

Manual de Referência do Derby

Version 10

Derby Document build:
October 2, 2006, 8:11:25 AM (PDT)

Contents

Direitos autorais reservados	7
Sobre este documento	8
Finalidade deste documento	8
Audiência	8
Organização deste documento	8
Referência da linguagem SQL	10
Letras maiúsculas e caracteres especiais	10
Identificadores SQL	10
Regras para identificadores SQL92.....	11
IdentificadorSQL92.....	11
nome-da-coluna.....	12
nome-da-correlação.....	12
novo-nome-da-tabela.....	13
nome-do-esquema.....	13
nome-de-coluna-simples.....	14
nome-do-sinônimo.....	14
nome-da-tabela.....	14
nome-da-visão.....	14
nome-do-índice.....	15
nome-da-restrição.....	15
nome-do-cursor.....	15
nome-do-gatilho.....	16
identificador-de-autorização.....	16
Instruções	16
Interação com o sistema de dependências.....	16
Instrução ALTER TABLE.....	17
Instruções CREATE.....	20
Instruções DROP.....	34
Instruções RENAME.....	36
Instruções SET.....	37
CALL (PROCEDIMENTO).....	38
Cláusula CONSTRAINT.....	39
Instrução DECLARE GLOBAL TEMPORARY TABLE.....	45
Instrução DELETE.....	48
Cláusula FOR UPDATE.....	48
Cláusula FROM.....	49
Cláusula GROUP BY.....	49
Cláusula HAVING.....	50
INNER JOIN.....	51
Instrução INSERT.....	52
Operação JOIN.....	53
LEFT OUTER JOIN.....	53
Instrução LOCK TABLE.....	54
Cláusula ORDER BY.....	55
Consulta.....	56
RIGHT OUTER JOIN.....	57
SubconsultaEscalar.....	58
ExpressãoSeleção.....	59
Instrução SELECT.....	61
ExpressãoTabela.....	62
SubconsultaTabela.....	63
Instrução UPDATE.....	64
VALUES Expressão.....	65
Cláusula WHERE.....	66
Cláusula WHERE CURRENT OF.....	67
Funções nativas	67

Funções nativas padrão.....	67
Agregações (funções de conjunto).....	68
ABS ou ABSVAL.....	69
AVG.....	70
BIGINT.....	70
CAST.....	71
CHAR.....	73
LENGTH.....	75
Concatenação.....	75
Expressões NULLIF e CASE.....	76
COUNT.....	77
COUNT(*).....	77
CURRENT DATE.....	77
CURRENT_DATE.....	77
CURRENT ISOLATION.....	78
CURRENT SCHEMA.....	78
CURRENT TIME.....	78
CURRENT_TIME.....	78
CURRENT TIMESTAMP.....	79
CURRENT_TIMESTAMP.....	79
CURRENT_USER.....	79
DATE.....	79
DAY.....	80
DOUBLE.....	80
HOUR.....	81
IDENTITY_VAL_LOCAL.....	81
INTEGER.....	83
LOCATE.....	83
LCASE ou LOWER.....	84
LTRIM.....	84
MAX.....	85
MIN.....	85
MINUTE.....	86
MOD.....	86
MONTH.....	87
RTRIM.....	87
SECOND.....	87
SESSION_USER.....	88
SMALLINT.....	88
SQRT.....	89
SUBSTR.....	89
SUM.....	90
TIME.....	90
TIMESTAMP.....	91
UCASE ou UPPER.....	91
USER.....	92
VARCHAR.....	92
YEAR.....	92
Funções do sistema nativas.....	93
SYSCS_UTIL.SYSCS_CHECK_TABLE.....	93
SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS.....	93
SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY.....	93
Procedimentos do sistema nativos.....	94
SYSCS_UTIL.SYSCS_COMPRESS_TABLE.....	94
SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE.....	95
SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS.....	97
SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING.....	97
SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY.....	98
SYSCS_UTIL.SYSCS_FREEZE_DATABASE.....	98

SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE.....	99
SYSCS_UTIL.SYSCS_CHECKPOINT_DATABASE.....	99
SYSCS_UTIL.SYSCS_BACKUP_DATABASE.....	100
SYSCS_UTIL.SYSCS_EXPORT_TABLE.....	100
SYSCS_UTIL.SYSCS_EXPORT_QUERY.....	101
SYSCS_UTIL.SYSCS_IMPORT_TABLE.....	102
SYSCS_UTIL.SYSCS_IMPORT_DATA.....	103
Tipos de dado.....	105
Visão geral dos tipos de dado nativos.....	105
Tipos numéricos.....	105
Tipo de dado - atribuições e comparação, classificação e ordenação.....	107
BIGINT.....	109
BLOB.....	110
CHAR.....	111
CHAR FOR BIT DATA.....	111
CLOB.....	112
DATE.....	113
DECIMAL.....	114
DOUBLE	115
DOUBLE PRECISION.....	115
FLOAT.....	116
INTEGER.....	116
LONG VARCHAR.....	117
LONG VARCHAR FOR BIT DATA.....	117
NUMERIC.....	117
REAL.....	118
SMALLINT.....	119
TIME.....	119
TIMESTAMP.....	120
VARCHAR.....	120
VARCHAR FOR BIT DATA.....	121
Expressões SQL.....	122
Precedência das expressões.....	124
Expressão booleana.....	124
Parâmetros dinâmicos.....	126
Palavras reservadas do SQL.....	130
Suporte do Derby às funcionalidades do SQL-92.....	134
Tabelas do sistema Derby.....	141
SYSALIASES.....	141
SYSCHECKS.....	141
SYSCOLUMNS.....	142
SYSCONGLOMERATES.....	143
SYSCONSTRAINTS.....	143
SYSDEPENDS.....	144
SYSFILES.....	144
SYSFOREIGNKEYS.....	145
SYSKEYS.....	145
SYSSCHEMAS.....	146
SYSSTATISTICS.....	146
SYSSTATEMENTS.....	146
SYSTABLES.....	147
SYSTRIGGERS.....	147
SYSVIEWS.....	149
Mensagens de exceção e estados SQL do Derby.....	150
Referência de SQLState e mensagem de erro.....	150
Referência do JDBC.....	177
Classes, interfaces e métodos java.sql do núcleo do JDBC.....	177
java.sql.Driver.....	177
java.sql.DriverManager.getConnection.....	178

Sintaxe da URL de conexão com banco de dados Derby.....	178
Sintaxe da URL de conexão com banco de dados para aplicativos com bancos de dados incorporados.....	179
Sintaxe SQL adicional.....	179
Atributos da URL de conexão com o banco de dados Derby.....	180
java.sql.Driver.getPropertyInfo.....	180
java.sql.Connection	181
java.sql.Connection.setTransactionIsolation	181
java.sql.Connection.setReadOnly	181
java.sql.Connection.isReadOnly	181
Funcionalidades de conexão não suportadas.....	181
java.sql.DatabaseMetaData.....	182
Conjuntos de resultados DatabaseMetaData.....	182
getProcedureColumns/getProcedureColumns.....	182
Parâmetros para getProcedureColumns.....	182
Colunas do ResultSet retornado por getProcedureColumns.....	182
Funcionalidades de DatabaseMetaData não suportadas.....	183
java.sql.Statement	183
Objetos ResultSet.....	184
java.sql.PreparedStatement.....	184
Instruções preparadas e colunas de fluxo.....	184
java.sql.CallableStatement	186
CallableStatement e parâmetros OUT.....	186
CallableStatement e parâmetros INOUT.....	186
java.sql.ResultSet	187
ResultSets e colunas de fluxo.....	187
java.sql.ResultSetMetaData.....	188
java.sql.SQLException.....	188
java.sql.SQLWarning.....	188
Mapeamento de java.sql.Types em tipos SQL.....	189
java.sql.Blob e java.sql.Clob.....	189
Notas.....	191
java.sql.Connection.....	192
java.sql.ResultSet.....	192
java.sql.Statement.....	193
java.sql.PreparedStatement.....	194
java.sql.CallableStatement	194
java.sql.DatabaseMetaData.....	194
java.sql.ResultSetMetaData.....	194
java.sql.BatchUpdateException.....	194
Pacote JDBC para Connected Device Configuration/Foundation Profile (JSR169).....	195
Funcionalidades apenas do JDBC 3.0.....	195
java.sql.Connection	196
java.sql.DatabaseMetaData	196
java.sql.ParameterMetaData.....	196
java.sql.PreparedStatement.....	197
java.sql.Savepoint.....	197
Definir e desfazer até um ponto de salvamento.....	197
Liberação de ponto de salvamento.....	198
Regras para pontos de salvamento.....	198
Restrições dos pontos de salvamento.....	198
java.sql.Statement.....	198
Chaves autogeradas.....	198
Sintaxe de escape do JDBC.....	199
Palavra chave de escape do JDBC para instruções call.....	200
Sintaxe de escape do JDBC.....	200
Sintaxe de escape do JDBC para cláusulas LIKE.....	200
Sintaxe de escape do JDBC para a palavra chave fn.....	201

Sintaxe de escape do JDBC para junções externas.....	203
Sintaxe de escape do JDBC para formatos de hora.....	203
Sintaxe de escape do JDBC para formatos de carimbo do tempo.....	204
Definição de atributos para a URL de conexão com o banco de dados.....	205
bootPassword=chave.....	205
create=true.....	205
databaseName=nomeBancoDados.....	206
dataEncryption=true.....	206
encryptionProvider=nomeProvedor.....	206
encryptionAlgorithm=algoritmo.....	207
territory=ll_CC.....	207
logDevice=caminhoDiretorioLog.....	208
password=senhaUsuario.....	209
rollForwardRecoveryFrom=Caminho.....	209
createFrom=Caminho.....	209
restoreFrom=Caminho.....	210
shutdown=true.....	210
user=nomeUsuário.....	211
(nenhum atributo).....	211
Conformidade com o J2EE: API de transação Java e extensões javax.sql.....	212
JVM e bibliotecas para as funcionalidades do J2EE.....	213
A API do JTA.....	213
Notas sobre o comportamento do produto.....	213
javax.sql: Extensões JDBC.....	213
API do Derby.....	215
Ferramentas e utilitários autônomos.....	215
Classes de implementação do JDBC.....	215
Driver de JDBC.....	215
Classes de fonte de dados.....	215
Utilitários e interfaces diversas.....	216
Territórios suportados.....	217
Limitações do Derby.....	218
Limitações do comprimento de identificador.....	218
Limitações numéricas.....	218
Limitações das cadeias.....	219
Limitações de DATE, TIME e TIMESTAMP.....	219
Limitações dos valores do gerenciador de banco de dados.....	219
Marcas registradas.....	221

Direitos autorais reservados

Segunda Edição (Julho de 2005)

Copyright 1997, 2005 Apache Software Foundation, ou seus concessionários de licença, conforme se aplique.

Licenciado sob a Licença Apache, Versão 2.0 (doravante chamada apenas de "Licença"); este arquivo não pode ser utilizado a não ser em conformidade com a Licença. Pode ser obtida uma cópia da Licença em

<http://www.apache.org/licenses/LICENSE-2.0>

A menos que seja requerido por lei aplicável, ou concordado por escrito, o programa distribuído sob esta Licença é distribuído na BASE DE "COMO É", SEM GARANTIAS OU CONDIÇÕES DE QUALQUER ESPÉCIE, sejam expressas ou implícitas. Veja na Licença as condições específicas que governam as permissões e limitações sob a Licença.

Sobre este documento

Para obter informações gerais sobre a documentação do Derby, como a relação completa de documentos, convenções e leitura adicional, deve ser consultada a *Introdução ao Derby*.

Finalidade deste documento

Este documento, o *Manual de Referência do Derby*, fornece informações de referência sobre o Derby. Cobre a linguagem SQL do Derby, a implementação do JDBC do Derby, os catálogos do sistema do Derby, as mensagens de erro do Derby, as propriedades do Derby, e as palavras chave do SQL.

Audiência

Este documento é a referência para os usuários do Derby, que são tipicamente desenvolvedores de aplicativos. Os usuários do Derby não familiarizados com o padrão SQL, ou com a linguagem de programação Java, serão beneficiados pela consulta a livros sobre estes tópicos.

Os usuários do Derby que desejarem uma abordagem para trabalhar com o Derby do tipo *como fazer*, ou uma introdução aos conceitos do Derby, devem consultar o *Guia do Desenvolvedor do Derby*.

Organização deste documento

Este documento inclui as seguintes seções:

- [Referência da linguagem SQL](#)

Informações de referência sobre a linguagem SQL do Derby, incluindo páginas de manual para instruções, funções e outros elementos da sintaxe.

- [Palavras reservadas do SQL](#)

Palavras chave do SQL além das palavras chave do padrão SQL-92.

- [Suporte do Derby às funcionalidades do SQL-92](#)

Uma lista de funcionalidades do SQL-92 que o Derby dá suporte e não dá suporte.

- [Tabelas do sistema Derby](#)

Informações de referência sobre os catálogos do sistema do Derby.

- [Mensagens de exceção e estados SQL do Derby](#)

Informações sobre as mensagens de exceção do Derby.

- [Referência do JDBC](#)

Informações sobre a implementação da interface de JDBC do Derby, incluindo suporte a funcionalidades do JDBC 2.0.

- [Definição de atributos para a URL de conexão com o banco de dados](#)

Informações sobre os atributos suportados pela URL de conexão com o banco de dados do JDBC do Derby.

- [Conformidade com o J2EE: API de transação Java e extensões javax.sql](#)

Informações sobre os atributos suportados pelo suporte do Derby à API de Transação Java.

- [API do Derby](#)

Notas sobre as APIs proprietárias para o Derby.

Referência da linguagem SQL

O Derby implementa um subconjunto do núcleo do SQL-92, assim como algumas funcionalidades do SQL-99.

Esta seção fornece uma visão geral da linguagem SQL corrente, através da descrição das instruções, funções nativas, tipos de dado, expressões e caracteres especiais presentes na linguagem.

Letras maiúsculas e caracteres especiais

Ao utilizar classes e métodos do JDBC, são submetidas instruções SQL ao Derby na forma de cadeias. O conjunto de caracteres permitido nas cadeias contendo instruções SQL é o Unicode. Nestas cadeias se aplicam as seguintes regras:

- Aspas delimitam os identificadores especiais, referidos no SQL-92 como *identificadores delimitados*.
- Apóstrofos delimitam cadeias de caracteres.
- Na cadeia de caracteres, para representar um apóstrofo (também chamado de aspas simples) são usados dois apóstrofos (Em outras palavras, o apóstrofo é o caractere de escape do apóstrofo).

As aspas não necessitam de caractere de escape. Para representar aspas deve-se, simplesmente, utilizar aspas. Entretanto, deve ser observado que no programa Java as aspas necessitam do caractere de escape contrabarra (\).

Exemplo:

```
-- o apóstrofo é o caractere de escape do apóstrofo
VALUES 'Maria D''Almeida'

-- no ij não há necessidade de caractere de escape para aspas
VALUES 'Ele disse, "Bom dia!"'

-- no programa Java aspas necessitam de escape
n = stmt.executeUpdate(
    "UPDATE minhaTabela SET stringCol = 'Ele disse, \"Bom dia!\"'");
```

- As palavras chave do SQL não são sensíveis a letras maiúsculas e minúsculas. Por exemplo, a palavra chave SELECT pode ser escrita como SELECT, Select, select, ou sELECT.
- Os identificadores no estilo SQL-92 não são sensíveis a letras maiúsculas e minúsculas (consulte [Identificador SQL92](#)), a menos que estejam delimitados.
- Os identificadores no estilo Java são sensíveis a letras maiúsculas e minúsculas.
- O * é um caractere curinga na [Expressão Seleção](#). Consulte [O curinga *](#). Também pode ser o operador de multiplicação. Em todos os outros casos é um meta-símbolo da sintaxe, sinalizando itens que podem ser repetidos zero ou mais vezes.
- % e _ são caracteres curinga quando utilizados nas cadeias de caracteres após o operador LIKE (exceto quando precedidos pelo caractere de escape). Consulte [Expressão booleana](#).
- De acordo com o padrão SQL-92, dois hífenes (--) e o caractere de nova-linha delimitam um comentário. Os dois hífenes iniciam o comentário, e o caractere de nova-linha termina o comentário.

Identificadores SQL

O *identificador* é a representação dentro da linguagem de itens criados pelos usuários, em oposição às palavras chave e comandos da linguagem. Alguns identificadores

representam *objetos do dicionário*, que são objetos criados pelo usuário (como tabelas, visões, índices, colunas e restrições), que são armazenados no banco de dados. São chamados de objetos do dicionário porque o Derby armazena as informações sobre estes objetos nas tabelas do sistema, algumas vezes chamado de dicionário de dados. O padrão SQL também define maneiras de criar aliases para estes objetos em certas instruções.

Cada tipo de identificador deve estar em conformidade com um conjunto diferente de regras. Os identificadores que representam objetos do dicionário devem estar em conformidade com as regras de identificador do SQL-92, sendo portanto chamados de [IdentificadorSQL92](#).

Regras para identificadores SQL92

Identificadores comuns são identificadores não envoltos por aspas. Os identificadores delimitados são identificadores envoltos por aspas.

Um identificador comum deve começar por uma letra, e conter apenas letras, caracteres de sublinhado (_) e dígitos. As letras e dígitos permitidos incluem todas as letras e dígitos do Unicode, mas o Derby não tenta garantir que os caracteres do identificador sejam válidos no idioma do banco de dados.

Um identificador delimitado pode conter qualquer caractere entre as aspas. As aspas envoltórias não fazem parte do identificador; servem apenas para marcar seu começo e seu fim. Os espaços no final do identificador delimitado não são significativos (são truncados). O Derby traduz duas aspas consecutivas no interior do identificador delimitado como uma aspa, ou seja, as duas aspas "traduzidas" se tornam um caractere do identificador delimitado.

Os pontos dentro do identificador delimitado não são separadores, são parte do identificador (o nome do objeto do dicionário sendo representado).

Portanto, no exemplo a seguir:

```
"A.B"
```

é um objeto do dicionário, enquanto

```
"A"."B"
```

é um objeto do dicionário qualificado por outro objeto do dicionário (como uma coluna chamada "B" pertencente à tabela "A").

IdentificadorSQL92

Um *IdentificadorSQL92* é um identificador de objeto do dicionário em conformidade com as regras do SQL-92. O padrão SQL-92 declara que os identificadores dos objetos do dicionário estão limitados a 128 caracteres, e não são sensíveis a letras maiúsculas e minúsculas (a menos que estejam delimitados por aspas), porque são traduzidos automaticamente para letras maiúsculas pelo sistema. Não é permitido utilizar palavras reservadas como identificadores para os objetos do dicionário, a menos que estejam delimitadas. Se for tentado utilizar um nome com mais de 128 caracteres, será lançada a *SQLException* X0X11.

