatíveis com as versões anteriores. O uso dos dialetos ajudam nessa transição. O Dialeto 1 garante compatibilidade com versões antigas dos bancos de dados e clientes. O Dialeto 3 permite total acesso aos novos recursos. O Dialeto 2 é usado como um modo de diagnóstico.

Características de transição :

Características que se comportam de maneira diferente nos dialetos 1 e 3 são chamadas de características de transição :
Qualquer coisa delimitada por aspas duplas (")
Campos Data_e_Hora
Campos numéricos exatos (Decimais e números com precisão maior que 9)

Ambos clientes e BDs devem ter designados um dialeto. Como padrão todo Banco de Dado novo é criado no dialeto 3. O cliente ISQL automaticamente determina qual o dialeto de uma base de dados, a não ser que você especifique um dialeto manualmente durante a conexão.

A seguir estão listadas as diferenças entre os dialetos.

Características que se comportam da mesma forma para todos os dialetos :

IBConsole
Cláusulas ALTER COLUMN e ALTER TABLE
Tipo de dado TIMESTAMP , que é equivalente ao tipo DATE nas versões anteriores do Firebird (armazenam DATA e HORA em conjunto)
A função EXTRACT() e CURRENT_TIMESTAMP
Bancos de Dados Read-Only
Avisos do SQL
API de Serviços, Instalação e Licença
Componentes InterBase Express (IBX) para o Delphi 5

DIALETO 1

Usando o dialeto 1, as características de transição se comportam como no Interbase 5:
Constantes alfanuméricas podem ser delimitadas por aspas simples e duplas. O dialeto 1 não reconhece identificadores delimitados.
O tipo DATE não está disponível, mas é substituído pelo tipo TIMESTAMP, que contém informações sobre data e hora. Quando um banco de dados versão 5 é gravado/restaurado na versão 6, os campos DATE são automaticamente atualizados para o tipo TIMESTAMP.
Os tipos DECIMAL e NUMERIC com precisão maior que 9 são gravados como ponto flutuante.

DIALECT 2

Os Clientes podem ser configurados para utilizar o dialeto 2. Nesse modo, eles reportam erros quando encontram aspas duplas, tipos DATE, ou campos NUMERIC/DECIMAL com precisão maior que 9. Esse

dialeto é utilizado para alertar o desenvolvedor para potenciais problemas durante a migração e não deve ser utilizado para uso normal no dia a dia. Para detectar áreas problemáticas na definição de um banco de dados que você está migrando, extraia a METADATA e rode-a através de um cliente utilizando o dialeto 2. Por exemplo :

```
isql -i v5metadata.sql
```

Lembre-se de NÃO utilizar o dialeto 2 para uso normal dos bancos de dados.

DIALETO 3

As seguintes características são específicas do DIALETO 3, e são incompatíveis com o dialeto 1 e todos os BDs e clientes antigos:
Constantes alfanuméricas devem ser delimitadas por aspas simples (apóstrofe). Aspas duplas (") são usadas somente em identificadores delimitados.
O tipo de dado DATE armazena somente a DATA. Dois novos tipos de dados estão disponíveis : TIME que armazena somente a informação de HORA, e TIMESTAMP que armazena ambos DATA e HORA. O tipo TIMESTAMP substitui a funcionalidade do tipo DATE das versões anteriores do IB. O Dialeto 3 também inclui os operadores funcionais CURRENT_DATE, CURRENT_TIME, e CURRENT_TIMESTAMP.
Tipos DECIMAL e NUMERIC com precisão maior que 9 são gravados utilizando inteiros de 64 bits se forem criados no dialeto 3. Note que todas os campos desse tipo continuam sendo armazenados como float se o BD foi trazido de alguma versão anterior do IB.

Identificadores Delimitados SQL

O Firebird agora suporta identificadores delimitados. Esses identificadores são objetos do banco de dados cujos nomes são delimitados por aspas duplas, e são permitidos somente em bancos da versão 6 usando dialeto 3.
No Firebird Dialeto 3, uma constante alfanumérica é delimitada por aspas simples e um identificador por aspas duplas. Como o nome do identificador agora pode ser delimitado pelas aspas, o tamanho do nome de um identificador é maior possibilitando muito mais variações do que nas versões anteriores.
Os nomes de um Objeto no Firebird agora podem:
Ser uma palavra-chave
Conter espaços (exceto espaços antes e depois do nome)
Usar caracteres não ASCII
Ser sensíveis à CAPS

Segurança com Usuários

A segurança de usuários do Firebird, fica guardado dentro do próprio Banco de Dados. Desta forma, você usuário SYSDBA e ou outro usuário que tenha os direitos de Administrador, restringe o acesso e manutenção a tabelas do Firebird.

Existe dois comandos SQL responsável pelo direito de acesso a tabelas do Banco de Dados :

GRANT

Este é o comando responsável para dizer o que o usuário e ou grupo de usuários "X" podem ou não fazer na(s) tabela(s) em que o GRANT definiu para estes.