O Derby define palavras chave além das especificadas pelo padrão SQL-92 (consulte [Palavras reservadas do SQL](#)).

Exemplo

```
-- o nome da visão é armazenado nos
-- catálogos do sistema como UMIDENTIFICADOR

CREATE VIEW UmIdentificador (RECEBIDO) AS VALUES 1

-- o nome da visão é armazenado nos
-- catálogos do sistema intacto

CREATE VIEW "UmIdentificadorComMaiúsculasEMinúsculas" (RECEBIDO) AS
VALUES 1
```

Esta seção descreve as regras para utilizar *IdentificadorSQL92* para representar os objetos do dicionário que se seguem.

Qualificação dos objetos do dicionário

Uma vez que alguns objetos do dicionário podem estar contidos em outros objetos, os nomes destes objetos do dicionário podem ser qualificados. Cada componente é separado do componente seguinte por um ponto. O *IdentificadorSQL92* é "separado-por-ponto". O nome do objeto do dicionário é qualificado para evitar ambiguidade.

nome-da-coluna

Na sintaxe do SQL, em muitos lugares o nome da coluna pode ser representado qualificado pelo *nome-da-tabela* ou pelo *nome-da-correlação*.

Em algumas situações, não se pode qualificar o *nome-da-coluna* com o *nome-da-tabela* ou com o *nome-da-correlação*. Em vez disso, é utilizado um *nome-de-coluna-simples*. Estas situações são:

- criação da tabela ([Instrução CREATE TABLE](#))
- especificação das colunas atualizáveis no cursor
- no nome de correlação da coluna em uma expressão SELECT (consulte [ExpressãoSeleção](#))
- no nome de correlação da coluna em uma *ExpressãoTabela* (consulte [ExpressãoTabela](#))

Não pode ser utilizado o nome-da-correlação em colunas atualizáveis; a utilização do nome-da-correlação desta maneira causa exceção SQL. Por exemplo:

```
SELECT C11 AS COL1, C12 AS COL2, C13 FROM T1 FOR UPDATE OF C11, C13
```

Neste exemplo, o nome-da-correlação COL1 para C11 não é permitido, porque C11 aparece na lista de colunas FOR UPDATE. Pode ser utilizado o nome-da-correlação COL2 para C12, porque C12 não aparece na lista de colunas de FOR UPDATE.

Sintaxe

```
[ { nome-da-tabela | nome-da-correlação } . ] IdentificadorSQL92
```

Exemplo

```
-- P.PAÍS é o nome da coluna qualificado pelo nome-da-correlação.

SELECT P.PAÍS
FROM APP.PAÍSES P
```

nome-da-correlação

O *nome-da-correlação* é atribuído à expressão de tabela na cláusula FROM como sendo o novo nome, ou aliás, para a tabela. O *nome-da-correlação* não é qualificado pelo

nome-do-esquema.

Não é permitido utilizar o nome-da-correlação em colunas atualizáveis; a utilização do nome-da-correlação desta maneira causa uma exceção SQL. Por exemplo:

```
SELECT C11 AS COL1, C12 AS COL2, C13 FROM T1 FOR UPDATE OF C11, C13
```

Neste exemplo, o nome-da-correlação COL1 para C11 não é permitido, porque C11 aparece na lista de colunas de FOR UPDATE. Pode ser utilizado o nome-da-correlação COL2 para C12, porque C12 não aparece na lista de colunas de FOR UPDATE.

Sintaxe

[*IdentificadorSQL92*](#)

Exemplo

```
-- F é o nome-da-correlação
SELECT F.NOME
FROM SAMP.FUNCIONÁRIOS F
```

novο-nome-da-tabela

O *novο-nome-da-tabela* representa uma tabela renomeada. Não é permitido qualificar o *novο-nome-da-tabela* com o *nome-do-esquema*.

Sintaxe

[*IdentificadorSQL92*](#)

Exemplo

```
-- VÔOS_DISPONÍVEIS é o novο-nome-da-tabela que não inclui o
nome-do-esquema
RENAME TABLE VÔOS_DISPONIBILIDADE TO VÔOS_DISPONÍVEIS
```

nome-do-esquema

O *nome-do-esquema* representa o *esquema*. Os esquemas contêm outros objetos do dicionário, como tabelas e índices. Os esquemas fornecem uma maneira de dar nomes a um subconjunto de tabelas e outros objetos do dicionário no banco de dados.

Os esquemas podem ser criados e removidos explicitamente. O esquema de usuário padrão é o esquema *APP* (se não for especificado nenhum nome de usuário no momento da conexão). Não é permitido criar objetos nos esquemas que começam por *SYS*.

Portanto, as referências a tabelas podem ser qualificadas pelo nome do esquema. Quando o nome-do-esquema não é especificado, o nome de esquema padrão é inserido implicitamente. As tabelas do sistema são colocadas no esquema *SYS*. Todas as referências a tabelas do sistema devem ser qualificadas pelo identificador de esquema *SYS*. Para obter mais informações sobre as tabelas do sistema deve ser consultado [Tabelas do sistema Derby](#).

O esquema é o objeto de dicionário no nível mais alto da hierarquia, portanto o nome-do-esquema não pode ser qualificado.

Sintaxe

IdentificadorSQL92

Exemplo

```
-- RH.FUNCIONÁRIOS é o nome-da-tabela qualificado pelo nome-do-esquema
SELECT COUNT(*) FROM RH.FUNCIONÁRIOS

-- Os nomes dos catálogos do sistema devem ser qualificados pelo seu
esquema, SYS
SELECT COUNT(*) FROM SYS.SYSCOLUMNS
```

nome-de-coluna-simples

O *nome-de-coluna-simples* é utilizado para representar a coluna quando seu nome não pode ser qualificado pelo *nome-da-tabela* ou pelo *nome-da-correlação*. Este é o caso quando a qualificação é fixa, como na definição da coluna na instrução CREATE TABLE.

Sintaxe

IdentificadorSQL92

Exemplo

```
-- NOME_PAÍS é um nome-de-coluna-simples
CREATE TABLE CONTINENTE (
    NOME_PAÍS    VARCHAR(26) NOT NULL PRIMARY KEY,
    COD_ISO_PAÍS CHAR(2),
    REGIÃO_PAÍS  VARCHAR(26))
```

nome-do-sinônimo

O *nome-do-sinônimo* representa um sinônimo para a tabela ou a visão. O *nome-do-sinônimo* pode ser qualificado pelo *nome-do-esquema*.

Sintaxe

[*nome-do-esquema*.] *IdentificadorSQL92*

nome-da-tabela

O *nome-da-tabela* representa a tabela. O *nome-da-tabela* pode ser qualificado pelo *nome-do-esquema*.

Sintaxe

[*nome-do-esquema*.] *IdentificadorSQL92*

Exemplo

```
-- ENG.PROJETO é um nome de tabela que inclui o nome-do-esquema
SELECT COUNT(*) FROM ENG.PROJETO
```

nome-da-visão

O *nome-da-visão* representa uma tabela ou uma visão. O *nome-da-visão* pode ser

qualificado pelo *nome-do-esquema*.

Sintaxe

```
[ nome-do-esquema. ] IdentificadorSQL92
```

Exemplo

```
-- Esta é uma visão qualificada pelo nome-do-esquema
SELECT COUNT(*) FROM RH.EMP_CURRICULUM
```

nome-do-índice

O *nome-do-índice* representa um índice. Os índices residem em esquemas, portanto seus nomes podem ser qualificados pelo *nome-do-esquema*. Os índices das tabelas do sistema residem no esquema SYS.

Sintaxe

```
[ nome-do-esquema . ] IdentificadorSQL92
```

Exemplo

```
DROP INDEX APP.ÍNDICE_ORIGEM;
-- ÍNDICE_ORIGEM é o nome-do-índice sem o nome-do-esquema
CREATE INDEX ÍNDICE_ORIGEM ON VÔOS (AEROPORTO_ORIGEM)
```

nome-da-restrição

Os nomes das restrições não podem ser qualificados.

Sintaxe

```
IdentificadorSQL92
```

Exemplo

```
-- FK2_PAÍS é o nome da restrição
CREATE TABLE MAPAS_DETALHADOS (
    COD_ISO_PAÍS CHAR(2)
    CONSTRAINT FK2_PAÍS REFERENCES PAÍSES
)
```

nome-do-cursor

O *nome-do-cursor* faz referência a um cursor. Não existe nenhum comando na linguagem SQL para *atribuir* nome a um cursor. Em vez disso, deve ser utilizada a API do JDBC para atribuir nomes a cursors ou obter os nomes gerados pelo sistema. Para obter mais informações deve ser consultado o *Guia do Desenvolvedor do Derby*. Se for atribuído nome a um cursor, poderá ser feita referência a este nome nas instruções SQL.

O *nome-do-cursor* não pode ser qualificado.

Sintaxe

```
IdentificadorSQL92
```

Exemplo

```
stmt.executeUpdate("UPDATE SAMP.EQUIPES SET COMISS = " +
"COMISS + 20 " + "WHERE CURRENT OF " + ResultSet.getCursorName());
```

nome-do-gatilho

O *nome-do-gatilho* faz referência a um gatilho criado pelo usuário.

Sintaxe

```
[ nome-do-esquema . ] IdentificadorSQL92
```

Exemplo

```
DROP TRIGGER GATILHO1
```

identificador-de-autorização

Os nomes de usuário no sistema Derby são conhecidos por *identificadores de autorização*. O identificador de autorização representa o nome do usuário, caso tenha sido fornecido o nome do usuário na requisição de conexão. O esquema padrão para o usuário é idêntico ao identificador de autorização. No sistema de autenticação pode haver diferença entre letras maiúsculas e minúsculas nos nomes dos usuários, mas não há diferença entre letras maiúsculas e minúsculas no sistema de autorização do Derby, a não ser que o nome esteja delimitado. Para obter mais informações deve ser consultado o *Guia do Desenvolvedor do Derby*.

Sintaxe

```
IdentificadorSQL92
```

Exemplo

```
CALL SYCS_UTIL.SYCS_SET_DATABASE_PROPERTY(
'derby.database.fullAccessUsers', 'Amber,FRED')
```

Instruções

Esta seção fornece páginas de manual tanto para as construções de alto nível da linguagem, como para partes desta. Por exemplo, a instrução CREATE INDEX é uma instrução de alto nível que pode ser executada diretamente através da interface de JDBC. Esta seção também inclui as cláusulas, que não são instruções de alto nível, e não podem ser executadas diretamente, mas apenas como parte das instruções de alto nível. As cláusulas ORDER BY e WHERE são exemplos deste tipo de cláusula. Por fim, esta seção também inclui algumas partes de instruções sintaticamente complexas chamada de expressões como, por exemplo, [ExpressãoSeleção](#) e [SubconsultaTabela](#). As cláusulas e expressões possuem suas próprias páginas de manual para facilitar a referência.

A menos que esteja explicitamente declarado o contrário, as instruções de alto nível, todas assinaladas com a palavra *instrução*, podem ser executadas, ou preparadas e executadas, através das interfaces fornecidas pelo JDBC. Este manual indica se a expressão pode ser executada como uma instrução de alto nível.

As seções fornecem informações gerais sobre a utilização de instruções, e descrições de instruções específicas.

Interação com o sistema de dependências

O Derby acompanha internamente as dependências das instruções preparadas, que são instruções SQL pré-compiladas antes de serem executadas. São tipicamente preparadas (pré-compiladas) uma vez, e executadas várias vezes.

As instruções preparadas dependem de objetos do dicionário, e de instruções referenciadas pelas mesmas (Os objetos do dicionário incluem tabelas, colunas, restrições, índices, visões e gatilhos). A remoção ou modificação de objetos do dicionário ou de instruções que a instrução preparada depende a invalida internamente, significando que o Derby vai tentar recompilar automaticamente a instrução quando esta for executada. Se a recompilação da instrução não for bem-sucedida, o pedido de execução falhará. Entretanto, se for tomada alguma ação para restaurar a dependência quebrada (como a restauração da tabela que falta), a instrução preparada poderá ser executada, porque o Derby irá recompilá-la automaticamente no próximo pedido de execução.

As instruções dependem umas das outras; uma instrução UPDATE WHERE CURRENT depende da instrução referenciada. Remover a instrução da qual esta depende, invalida a instrução UPDATE WHERE CURRENT.

Além disso, as instruções preparadas não permitem a execução de certas instruções de DDL quando existem conjuntos de resultados abertos para as mesmas.

As páginas do manual de cada instrução detalham quais ações invalidam a instrução, caso esteja preparada.

Abaixo segue um exemplo utilizando a ferramenta ij do Derby:

```
ij> CREATE TABLE MINHA_TABELA (MINHA_COLUNA INT);
0 rows inserted/updated/deleted
ij> INSERT INTO MINHA_TABELA VALUES (1), (2), (3);
3 rows inserted/updated/deleted
-- este exemplo utiliza o comando prepare do ij,
-- que prepara a instrução
ij> prepare p1 AS 'INSERT INTO MINHA_TABELA VALUES (4)';
-- p1 depende de MINHA_TABELA;
ij> execute p1;
1 row inserted/updated/deleted
-- O Derby executa sem recompilar
ij> CREATE INDEX i1 ON MINHA_TABELA(MINHA_COLUNA);
0 rows inserted/updated/deleted
-- p1 está temporariamente inválido por causa do novo índice
ij> execute p1;
1 row inserted/updated/deleted
-- O Derby recompila automaticamente e executa p1
ij> DROP TABLE MINHA_TABELA;
0 rows inserted/updated/deleted
-- O Derby permite remover a tabela
-- porque o conjunto de resultados de p1 está fechado,
-- entretanto a instrução p1 está temporariamente inválida
ij> CREATE TABLE MINHA_TABELA (MINHA_COLUNA INT);
0 rows inserted/updated/deleted
ij> INSERT INTO MINHA_TABELA VALUES (1), (2), (3);
3 rows inserted/updated/deleted
ij> execute p1;
1 row inserted/updated/deleted
-- Como p1 está inválida, o Derby tenta recompilar
-- antes de executar.
-- É bem-sucedido e executa.
ij> DROP TABLE MINHA_TABELA;
0 rows inserted/updated/deleted
-- a instrução p1 agora está inválida,
-- e desta vez a tentativa de recompilar
-- na hora de executar vai falhar
ij> execute p1;
ERROR 42X05: Table/View 'MINHA_TABELA' does not exist.
```

Instrução ALTER TABLE

A instrução ALTER TABLE permite:

- adicionar coluna à tabela

- adicionar restrição à tabela
- remover da tabela uma restrição existente
- aumentar o comprimento de coluna VARCHAR, CHAR VARYING e CHARACTER VARYING
- sobrepor o bloqueio no nível de linha para a tabela (ou remover a sobreposição)

Sintaxe

```
ALTER TABLE nome-da-tabela
{
    ADD COLUMN definição-da-coluna |
    ADD CONSTRAINT cláusula |
    DROP { PRIMARY KEY | FOREIGN KEY nome-da-restrição | UNIQUE
nome-da-restrição |
        CHECK nome-da-restrição | CONSTRAINT nome-da-restrição }
    ALTER alteração-da-coluna |
    LOCKSIZE { ROW | TABLE }
}
```

definição-da-coluna

```
nome-de-coluna-simplestipo-de-dado
[ restrição no nível-de-coluna ]*
[ [ WITH ] DEFAULT {ExpressãoConstante | NULL } ]
```

alteração-da-coluna

```
nome-da-coluna SET DATA TYPE VARCHAR(inteiro) |
nome-da-coluna SET INCREMENT BY constante-inteira
```

Na alteração-da-coluna, SET INCREMENT BY constante-inteira especifica o intervalo entre valores consecutivos da coluna de identidade. O próximo valor a ser gerado para a coluna de identidade será determinado a partir do último valor atribuído e o incremento a ser aplicado. A coluna deve ter sido definida com o atributo IDENTITY.

ALTER TABLE não afeta nenhuma visão que faz referência à tabela sendo alterada. Isto inclui as visões que possuem "*" na lista do SELECT. Estas visões devem ser removidas e recriadas para que retornem a nova coluna.

Adição de colunas

A sintaxe para a *definição-da-coluna* de uma nova coluna é a mesma que para a coluna na instrução CREATE TABLE. Isto significa que pode ser colocada uma restrição de coluna para a nova coluna na instrução ALTER TABLE ADD COLUMN. Entretanto, somente pode ser adicionada uma coluna com a restrição NOT NULL a uma tabela existente se for fornecido um valor padrão; caso contrário, é lançada uma exceção ao ser executada a instrução ALTER TABLE.

Assim como em CREATE TABLE, se a definição da coluna incluir uma restrição de unicidade ou de chave primária, a coluna não poderá conter valores nulos e, portanto, também deve ser especificado o atributo NOT NULL (SQLState 42831).

Note: Se a tabela possuir um gatilho de UPDATE sem uma lista de colunas explícita, adicionar uma coluna à tabela adicionará esta coluna à lista de colunas de atualização implícita sobre a qual o gatilho está definido, e todas as referências a variáveis de transição serão invalidadas para que incluam a nova coluna.

Adição de restrições

ALTER TABLE ADD CONSTRAINT adiciona restrição no nível-de-tabela a uma tabela existente. Pode ser adicionado através de ALTER TABLE qualquer tipo de restrição no nível-de-tabela suportado. Existem as seguintes limitações para adicionar restrição a uma tabela existente:

- Ao adicionar uma restrição de chave estrangeira ou de verificação a uma tabela existente, o Derby verifica a tabela para ter certeza que as linhas existentes

satisfazem a restrição. Se alguma linha for inválida, o Derby lançará uma exceção de instrução, e a restrição não será adicionada.

- Todas as colunas incluídas na chave primária devem conter dados não nulos e serem únicas.

ALTER TABLE ADD UNIQUE e PRIMARY KEY dispõem de um método resumido para definir uma chave primária formada por uma única coluna. Se for especificado PRIMARY KEY na definição da coluna C, o efeito será o mesmo de especificar a cláusula PRIMARY KEY(C) como uma cláusula em separado. A coluna não pode conter valores nulos, portanto o atributo NOT NULL também deve ser especificado.

Para obter informações sobre a sintaxe das restrições deve ser consultada a [Cláusula CONSTRAINT](#). Ao se adicionar restrições com a sintaxe ALTER TABLE ADD CONSTRAINT, deve ser utilizada a sintaxe para restrição no nível-de-tabela.

Remoção de restrições

ALTER TABLE DROP CONSTRAINT remove uma restrição de uma tabela existente. Para remover uma restrição sem nome, deve ser especificado o nome gerado para a restrição armazenado em SYS.SYSCONSTRAINTS na forma de um identificador delimitado.

A remoção da restrição de chave primária, de unicidade ou de chave estrangeira remove o índice físico que impõe a restrição (também conhecido por *índice de apoio*).

Modificação de colunas

A [alteração-da-coluna](#) permite alterar a coluna especificada das seguintes maneiras:

- Aumentar o comprimento de uma coluna VARCHAR existente. Pode ser utilizado CHARACTER VARYING ou CHAR VARYING como sinônimo para a palavra chave VARCHAR.

Para aumentar o comprimento de uma coluna deste tipo, deve ser especificado o tipo de dado e o novo tamanho após o nome da coluna.

Não é permitido diminuir o tamanho ou mudar o tipo de dado. Não é permitido aumentar o comprimento de uma coluna que faz parte de uma chave primária ou chave de unicidade referenciada por uma restrição de chave estrangeira, ou que seja parte de uma restrição de chave estrangeira.

- Especificação do intervalo entre valores consecutivos da coluna de identidade.

Para definir o intervalo entre valores consecutivos da coluna de identidade deve ser especificada uma constante-inteira. A coluna deve ter sido definida anteriormente com o atributo IDENTITY (SQLState 42837). Caso existam linhas na tabela, os valores na coluna para a qual o padrão para SET INCREMENT foi adicionado não mudam.

Definição de padrão

Pode ser especificado um valor padrão para a nova coluna. O valor padrão é o valor inserido na coluna se não for especificado nenhum outro valor. Quando não é especificado explicitamente, o valor padrão da coluna é NULL. Se for especificado valor padrão para uma nova coluna, as linhas existentes na tabela receberão o valor padrão na nova coluna.

Para obter mais informações sobre valor padrão deve ser consultada a [Instrução CREATE TABLE](#).

Alteração da granularidade do bloqueio da tabela

A cláusula LOCKSIZE permite sobrepor o bloqueio no nível-de-linha para uma

determinada tabela, se o sistema utilizar a definição padrão de bloqueio no nível-de-linha (Se o sistema estiver definido com bloqueio no nível-de-tabela, não será possível alterar a granularidade do bloqueio, embora o Derby permita utilizar a cláusula LOCKSIZE nesta situação sem lançar uma exceção). Para sobrepor o bloqueio no nível-de-linha de uma determinada tabela, deve ser definido o bloqueio da tabela como TABLE. Se a tabela tiver sido criada com a granularidade de bloqueio no nível-de-tabela, o bloqueio poderá passar a ser no nível-de-linha especificando ROW na cláusula LOCKSIZE da instrução ALTER TABLE. Para obter informações sobre porque algumas vezes esta alteração é útil, deve ser consultado o *Ajuste do Derby*.

Exemplos

```
-- Adicionar uma nova coluna com restrição no
-- nível-de-coluna a uma tabela existente.
-- Se a tabela tiver alguma linha será lançada uma exceção,
-- uma vez que a nova coluna será inicializada com NULL
-- em todas as linhas da tabela.

ALTER TABLE CIDADES ADD COLUMN REGIÃO VARCHAR(26)
CONSTRAINT NOVA_RESTRIÇÃO CHECK (REGIÃO IS NOT NULL);

-- Adicionar uma restrição de unicidade a uma tabela existente.
-- Será lançada uma exceção se forem encontradas chaves duplicadas.

ALTER TABLE SAMP.DEPARTAMENTOS
ADD CONSTRAINT NOVA_UNICIDADE UNIQUE (NUM_DEP);

-- Adicionar uma restrição de chave estrangeira à
-- tabela cidades. Cada linha de cidades é verificada
-- para ter certeza que satisfaz a restrição.
-- Se alguma linha não satisfizer a restrição, a
-- restrição não será adicionada

ALTER TABLE CIDADES ADD CONSTRAINT FK_PAÍSES
FOREIGN KEY (PAÍS) REFERENCES PAÍSES (PAÍS);

-- Adicionar uma restrição de chave primária à tabela.
-- Primeiro, criar a tabela.
CREATE TABLE ATIVIDADES (ID_CIDADE INT NOT NULL,
ESTAÇÃO CHAR(2), ATIVIDADE VARCHAR(32) NOT NULL);

-- Não será possível adicionar esta restrição se as
-- colunas que compõem a chave primária possuírem
-- dados nulos ou valores duplicados.
ALTER TABLE ATIVIDADES ADD PRIMARY KEY (ID_CIDADE, ATIVIDADE);

-- Remover a restrição de chave primária da tabela cidades.
ALTER TABLE CIDADES DROP CONSTRAINT PK_CIDADES;

-- Remover a restrição de chave estrangeira da tabela cidades.
ALTER TABLE CIDADES DROP CONSTRAINT FK_PAÍSES;

-- Adicionar a coluna NUM_DEP com valor padrão igual a 1.
ALTER TABLE SAMP.ATIV_EMP ADD COLUMN NUM_DEP INT DEFAULT 1;

-- Aumentar o comprimento da coluna VARCHAR.
ALTER TABLE SAMP.EMP_FOTO ALTER FORMATO_FOTO SET DATA TYPE VARCHAR(30);

-- Alterar a granularidade do bloqueio da tabela.
ALTER TABLE SAMP.VENDAS LOCKSIZE TABLE;
```

Resultados

A instrução ALTER TABLE faz com que todas as instruções que dependem da tabela sendo alterada sejam recompiladas antes de sua próxima execução. A instrução ALTER TABLE não é permitida caso existam cursores abertos fazendo referência à tabela sendo alterada.

Instruções CREATE

As instruções CREATE são utilizadas com funções, índices, procedimentos, esquemas, sinônimos, tabelas, gatilhos e visões.

Instrução CREATE FUNCTION

A instrução CREATE FUNCTION permite criar funções Java que podem ser utilizadas em expressões.

Sintaxe

```
CREATE FUNCTION nome-da-função ( [ parâmetro-da-função
[ , parâmetro-da-função ] ] * ) RETURNS tipo-de-dado [
elemento-da-função ] *
```

nome-da-função

```
[ nome-do-esquema . ] IdentificadorSQL92
```

Se o nome-do-esquema não for fornecido, o esquema corrente se tornará o esquema padrão. Se for especificado um nome de procedimento qualificado, o nome do esquema não poderá começar por SYS.

parâmetro-da-função

```
[ nome_do_parâmetro ] tipo-de-dado
```

O nome_do_parâmetro deve ser único na função.

A sintaxe do *tipo-de-dado* está descrita em [Tipos de dado](#).

Note: Os tipos-de-dado longos, como LONG VARCHAR, LONG VARCHAR FOR BIT DATA, CLOB e BLOB, não são permitidos como parâmetros da instrução CREATE FUNCTION.

elemento-da-função

```
{
  LANGUAGE { JAVA }
  EXTERNAL NAME cadeia-de-caracteres
  PARAMETER STYLE JAVA
  { NO SQL | CONTAINS SQL | READS SQL DATA }
  { RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT }
}
```

LANGUAGE

JAVA- o gerenciador de banco de dados chama a função como um método estático público de uma classe Java.

EXTERNAL NAME *cadeia-de-caracteres*

A *cadeia-de-caracteres* descreve o método Java a ser chamado quando a função for executada, e possui a seguinte forma:

```
nome-da-classe.nome-do-método
```

O EXTERNAL NAME (nome externo) não pode conter espaços indevidos.

PARAMETER STYLE

JAVA - A função utiliza uma convenção de passagem de parâmetros em conformidade com a linguagem Java e com a especificação de rotinas SQL. Os parâmetros INOUT e OUT são passados como matrizes de uma única entrada para facilitar o retorno de valores. Os conjuntos de resultados são retornados através de parâmetros adicionais para um método Java do tipo java.sql.ResultSet[] passados como matrizes de uma única entrada.

O Derby não suporta tipos de coluna longos (por exemplo, LONG VARCHAR, BLOB, etc.). Ocasionalmente tenta utilizar um destes tipos de coluna longos.

NO SQL, CONTAINS SQL, READS SQL DATA

Indica se a função emite instruções SQL e, se o fizer, de que tipo.

CONTAINS SQL

Indica que podem ser executadas pela função instruções SQL que não lêem nem modificam dados SQL. As instruções não suportadas por qualquer função retornam um erro diferente.

NO SQL

Indica que a função não pode executar nenhuma instrução SQL.

READS SQL DATA

Indica que podem ser incluídas na função instruções SQL que não modificam dados SQL. As instruções não suportadas por qualquer função retornam um erro diferente. Este é o valor padrão.

RETURNS NULL ON NULL INPUT ou CALLED ON NULL INPUT

Especifica se a função será chamada quando algum de seus argumentos de entrada for nulo.

RETURNS NULL ON NULL INPUT

Especifica que a função não será chamada quando o valor de algum de seus argumentos de entrada for nulo. O resultado será o valor nulo.

CALLED ON NULL INPUT

Especifica que a função será chamada quando o valor de algum de seus argumentos de entrada for nulo. Esta especificação significa que a função deve ser codificada para testar nulos nos valores dos argumentos. A função poderá retornar um valor nulo, ou não nulo. Esta é a definição padrão.

Os elementos da função podem estar em qualquer ordem, mas cada tipo de elemento só pode aparecer uma vez. A definição da função deve conter estes elementos:

- **LANGUAGE**
- **PARAMETER STYLE**
- **EXTERNAL NAME**

Exemplo

```
CREATE FUNCTION rad_graus(radianos DOUBLE) RETURNS DOUBLE  
PARAMETER STYLE JAVA NO SQL LANGUAGE JAVA  
EXTERNAL NAME 'java.lang.Math.toDegrees'
```

Instrução CREATE INDEX

A instrução CREATE INDEX cria um índice em uma tabela. Os índices podem incluir uma ou mais colunas da tabela.

Sintaxe

```
CREATE [UNIQUE] INDEX nome-do-índice  
ON nome-da-tabela ( nome-de-coluna-simples [ ASC | DESC ]  
[ , nome-de-coluna-simples [ ASC | DESC ] ] * )
```

No Derby, o número máximo de colunas para chave do índice é 16.

O nome do índice não pode ter mais que 128 caracteres.

O nome da coluna não pode aparecer mais de uma vez na mesma instrução CREATE INDEX. Entretanto, índices diferentes podem incluir a mesma coluna.

O Derby pode utilizar os índices para melhorar o desempenho das instruções de manipulação de dados (consulte o *Ajuste do Derby*). Além disso, os índices UNIQUE fornecem uma maneira de verificar a integridade dos dados.

Os nomes dos índices são únicos no esquema (Alguns sistemas de banco de dados

permitem que tabelas diferentes no mesmo esquema possuam índices com o mesmo nome, mas o Derby não permite). É assumido que a tabela e o índice estão no mesmo esquema, se o nome do esquema for especificado para um dos nomes, mas não para o outro. Se o nome do esquema for especificado tanto para o índice quanto para a tabela, será lançada uma exceção se os nomes dos esquemas não forem o mesmo. Se não for especificado o nome do esquema nem para a tabela e nem para o índice, será utilizado o esquema corrente.

O Derby utiliza, por padrão, a ordem ascendente de cada coluna para criar o índice. Especificar ASC após o nome da coluna não modifica o comportamento padrão. A palavra chave DESC após o nome da coluna faz com que o Derby utilize a ordem descendente da coluna para criar o índice. Utilizar a ordem descendente para uma coluna pode ajudar a melhorar o desempenho dos comandos que requerem resultados em uma ordem de classificação mista ou na ordem descendente, e para os comandos que selecionam o valor mínimo ou máximo de uma coluna indexada.

Se for especificado um nome de índice qualificado, o nome do esquema não poderá começar por SYS.

Índices e restrições

As restrições de unicidade, de chave primária, e de chave estrangeira, geram índices que impõem, ou "apoiam", a restrição (por isso, algumas vezes são chamados de *índices de apoio*). Se uma coluna, ou conjunto de colunas, tiver uma restrição UNIQUE ou PRIMARY KEY aplicada, não será permitido criar índice com estas colunas. O Derby já terá criado um índice com nome gerado pelo sistema. Os nomes gerados pelo sistema para os índices que apoiam as restrições são facilmente encontrados consultando as tabelas do sistema, se for especificado o nome da restrição. Por exemplo, para descobrir o nome do índice que apoia a restrição PK_VÔOS:

```
SELECT CONGLOMERATENAME FROM SYS.SYSCONGLOMERATES,
SYS.SYSCONSTRAINTS WHERE
SYS.SYSCONGLOMERATES.TABLEID = SYSCONSTRAINTS.TABLEID
AND CONSTRAINTNAME = 'PK_VÔOS'
```

```
CREATE INDEX ÍNDICE_ORIGEM ON VÔOS(AEROPORTO_ORIGEM);

-- valores monetários são geralmente ordenados do maior para o menor,
-- portanto o índice é criado na ordem descendente

CREATE INDEX PAG_DESC ON SAMP.EMPREGADOS (SALÁRIO);

-- utilizar um tamanho de página maior para o índice

CALL
SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY('derby.storage.pageSize','8192');
CREATE INDEX IDX_VENDAS ON SAMP.VENDAS (VENDAS);
CALL
SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY('derby.storage.pageSize',NULL);
```

Tamanho da página e comprimento da chave

Note: Em um índice, o tamanho das colunas chave deve ser igual ou menor que a metade do tamanho da página. Se o comprimento das colunas chave, de uma linha existente na tabela, for maior que a metade do tamanho da página do índice, a criação do índice para a tabela com estas colunas chave falhará. Este erro somente ocorre ao criar o índice, quando uma linha existente na tabela não respeita este critério. Após o índice ser criado, as inserções falham se o tamanho da chave associada não respeitar este critério.

Sistema de dependência de instruções

As instruções preparadas envolvendo SELECT, INSERT, UPDATE, UPDATE WHERE CURRENT, DELETE e DELETE WHERE CURRENT na tabela referenciada pela

instrução CREATE INDEX são invalidadas quando o índice é criado. Os cursores abertos nas tabelas não são afetados.

Instrução CREATE PROCEDURE

A instrução CREATE PROCEDURE permite criar procedimentos armazenados em Java, que podem ser chamados utilizando a instrução CALL PROCEDURE.

Sintaxe

```
CREATE PROCEDURE nome-do-procedimento ( [ parâmetro-do-procedimento
[ , parâmetro-do-procedimento ] * )
[ elemento-do-procedimento ] *
```

nome-do-procedimento

```
[ nome-do-esquema . ] IdentificadorSQL92
```

Se o nome-do-esquema não for fornecido, o esquema corrente será o esquema padrão. Se for especificado um nome de procedimento qualificado, o nome do esquema não poderá começar por SYS.

parâmetro-do-procedimento

```
[ { IN | OUT | INOUT } ] [ nome-do-parâmetro ] tipo-de-dado
```

O valor padrão para o parâmetro é IN. O nome-do-parâmetro deve ser único no procedimento.

A sintaxe do *tipo-de-dado* está descrita em [Tipos de dado](#).

Note: Os tipos-de-dado longos, como LONG VARCHAR, LONG VARCHAR FOR BIT DATA, CLOB e BLOB, não podem ser usados como parâmetros nas instruções CREATE PROCEDURE.

elemento-do-procedimento

```
{
[ DYNAMIC ] RESULT SETS INTEGER
LANGUAGE { JAVA }
EXTERNAL NAME cadeia-de-caracteres
PARAMETER STYLE JAVA
{ NO SQL | MODIFIES SQL DATA | CONTAINS SQL | READS SQL DATA }
}
```

DYNAMIC RESULT SETS *integer*

Indica o limite superior estimado de conjuntos de resultados retornados pelo procedimento. O padrão é sem (zero) conjuntos de resultados dinâmicos.

LANGUAGE

JAVA - o gerenciador de banco de dados chama o procedimento como um método estático público de uma classe Java.

EXTERNAL NAME *cadeia-de-caracteres*

A *cadeia-de-caracteres* descreve o método Java a ser chamado quando o procedimento for executado, e possui a seguinte forma:

```
nome_da_classe.nome_do_método
```

O nome externo não pode conter espaços.

PARAMETER STYLE

JAVA - O procedimento utiliza uma convenção de passagem de parâmetros em conformidade com a linguagem Java e com a especificação de rotinas SQL. Os parâmetros INOUT e OUT são passados como matrizes de uma única entrada para facilitar o retorno de valores. Os conjuntos de resultados são retornados através de parâmetros adicionais para um método Java do tipo `java.sql.ResultSet []` que são passados como matrizes de uma única entrada.

O Derby não suporta tipos de coluna longos (por exemplo, LONG VARCHAR, BLOB, etc.). Ocasionalmente tenta utilizar um destes tipos de coluna longos.

NO SQL, CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA

Indica se o procedimento armazenado emite instruções SQL e, se o fizer, de que tipo.

CONTAINS SQL

Indica que podem ser executadas pelo procedimento armazenado instruções SQL que não lêem nem modificam dados SQL. As instruções não suportadas por qualquer procedimento armazenado retornam um erro diferente. O valor padrão é MODIFIES SQL DATA.

NO SQL

Indica que o procedimento armazenado não pode executar nenhuma instrução SQL.

READS SQL DATA

Indica que podem ser incluídas no procedimento armazenado algumas instruções SQL que não modificam dados SQL. As instruções não suportadas por qualquer procedimento armazenado retornam um erro diferente.

MODIFIES SQL DATA

Indica que o procedimento armazenado pode executar qualquer instrução SQL, exceto as instruções não suportadas por procedimentos armazenados.

Os elementos do procedimento podem estar em qualquer ordem, mas cada tipo de elemento pode aparecer somente uma única vez. A definição do procedimento deve conter os seguintes elementos:

- **LANGUAGE**
- **PARAMETER STYLE**
- **EXTERNAL NAME**

Exemplo

```
CREATE PROCEDURE VENDAS.RENDIMENTO_TOTAL(IN V_MÊS INTEGER,  
IN V_ANO INTEGER, OUT TOTAL DECIMAL(10,2))  
PARAMETER STYLE JAVA READS SQL DATA LANGUAGE JAVA EXTERNAL NAME  
'com.acme.vendas.calculaRendimentoPorMes'
```

Instrução CREATE SCHEMA

O esquema é uma forma de agrupar logicamente objetos em uma única coleção, e fornecer um espaço de nomes único para os objetos.