Sintaxe :

GRANT
< privilegies> ON [TABLE] { tablename | viewname}
TO { <object>|<userlist> | GROUP UNIX_group}
|<role_granted> TO {PUBLIC | < role_grantee_list>}};
< privileges> = {ALL [PRIVILEGES] | < privilege_list>}
< privilege_list>={
SELECT
| DELETE
| INSERT
| UPDATE [(col [, col …])]
| REFERENCES [(col [, col …])]
[, < privilege_list>…]}}
<object> ={
PROCEDURE procname
| TRIGGER trigname
|VIEWviewname
| PUBLIC
[, <object> …]}
<userlist> ={
[USER] username
| rolename
| UNIX_user}
[, <userlist>…]
[WITH GRANT OPTION]
< role_granted> = rolename [, rolename …]
< role_grantee_list> = [USER] username [, [USER] username …]
[WITH ADMIN OPTION]

Ex :

```
GRANT ALL ON TAB_FORNECEDORES TO EDUARDO;
GRANT SELECT ON TAB_FORNECEDORES TO AIRTON;
GRANT SELECT,INSERT,UPDATE ON TAB_FORNECEDORES TO ANDREA;
```

“Grant para dar direitos a todos os usuários” :

```
GRANT SELECT,INSERT,UPDATE,DELETE ON TAB_FORNECEDORES TO PUBLIC;
```

REVOKE

O comando Revoke é responsável pelo processo contrário do Grant. Isto é, REVOGAR o(s) direito(s) do(s) usuário(s).

Sintaxe :

```
REVOKE < privileges> ON [TABLE] { tablename | viewname}
FROM { <object>|<userlist> | GROUP UNIX_group};
< privileges> = {ALL [PRIVILEGES] | < privilege_list>}
< privilege_list>={
SELECT
| DELETE
| INSERT
| UPDATE [( col [, col …])]
| REFERENCES [(col [, col …])]
[, < privilege_list>…]}}
<object>={
PROCEDURE procname
| TRIGGER trigname
|VIEWviewname
| PUBLIC
[, <object>]}
<userlist> = [USER] username [, [USER] username …]
```

Ex:

```
REVOKE ALL ON TAB_FORNECEDORES TO EDUARDO;
REVOKE SELECT ON TAB_FORNECEDORES TO AIRTON;
REVOKE SELECT,INSERT,UPDATE ON TAB_FORNECEDORES TO ANDREA;
```

“Revoke para todos os usuários. Particularmente interessante, para começar a definir as regras, caso todo mundo tenha acesso a tudo.” :

```
REVOKE SELECT,INSERT,UPDATE,DELETE ON TAB_FORNECEDORES TO PUBLIC;
```

Tipos de Dados do Firebird

O Firebird Dialect 3, suporta a maioria dos tipos de Dados do SQL. O Firebird, apenas não tem como tipo de dado, o tipo Boolean. Mas, isto não é uma falha do Firebird, outro SGDB´s também não tem este tipo de dado. Apesar de não ter este tipo de dado, podemos criar o nosso "tipo boolean" através de DOMAINS. Neste capitulo sobre tipo de dados, iremos utilizar um exemplo de como criar um tipo Boolean para o Firebird.

BLOB

O tipo de Dado BLOB, tem o tamanho variável, isto é, não sabemos na hora da criação do campo BLOB qual será o seu tamanho realmente, mas, o limite do campo Blob que está na documentação do Firebird, é de 64k por segmento.

Este tipo de campo é o tipo indicado para armazenar Textos Grandes "Memos", Fotos, Gráficos, Ícones, isto é, aparentemente não tem um tipo de dado que não possa ser armazenado no Campo Blob. Campos Blob´s não podem ser indexados.

Saber qual o sub-tipo correto utilizar é essencial para criar aplicativos que se utilizem dos campos BLOBs. Os BLOBs se apresentam em 3 versões :

Sub-tipo 0 - Armazena dados em formato binário – Fotos, etc.
Sub-tipo 1 - Armazena dados em formato texto – Memos.
Sub-tipos definidos pelo usuário.

Além dos 2 Sub-tipos pré-definidos, também existem os Sub-tipos definidos pelo usuário. Esses tipos são determinados com o uso de valores negativos logo após a palavra SUB_TYPE. O número utilizado é um inteiro determinado arbitrariamente pelo usuário de acordo com sua preferência, desde que seja negativo. O uso de -1 é funcionalmente equivalente ao uso de -2, -3, Tc...

A única consideração que deve ser tomada é a de se certificar de sempre armazenar o tipo pré-determinado de informação no respectivo sub-tipo de BLOB. O Firebird não faz nenhuma análise dos dados que estão sendo gravados, portanto essa é uma responsabilidade do aplicativo. Nenhum erro será retornado pelo Firebird se um tipo errado de dado for inserido em um BLOB de sub-tipo incorreto, mas um aplicativo pode ser prejudicado se ao recuperar as informações do BLOB, a mesma não corresponder ao formato esperado.