Sintaxe

```
CREATE SCHEMA nome-do-esquema
```

A instrução CREATE SCHEMA é utilizada para criar esquemas. O nome do esquema não pode possuir mais de 128 caracteres. Os nomes dos esquemas devem ser únicos no banco de dados.

```
-- Criar um esquema para as tabelas relacionadas com os empregados  
CREATE SCHEMA EMP;  
  
-- Criar um esquema para as tabelas relacionadas com as linhas aéreas  
CREATE SCHEMA VÔOS
```

```
-- Criar uma tabela chamada "disponibilidade" em cada esquema

CREATE TABLE VÔOS.DISPONIBILIDADE (
    ID_VÔO          CHAR(6) NOT NULL,
    NUM_SEGMENTO    INT NOT NULL,
    DATA_VÔO       DATE NOT NULL,
    ASSENTOS_ECONÔMICOS_OCUPADOS INT,
    ASSENTOS_EXECUTIVOS_OCUPADOS INT,
    ASSENTOS_PRIMEIRA_CLASSE_OCUPADOS INT,
    CONSTRAINT PK_VÔOS_DISPONIBILIDADE
    PRIMARY KEY (ID_VÔO, NUM_SEGMENTO, DATA_VÔO));

CREATE TABLE EMP.DISPONIBILIDADE (
    ID_HOTEL          INT NOT NULL,
    DATA_RESERVA     DATE NOT NULL,
    QUARTOS_OCUPADOS INT,
    CONSTRAINT PK_HOTEL_DISPONIBILIDADE PRIMARY KEY (ID_HOTEL,
    DATA_RESERVA));
```

Instrução CREATE SYNONYM

A instrução CREATE SYNONYM é utilizada para fornecer um nome alternativo para uma tabela ou visão presente no mesmo esquema ou em outro esquema. Também podem ser criados sinônimos para outros sinônimos, resultando em sinônimos aninhados. O sinônimo pode ser utilizado no lugar do nome qualificado original da tabela ou da visão, nas instruções SELECT, INSERT, UPDATE, DELETE e LOCK TABLE. Pode ser criado um sinônimo para uma tabela ou uma visão que não existe, mas a tabela ou visão de destino deverá existir antes do sinônimo poder ser utilizado.

Os sinônimos compartilham o mesmo espaço de nomes das tabelas e visões. Não pode ser criado um sinônimo com o mesmo nome de uma tabela que já existe no mesmo esquema. De maneira semelhante, não pode ser criada uma tabela ou visão com nome correspondente a um sinônimo já existente.

O sinônimo pode ser definido para uma tabela ou visão que não existe quando o sinônimo é criado. Se a tabela ou visão não existir, será recebida uma mensagem de advertência (SQLState 01522). O objeto referenciado deverá existir quando o sinônimo for utilizado em uma instrução da DML.

Pode ser criado sinônimo aninhado (um sinônimo para outro sinônimo), mas qualquer tentativa de criar um sinônimo que resulte em uma referência circular retorna uma mensagem de erro (SQLState 42916).

Não podem ser definidos sinônimos nos esquemas do sistema. Todos os esquemas começando por 'SYS' são considerados esquemas do sistema, e são reservados pelo Derby.

Não pode ser definido um sinônimo para uma tabela temporária. A tentativa de definir um sinônimo para uma tabela temporária retorna uma mensagem de erro (SQLState XCL51).

Sintaxe

```
CREATE SYNONYM nome-do-sinônimo FOR { nome-da-visão | nome-da-tabela }
```

Na instrução, **nome-do-sinônimo** representa o nome do sinônimo sendo atribuído à tabela ou visão de destino, enquanto **nome-da-visão** e **nome-da-tabela** representam o nome original da tabela ou visão de destino.

Exemplo

```
CREATE SYNONYM SAMP.T1 FOR SAMP.TABELA_COM_NOME_COMPRIDO
```

Instrução CREATE TABLE

A instrução CREATE TABLE cria uma tabela. As tabelas contêm colunas e restrições, que são regras que os dados devem estar em conformidade. A restrição no

nível-de-tabela especifica uma coluna ou várias colunas. As colunas possuem um tipo de dado e podem especificar restrições de coluna (restrições no nível-de-coluna).

Para obter informações sobre as restrições deve ser consultada a [Cláusula CONSTRAINT](#).

Pode ser especificado um valor padrão para a coluna. O valor padrão é o valor a ser inserido na coluna quando não é especificado nenhum outro valor. Quando não é especificado explicitamente, o valor padrão para a coluna é NULL. Consulte [Valor padrão da coluna](#).

Podem ser especificadas propriedades de armazenamento, como o tamanho da página para a tabela, chamando o procedimento do sistema SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY.

Se for especificado um nome de tabela qualificado, o nome do esquema não poderá começar por SYS.

Sintaxe

```
CREATE TABLE nome-da-tabela
( { definição-da-coluna | restrição no nível-de-tabela }
[ , { definição-da-coluna | restrição no nível-de-tabela } ] * )
```

Exemplos

```
CREATE TABLE DISPONIBILIDADE_HOTEL (
    ID_HOTEL          INT NOT NULL,
    DATA_RESERVA     DATE NOT NULL,
    QUARTOS_RESERVADOS INT DEFAULT 0,
    PRIMARY KEY (ID_HOTEL, DATA_RESERVA));

-- A definição de chave primária no nível-de-tabela permite
-- incluir duas colunas na definição da chave primária
PRIMARY KEY (ID_HOTEL, DATA_RESERVA))

-- Atribuir um atributo de coluna de identidade a uma coluna INTEGER,
-- e também definir uma restrição de chave primária na coluna

CREATE TABLE PESSOAS (
    ID_PESSOA INT NOT NULL GENERATED ALWAYS AS IDENTITY
    CONSTRAINT PK_PESSOAS PRIMARY KEY,
    PESSOA VARCHAR(26));

-- Atribuir um atributo de coluna de identidade a uma coluna SMALLINT
-- com valor inicial igual a 5 e valor do incremento igual a 5.

CREATE TABLE GRUPOS (
    ID_GRUPO SMALLINT NOT NULL
    GENERATED ALWAYS AS IDENTITY (START WITH 5, INCREMENT BY 5),
    ENDEREÇO VARCHAR(100),
    TEL VARCHAR(15));
```

Note: Para obter mais exemplos de instruções CREATE TABLE utilizando várias restrições deve ser consultada a [Cláusula CONSTRAINT](#).

definição-da-coluna:

```
nome-de-coluna-simple tipo-de-dado
[ restrição no nível-de-coluna ] *
[ [ WITH ] DEFAULT { expressão-constante | NULL }
  | especificação-de-coluna-gerada ]
[ restrição no nível-de-coluna ] *
```

A sintaxe do *tipo-de-dado* está descrita em [Tipos de dado](#).

As sintaxes de *restrição no nível-de-coluna* e *restrição no nível-de-tabela* estão descritas na [Cláusula CONSTRAINT](#).

Valor padrão da coluna

Na definição do valor padrão, a *expressão-constante* é uma expressão que não faz referência a nenhuma tabela. Pode incluir constantes, registros especiais de data e hora, esquemas correntes, usuários e nulo.

especificação-de-coluna-gerada:

```
[ GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY
[ ( START WITH ConstanteInteira
[ , INCREMENT BY ConstanteInteira ] ) ] ] ]
```

Atributos da coluna de identidade

Nas colunas SMALLINT, INT e BIGINT com atributo de identidade, o Derby atribui automaticamente valores inteiros incrementados para a coluna. Os atributos da coluna de identidade se comportam como os outros valores padrão, ou seja, quando a instrução de inserção não especifica o valor para a coluna, o Derby fornece automaticamente o valor. Entretanto, o valor não é uma constante; o Derby incrementa automaticamente o valor padrão na hora da inserção.

A palavra chave IDENTITY somente pode ser especificada quando o tipo de dado associado à coluna for um dos seguintes tipos inteiros exatos.

- SMALLINT
- INT
- BIGINT

Existem dois tipos de coluna de identidade no Derby: aquelas que são sempre geradas (GENERATED ALWAYS), e aquelas que são geradas por padrão (GENERATED BY DEFAULT).

GENERATED ALWAYS

Uma coluna de identidade que é sempre gerada, incrementa o valor padrão em todas as inserções e armazena o valor incrementado na coluna. Ao contrário dos outros valores padrão, não é possível inserir diretamente um valor ou atualizar uma coluna de identidade que é sempre gerada. Por isso, deve ser especificada a palavra chave DEFAULT ao fazer inserção na coluna de identidade, ou não incluir a coluna de identidade na lista de colunas na inserção. Por exemplo:

```
CREATE TABLE SAUDAÇÕES (
  I INT GENERATED ALWAYS AS IDENTITY,
  CH CHAR(50));
INSERT INTO SAUDAÇÕES VALUES (DEFAULT, 'alô');
INSERT INTO SAUDAÇÕES(CH) VALUES ('bom dia');
```

Os valores gerados automaticamente na coluna de identidade GENERATED ALWAYS são únicos. A criação de uma coluna de identidade não cria um índice para a coluna.

GENERATED BY DEFAULT

Uma coluna de identidade gerada por padrão, somente incrementa e utiliza o valor padrão nas inserções quando não é fornecido nenhum valor explícito. Ao contrário das colunas sempre geradas, pode ser especificado um valor na instrução de inserção a ser utilizado no lugar do valor padrão gerado.

Para utilizar o valor gerado, deve ser especificada a palavra chave DEFAULT ao inserir na coluna de identidade, ou não incluir a coluna de identidade na lista de colunas da inserção. Para especificar um valor, este deve ser incluído na instrução de inserção. Por exemplo:

```
CREATE TABLE SAUDAÇÕES (
  I INT GENERATED BY DEFAULT AS IDENTITY,
  CH CHAR(50));
-- especificar o valor "1":
INSERT INTO SAUDAÇÕES VALUES (1, 'olá');
-- usar o padrão gerado
INSERT INTO SAUDAÇÕES VALUES (DEFAULT, 'salut');
-- usar o padrão gerado
```

```
INSERT INTO SAUDAÇÕES(CH) VALUES ('bonjour');
```

Deve ser observado que ao contrário das colunas GENERATED ALWAYS, as colunas GENERATED BY DEFAULT não garantem unicidade. Portanto, no exemplo acima as linhas `olá` e `salut` possuem o valor de identidade igual a "1", porque a coluna gerada começa por "1" e o valor especificado pelo usuário também é "1". Para não permitir duplicidade, especialmente ao carregar ou importar dados, a tabela deve ser criada utilizando um valor para START WITH correspondente ao primeiro valor de identidade que o sistema deve atribuir. Para verificar esta condição e não permiti-la, pode ser utilizada uma chave primária ou restrição de unicidade na coluna de identidade GENERATED BY DEFAULT.

Por padrão, o valor inicial da coluna de identidade é 1, e o valor do incremento é 1. Podem ser especificados valores diferentes do padrão para o valor inicial e para o incremento ao definir a coluna utilizando as palavras chave STARTS WITH e INCREMENT BY. Se for especificado um número negativo para o valor do incremento, o Derby *decrementará* o valor a cada inserção. Se o valor for 0, ou positivo, o Derby incrementará o valor a cada inserção.

Os valores máximos e mínimos permitidos nas colunas de identidade são determinados pelo tipo de dado da coluna. A tentativa de inserir um valor fora da faixa de valores suportados pelo tipo de dado lança uma exceção.

Tabela 1. Valores máximos e mínimos para colunas com especificação de coluna gerada

Tipo de dado	Valor máximo	Valor mínimo
SMALLINT	32.767 (<i>java.lang.Short.MAX_VALUE</i>)	-32.768 (<i>java.lang.Short.MIN_VALUE</i>)
INT	2.147.483.647 (<i>java.lang.Integer.MAX_VALUE</i>)	-2.147.483.648 (<i>java.lang.Integer.MIN_VALUE</i>)
BIGINT	9.223.372.036.854.775.807 (<i>java.lang.Long.MAX_VALUE</i>)	-9.223.372.036.854.775.808 (<i>java.lang.Long.MIN_VALUE</i>)

Os valores gerados automaticamente na coluna de identidade são únicos. Deve ser utilizada uma restrição de chave primária ou de unicidade para garantir a unicidade. A criação da coluna de identidade *não* cria um índice para a coluna.

A função `IDENTITY_VAL_LOCAL` é uma função não determinística que retorna o valor atribuído mais recentemente a uma coluna de identidade. Consulte [IDENTITY_VAL_LOCAL](#) para obter mais informações.

Note: Deve ser especificado o nome do esquema, da tabela e da coluna utilizando letras maiúsculas e minúsculas conforme os nomes estejam armazenados nas tabelas do sistema -- ou seja, todas as letras em maiúsculo, a menos que tenham sido utilizados identificadores delimitados para criar estes objetos no banco de dados.

O Derby mantém o último valor incrementado para a coluna no cache. Também armazena o próximo valor incrementado para a coluna no disco na coluna `AUTOINCREMENTVALUE` da tabela do sistema `SYS.SYSCOLUMNS`. Desfazer a transação não desfaz este valor, portanto transações desfeitas podem deixar "espaços" nos valores inseridos automaticamente na coluna de identidade. O Derby se comporta desta maneira para evitar bloqueio na linha de `SYS.SYSCOLUMNS` pela duração da transação, e para manter a simultaneidade alta.

Quando acontece uma inserção em uma instrução-SQL-engatilhada, o valor inserido pela instrução-SQL-engatilhada na coluna de identidade é disponível a partir de `ConnectionInfo` somente dentro do código do gatilho. O código do gatilho também pode ver o valor inserido pela instrução que fez o gatilho disparar. Entretanto, a instrução que fez o gatilho disparar não pode ver o valor inserido pela instrução-SQL-engatilhada na

coluna de identidade. Da mesma maneira, os gatilhos podem ser aninhados (ou recursivos). Uma instrução SQL pode fazer o gatilho T1 disparar. T1, por sua vez, executa uma instrução SQL que faz o gatilho T2 disparar. Se tanto T1 quanto T2 inserem linhas em uma tabela que faz o Derby inserir em uma coluna de identidade, o gatilho T1 não poderá ver o valor produzido pela inserção feita por T2, mas T2 poderá ver o valor produzido pela inserção feita por T1. Cada nível de aninhamento pode ver os valores incrementados gerados por si próprio e pelos níveis de aninhamento anteriores, por todo o percurso até a instrução SQL de nível superior que deu início aos gatilhos recursivos. Somente podem existir 16 níveis de recursividade de gatilho.

Exemplo

```
CREATE TABLE SAUDAÇÕES (
  I INT GENERATED BY DEFAULT AS IDENTITY (START WITH 2, INCREMENT BY
1),
  CH CHAR(50));
-- especificar o valor "1":
INSERT INTO SAUDAÇÕES VALUES (1, 'olá');
-- usar o valor gerado
INSERT INTO SAUDAÇÕES VALUES (DEFAULT, 'salut');
-- usar o valor gerado
INSERT INTO SAUDAÇÕES(CH) VALUES ('bonjour');
```

Instrução CREATE TRIGGER

O gatilho define um conjunto de ações a serem executadas quando ocorre um evento de banco de dados em uma determinada tabela. O *evento de banco de dados* é uma operação de exclusão, inserção ou de atualização. Por exemplo, se for definido um gatilho para exclusão em uma determinada tabela, a ação do gatilho ocorre sempre que se remove uma ou mais linhas da tabela.

Junto com as restrições, os gatilhos podem ajudar a impor regras de integridade com ações como exclusões ou atualizações em cascata. Os gatilhos também podem realizar várias funções como emitir alertas, atualizar outras tabelas, enviar e-mail, e outras ações úteis.

Pode ser definido qualquer número de gatilhos para uma única tabela, inclusive vários gatilhos para a mesma tabela para o mesmo evento.

Pode ser criado gatilho em qualquer esquema, exceto os começados por SYS. O gatilho não precisa residir no mesmo esquema da tabela para a qual é definido.

Se for especificado um nome de gatilho qualificado, o nome do esquema não poderá começar por SYS.

Sintaxe

```
CREATE TRIGGER nome-do-gatilho
{ AFTER | NO CASCADE BEFORE }
{ INSERT | DELETE | UPDATE } [ OF nome-da-coluna [, nome-da-coluna]* ]
ON nome-da-tabela
[ CláusulaReferência ]
FOR EACH { ROW | STATEMENT } MODE DB2SQL
Instrução-SQL-engatilhada
```

Antes ou depois: quando os gatilhos disparam

Os gatilhos são definidos para *BEFORE* (antes) ou *AFTER* (depois).

- Os gatilhos *BEFORE* disparam antes das modificações da instrução serem aplicadas, e antes de qualquer restrição ser aplicada. Os gatilhos para antes podem ser tanto de linha quanto de instrução (consulte [Gatilhos de instrução versus gatilhos de linha](#)).
- Os gatilhos *AFTER* disparam após todas as restrições terem sido satisfeitas, e após todas as alterações terem sido aplicadas à tabela de destino. Os gatilhos *AFTER* podem ser tanto de linha quanto de instrução (consulte [Gatilhos de instrução versus gatilhos de linha](#)).

Inserção, exclusão e atualização: o que faz o gatilho disparar

O gatilho é disparado por um dos seguintes eventos do banco de dados, dependendo de como foi definido (consulte [Sintaxe](#) acima):

- INSERT
- UPDATE
- DELETE

Pode ser definido qualquer número de gatilhos para um determinado evento em uma determinada tabela. No caso de gatilho para atualização podem ser especificadas as colunas.

Referência a valores antigos e novos: a cláusula de referência

Muitas instruções-SQL-engatilhadas necessitam fazer referência aos dados que estão sendo alterados no momento pelo evento de banco de dados que causou o disparo do gatilho. A instrução-SQL-engatilhada pode necessitar fazer referência aos valores novos (pós-alteração ou "após").

O Derby fornece várias maneiras para fazer referência aos dados que estão sendo alterados no momento pelo evento do banco de dados que fez o gatilho disparar. Os dados alterados podem ser referenciados pela instrução-SQL-engatilhada utilizando *variáveis de transição* ou *tabelas de transição*. A cláusula de referência permite o fornecimento do nome da correlação ou aliás para estas variáveis de transição especificando OLD/NEW AS *nome-da-correlação*.

Por exemplo, se for adicionada a seguinte cláusula à definição do gatilho:

```
REFERENCING OLD AS LINHA_APAGADA
```

pode-se então fazer referência a este nome de correlação na instrução-SQL-engatilhada:

```
DELETE FROM DISPONIBILIDADE_HOTEL WHERE ID_HOTEL = LINHA_APAGADA.ID_HOTEL
```

As variáveis de transição OLD e NEW são mapeadas a um *java.sql.ResultSet* de uma única linha.

Note: Somente os gatilhos de linha (consulte [Gatilhos de instrução versus gatilhos de linha](#)) podem utilizar variáveis de transição. Os gatilhos de linha para INSERT não podem fazer referência à linha antiga. Os gatilhos de linha para DELETE não podem fazer referência à linha nova.

Para os gatilhos de instrução, as *tabelas* de transição servem como identificador de tabela para a instrução-SQL-engatilhada ou para a qualificação do gatilho. A cláusula de referência permite fornecer um nome de correlação ou aliás para estas tabelas de transição especificando OLD_TABLE/NEW_TABLE AS *nome-da-correlação*

Por exemplo:

```
REFERENCING OLD_TABLE AS HOTÉIS_EXCLUÍDOS
```

permite utilizar este novo identificador (*HOTÉIS_EXCLUÍDOS*) na instrução-SQL-engatilhada:

```
DELETE FROM DISPONIBILIDADE_HOTEL WHERE ID_HOTEL IN  
(SELECT ID_HOTEL FROM HOTÉIS_EXCLUÍDOS)
```

As tabelas de transição antiga e nova são mapeadas a um *java.sql.ResultSet* com cardinalidade equivalente ao número de linhas afetadas pelo evento do gatilho.

Note: Somente os gatilhos de instrução (consulte [Gatilhos de instrução versus gatilhos de linha](#)) podem utilizar tabelas de transição. Os gatilhos para a instrução INSERT não podem fazer referência à tabela OLD. Os gatilhos para a instrução DELETE não podem fazer referência à tabela NEW.

A cláusula REFERENCING pode designar apenas uma correlação ou identificador para novo, e apenas uma correlação ou identificador para antigo. Os gatilhos de linha não podem designar um identificador para a tabela de transição, e os gatilhos de instrução não podem designar uma correlação para as variáveis de transição.

Gatilhos de instrução versus gatilhos de linha

Deve ser especificado se o gatilho é um *gatilho de instrução* ou um *gatilho de linha*:

- *gatilho de instrução*

O gatilho de instrução dispara uma vez por evento de gatilho, independentemente de alguma linha ser modificada pelo evento de inserção, atualização ou exclusão.

- *gatilho de linha*

O gatilho de linha dispara uma vez para cada linha afetada pelo evento de gatilho. Se nenhuma linha for afetada, o gatilho não vai disparar.

Note: Uma atualização que define o valor da coluna como o valor originalmente contido (por exemplo, UPDATE T SET C = C) faz com que o gatilho de linha dispare, embora o valor da coluna seja o mesmo que era antes do evento de gatilho.

Instrução-SQL-engatilhada

A ação definida pelo gatilho é chamada de instrução-SQL-engatilhada (na [Sintaxe](#) acima, veja a última linha). Possui as seguintes limitações:

- Não pode conter parâmetros dinâmicos (?).
- Não pode criar, alterar ou remover a tabela para a qual o gatilho está definido.
- Não pode adicionar ou remover um índice na tabela para a qual o gatilho está definido.
- Não pode adicionar ou remover um gatilho na tabela para a qual o gatilho está definido.
- Não pode efetivar ou desfazer a transação corrente, ou mudar o nível de isolamento.
- Não pode executar a instrução CALL.
- Os gatilhos para antes não podem possuir instruções INSERT, UPDATE ou DELETE como sua ação.

A instrução-SQL-engatilhada pode fazer referência a outros objetos do banco de dados além da tabela para a qual o gatilho é declarado. Se algum destes objetos for removido, o gatilho será invalidado. Se na próxima execução a recompilação do gatilho não for bem-sucedida, a chamada lançará uma exceção e a instrução que causou o disparo será desfeita.

Para obter mais informações sobre instrução-SQL-engatilhada deve ser consultado o *Guia do Desenvolvedor do Derby*.

Ordem de execução

Quando ocorre um evento de banco de dados que dispara um gatilho, o Derby realiza ações nesta ordem:

- Dispara os gatilhos *NO CASCADE BEFORE*.
- Realiza a verificação das restrições (verifica chave primária, chave única e chave estrangeira).
- Realiza a inserção, atualização ou exclusão.
- Dispara os gatilhos *AFTER*.

Quando são definidos vários gatilhos para o mesmo evento de banco de dados, para a mesma tabela, e para o mesmo instante (antes ou depois), os gatilhos são disparados na ordem em que foram criados.

```
-- Instruções e gatilhos:
```

```

CREATE TRIGGER T1 NO CASCADE BEFORE UPDATE ON X
  FOR EACH ROW MODE DB2SQL
    values app.notifyEmail('Jerry', 'A tabela x está para ser atualizada');

CREATE TRIGGER EXCLUI VÔOS
  AFTER DELETE ON VÔOS
  REFERENCING OLD_TABLE AS VÔOS_EXCLUÍDOS
  FOR EACH STATEMENT MODE DB2SQL
  DELETE FROM VÔOS_DISPONIBILIDADE WHERE ID_VÔO IN
    (SELECT ID_VÔO FROM VÔOS_EXCLUÍDOS);

CREATE TRIGGER EXCLUI VÔOS3
  AFTER DELETE ON VÔOS
  REFERENCING OLD AS OLD
  FOR EACH ROW MODE DB2SQL
  DELETE FROM VÔOS_DISPONIBILIDADE WHERE ID_VÔO = OLD.ID_VÔO;

```

Note: Podem ser encontrados mais exemplos no *Guia do Desenvolvedor do Derby*.

Recursividade de gatilho

A profundidade de recursividade máxima de gatilho é 16.

Informações relacionadas

As funções de sistema especiais que retornam informação sobre a hora corrente ou o usuário corrente são avaliadas quando o gatilho dispara, e não quando o gatilho é criado. Estas funções incluem:

- [CURRENT_DATE](#)
- [CURRENT_TIME](#)
- [CURRENT_TIMESTAMP](#)
- [CURRENT_USER](#)
- [SESSION_USER](#)
- [USER](#)

CláusulaReferência:

```

REFERENCING
{
  OLD | NEW } [ AS ] nome-da-correlação [ { OLD | NEW } [ AS ]
  nome-da-correlação ] |
{
  OLD_TABLE | NEW_TABLE } [ AS ] Identificador [ { OLD_TABLE | NEW_TABLE
  [AS] Identificador ]
}

```

Instrução CREATE VIEW

As visões são tabelas virtuais formadas por uma consulta. A visão é um objeto do dicionário que pode ser utilizado, até ser removido.

As visões não são atualizáveis.

Se for especificado um nome de visão qualificado, o nome do esquema não poderá começar por SYS.

Sintaxe

```

CREATE VIEW nome-da-visão
  [ ( nome-de-coluna-simples [, nome-de-coluna-simples] * ) ]
AS Consulta

```

A definição da visão pode conter uma lista de colunas da visão, opcional, para atribuir nomes explícitos às colunas da visão. Se não houver uma lista de colunas, a visão herdar os nomes de coluna da consulta subjacente. Todas as colunas da visão devem possuir nomes únicos.

```

CREATE VIEW SAMP.V1 (COL_SUM, COL_DIFF)
  AS SELECT COMISS + BÔNUS, COMISS - BÔNUS

```

```

        FROM SAMP.EMPREGADOS;

CREATE VIEW SAMP.V_EMP_CURR (CURRICULUM)
AS VALUES 'Delores M. Quintana', 'Heather A. Nicholls', 'Bruce
Adamson';

CREATE VIEW SAMP.PROJ_COMBO
(NUM_PROJ, PROJ_DATA_FIM, PROJ_EQUIPE, PROJ_CHEFE)
AS SELECT NUM_PROJ, PROJ_DATA_FIM, PROJ_EQUIPE, PROJ_CHEFE
FROM SAMP.PROJETO
UNION ALL
SELECT NUM_PROJ, EMSTDAT, EMPTIME, NUM_EMP
FROM SAMP.ATIV_EMP
WHERE NUM_EMP IS NOT NULL;

```

Sistema de dependência de instrução

As definições das visões são dependentes das tabelas e visões referenciadas na definição da visão. As instruções de DML (linguagem de manipulação de dados) que contêm referência a visões dependem destas visões, assim como dos objetos nas definições das visões dos quais as visões dependem. As instruções que fazem referência à visão dependem dos índices que as visões utilizam; o índice utilizado pela visão pode mudar de uma instrução para outra, baseado em como a consulta é otimizada. Por exemplo, dado:

```

CREATE TABLE T1 (C1 DOUBLE PRECISION);

CREATE FUNCTION SIN (DATA DOUBLE)
RETURNS DOUBLE EXTERNAL NAME 'java.lang.Math.sin'
LANGUAGE JAVA PARAMETER STYLE JAVA;

CREATE VIEW V1 (C1) AS SELECT SIN(C1) FROM T1;

```

então a instrução a seguir:

```
SELECT * FROM V1
```

é dependente da visão *V1*, da tabela *T1*, e da função escalar externa *SIN*.

Instruções DROP

As instruções DROP (remover) são utilizadas com funções, índices, procedimentos, esquemas, sinônimos, tabelas, gatilhos e visões.

Instrução DROP FUNCTION

Sintaxe

```
DROP FUNCTION nome-da-função
```

Identifica uma determinada função a ser removida, sendo válida somente se existir exatamente uma instância da função com o *nome-da-função* no esquema. A função identificada pode ter qualquer número de parâmetros definidos na mesma. Se não houver nenhuma função com o nome indicado no esquema nomeado ou implícito, ocorrerá um erro (SQLState 42704). Também ocorrerá um erro quando existir mais de uma instância da função especificada no esquema nomeado ou implícito.

Instrução DROP INDEX

A instrução DROP INDEX remove o índice especificado.

Sintaxe

```
DROP INDEX nome-do-índice
```

```
DROP INDEX INDICE_ORIGEM
```

```
DROP INDEX INDICE_DESTINO
```

Sistema de dependência de instrução

Se houver um cursor aberto na tabela de onde o índice será removido, a instrução DROP INDEX irá gerar um erro e não removerá o índice. Se não houver, as instruções que dependem da tabela do índice serão invalidadas.

Instrução DROP PROCEDURE

Sintaxe

```
DROP PROCEDURE nome-do-procedimento
```

Identifica um determinado procedimento a ser removido, sendo válida somente se houver exatamente uma instância do procedimento com o *nome-do-procedimento* no esquema. O procedimento identificado pode ter qualquer número de parâmetros definidos no mesmo. Se não houver nenhum procedimento com o nome indicado no esquema nomeado ou implícito, ocorrerá um erro (SQLState 42704). Também ocorrerá um erro quando existir mais de uma instância do procedimento especificado no esquema nomeado ou implícito.

Instrução DROP SCHEMA

A instrução DROP SCHEMA remove um esquema. O esquema de destino deve estar vazio para a remoção ser bem-sucedida.

Nem o esquema *APP* (o esquema padrão do usuário), e nem o esquema *SYS*, podem ser removidos.

Sintaxe

```
DROP SCHEMA nome-do-esquema RESTRICT
```

A palavra chave RESTRICT impõe a regra que, para o esquema poder ser removido do banco de dados, não pode existir nenhum objeto definido no esquema especificado. A palavra chave RESTRICT é requerida.

```
-- Remover o esquema SAMP
-- O esquema SAMP somente poderá ser removido do banco de dados
-- se não existir nenhum objeto definido no esquema SAMP.
```

```
DROP SCHEMA SAMP RESTRICT
```

Instrução DROP SYNONYM

Remove o sinônimo especificado da tabela ou da visão.

Sintaxe

```
DROP SYNONYM nome-do-sinônimo
```

Instrução DROP TABLE

A instrução DROP TABLE remove a tabela especificada.

Sintaxe

```
DROP TABLE nome-da-tabela
```

Sistema de dependência de instrução

Os gatilhos, as restrições (chave primária, unicidade, verificação e referencial da tabela sendo removida) e os índices da tabela são removidos em silêncio. A existência de um cursor aberto fazendo referência à tabela sendo removida faz com que a instrução DROP TABLE gere um erro, e a tabela não seja removida.

A remoção da tabela invalida as instruções que dependem da tabela (Invalidar uma instrução faz com que esta seja recompilada na próxima execução. Consulte a [Interação com o sistema de dependências](#).)

Instrução DROP TRIGGER

A instrução DROP TRIGGER remove o gatilho especificado.

Sintaxe

```
DROP TRIGGER nome-do-gatilho
```

```
DROP TRIGGER GAT1
```

Sistema de dependência de instrução

Quando uma tabela é removida, todos os gatilhos da tabela são removidos automaticamente (Não há necessidade de remover os gatilhos da tabela antes de remover a tabela).

Instrução DROP VIEW

Remove a visão especificada.

Sintaxe

```
DROP VIEW nome-da-visão
```

```
DROP VIEW um_identificador
```

Sistema de dependência de instrução

Todas as instruções que fazem referência à visão são invalidadas pela instrução DROP VIEW. A instrução DROP VIEW não é permitida quando existem visões ou cursores abertos dependentes da visão. A visão deve ser removida antes que qualquer objeto do qual seja dependente possa ser removido.

Instruções RENAME

As instruções RENAME são utilizadas com índices e tabelas.

Instrução RENAME INDEX

Esta instrução permite mudar o nome de um índice no esquema corrente. Os usuários não podem mudar os nomes dos índices do esquema SYS.

Sintaxe

```
RENAME INDEX nome-do-índice TO novo-nome-do-índice
```

```
RENAME INDEX ÍNDICE_DESTINO TO ÍNDICE_CHEGADA
```

Sistema de dependência de instrução

A instrução RENAME INDEX não é permitida quando há cursores abertos fazendo referência ao índice sendo renomeado.

Instrução RENAME TABLE

A instrução RENAME TABLE permite renomear uma tabela existente em qualquer esquema (exceto o esquema SYS).

Sintaxe

```
RENAME TABLE nome-da-tabela TO novο-nome-da-tabela
```

Se houver uma visão ou uma chave estrangeira fazendo referência à tabela, a tentativa de renomear a tabela vai gerar um erro. Além disso, caso exista alguma restrição de verificação ou gatilho na tabela, a tentativa de renomeá-la vai gerar um erro.

```
RENAME TABLE SAMP.ATIV_EMP TO ATIVIDADES_EMPREGADOS
```

Para obter mais informações consulte também a [Instrução ALTER TABLE](#).

Sistema de dependência de instrução

Mesmo que exista um índice definido na tabela, a tabela pode ser renomeada.

A instrução RENAME TABLE não é permitida quando há cursores abertos fazendo referência à tabela sendo alterada.

Instruções SET

As instruções SET são utilizadas com esquemas e para definir o nível de isolamento corrente.

Instrução SET SCHEMA

A instrução SET SCHEMA define o esquema padrão para a sessão da conexão como sendo o esquema designado. O esquema padrão é utilizado como esquema de destino por todas as instruções emitidas pela conexão que não especificam explicitamente o nome do esquema.

Para a instrução SET SCHEMA ser bem-sucedida, o esquema de destino deve existir. Se o esquema não existir, retornará um erro. Consulte a [Instrução CREATE SCHEMA](#).

A instrução SET SCHEMA não é transacional: Se a instrução SET SCHEMA for parte de uma transação desfeita, a mudança de esquema permanecerá aplicada.

Sintaxe

```
SET [CURRENT] SCHEMA [=]
{ nome-do-esquema |
  USER | ? | '<constante-cadeia-de-caracteres>' } | SET CURRENT SQLID [=]
{ nome-do-esquema | USER | ? | '<constante-cadeia-de-caracteres>' }
```

O *nome-do-esquema* é um identificador com comprimento máximo igual a 128. Não diferencia letras maiúsculas de minúsculas, a menos que esteja entre aspas (Por exemplo, SYS é equivalente a sYS, SYs, sys, etc.).

USER é o usuário corrente. Se não houver usuário corrente definido, o esquema corrente padrão será o esquema APP (Se for especificado um nome de usuário na conexão, o nome de usuário será o esquema padrão para a conexão, caso exista um

esquema com este nome).

? é a especificação de um parâmetro dinâmico que pode ser utilizado em instruções preparadas. A instrução SET SCHEMA pode ser preparada uma vez e executada com valores de esquema diferentes. Os valores do esquema são tratados como constantes cadeia de caracteres, portanto diferenciando letras maiúsculas e minúsculas. Por exemplo, para designar o esquema *APP* deve ser utilizada a cadeia de caracteres "APP", em vez de "app".

```
-- os comandos a seguir são todos equivalentes,
-- e funcionam assumindo que exista um esquema chamado HOTEL
SET SCHEMA HOTEL
SET SCHEMA hotel
SET CURRENT SCHEMA hotel
SET CURRENT SQLID hotel
SET SCHEMA = hotel
SET CURRENT SCHEMA = hotel
SET CURRENT SQLID = hotel
SET SCHEMA "HOTEL" -- identificador entre aspas
SET SCHEMA 'HOTEL' -- cadeia de caracteres entre apóstrofos-- Este
exemplo produz um erro, porque
-- hotel em minúsculas não será encontrado
SET SCHEMA = 'hotel'

-- Este exemplo produz um erro, porque SQLID
-- não é permitido sem CURRENT
SET SQLID hotel

-- Este exemplo define o esquema como o ID do usuário corrente
SET CURRENT SCHEMA USER

// Abaixo está um exemplo da utilização de SET SCHEMA em um programa Java
PreparedStatement ps = conn.prepareStatement("set schema ?");
ps.setString(1,"HOTEL");
ps.executeUpdate();
... fazer alguma coisa
ps.setString(1,"APP");
ps.executeUpdate();

ps.setString(1,"app"); //erro - a cadeia de caracteres diferencia
// letras maiúsculas e minúsculas; app não será encontrado
ps.setNull(1, Types.VARCHAR); //erro - não é permitido nulo
```

Instrução SET CURRENT ISOLATION

A instrução SET CURRENT ISOLATION LEVEL permite que o usuário mude o nível de isolamento de sua conexão. Os níveis válidos são: SERIALIZABLE, REPEATABLE READ, READ COMMITTED e READ UNCOMMITTED.

A emissão deste comando efetiva a transação corrente, o que é consistente com o método *java.sql.Connection.setTransactionLevel*.

Para obter informações sobre os níveis de isolamento deve ser consultado "Bloqueio, Simultaneidade e Isolamento" no *Guia do Desenvolvedor do Derby*.