Sintaxe :

Estas declarações é na criação da tabela :

```
MEMO BLOB SUB_TYPE 1;
FOTO BLOB SUB_TYPE 0;
```

Ex :

```
CREATE TABLE FUNCIONARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
NOME VARCHAR(50) NOT NULL,
....,
FOTO BLOB SUB_TYPE 0,
EXPERIENCIA BLOB SUB_TYPE 1 SEGMENT SIZE 80,
)
```

Dica :

Na criação da tabela acima, você verificou uma nova informação “SEGMENT”. Abaixo a explicação :

O tamanho de segmento (Segment Size), é um pequeno pedaço de informação, tipo um conselho, que é mantido com a definição de um Blob. Quando se abre um Blob, pode-se perguntar por segmentos de qualquer tamanho, mas para alguns Blobs um determinado tamanho é mais conveniente que um outro. Blobs que armazenam texto, por exemplo, freqüentemente utilizam segmentos de tamanho 80. O pré-processador e outros programas utilitários usam o tamanho do segmento para determinar o tamanho de buffers que são necessários para transferência de dados para e do Blob.

CHAR(n)

O tipo de Dado CHAR, tem o seu tamanho definido na hora da criação da tabela. Seu tamanho máximo é de 32767, 32k. Este tipo tem o seu tamanho fixo.

Ex :

```
CREATE TABLE FUNCIONARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
NOME VARCHAR(50) NOT NULL,
...
SEXO CHAR(01)
)
```

Este tipo de dado é usado quando você realmente souber o tamanho da coluna/campo a ser criada. Outro exemplo, é criar o coluna de CNPJ, DOMAIN BOLLEAN.

VARCHAR(n)

O tipo de Dado VARCHAR, tem o seu tamanho definido na hora da criação da tabela. Seu tamanho máximo é de 32767, 32k. Este tipo tem o seu tamanho variado na tabela. Isto é, se você criar uma coluna de 45 Caracteres, mas, a coluna tenha apenas 20 Caracteres gravados, o restante, os 25 Caracteres são descartados.

Ex :

```
CREATE TABLE FUNCIONARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
NOME VARCHAR(50) NOT NULL
```

)

Este tipo de dado é usado quando você realmente não souber o tamanho da coluna/campo a ser criada. Outros exemplos, são criar campos Descrições, Inscrições Estaduais.

DATE

O tipo de Dado DATE, no DIALECT 3, armazena a Data, e seu tamanho é de 32 bits inteiros longos.

Ex :

```
CREATE TABLE FUNCIONARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
NOME VARCHAR(50) NOT NULL,
...,
DATA_ADMISSAO DATE
)
```

O Tipo Date, no Dialect 1, armazena Data e Hora ao mesmo tempo.

TIME

O tipo de Dado TIME, no DIALECT 3, armazena a hora, e seu tamanho é de 32 bits inteiros longos.

Ex :

```
CREATE TABLE FUNCIONARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
NOME VARCHAR(50) NOT NULL,
...,
HORA_ENTRADA TIME,
HORA_SAIDA TIME
)
```

O Tipo Time, no Dialect 1, não existia.

TIMESTAMP

O tipo de Dado TIMESTAMP, no DIALECT 3, armazena a Data e a hora ao mesmo tempo, e seu tamanho é de 32 bits inteiros longos.

Ex :

```
CREATE TABLE PRODUTOS (
ID INTEGER NOT NULL PRIMARY KEY,
DESCRICAO VARCHAR(50) NOT NULL,
...,
DATA_HORA_MOVIMENTACAO TIMESTAMP
```

)

O Tipo TimeStamp, no Dialect 1, não existia.

DECIMAL

O tipo de Dado DECIMAL, armazena dígitos a serem gravados na precisão especificada na criação da tabela.

Ex :

```
CREATE TABLE FUNCIOARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
DESCRICAO VARCHAR(50) NOT NULL,
...,
SALARIO DECIMAL(15,02)
)
```

NUMERIC

O tipo de Dado NUMERIC, armazenas dígitos a serem gravados na precisão especificada na criação da tabela.

Ex :

```
CREATE TABLE FUNCIOARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
DESCRICAO VARCHAR(50) NOT NULL,
...,
SALARIO NUMERIC(15,02)
)
```

SMALLINT

O tipo de Dado SMALLINT, armazena dígitos a serem gravados, mas, com o limite de : -32768 a 32767. Serve para armazenar dados numéricos pequenos.