Sintaxe

```
SET [ CURRENT ] ISOLATION [ = ]
{
  UR | DIRTY READ | READ UNCOMMITTED
  CS | READ COMMITTED | CURSOR STABILITY
  RS |
  RR | REPEATABLE READ | SERIALIZABLE
  RESET
}
```

```
SET ISOLATION SERIALIZABLE;
```

CALL (PROCEDIMENTO)

A instrução CALL (PROCEDIMENTO) é utilizada para chamar procedimentos. A chamada de procedimento não retorna nenhum valor.

Sintaxe

```
CALL nome-do-procedimento ( [ ? [, ?]* ] )
```

Exemplo

```
CREATE PROCEDURE VENDAS.RENDIMENTO_TOTAL(IN V_MÊS INTEGER,
IN V_ANO INTEGER, OUT TOTAL DECIMAL(10,2))
PARAMETER STYLE JAVA READS SQL DATA LANGUAGE JAVA EXTERNAL NAME
'com.acme.vendas.calcularRendimentoTotal';
CALL VENDAS.RENDIMENTO_TOTAL(?,?,?);
```

Cláusula CONSTRAINT

A cláusula CONSTRAINT é uma parte opcional da [Instrução CREATE TABLE](#) e da [Instrução ALTER TABLE](#). A restrição é uma regra com a qual os dados devem estar em conformidade. O nome da restrição é opcional.

A restrição pode ser:

- restrição no nível-de-coluna

As restrições no nível-de-coluna fazem referência a uma única coluna da tabela, e não especificam o nome da coluna (exceto as restrições de verificação). Se referem à coluna a qual seguem.

- restrição no nível-de-tabela

As restrições no nível-de-tabela fazem referência a uma ou mais colunas da tabela. As restrições no nível-de-tabela especificam os nomes das colunas às quais se aplicam. As restrições de verificação (CHECK) no nível de tabela podem fazer referência a zero ou mais colunas da tabela.

As restrições de coluna incluem:

- NOT NULL

Especifica que a coluna não pode conter valores nulos (não pode ser dado nome a restrições deste tipo).

- PRIMARY KEY

Especifica a coluna que identifica unicamente uma linha da tabela. A coluna identificada deve ser definida como NOT NULL.

Note: Se for tentado adicionar uma chave primária utilizando a instrução ALTER TABLE, e alguma coluna incluída na chave primária contiver valores nulos, será gerado um erro e a chave primária não será adicionada. Para obter mais informações deve ser consultada a [Instrução ALTER TABLE](#).

- UNIQUE

Especifica que os valores na coluna devem ser únicos. Não são permitidos valores nulos.

- FOREIGN KEY

Especifica que os valores na coluna devem corresponder a valores em uma coluna de chave primária ou de chave única referenciada, ou que são nulos.

- CHECK

Especifica regras para os valores da coluna.

As restrições de tabela incluem:

- PRIMARY KEY

Especifica a coluna ou colunas que identificam unicamente uma linha da tabela. Não são permitidos valores nulos.

- UNIQUE

Especifica que os valores nas colunas devem ser únicos. As colunas identificadas devem ser definidas como NOT NULL.

- FOREIGN KEY

Especifica que os valores nas colunas devem corresponder a valores em colunas de uma chave primária ou chave única referenciada, ou que são nulos.

Note: Se a chave estrangeira for formada por várias colunas, e se *alguma* coluna for nula, toda a chave será considerada nula. A inserção será permitida não importando o que esteja presente nas colunas não-nulas.

- CHECK

Especifica diversas regras para os valores na tabela.

As restrições de coluna e restrições de tabela possuem a mesma função; a diferença é onde são especificadas. As restrições de tabela permitem especificar mais de uma coluna na definição da restrição PRIMARY KEY, UNIQUE, CHECK e FOREIGN KEY. As restrições no nível-de-coluna (exceto as restrições de verificação) fazem referência a apenas uma coluna.

Sintaxe

Restrições de chave primária e de unicidade

A chave primária define o conjunto de colunas que identificam unicamente as linhas da tabela.

Quando se cria uma restrição de chave primária, nenhuma das colunas incluídas na chave primária pode ter a restrição NULL; ou seja, não podem permitir valores nulos.

A instrução ALTER TABLE ADD PRIMARY KEY permite incluir colunas existentes em uma chave primária, caso estas tenham sido anteriormente definidas como NOT NULL. Os valores NULL não são permitidos. Se as colunas contiverem valores nulos, o sistema não adicionará a restrição de chave primária. Para obter mais informações deve ser consultada a [Instrução ALTER TABLE](#).

A tabela pode ter no máximo uma restrição PRIMARY KEY, mas pode ter várias restrições UNIQUE.

Restrições de chave estrangeira

As chaves estrangeiras fornecem um meio de impor a integridade referencial de um banco de dados. A chave estrangeira é uma coluna, ou grupo de colunas, dentro da tabela que fazem referência a uma chave de alguma outra tabela (ou algumas vezes, embora raramente, à mesma tabela). A chave estrangeira deve incluir sempre colunas cujos tipos correspondem exatamente aos tipos das colunas da restrição de chave primária ou de unicidade referenciada.

Em uma restrição de chave estrangeira no nível-de-tabela, para a qual são especificadas as colunas da tabela que compõem a restrição, a mesma coluna não pode ser usada mais de uma vez.

Se houver uma lista de colunas na *EspecificaçãoReferencias* (lista das colunas na tabela referenciada), esta lista deverá corresponder a uma restrição de chave primária ou a uma restrição de unicidade da tabela referenciada. A *EspecificaçãoReferencias* pode

omitir a lista de colunas da tabela referenciada, se esta tabela possuir uma chave primária declarada.

Se não houver uma lista de colunas na *EspecificaçãoReferencias*, e a tabela referenciada não possuir uma chave primária, será lançada uma exceção de instrução (Isto significa que se a tabela referenciada possuir apenas chaves únicas, é necessário incluir a lista de colunas na *EspecificaçãoReferencias*).

A restrição de chave estrangeira está satisfeita quando há um valor correspondente na coluna de unicidade ou de chave primária correspondente. Se a chave estrangeira for composta por várias colunas, o valor da chave estrangeira será considerado nulo quando qualquer uma de suas colunas tiver o valor nulo.

Note: É possível que uma chave estrangeira formada por várias colunas permita que uma de suas colunas contenha um valor para o qual não exista valor correspondente nas colunas referenciadas, de acordo com o padrão SQL-92. Para evitar esta situação, devem ser criadas restrições NOT NULL em todas as colunas da chave estrangeira.

Restrições de chave estrangeira e a DML

Quando é feita uma inserção ou atualização em uma tabela que possui uma restrição de chave estrangeira habilitada, o Derby verifica se a linha não viola a restrição de chave estrangeira procurando a chave referenciada correspondente na tabela referenciada. Se a restrição não for satisfeita, o Derby rejeitará a inserção ou a atualização através de uma exceção de instrução.

Quando uma linha de uma tabela com uma chave referenciada (uma restrição de chave primária ou de unicidade referenciada por uma chave estrangeira) é atualizada ou removida, o Derby verifica todas as restrições de chave estrangeira que fazem referência à chave, para ter certeza que a remoção ou a modificação da linha não causa violação da restrição. Se a remoção ou a modificação da linha causar uma violação da restrição, a atualização ou remoção não será permitida, e o Derby lançará uma exceção de instrução.

O Derby realiza as verificações de restrição no momento em que a instrução é executada, e não quando a transação é efetivada.

Índices de apoio

As restrições UNIQUE, PRIMARY KEY e FOREIGN KEY geram índices que impõem, ou "apoiam", à restrição (sendo algumas vezes chamados de *índices de apoio*). As restrições UNIQUE e PRIMARY KEY geram índices únicos. As restrições FOREIGN KEY geram índices que não são únicos. Portanto, se uma coluna, ou conjunto de colunas, estiverem envolvidas em uma restrição UNIQUE, PRIMARY KEY ou FOREIGN KEY, não é necessário criar índices nestas colunas para melhorar o desempenho, o Derby já terá criado o índice. Consulte [Índices e restrições](#).

Estes índices ficam disponíveis para o otimizador para a otimização de comandos (consulte a [Instrução CREATE INDEX](#)), e possuem nomes gerados pelo sistema.

Não é permitido remover um índice de apoio através da instrução DROP INDEX; é necessário remover a restrição ou a tabela.

Restrições de verificação

As restrições de verificação podem ser utilizadas para especificar diversas regras para o conteúdo da tabela. É especificada uma condição de procura (que é uma expressão booleana), para a restrição de verificação. Esta condição de procura deve ser satisfeita por todas as linhas da tabela. A condição de procura é aplicada a toda linha modificada por uma instrução INSERT ou UPDATE, na hora em que a linha é modificada. A instrução é interrompida por completo quando qualquer restrição de verificação é violada.

Requisitos da condição de procura

Se a restrição de verificação for especificada como parte da definição-da-coluna, a referência a coluna somente poderá ser feita para a mesma coluna. As restrições de verificação especificadas como parte da definição da tabela, podem possuir referência a colunas que identificam colunas definidas anteriormente na instrução CREATE TABLE.

A condição de procura deve retornar sempre o mesmo valor quando for aplicada aos mesmos valores. Portanto, não pode conter:

- Parâmetros dinâmicos (?)
- Funções de Data/Hora (CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP)
- Subconsultas
- Funções de usuário (tal como USER, SESSION_USER, CURRENT_USER)

Ações referenciais

Quando a chave estrangeira é definida, pode ser especificada uma cláusula ON DELETE e/ou ON UPDATE seguida pela ação apropriada (CASCADE, RESTRICT, SET NULL ou NO ACTION). Estas cláusulas especificam se o Derby deve modificar os valores da chave estrangeira correspondente ou não permitir a operação, para manter o relacionamento de chave estrangeira intacto quando o valor da chave primária for atualizado ou excluído da tabela.

A regra de atualização ou de exclusão da restrição referencial é especificada quando a restrição referencial é definida.

A regra de atualização é aplicada quando uma linha da tabela mãe ou da tabela dependente é atualizada. As escolhas são NO ACTION e RESTRICT.

Quando se atualiza o valor de uma coluna da chave primária da tabela mãe, e a regra de atualização está especificada como RESTRICT, o Derby verifica as tabelas dependentes com relação às restrições de chave estrangeira. Se alguma linha de tabela dependente violar a restrição de chave estrangeira, a transação será desfeita.

Se a regra de atualização for NO ACTION, o Derby verificará as tabelas dependentes com relação às restrições de chave estrangeira *após* todas as exclusões terem sido executadas, mas *antes* dos gatilhos serem executados. Se alguma linha de tabela dependente violar a restrição de chave estrangeira, a instrução será rejeitada.

Quando é atualizado o valor de uma coluna em uma tabela dependente, e este valor faz parte da chave estrangeira, a regra de atualização implícita é NO ACTION. NO ACTION significa que se a chave estrangeira for atualizada com um valor não-nulo, o valor atualizado deverá corresponder a um valor na chave primária da tabela mãe quando a instrução estiver completa. Se a atualização não corresponder a um valor na chave primária da tabela mãe, a instrução será rejeitada.

A regra de exclusão é aplicada quando uma linha da tabela mãe é excluída, e esta linha possui dependentes na tabela dependente da restrição referencial. Quando são excluídas linhas da tabela dependente, a operação de exclusão da tabela mãe é dita como *propagada* para a tabela dependente. Se a tabela dependente também for uma tabela mãe, a ação especificada será aplicada, por sua vez, às suas tabelas dependentes.

As escolhas são NO ACTION, RESTRICT, CASCADE e SET NULL. SET NULL somente poderá ser especificada quando alguma coluna da chave estrangeira permitir valores nulos.

Se a regra de exclusão for:

NO ACTION, o Derby verificará as tabelas dependentes com relação às restrições referenciais *após* todas as exclusões terem sido executadas, mas *antes* dos gatilhos serem executados. Se alguma linha da tabela dependente violar a restrição de chave estrangeira, a instrução será rejeitada.

RESTRICT, o Derby verificará as tabelas dependentes com relação às chaves estrangeiras. Se alguma linha da tabela dependente violar a restrição de chave estrangeira, a transação será desfeita.

CASCADE, a operação de exclusão será propagada para a tabela dependente (e para as dependentes desta tabela, caso se aplique).

SET NULL, todas as colunas da chave estrangeira da tabela dependente que aceitam o valor nulo, receberão o valor nulo (Novamente, se a tabela dependente também possuir tabelas dependentes, as colunas das chaves estrangeiras destas tabelas que aceitam o valor nulo, receberão o valor nulo)

Cada restrição referencial onde a tabela é a tabela mãe, possui suas próprias regras de exclusão; todas as regras de exclusão aplicáveis são utilizadas para determinar o resultado da operação de exclusão. Portanto, não poderá ser excluída uma linha que possua dependentes em uma restrição referencial com regra de exclusão RESTRICT ou NO ACTION. De forma semelhante, a linha não poderá ser excluída quando a exclusão se propagar em cascata para alguma de suas descendentes que seja dependente em uma restrição referencial com a regra de exclusão RESTRICT ou NO ACTION.

A exclusão de uma linha da tabela mãe envolve outras tabelas. Qualquer tabela envolvida em uma operação de exclusão na tabela mãe é dita como sendo conectada-para-exclusão com a tabela mãe. A exclusão pode afetar as linhas destas tabelas das seguintes maneiras:

- Se a regra de exclusão for RESTRICT ou NO ACTION, a tabela dependente estará envolvida na operação, mas não será afetada pela operação (ou seja, o Derby verificará os valores na tabela, mas não excluirá qualquer valor).
- Se a regra de exclusão for SET NULL, as linhas da tabela dependente poderão ser atualizadas quando uma linha da tabela mãe for objeto de uma exclusão ou de uma operação de exclusão propagada.
- Se a regra de exclusão for CASCADE, as linhas da tabela dependente poderão ser excluídas quando a tabela mãe for objeto de uma exclusão.
- Se a tabela dependente também for uma tabela mãe, as ações descritas nesta lista serão aplicadas, por sua vez, às suas tabelas dependentes.

Exemplos

```
-- restrição de chave primária no nível-de-coluna chamada PK_SAÍDA:
CREATE TABLE SAMP.CAIXA_SAÍDA
(
    ENVIO          TIMESTAMP,
    DESTINO        CHAR(8),
    ASSUNTO        CHAR(64) NOT NULL CONSTRAINT PK_CAIXA_SAÍDA PRIMARY
KEY,
    TEXTO_NOTA     VARCHAR(3000)
);

-- a definição de chave primária no nível-de-tabela permite
-- incluir duas colunas na definição da chave primária:
CREATE TABLE SAMP.AGENDA
(
    COD_CLASSE CHAR(7) NOT NULL,
    DIA        SMALLINT NOT NULL,
    INÍCIO     TIME,
    FIM        TIME,
    PRIMARY KEY (COD_CLASSE, DIA)
);

-- Uso de uma restrição no nível-de-coluna para verificação aritmética.
-- Uso de uma restrição no nível-de-tabela para ter certeza que os
-- impostos do empregado não são maiores que os bônus.
CREATE TABLE SAMP.EMP
```

```

        (
            NUM_EMP      CHAR(6)      NOT NULL CONSTRAINT PK_EMP PRIMARY KEY,
            NOME          CHAR(12)     NOT NULL,
            INICIAL_MEIO  VARCHAR(12)  NOT NULL,
            SOBRENOME      VARCHAR(15)  NOT NULL,
            SALÁRIO       DECIMAL(9,2)  CONSTRAINT VERIF_SAL CHECK (SALÁRIO >=
10000),
            BÔNUS         DECIMAL(9,2),
            IMPOSTOS       DECIMAL(9,2),
            CONSTRAINT VERIF_BÔNUS CHECK (BÔNUS > IMPOSTOS)
        );

-- Uso de uma restrição de verificação para permitir
-- apenas as abreviaturas apropriadas para as refeições
CREATE TABLE VÔOS
(
    ID_VÔO              CHAR(6) NOT NULL ,
    NÚMERO_SEGMENTO     INTEGER NOT NULL ,
    AEROPORTO_ORIGEM    CHAR(3),
    HORA_PARTIDA        TIME,
    AEROPORTO_DESTINO    CHAR(3),
    HORA_CHEGADA        TIME,
    REFEIÇÃO            CHAR(1) CONSTRAINT VERIF_REFEIÇÃO
                        CHECK (REFEIÇÃO IN ('B', 'L', 'D',
'S')),
    PRIMARY KEY (ID_VÔO, NÚMERO_SEGMENTO)
);

CREATE TABLE METROPOLITANO
(
    ID_HOTEL    INT NOT NULL CONSTRAINT PK_HOTÉIS PRIMARY KEY,
    NOME_HOTEL  VARCHAR(40) NOT NULL,
    ID_CIDADE   INT CONSTRAINT FK_METRO REFERENCES CIDADES
);

-- criação de uma tabela com uma restrição de chave primária
-- e uma restrição de chave estrangeira no nível-de-tabela
CREATE TABLE VÔOS_DISP
(
    ID_VÔO              CHAR(6) NOT NULL,
    NÚMERO_SEGMENTO     INT NOT NULL,
    DATA_VÔO           DATE NOT NULL,
    ASSENTOS_ECONÔMICOS_OCUPADOS INT,
    ASSENTOS_EXECUTIVOS_OCUPADOS INT,
    ASSENTOS_PRIMEIRA_CLASSE_OCUPADOS INT,
    CONSTRAINT PK_VÔOS_DISP PRIMARY KEY (ID_VÔO, NÚMERO_SEGMENTO),
    CONSTRAINT FK_VÔOS
        FOREIGN KEY (ID_VÔO, NÚMERO_SEGMENTO)
        REFERENCES VÔOS (ID_VÔO, NÚMERO_SEGMENTO)
);

-- adicionar uma restrição de unicidade a uma coluna
ALTER TABLE SAMP.PROJETO
ADD CONSTRAINT UNQ_PROJ UNIQUE (NOME_PROJ);

-- criar uma tabela cuja coluna ID_CIDADE faça referência
-- à chave primária da tabela CIDADES utilizando uma
-- restrição de chave estrangeira no nível-de-coluna
CREATE TABLE CONDOMÍNIOS
(
    ID_COND    INT NOT NULL CONSTRAINT PK_HOTÉIS PRIMARY KEY,
    NOME_COND  VARCHAR(40) NOT NULL,
    ID_CIDADE  INT CONSTRAINT FK_CIDADE
        REFERENCES CIDADES
        ON DELETE CASCADE
        ON UPDATE RESTRICT
);

```

Sistema de dependência de instrução

As instruções INSERT e UPDATE dependem de todas as restrições da tabela de destino. As instruções DELETE dependem das restrições de unicidade, chave primária e de chave estrangeira. Estas instruções são invalidadas quando é adicionada ou removida uma restrição na tabela de destino.

restrição no nível-de-coluna

```

{
    NOT NULL |
    { [CONSTRAINT nome-da-restrição]
    {
        CHECK (CondiçãoProcura) |

```

```

    {
        PRIMARY KEY |
        UNIQUE |
        REFERENCES cláusula
    }
}

```

restrição no nível-de-tabela

```

[CONSTRAINT nome-da-restrição]
{
    CHECK (CondiçãoProcura) |
    {
        PRIMARY KEY ( nome-de-coluna-simples [ , nome-de-coluna-simples
]* ) |
        UNIQUE ( nome-de-coluna-simples [ , nome-de-coluna-simples ]* ) |
        FOREIGN KEY ( nome-de-coluna-simples [ , nome-de-coluna-simples
]* )
        REFERENCES cláusula
    }
}

```

Especificação da referência

```

REFERENCES nome-da-tabela [ ( nome-de-coluna-simples [ ,
nome-de-coluna-simples ]* ) ]
[ ON DELETE {NO ACTION | RESTRICT | CASCADE | SET NULL} ]
[ ON UPDATE {NO ACTION | RESTRICT} ]
|
[ ON UPDATE {NO ACTION | RESTRICT} ] [ ON DELETE
{NO ACTION | RESTRICT | CASCADE | SET NULL} ]

```

CondiçãoProcura

A *CondiçãoProcura* é qualquer [Expressão booleana](#) que atenda aos requisitos especificados nos [Requisitos da condição de procura](#).

Quando não é especificado o *nome-da-restrição*, o Derby gera um nome de restrição único (tanto para a restrição de coluna quanto para a restrição de tabela).

Instrução DECLARE GLOBAL TEMPORARY TABLE

A instrução DECLARE GLOBAL TEMPORARY TABLE define uma tabela temporária para a conexão corrente. Estas tabelas não residem nos catálogos do sistema, e não são persistentes. As tabelas temporárias existem somente durante a conexão em que foram declaradas, não podendo ser referenciadas de fora da conexão. Ao fechar a conexão, as linhas da tabela são excluídas e a descrição em-memória da tabela temporária é removida.

As tabelas temporárias são úteis quando:

- a estrutura da tabela não é conhecida antes de utilizar o aplicativo.
- os outros usuários não necessitam da mesma estrutura de tabela.
- os dados da tabela temporária são necessários durante o uso do aplicativo.
- a tabela pode ser declarada e removida sem manter bloqueios no catálogo do sistema.

Sintaxe

```

DECLARE GLOBAL TEMPORARY TABLE nome-da-tabela
{ definição-da-coluna [ , definição-da-coluna ] * }
[ ON COMMIT {DELETE | PRESERVE} ROWS ]
NOT LOGGED [ON ROLLBACK DELETE ROWS]

```

nome-da-tabela

O nome da tabela temporária. Se for especificado um nome-do-esquema diferente de SESSION, ocorrerá um erro (SQLState 428EK). Se o nome-do-esquema não for especificado, será atribuído SESSION. Várias conexões podem definir tabelas temporárias globais declaradas com o mesmo nome, porque cada conexão possui seu próprio descritor de tabela único.

A utilização de SESSION como nome do esquema de uma tabela física não gera erro, mas é desestimulado. O nome de esquema SESSION deve ser reservado para o esquema de tabelas temporárias.

definição-da-coluna

Para obter mais informações sobre a [definição-da-coluna](#) deve ser consultada [definição-da-coluna](#) em CREATE TABLE. A instrução DECLARE GLOBAL TEMPORARY TABLE não permite especificação-de-coluna-gerada na definição-da-coluna.

Tipo-de-dado

Os tipos-de-dado suportados são:

- BIGINT
- CHAR
- DATE
- DECIMAL
- DOUBLE PRECISION
- FLOAT
- INTEGER
- NUMERIC
- REAL
- SMALLINT
- TIME
- TIMESTAMP
- VARCHAR

ON COMMIT

Especifica a ação a ser executada na tabela temporária global quando é realizada uma operação de COMMIT.

DELETE ROWS

Todas as linhas da tabela serão excluídas, se não houver nenhum cursor com possibilidade de HOLD aberto na tabela. Este é o valor padrão para ON COMMIT. Se for especificado ON ROLLBACK DELETE ROWS, serão excluídas todas as linhas da tabela somente se a tabela temporária tiver sido utilizada. ON COMMIT DELETE ROWS exclui as linhas da tabela, mesmo que a tabela não tenha sido utilizada (se a tabela não possuir nenhum cursor com possibilidade de HOLD aberto na mesma).

PRESERVE ROWS

As linhas da tabela são preservadas.

NOT LOGGED

Especifica a ação realizada na tabela temporária global quando uma operação de desfazer é realizada. Quando é realizada uma operação de ROLLBACK (ou ROLLBACK TO SAVEPOINT), se a tabela foi criada na unidade de trabalho (ou no ponto de salvamento), a tabela será removida. Se a tabela foi removida na unidade de trabalho (ou no ponto de salvamento), a tabela será restaurada sem nenhuma linha.

ON ROLLBACK DELETE ROWS

Este é o valor padrão para NOT LOGGED. NOT LOGGED [ON ROLLBACK DELETE ROWS]] especifica a ação a ser realizada na tabela temporária global quando é realizada uma operação de ROLLBACK (ou ROLLBACK TO SAVEPOINT). Se os dados da tabela foram modificados, todas as linhas serão excluídas.

Exemplos

```
SET SCHEMA MYAPP;

CREATE TABLE T1(C11 INT, C12 DATE);

DECLARE GLOBAL TEMPORARY TABLE SESSION.T1(C11 INT) NOT LOGGED;
-- A qualificação SESSION é redundante, porque as tabelas temporárias
-- somente podem existir no esquema SESSION.

DECLARE GLOBAL TEMPORARY TABLE T2(C21 INT) NOT LOGGED;
-- A tabela temporária não é qualificada neste caso com SESSION, porque
-- as tabelas temporárias somente podem existir no esquema SESSION.

INSERT INTO SESSION.T1 VALUES (1);
-- Neste caso a qualificação SESSION é obrigatória para utilizar a
-- tabela temporária, porque o esquema corrente é MYAPP.

SELECT * FROM T1;
-- Esta instrução SELECT está fazendo referência à tabela física
-- "MYAPP.T1", uma vez que a tabela não foi qualificada com SESSION.
```

Deve ser observado que as tabelas temporárias somente podem ser declaradas no esquema SESSION. Nunca deve ser declarado um esquema físico com o nome SESSION.

A seguir está mostrada uma lista de funções da instrução DECLARE GLOBAL TEMPORARY TABLE do DB2 UDB que não são suportadas pelo Derby:

- IDENTITY opções-de-coluna
- IDENTITY atributo nas opções-de-cópia
- AS (FULLSELECT) DEFINITION ONLY
- NOT LOGGED ON ROLLBACK PRESERVE ROWS
- IN nome-do-espaco-de-tabelas
- PARTITIONING KEY
- WITH REPLACE

Restrições das tabelas temporárias globais declaradas

As tabelas temporárias não podem ser especificadas nas seguintes instruções:

- ALTER TABLE
- CREATE SYNONYM
- CREATE TRIGGER
- CREATE VIEW
- LOCK
- RENAME

As tabelas temporárias não pode ser especificadas nas restrições referenciais.

Não há suporte para restrição de verificação nas colunas.

Não podem ser utilizados os seguintes tipos de dado nas tabelas temporárias globais declaradas:

- BLOB
- CLOB
- LONG VARCHAR

As tabelas temporárias não podem ser referenciadas por uma instrução-SQL-engatilhada.

Se uma instrução realizando uma inserção, atualização ou exclusão em uma tabela

temporária encontrar um erro, todas as linhas da tabela serão excluídas.

Restrições específicas do Derby

O Derby não dá suporte em tabelas temporárias a:

- índices
- sinônimos, gatilhos e visões em tabelas no esquema SESSION (incluindo as tabelas físicas e as tabelas temporárias)
- LOCK TABLE
- restrições e chaves primárias
- especificação-de-coluna-gerada
- importação em tabelas temporárias

As instruções fazendo referência a tabelas e visões no esquema SESSION não ficam no *cache*.

Instrução DELETE

Sintaxe

```
{  
    DELETE FROM nome-da-tabela  
    [WHERE cláusula] |  
    DELETE FROM nome-da-tabela WHERE CURRENT OF  
}
```

A primeira forma sintática, chamada de exclusão procurada, exclui todas as linhas identificadas pelo nome da tabela e pela cláusula WHERE.

A segunda forma sintática, chamada de exclusão posicionada, exclui a linha corrente de um cursor atualizável aberto. Se não houver uma linha corrente, ou se a linha não mais satisfizer o comando do cursor, será lançada uma exceção. Para obter mais informações sobre cursores atualizáveis deve ser consultada a [Instrução SELECT](#).

Exemplos

```
DELETE FROM SAMP.CAIXA_ENTRADA;  
  
stmt.executeUpdate("DELETE FROM SAMP.CAIXA_ENTRADA WHERE CURRENT OF " +  
    resultSet.getCursorName())
```

Uma instrução de exclusão procurada depende da tabela sendo atualizada, todos os seus conglomerados (unidades de armazenamento como *heaps* e índices), e todas as outras tabelas citadas na cláusula WHERE. Uma instrução CREATE ou DROP INDEX na tabela de destino de uma instrução de exclusão procurada preparada, invalida a instrução de exclusão procurada preparada.

A instrução de exclusão posicionada depende do cursor e de todas as tabelas referenciadas pelo cursor. A exclusão posicionada pode ser compilada mesmo que o cursor ainda não tenha sido aberto. Entretanto, a remoção do cursor aberto através do método *close* do JDBC invalida a exclusão posicionada.

Uma instrução CREATE ou DROP INDEX na tabela de destino de uma exclusão posicionada preparada, invalida a instrução de exclusão posicionada preparada.

Cláusula FOR UPDATE

A cláusula FOR UPDATE é uma parte opcional da [Instrução SELECT](#). A cláusula FOR UPDATE especifica se o *ResultSet* de uma [Instrução SELECT](#) simples, que atende os requisitos para um *cursor*, é atualizável ou não. Para obter mais informações sobre poder ser atualizável, deve ser consultado [Requisitos para cursores atualizáveis e ResultSets atualizáveis](#).

Sintaxe

```
FOR
{
    READ ONLY | FETCH ONLY |
    UPDATE [ OF nome-de-coluna-simples [ , nome-de-coluna-simples]* ]
}
```

nome-de-coluna-simples faz referência aos nomes visíveis para a tabela especificada na cláusula FROM da consulta subjacente.

Por padrão, os cursores são apenas-de-leitura. Para o cursor ser atualizável, deve ser especificado FOR UPDATE.

O otimizador é capaz de utilizar um índice mesmo que a coluna do índice esteja sendo atualizada. Para obter mais informações sobre como os índices afetam os cursores deve ser consultado o [Ajuste do Derby](#).

```
SELECT RECEBIDO, ORIGEM, ASSUNTO, TEXTO_NOTA FROM SAMP.CAIXA_ENTRADA FOR
UPDATE;
```

Cláusula FROM

A cláusula FROM é uma cláusula obrigatória na [ExpressãoSeleção](#). Especifica as tabelas ([ExpressãoTabela](#)) das quais as outras cláusulas da consulta podem acessar as colunas a serem utilizadas nas expressões.

Sintaxe

```
FROM ExpressãoTabela [ , ExpressãoTabela ] *
```

```
SELECT CIDADES.ID_CIDADE
FROM CIDADES
WHERE ID_CIDADE < 5

-- outros tipos de ExpressãoTabela
SELECT TABLENAME, ISINDEX
FROM SYS.SYSTABLES T, SYS.SYSCONGLOMERATES C
WHERE T.TABLEID = C.TABLEID
ORDER BY TABLENAME, ISINDEX;

-- forçar a ordem de junção
SELECT *
FROM VÔOS, VÔOS_DISPONIBILIDADE
WHERE VÔOS_DISPONIBILIDADE.ID_VÔO = VÔOS.ID_VÔO
AND VÔOS_DISPONIBILIDADE.NÚMERO_SEGMENTO = VÔOS.NÚMERO_SEGMENTO
AND VÔOS.ID_VÔO < 'AA1115'

-- a ExpressãoTabela pode ser uma OperaçãoJunção. Portanto,
-- podem haver várias operações de junção na cláusula FROM
SELECT PAÍSES.PAÍS, CIDADES.NOME_CIDADE, VÔOS.AEROPORTO_DESTINO
FROM PAÍSES LEFT OUTER JOIN CIDADES
ON PAÍSES.COD_ISO_PAÍS = CIDADES.COD_ISO_PAÍS
LEFT OUTER JOIN VÔOS
ON CIDADES.AEROPORTO = VÔOS.AEROPORTO_DESTINO;
```

Cláusula GROUP BY

A cláusula GROUP BY, que faz parte da [ExpressãoSeleção](#), agrupa o resultado em

subconjuntos que possuem valores correspondentes em uma ou mais colunas. Em cada grupo não há duas linhas com o mesmo valor na coluna, ou colunas, de agrupamento. Para as finalidades de agrupamento os valores nulos são considerados equivalentes.

Normalmente a cláusula GROUP BY é utilizada junto com expressões de agregação.

Sintaxe

```
GROUP BY nome-da-coluna [ , nome-da-coluna ] *
```

O *nome-da-coluna* deve ser uma coluna do escopo corrente da consulta; não pode haver nenhuma coluna do bloco de consulta fora do escopo corrente. Por exemplo, se a cláusula GROUP BY estiver na subconsulta, então não poderá fazer referência a colunas da consulta externa.

Os *ItensSelecionados* em uma *ExpressãoSeleção* com cláusula GROUP BY devem conter somente agregações ou colunas de agrupamento.

```
-- calcular o TEMPO_VÔO médio agrupado por aeroporto
SELECT AVG (TEMPO_VÔO), AEROPORTO_ORIGEM
FROM VÔOS
GROUP BY AEROPORTO_ORIGEM

SELECT MAX(CIDADE), REGIÃO
FROM CIDADES, PAÍSES
WHERE CIDADES.COD_ISO_PAÍS = PAÍSES.COD_ISO_PAÍS
GROUP BY REGIÃO

-- agrupar por um SMALLINT
SELECT ID, AVG(SALÁRIO)
FROM SAMP.EQUIPES
GROUP BY ID

-- Obter as colunas SALÁRIO_MÉDIO, NUM_EMP e NUM_DEP
-- utilizando a cláusula AS
-- Agrupar pela coluna DEP_TRAB utilizando o nome de
-- correlação OUTROS
SELECT OUTROS.DEP_TRAB AS NUM_DEP,
       AVG(OUTROS.SALÁRIO) AS SALÁRIO_MÉDIO,
       COUNT(*) AS NUM_EMP
FROM SAMP.EMPREGADOS OUTROS
GROUP BY OUTROS.DEP_TRAB;
```

Cláusula HAVING

A cláusula HAVING restringe os resultados do GROUP BY na *ExpressãoSeleção*. A cláusula HAVING é aplicada a cada grupo da tabela agrupada, de forma parecida como a cláusula WHERE é aplicada à lista de seleção. Se não houver uma cláusula GROUP BY, a cláusula HAVING será aplicada a todo o resultado como um único grupo. A cláusula SELECT não pode fazer referência direta a qualquer coluna que não possua uma cláusula GROUP BY. Entretanto, pode fazer referência a constantes, agregações, e registros especiais.

Sintaxe

```
HAVING CondiçãoProcura
```

A *CondiçãoProcura*, que é uma *ExpressãoBooleana* especializada, pode conter apenas as colunas de agrupamento (consulte a [Cláusula GROUP BY](#)), colunas que fazem parte das expressões de agregação, e as colunas que fazem parte da subconsulta. Por exemplo, a seguinte consulta é ilegal, porque a coluna SALÁRIO não é uma coluna de agrupamento, não aparece em uma agregação, e não está em uma subconsulta:

```
-- SELECT COUNT(*)
-- FROM SAMP.EQUIPES
-- GROUP BY ID
-- HAVING SALÁRIO > 15000;
```

As agregações na cláusula HAVING não precisam aparecer na lista de seleção. Se a cláusula HAVING possui uma subconsulta, a subconsulta poderá fazer referência ao bloco de consulta externo se, e somente se, fizer referência a uma coluna de agrupamento.

```
-- Descobrir o número total de assentos econômicos
-- ocupados no voo, agrupado por linha aérea,
-- somente quando o grupo possuir pelo menos 2 linhas.
SELECT SUM(ASSENTOS_ECONÔMICOS_OCUPADOS), LINHA_AÉREA_CHEIA
FROM VÔOS_DISPONIBILIDADE, LINHAS_AÉREAS
WHERE SUBSTR(VÔOS_DISPONIBILIDADE.ID_VÔO, 1, 2) = LINHA_AÉREA
GROUP BY LINHA_AÉREA_CHEIA
HAVING COUNT(*) > 1
```

INNER JOIN

INNER JOIN (junção interna) é uma [Operação JOIN](#) que permite especificar uma cláusula de junção explícita.

Sintaxe

```
ExpressãoTabela [ INNER ] JOIN ExpressãoTabela { ON ExpressãoBooleana }
```

A cláusula de junção pode ser especificada utilizando ON com uma expressão booleana.