Ex :

```
CREATE TABLE FUNCIOARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
DESCRICAO VARCHAR(50) NOT NULL,
...,
ALTURA SMALLINT
)
```

INTEGER

O tipo de Dado INTEGER, armazena dígitos a serem gravados, mas, diferente do SMALLINT, não existe um limite aparentemente, este tipo é de 32 bits, tem a escala de valores em : -2.147.483.648 até 2.147.483.648

Ex :

```
CREATE TABLE FUNCIOARIOS (
ID INTEGER NOT NULL PRIMARY KEY,
DESCRICAO VARCHAR(50) NOT NULL,
...)
```

FLOAT

O tipo de Dado FLOAT, armazena dígitos a serem gravados, mas, com precisão simples de 7 dígitos.

Ex :

```
CREATE TABLE PRODUTOS (
ID INTEGER NOT NULL PRIMARY KEY,
DESCRICAO VARCHAR(50) NOT NULL,
VLR_ULT_CMP_ITEM FLOAT
...)
```

DOUBLE PRECISION

Este é o tipo de campo no qual eu recomenda para uso monetário/valores no Firebird, Dialect 3. Sua precisão é de 64 bits, na documentação fala em usar apenas para valores científicos, mas, eu o uso em todos os sistemas, e obtenho sempre o arredondamento e precisão desejada.

Ex :

```
CREATE TABLE MOV_FINANCEIRA (
IDCODMOEDA CHAR(03) NOT NULL PRIMARY KEY,
DATA_MOVIMENTACAO DATE NOT NULL PRIMARY KEY,
VALOR_TOTAL_MOVIMENTACAO_DIA DOUBLE PRECISION
);
```

JOINS

Pois bem, começaremos explicando o que acontece no motor do Firebird quando o usuário solicita ao Firebird a Junção.

Quando o Usuário/Administrador do Banco de Dados pede ao Firebird para realizar a junção, usando, digamos o exemplo mais simples de JOIN :

SELECT UF,NOME FROM TABLEA INNER JOIN TABLEB ON UF = SIGLA

o Firebird antes de processar a recuperação de Dados da Tabela, ele precisará fazer uma combinação das Fontes de Dados entre as duas tabelas “neste caso” acima descritas. Após a combinação de Colunas o Firebird combina cada linha da primeira tabela com todas as linhas da segunda tabela. Após está busca, o Firebird começará buscar as informações “Dados” nas tabelas e levar ao Result Set os dados em forma de Junção. Está combinação também é chamada de Produto Cartesiano.

Ok. Explicado o que acontece no motor/bastidores do Firebird vamos agora mostras as opções de JOIN.

Existe dois tipos de JOIN. Vamos explicar o que é cada uma.

INNER JOIN – Liga em linhas, baseando-se em condições de ligação e, somente retornará as linhas que realmente se combinam entre si. O INNER JOIN contém três
tipos :

EQUI-JOINS - Comparação = “Igual”.
Non-EQUI-JOINS - Não comparação. “>, <, >=, <=, <>”.
Reflexive Joins - Sobre si próprio.

2) OUTER JOIN - Liga as linhas de tabelas, mas, não necessariamente precisam casar entre si para aparecer no Result Set. Desta forma, mesmo as linhas que não se encontrou referência no cruzamento das tabelas aparecerá no Resultado de Dados.
Existem três tipos de OUTER JOIN´s, são eles :

LEFT OUTER JOIN;
RIGHT OUTER JOIN;
FULL OUTER JOIN;

LEFT OUTER JOIN – Diz ao motor do Firebird, que todas as linhas da tabela à esquerda, serão recuperadas. Independente de casarem ou não com o identificado ON.

RIGHT OUTER JOIN - Diz ao motor do Firebird, que todas as linhas da tabela à direita, serão recuperadas. Independente de casarem ou não com o identificado ON.
FULL OUTER JOIN - Diz ao motor do Firebird, para fazer um balanceamento do que atente e o que não atende na requisição de Consulta.
Observações :

Salienta que pode haver uma queda de performance, se não houver alguns cuidados de Otimização de Consulta.

É recomendável, apelidar as tabelas envolvidas em Join´s explícitos e ou não explícitos, para uma melhor compreensão das tabelas envolvidas e também evitar Ambigüidade de colunas.

Transação :

Transações são isolamentos no Banco de Dados, desta forma, você "protege" os dados envolvidos na transação de algum erro ocorrido no processamento dessa transação. Isto é, sempre que você iniciar uma transação, é garantido que os dados envolvidos na mesma, serão, ou gravados e ou cancelados, isto serve para todos os dados envolvidos na transação. Um exemplo típico é Notas Fiscais :

Você inicia uma transação no Cabeçalho das notas, envolve os itens, a baixa de estoque, movimentação financeira, parte fiscal, históricos, e, se por algum caso ocorrer um erro, digamos em "movimentação financeira", todo o processamento até o presente momento, não será efetivado. Desta forma, podemos dizer que : Ou Grava/Confirma tudo, ou Cancela/Não confirma nada.

Veremos os tipos de Transação/Isolamentos que o Firebird suporta :

SNAPSHOT :

Fornece a visão atual dos dados do Banco de Dados. Este é o isolamento padrão do Firebird. Isto é, outras transações, incluem, alteram, excluem dados, mas, está transação, não os enxergará, até que seja iniciada outra transação, e aí teremos acesso a o que realmente existe no Banco de Dados. Este isolamento, é mais usado para geração de Relatórios e Processamentos.

SNAPSHOT TABLE STABILITY

Impedem que outras transações atualizem registros, nas tabelas envolvidas nesta transação. Mas, podem serem feitas leituras as tabelas envolvidas.

READ COMMITED

Permite que a transação enxergue todos os dados que foram "commitados" por outras transações. Isto é, outras transações estão incluindo dados, e a transação que estiver com está configuração, terá acesso a todos os dados atualizados. É o nível de transação mais utilizado, pois, é usado para sistemas Multi-Usuários, Multi-Empresas e Sistemas para Internet.

Acesso nativo:

Acesso nativo ao Firebird, através dos componentes InterBase Express “IBX”.

Acesso nativo significa que existem a sua aplicação e os componentes de Acesso Nativo para acessar a base de dados. Neste meio, não existe nada “BDE/ODBC/OLE DB” e ou outro driver. Isto é, e a sua Aplicação e a base de dados, nada mais envolve na conversação dos dados. E isto é feito através de Funções da API do SGDB Firebird.

O que é InterBase Express “IBX” ?

Está é a palheta responsável pela comunicação de dados entre seu aplicativo
e o Banco de Dados Firebird. Nela existem 12 componentes “até a versão 4.4 do IBX”.

Vamos agora, analisar cada componente, sua Função, suas Propriedades, Métodos e Eventos que cada componente possuí.

Acesso nativo/direto ao Firebird.

TIBDataBase

É o responsável pela conexão entre a sua aplicação e o Banco de Dados
Firebird.

Propriedades :

Connected : Ativa ou desativa a conexão com a base de Dados.

DataBaseName : Nome do arquivo de Base de Dados do Firebird ".GDB".

DefaultTransaction : Especifica qual Transação “IBTransaction”, é ligado automaticamente ao IBDataBase. Serve para aplicações onde existe apenas uma transação envolvida em todo o sistema. Isto é, para sistemas pequenos, onde o controle de transação não é tão importante para o bom funcionamento da aplicação. Se a sua aplicação necessita de um controle de várias transações ao mesmo tempo, deixe está propriedade em branco.

IdleTimer : Especifica quanto tempo o Cliente irá esperar por uma resposta
do servidor. Caso o tempo tenha se excedido, a conexão será desfeita.

LoginPrompt : Se ativa ou não o pedido de senha quando houver a conexão com
o Banco de Dados Firebird.

SQLDialect : Indica qual o Dialeto que será utilizado pela conexão. Caso necessita saber mais sobre Dialect, dê uma olhada no capitulo de Firebird.

TraceFlags : Indica quais serão as ações monitoradas pelo TIBSQLMonitor.

Eventos :

AfterConnected : Ocorre após a conexão.

AfterDisconnected : Ocorre após ter terminado a conexão.

BeforeConnected : Ocorre antes da conexão.

BeforeDisconnected : Ocorre quando for pedido o término de conexão.

OnDialectDowngradeWarning : Ocorre quando o Dialect da Aplicação e do Banco de Dados não estão configurados da mesma forma. Isto é, este evento é chamado, se o Dialect de sua aplicação for 3 e do seu Banco de Dados for 1, este evento será disparado e você pode tomar as decisões necessárias para avisar ao usuário e ou escrever algum código na sua aplicação para tentar contornar este erro.

OnIdleTimer : Ocorre quando o tempo especificado na propriedade IdleTimer
ter se esgotado.

OnLogin : Este evento substitui o UserName e Password definido em
LoginPrompt, e passa toda a rotina de tratamento de login ao programador.

Principais métodos :

CloseDataSets : Fecha todos os DataSet´s ativos no momento na aplicação no qual esteja usando está conexão.

CreateDataBase : Método responsável pela Criação do seu “.GDB”. Muito útil, se a sua aplicação é independente na criação do “.GDB”. Isto é, é com este método em conjunto com o Params que você cria a sua base de dados, sem a necessidade de criar através do IBConsole e ou outra ferramenta de Administração do Firebird.

DropDataBase : Método responsável pela exclusão do Banco de Firebird “.GDB” do servidor e ou maquina local. É necessário antes de chamar este método, verificar se a conexão não está ativa no momento.

ForceClose : Força a desconexão da sua aplicação com o Banco de Dados Firebird. Este método, não verifica transação ativa, ele simplesmente ignora todas as transações e fecha a conexão.

GetFieldNames(const TableName:String;List:Tstrings) : Retorna para a variável do StringList, todas colunas da tabela passada como parâmetro.

GetTableNames(List:Tstrings;[System Tables :Boolean=False) : Retorna a para a variável do StringList, todas as tabelas da conexão. O segundo parâmetro indica se você deseja ver também as tabelas de sistema do Firebird. Como default, é não “False”.

SQLDialect : Diz qual o Dialect que será usado pela conexão.

TestConnected : Testa se a conexão está ativa ou não.

TransactionCount : Retorna quantas transações estão associadas ao Data Base.

TIBTransaction

É o responsável pelo controle de transações da sua aplicação. Você pode controlar transações concorrentes, ou em threads independentes. Para tratarmos de transações, precisaríamos de um artigo a parte.

Propriedades :

Active : Inicia a transação, tem o mesmo efeito do método StartTransaction.

DefaultAction : Indica a sua transação qual será o método executado quando o parâmetro IdleTimer exceder.

DefaultDataBase : Indica a qual conexão a transação pertence.

Params : Propriedade onde você especifica o tipo de transação, isto é, como a sua transação se portará na sua aplicação. Em versões atualizadas do IBX, você clica duas vezes no componente para abrir as opções de transações. Estas opções estão explicadas no capitulo de transações.

IdleTimer : Especifica quanto tempo a transação ira esperar para executar a propriedade DefaultAction

Eventos :

OnIdleTimer : Ocorre quando o tempo excedido por IdleTimer.

Principais Métodos :

Commit : Método responsável pela confirmação e gravação de toda a transação. Quando se inicia uma transação, você precisa fechá-la e confirmar ou não os dados envolvidos na transação. Para confirmar os dados e fechar a transação, você chama o método Commit.

CommitRetaining : Método semelhante ao Commit, mas, diferente do Commit, não fecha a transação, isto é, a transação ainda fica ativa, mas, foi confirmado a gravação dos Dados envolvidos na transação.

RollBack : Método responsável pelo cancelamento e a não gravação de toda a transação. Quando se inicia uma transação, você precisa fechá-la e confirmar ou não os dados envolvidos na transação. Para cancelar a transação e os dados envolvidos nela, você executa o RollBack.

RollBackRetaining : Método semelhante ao RollBack, mas, diferente do RollBack, não fecha a transação, isto é, a transação ainda fica ativa, mas, foi o cancelado os Dados envolvidos na transação.

StartTransaction : Inicia uma transação, no qual ainda tenha sido iniciada.

InTransaction : Verifica se a transação está ativa “Iniciada” ou não.
TIBTable

Faz a conexão com a sua Base de Dados no mesmo modelo de TTable do Data Access do Delphi. O uso de TIBTable em modelo Client-Server não é

recomendado, por causa do número de instruções muito grande enviadas ao servidor. Por causa disto, é recomendável o uso de TIBQuery e ou TIBDataSet.

Diante deste contexto, não entraremos em maiores detalhes sobre o uso do TIBTable.

TIBQuery

Faz a conexão SQL com a sua Base de Dados Firebird. Este componente aceita quase todas as instruções DDL, DML e DQL. Para utilizar o TIBQuery como um componente de edição, é necessário o uso do TIBUpdateSQL. As propriedades e eventos, são praticamente as mesmas do componente TQuery de Data Access do Delphi. Explicarei algumas que estão diretamente ligada ao Firebird.

Propriedades :

BufferChunks : Número de Registros no Buffer.

DataBase : Onde você especifica a qual Data Base “IBDataBase” a Query está ligada.

Transaction : Onde você especifica qual a Transação “IBTransaction” a Query está ligada.

Unidirectional* : Especifica se a navegação será Unidirecional, isto é, em um sentido apenas. E este sentido é somente para navegação para os próximos registros.

*Este método não está ligado diretamente ao Firebird, mas, achei interessante explicar, pois o mesmo não está muito explicado em livros e na Internet, e está propriedade está ligada diretamente a performance do Banco de Dados.

UpdateObject : Propriedade ligada ao TIBUpdateSQL, onde indica que a sua Query é Editável, isto é, podem ser feitas Inclusões, Alterações e Exclusões.

Eventos :

AfterDatabaseDisconnect : Ocorre após a desconexão do IBDataBase.

AfterTransactionEnd : Ocorre após a transação for encerrada. Isto é, após um Commit e ou RollBack.

BeforeDatabaseDisconnect : Ocorre antes da desconexão do IBDataBase.

BeforeTransactionEnd : Ocorre antes do encerramento da transação.

DatabaseFree : Ocorre depois que o IBDataBase seja eliminado da memória.

TransactionFree : Ocorre depois que o IBTransaction seja eliminado da memória.

Principais Métodos :

ExecSQL : Executa as instruções DML contidas na propriedade SQL. Este método é válido para Instruções Insert, Update, Delete. Para instruções SQL, execute o método Open.

Open : Executa as instruções DSL do Banco de Dados Firebird. Executa as instruções contidas na propriedade SQL. Este método é válido para Instruções Select. Para instruções DML, execute o método ExecSQL.

Os demais métodos são os mesmos do TQuery .

TIBStoredProc

Executa uma procedure armazenada no Servidor Firebird. Este componente é equivalente ao componente TStoreProc de Data Access do Delphi.

Propriedades :

DataBase : Indica o qual IBDataBase pertence o seu Stored Procedure.

Filtered : Especifica se as condições de filtro para a tabela em questão, estarão ativas na Stored Procedure ou não.

ForcedRefresh : Especifica se será forçado o Refresh no DataSet ou não.

Params : Especifica os parâmetros de entrada da Stored Procedure. Este é o método mais importante da Stored Procedure, pois, ao mesmo tempo que é método responsável pela passagem de parâmetros a sua Stored Procedure, após a execução da Stored Procedure, é onde ficará armazenado os parâmetros de retorno da sua Stored Procedure. Isto é, há uma troca de função da propriedade Params, após a execução do Params.

Transaction : Indica a qual transação “IBTransaction” a Stored Procedure pertence. Não é possível executar uma Stored Procedure “e isto também serve para Trigger”, em uma transação separada a da aplicação. Pois, desta forma, poderia ser quebrada a integridade referencial do Banco de Dados.

Eventos :

São os mesmos da TIBQuery. Não existem diferenças.

Principais Métodos :

CopyParams : Faz uma cópia dos parâmetros para outro TIBStoredProc.

ExecProc : Executa as instruções contidas na Stored Procedure e, faz a troca de parâmetros de Entrada para os de saída.

TIBUpdateSQL

Propriedades :

Permite definir instruções DML para cada método Insert, Edit e Delete. TIBUpdateSQL+TIBQuery representa toda a funcionalidade SQL de manipulação de Dados e Live Result Set. A seguir, mostraremos as principais propriedades.

InsertSQL - Instrução SQL de Inserção de Dados. É executado quando for chamado o método Append/Insert.

ModifySQL - Instrução SQL de alteração de Dados. É executada quando a tabela for colocada em modo de Edição

DeleteSQL - Instrução SQL para deletar Dados. É executada quando o método Delete for chamado.

RefreshSQL - Instrução SQL para executar o Refresh. É executada quando for chamado o método Refresh.

Eventos :

Não existem eventos associados a este componente.

Principais Métodos :

Não existem métodos relacionados diretamente ao Firebird.

TIBDataSet

Engloba toda a funcionalidade de TIBQuery+TIBUpdateSQL, e ainda é mais rápido. É o recomendado pelo Autor e pelos criadores dos componentes Interbase® Express "IBX".

Propriedades :

Active : Indica que a Tabela está aberta.

BufferChuncks : Número de Registros no Buffer.

DataBase : Indica o TIBDataBase da tabela.

DeleteSQL, InsertSQL, ModifySQL, RefreshSQL : São os mesmos explicados no componente TIBUpdateSQL.

SelectSQL : Instrução SQL de Result Live executada quando a tabela for aberta.

Transaction : Indica a qual transação “IBTransaction” pertence o IBDataSet.

Eventos :

OnAfterDataBaseDisconnect : Ocorre após o término da conexão com o Banco de Dados.

OnAfterTransaction : Ocorre após o término de transação. Este evento captura apenas o término de transação conhecida como "Hard". Isto é, não é chamado após o CommitRetaining e RollBackRetaining

BeforeDataBaseDisconnect : Ocorre antes do término da conexão com o Banco de Dados.

OnBeforeTransactionEnd : Ocorre antes do término "Hard" da transação.

DataBaseFree : Ocorre após a liberação dos Handle´s aloucados pelo TIBDataBase.

TransactionFree : Ocorre após a liberação dos Handle´s aloucados pelo TIBTransaction.

TIBSQL

Executa instruções SQL. Recomendável para o uso de instruções DML. Este componente não oferece condições de controles de Data Control, desta forma, um SELECT não poderá ser recuperado. A seguir as principais propriedades, eventos e métodos.

Propriedades :

GotoFirstRecordOnExecute : Se True vai para o primeiro registro após a execução de Instrução DQL.

ParamCheck : True indica que a instrução TIBSQL pode e irá receber parâmetros.

SQL : Instrução que será executada.

Transaction : Indica a qual transação “IBTransaction” pertence o IBSQL.

Eventos :

OnSQLChanging : Ocorre quando a instrução SQL é modificada.

Principais Métodos :

ExecQuery : Executa a Query SQL.

TIBDataBaseInfo

Componente que retorna várias informações do seu Banco de Dados especificado. Para capturar as informações, apenas coloque o componente no formulário e sete DataBase para o objeto TIBDataBase correspondente.

TIBSQLMonitor

Cria um LOG para acompanhamento de todas as instruções enviadas para o servidor. Este componente trabalha em conjunto com a propriedade TraceFlags de TIBDataBase. Para utilizar o componente, basta coloca-lo no formulário. Para capturar as informações enviadas pelo servidor, utilize um List Box e ou outro componente de Lista de String e configure o evento OnSQL.

TIBEvents

Componente que captura eventos do Banco de Dados Firebird. O SGDB Firebird pode emitir eventos e a sua aplicação capturar. Este eventos são programados dentro de Trigger e Stored Procedure.

Propriedades :

Events : Nome dos eventos recebidos pelo TIBEvents. Você pode incluir até 15 Eventos por cada objeto IBEvents. Se você precisar de mais eventos, você precisará colocar mais IBEvents na sua aplicação.

Registered : Se True, indica que o objeto estará ativo.

Eventos :

OnEventAlert : Ocorre quando algum evento é recebido pelo Objeto.

Principais Métodos :

CancelEvents : Cancela os eventos pendentes.

QueueEvents : Indica a sua aplicação, que foi inicializada o recebimento de eventos.

RegisterEvents : Registra ao IBDataBase, os eventos listados em IBEvent´s.

UnRegisterEvents : Diz ao Banco de Firebird, que está retirando os registros do IBEvent´s.

TIBExtract.

Componente responsável por extrair informações das Tabelas de Sistemas no Firebird. Tem o mesmo papel do Extract MetaData do IBConsole. Existem dois tipos relacionados ao IBExtract responsáveis pela Extração das informações do Banco de Dados :

```
TExtractObjectTypes =
  (eoDatabase, eoDomain, eoTable, eoView, eoProcedure, eoFunction,
   eoGenerator, eoException, eoBLOBFilter, eoRole, eoTrigger, eoForeign,
   eoIndexes, eoChecks, eoData);

TExtractType =
  (etDomain, etTable, etRole, etTrigger, etForeign,
   etIndex, etData, etGrant, etCheck);
```

Propriedades :

DataBase : Nome do IBDataBase no qual você liga ao IBExtract.

ShowSystem : Se retorna as informações das Tabelas de Sistemas.

Eventos :

Não existem eventos associados a este componente.

Principais Métodos :

IBExtract1.ExtractObject : Método responsável pela extração das informações das tabelas do Banco de Dados.

Exemplo :

```
IBExtract1.ExtractObject(eoDatabase);  //  Extrair todas informações do Banco Firebird

IBExtract1.ExtractObject(eoTable, 'EMPLOYEE');  // Extrai informações referente a apenas a tabela EMPLOYEE

IBExtract1.ExtractObject(eoTable, 'EMPLOYEE', [etDomain, etForeign, etIndex, etGrant]);  // Extrai informações referente a apenas a tabela EMPLOYEE, mas, trará apenas informações referentes a domínio, Foreign Key, Índices, e Direitos.
```

Foi citado mais acima, a recomendação do uso do IBDataSet para manutenção dos Dados. Mas, não posso deixar de citar a dobradinha “IBQuery+IBSQL”, pois, estes dois componentes tem uma melhor performance sobre os demais componentes. Vale ressaltar também, que se for apenas fazer pesquisa e mostrar num GRID “por exemplo” os dados, use sempre o IBQuery !.

Palheta Interbase® Admin.

Interbase® Admin é o conjunto de componentes para a Administração de seu Banco de Dados Firebird. Neste capitulo, mostraremos apenas a funcionalidade de cada componente.

IBConfigService

Este componente tem função de enviar e mudar os parâmetros do Banco de Dados Firebird, entre eles, o Intervalo de Sweap, Page Control´s e outros.

IBBackupService

Este componente tem a função de realizar o Backup de sua base de dados. Você tem pode especificar os parâmetros do Backup.

IBRestoreService

Este componente tem a função de restaurar o Backup feito pelo IBConsole e ou pelo componente de Backup.

IBValidationService

Este componente tem a função de validar a sua base de dados. Sendo assim, você poderá validar as transações em Limbo, as transações Default e outras validações.

IBStaticalService

Este componente tem a função de mostrar as estatísticas do Banco de Dados, entre eles : Log, Header Pages, Índices e outros.

IBLogService

Este componente tem a função de lhe ajudar a criar seu próprio LOG do Banco de Dados Firebird.

IBSecurityService

Este componente tem a função de gerênciar o acesso a usuários ao seu Banco de Dados Firebird® e também a manutenção dos mesmos.

IBLicensingService

Este componente tem a função de dar a manutenção nos certificados do Software InterBase. Não é utilizada no Firebird.

IBServerProperties

Este componente tem a função de retornar informações de configuração do servidor Firebird.

IBInstall

Este componente tem a função de Instalar o Firebird, configurar os diretórios de Instalação e os componente que serão instalados.

IBUninstall

Este componente tem a função de desinstalar os componentes usados na instalação do Firebird.

Artigo Original

Anderson Haertel Rodrigues
Colaboração de Marcus Boi

anderson.hr@bol.com.br

Comunidade Firebird de Língua Portuguesa

Visite a Comunidade em:

http://www.comunidade-firebird.org