O escopo das expressões na cláusula ON inclui as tabelas correntes, e as tabelas nos blocos de consulta externos ao SELECT corrente. No exemplo a seguir, a cláusula ON faz referência às tabelas correntes:

```
SELECT *
FROM SAMP.EMPREGADOS INNER JOIN SAMP.EQUIPES
ON EMPREGADOS.SALÁRIO < EQUIPES.SALÁRIO;
```

A cláusula ON pode fazer referência a tabelas que não estão sendo juntadas, e não é obrigada a fazer referência a nenhuma das tabelas sendo juntadas (embora tipicamente o faça).

```
-- Junção das tabelas ATIV_EMP e EMPREGADOS
-- selecionar todas as colunas da tabela ATIV_EMP e
-- adicionar o sobrenome do empregado (ÚLTIMO_NOME) da tabela
-- EMPREGADOS a todas as linhas do resultado
SELECT SAMP.ATIV_EMP.*, ÚLTIMO_NOME
FROM SAMP.ATIV_EMP JOIN SAMP.EMPREGADO
ON ATIV_EMP.NUM_EMP = EMPREGADOS.NUM_EMP;

-- Juntar as tabelas EMPREGADOS e DEPARTAMENTOS,
-- selecionar o número do empregado (NUM_EMP),
-- o sobrenome do empregado (ÚLTIMO_NOME),
-- o número do departamento (DEP_TRAB na tabela EMPREGADOS e
-- NUM_DEP na tabela DEPARTAMENTOS)
-- e o nome do departamento (NOME_DEP)
-- de todos os empregados nascidos (DATA_NASC) antes de 1930.
SELECT NUM_EMP, ÚLTIMO_NOME, DEP_TRAB, NOME_DEP
FROM SAMP.EMPREGADOS JOIN SAMP.DEPARTAMENTOS
ON DEP_TRAB = NUM_DEP
AND YEAR(DATA_NASC) < 1930;

-- Outro exemplo de "gerar" novos valores de dado,
-- utilizando uma consulta que seleciona da cláusula VALUES
-- (que é uma forma alternativa de FULLSELECT).
SELECT *
FROM (VALUES (3, 4), (1, 5), (2, 6))
AS TABELA1_VALORES(C1, C2)
JOIN (VALUES (3, 2), (1, 2), (0, 3))
AS TABELA2_VALORES(C1, C2)
ON TABELA1_VALORES.C1 = TABELA2_VALORES.C1;
```

O que resulta em:

C1	C2	C1	2
----	----	----	---

```

-----
3          |4          |3          |2
1          |5          |1          |2

-- Listar todos os departamentos, juntamente com o
-- número do empregado e o último nome do gerente
SELECT NUM_DEP, NOME_DEP, NUM_EMP, ÚLTIMO_NOME
FROM DEPARTAMENTOS
     INNER JOIN EMPREGADOS
       ON NUM_GER = NUM_EMP;

-- Listar todos os números do empregado e último nome, juntamente
-- com o número do empregado e último nome de seus gerentes
SELECT E.NUM_EMP, E.ÚLTIMO_NOME, M.NUM_EMP, M.ÚLTIMO_NOME
FROM EMPREGADOS E INNER JOIN
     DEPARTAMENTOS INNER JOIN EMPREGADOS M
       ON NUM_GER = M.NUM_EMP
       ON E.DEP_TRAB = NUM_DEP;

```

Instrução INSERT

A instrução INSERT cria uma ou mais linhas, e as armazena na tabela especificada. O número de valores especificados na instrução INSERT deve ser idêntico ao número de colunas especificadas ou implícitas.

Sintaxe

```

INSERT INTO nome-da-tabela
[ ( nome-de-coluna-simples [ , nome-de-coluna-simples ]* ) ]
  Consulta

```

A *Consulta* pode ser:

- uma [ExpressãoSeleção](#)
- uma lista VALUES
- uma expressão VALUES de várias linhas

As listas de uma única linha e de várias linhas podem incluir a palavra chave DEFAULT. A especificação de DEFAULT para uma coluna insere o valor padrão da coluna na coluna. Outra forma de inserir o valor padrão na coluna é omitir a coluna na lista de colunas, e somente inserir valores nas outras colunas da tabela. Para obter mais informações deve ser consultado [VALUES Expressão](#).

- expressões UNION

Para obter mais informações sobre a *Consulta* deve ser consultada [Consulta](#).

```

INSERT INTO PAÍSES
VALUES ('Taiwan', 'TW', 'Ásia');

-- Inserir um novo departamento na tabela DEPARTAMENTOS,
-- sem atribuir gerente ao novo departamento
INSERT INTO DEPARTAMENTOS (NUM_DEP, NOME_DEP, ADMRDEPT)
VALUES ('E31', 'ARQUITETURA', 'E01');

-- Inserir dois novos departamentos na tabela DEPARTAMENTOS
-- utilizando uma instrução, como no exemplo anterior,
-- sem atribuir gerente aos novos departamentos.
INSERT INTO DEPARTAMENTOS (NUM_DEP, NOME_DEP, ADMRDEPT)
VALUES ('B11', 'COMPRAS', 'B01'),
       ('E41', 'ADMINISTRAÇÃO DE BANCO DE DADOS', 'E01');

-- Criar a tabela temporária MA_ATIV_EMP com as mesmas
-- colunas da tabela ATIV_EMP.
-- Carregar a tabela MA_ATIV_EMP com as linhas da tabela ATIV_EMP
-- onde o número do projeto (NUM_PROJ)
-- começa pelas letras 'MA'.
CREATE TABLE MA_ATIV_EMP
(
    NUM_EMP    CHAR(6)    NOT NULL,
    NUM_PROJ   CHAR(6)    NOT NULL,
    ACTNO      SMALLINT   NOT NULL,
    EMPTIME    DEC(5,2),
    EMSTDATE   DATE,

```

```

    );
    EMENDATE DATE
);
INSERT INTO MA_ATIV_EMP
  SELECT * FROM ATIV_EMP
  WHERE SUBSTR(NUM_PROJ, 1, 2) = 'MA';
-- Inserir o valor DEFAULT para a coluna LOCALIZAÇÃO
INSERT INTO DEPARTAMENTOS
  VALUES ('E31', 'ARQUITETURA', '00390', 'E01', DEFAULT);

```

Sistema de dependência de instrução

A instrução INSERT depende da tabela onde está sendo feita a inserção, todos os seus conglomerados (unidades de armazenamento como *heaps* e índices), e todas as outras tabelas citadas na consulta. Qualquer instrução que cria ou remove um índice ou uma restrição da tabela de destino de uma instrução INSERT preparada, invalida a instrução INSERT preparada.

Operação JOIN

As operações de junção (JOIN), que estão entre as [Expressões Tabela](#) possíveis na [Cláusula FROM](#), realizam junções entre duas tabelas (Também pode ser realizada a junção entre duas tabelas utilizando um teste de igualdade explícito na cláusula WHERE, como "WHERE t1.col1 = t2.col2".)

Sintaxe

Operação de junção

As operações de junção são:

- [INNER JOIN](#)

Especifica a junção entre duas tabelas com uma cláusula de junção explícita. Consulte [INNER JOIN](#).

- [LEFT OUTER JOIN](#)

Especifica a junção entre duas tabelas com uma cláusula de junção explícita, preservando as linhas sem correspondência da primeira tabela. Consulte [LEFT OUTER JOIN](#).

- [RIGHT OUTER JOIN](#)

Especifica a junção entre duas tabelas com uma cláusula de junção explícita, preservando as linhas sem correspondência da segunda tabela. Consulte [RIGHT OUTER JOIN](#).

Em todos os casos podem ser especificadas restrições adicionais para uma ou mais tabelas sendo juntadas nas cláusulas de junção externa, ou na [Cláusula WHERE](#).

Expressões de junção e otimização de consulta

Para obter informações sobre quais tipos de junção são otimizadas, consulte *Ajuste do Derby*.

LEFT OUTER JOIN

LEFT OUTER JOIN é uma [Operação JOIN](#) que permite especificar a cláusula de junção. Preserva as linhas sem correspondência da primeira tabela (esquerda), juntando-as com uma linha nula na forma da segunda tabela (direita).

Sintaxe

```
ExpressãoTabela LEFT [ OUTER ] JOIN ExpressãoTabela
{
    ON ExpressãoBooleana
}
```

O escopo das expressões na cláusula ON inclui as tabelas correntes, e as tabelas nos blocos de consulta externos ao SELECT corrente. A cláusula ON pode fazer referência a tabelas que não estão sendo juntadas, e não é obrigada a fazer referência a nenhuma das tabelas sendo juntadas (embora tipicamente o faça).

```
--correspondência entre cidades e países
SELECT CIDADES.PAÍS, REGIÃO
FROM PAÍSES
    LEFT OUTER JOIN CIDADES
        ON ID_CIDADE=ID_CIDADE
WHERE REGIÃO = 'Ásia';

-- uso da sintaxe sinônimo, LEFT JOIN, para obter exatamente
-- os mesmos resultados da exemplo acima
SELECT CIDADES.PAÍS, REGIÃO
FROM PAÍSES
    LEFT JOIN CIDADES
        ON ID_CIDADE=ID_CIDADE
WHERE REGIÃO = 'Ásia';

-- Junção das tabelas EMPREGADOS e DEPARTAMENTOS,
-- selecionar o número do empregado (NUM_EMP),
-- o sobrenome do empregado (ÚLTIMO_NOME),
-- o número do departamento (DEP_TRAB na tabela EMPREGADOS e
-- NUM_DEP na tabela DEPARTAMENTOS)
-- e o nome do departamento (NOME_DEP)
-- de todos os empregados nascidos (DATA_NASC) antes de 1930
SELECT NUM_EMP, ÚLTIMO_NOME, DEP_TRAB, NOME_DEP
FROM SAMP.EMPREGADOS
    LEFT OUTER JOIN SAMP.DEPARTAMENTOS
        ON DEP_TRAB = NUM_DEP
        AND YEAR(DATA_NASC) < 1930;

-- Listar todos os departamentos, juntamente com o
-- número do empregado e o último nome do gerente,
-- incluindo os departamentos sem gerente
SELECT NUM_DEP, NOME_DEP, NUM_EMP, ÚLTIMO_NOME
FROM DEPARTAMENTOS
    LEFT OUTER JOIN EMPREGADOS
        ON NUM_GER = NUM_EMP;
```

Instrução LOCK TABLE

Permite ao usuário obter explicitamente um bloqueio de tabela, exclusivo ou compartilhado, na tabela especificada. O bloqueio da tabela permanece até o término da transação corrente.

O bloqueio explícito da tabela é útil para:

- evitar a sobrecarga devido a vários bloqueios na tabela (em outras palavras, escalada de bloqueio iniciada pelo usuário)
- evitar impasses (*deadlocks*)

Não é possível bloquear tabelas do sistema com esta instrução.

Sintaxe

```
LOCK TABLE nome-da-tabela IN { SHARE | EXCLUSIVE } MODE
```

Uma vez que alguma tabela esteja bloqueada em um dos modos, a transação não obterá bloqueios subseqüentes no nível-de-linha. Por exemplo, se uma transação bloquear toda a tabela VÔOS no modo compartilhado para ler os dados, e uma determinada instrução desta transação necessitar bloquear uma determinada linha no modo exclusivo para poder atualizar a linha, o bloqueio anterior no nível-de-tabela força o bloqueio no modo exclusivo ser no nível-de-tabela também.

Se o bloqueio especificado não puder ser obtido porque outra conexão já possui um bloqueio na tabela, será lançada uma exceção no nível-de-instrução (*SQLState X0X02*) após ser esgotado o tempo limite de impasse.

```
-- bloquear toda a tabela no modo compartilhado
-- para evitar um número grande de bloqueios de linha
LOCK TABLE VÔOS IN SHARE MODE;

SELECT *
FROM VÔOS
WHERE AEROPORTO_ORIGEM > '000';

-- bloquear toda a tabela no modo exclusivo
-- para uma transação que irá atualizar muitas linhas,
-- mas onde nenhuma instrução atualizará isoladamente um
-- número suficiente de linhas para obter um bloqueio
-- da tabela no modo exclusivo.
-- No sistema de bloqueio no nível-de-linha, a transação
-- iria requerer um número grande de bloqueios e poderia
-- causar um impasse.
LOCK TABLE DISPONIBILIDADE_HOTEL IN EXCLUSIVE MODE;

UPDATE DISPONIBILIDADE_HOTEL
SET QUARTOS_RESERVADOS = (QUARTOS_RESERVADOS + 2)
WHERE ID_HOTEL = 194 AND DATA_DE_RESERVA = DATE('1998-04-10');

UPDATE DISPONIBILIDADE_HOTEL
SET QUARTOS_RESERVADOS = (QUARTOS_RESERVADOS + 2)
WHERE ID_HOTEL = 194 AND DATA_DE_RESERVA = DATE('1998-04-11');

UPDATE DISPONIBILIDADE_HOTEL
SET QUARTOS_RESERVADOS = (QUARTOS_RESERVADOS + 2)
WHERE ID_HOTEL = 194 AND DATA_DE_RESERVA = DATE('1998-04-12');

UPDATE DISPONIBILIDADE_HOTEL
SET QUARTOS_RESERVADOS = (QUARTOS_RESERVADOS + 2)
WHERE ID_HOTEL = 194 AND DATA_DE_RESERVA = DATE('1998-04-12');

-- se a transação necessitar bloquear a tabela antes de
-- atualizá-la, deverá obter um bloqueio exclusivo antes
-- de selecionar para evitar impasses.
LOCK TABLE PESSOAS IN EXCLUSIVE MODE;

SELECT MAX(ID_PESSOA) + 1 FROM PESSOAS;
-- INSERT INTO PESSOAS . . .
```

Cláusula ORDER BY

A cláusula ORDER BY é um elemento opcional da [Instrução SELECT](#). A cláusula ORDER BY permite especificar a ordem em que as linhas aparecem no *ResultSet*.

Sintaxe

```
ORDER BY { nome-da-coluna | PosiçãoColuna }
        [ ASC | DESC ]
        [ , nome-da-coluna | PosiçãoColuna
          [ ASC | DESC ] ] *
```

A *PosiçãoColuna* é um valor inteiro que identifica o número da coluna no *ItemSeleção* na consulta subjacente da [Instrução SELECT](#). A *PosiçãoColuna* deve ser maior que zero, e não pode ser maior que o número de colunas na tabela de resultado. Em outras palavras, se for desejado ordenar pela posição da coluna, a coluna deverá estar presente na lista de seleção.

O [nome-da-coluna](#) se refere aos nomes visíveis dos *ItensSelecionados* na consulta subjacente da [Instrução SELECT](#). O nome da coluna de ordenação não precisa estar na lista de seleção.

ASC especifica que os resultados devem ser retornados na ordem ascendente; DESC especifica que os resultados devem ser retornados na ordem descendente; Quando a ordem não é especificada, o padrão é ASC.

A cláusula ORDER BY impede que a instrução SELECT se torne um cursor atualizável (Para obter mais informações deve ser consultado [Requisitos para cursores atualizáveis e ResultSets atualizáveis](#).)

Por exemplo, se uma coluna INTEGER contiver números inteiros, NULL é considerado maior que 1 para as finalidades de classificação. Em outras palavras, os valores nulos são classificados como sendo maiores.

```
-- ordenar pelo nome de correlação NAÇÃO
SELECT NOME_CIDADE, PAÍS AS NAÇÃO
FROM CIDADES
ORDER BY NAÇÃO;
```

Consulta

A consulta cria uma tabela virtual baseada em tabelas existentes ou constantes transformadas em tabelas.

Sintaxe

```
{
  ( Consulta ) |
  Consulta INTERSECT [ ALL | DISTINCT ] Consulta |
  Consulta EXCEPT [ ALL | DISTINCT ] Consulta |
  Consulta UNION [ ALL | DISTINCT ] Consulta |
  ExpressãoSeleção | VALUES Expressão
}
```

Pode-se colocar parênteses arbitrariamente em torno das consultas, ou utilizar parênteses para controlar a ordem de avaliação das operações INTERSECT, EXCEPT e UNION. Estas operações são avaliadas da esquerda para a direita quando não existem parênteses presentes, com exceção das operações INTERSECT, que são avaliadas antes das operações de UNION e EXCEPT.

Linhas duplicadas nos resultados de UNION, INTERSECT e EXCEPT ALL

As palavras chave ALL e DISTINCT determinam se as linhas duplicadas são eliminadas do resultado da operação. Se for especificada a palavra chave DISTINCT, então o resultado não terá linhas duplicadas. Se for especificada a palavra chave ALL, então poderão existir linhas duplicadas no resultado, dependendo da existência de linhas duplicadas na entrada. DISTINCT é o padrão, portanto se não for especificado nem ALL nem DISTINCT as linhas duplicadas serão eliminadas. Por exemplo, UNION constrói um *ResultSet* intermediário com todas as linhas das duas consultas, e elimina as linhas duplicadas antes de retornar as linhas remanescentes. UNION ALL retorna todas as linhas das duas consultas como resultado.

Dependendo da operação especificada, se o número de cópias de uma determinada linha na tabela à esquerda for L, e o número de cópias desta linha na tabela à direita for R, então o número de linhas duplicadas desta determinada linha contidas na tabela de saída será (assumindo que a palavra chave ALL foi especificada):

- UNION: (L + R).
- EXCEPT: o maior entre (L – R) e 0 (zero).
- INTERSECT: o menor entre L e R.

Exemplos

```
-- Uma expressão de seleção
SELECT *
FROM ORG;

-- uma subconsulta
SELECT *
FROM (SELECT COD_CLASSE FROM CLASSE_AGENDA) AS CS;
```

```

-- uma subconsulta
SELECT *
FROM (SELECT COD_CLASSE FROM CLASSE_AGENDA) AS CS (COD_CLASSE);

-- uma união
-- retornar todas as linhas das colunas NUM_DEP e GERENTE
-- da tabela ORG
-- e (1,2) e (3,4)
-- NUM_DEP e GERENTE são colunas do tipo SMALLINT.
SELECT NUM_DEP, GERENTE
FROM ORG
UNION ALL
VALUES (1,2), (3,4);

-- uma expressão de valores
VALUES (1,2,3);

-- Listar os números dos empregados (NUM_EMP)
-- de todos os empregados na tabela EMPREGADOS
-- cujo número do departamento (DEP TRAB) começa por 'E', ou
-- quem está alocado a projetos na tabela ATIV_EMP
-- cujo número do projetor (NUM_PROJ) é igual a
-- 'MA2100', 'MA2110' ou 'MA2112'.
SELECT NUM_EMP
FROM EMPREGADOS
WHERE DEP TRAB LIKE 'E%'
UNION
SELECT NUM_EMP
FROM ATIV_EMP
WHERE NUM_PROJ IN('MA2100', 'MA2110', 'MA2112');

-- Realizar a mesma consulta do exemplo anterior
-- e "marcar" as linhas da tabela EMPREGADOS com 'emp' e
-- as linhas da tabela ATIV_EMP com 'ativ_emp'.
-- Diferentemente do resultado do exemplo anterior,
-- esta consulta pode retornar o mesmo NUM_EMP mais de uma vez,
-- identificando de que tabela veio pela "marca" associada.
SELECT NUM_EMP, 'emp'
FROM EMPREGADOS
WHERE DEP TRAB LIKE 'E%'
UNION
SELECT NUM_EMP, 'ativ_emp'
FROM ATIV_EMP
WHERE NUM_PROJ IN('MA2100', 'MA2110', 'MA2112');

-- Realizar a mesma consulta do exemplo anterior,
-- porém utilizando UNION ALL para que as linhas
-- duplicadas não sejam eliminadas.
SELECT NUM_EMP
FROM EMPREGADOS
WHERE DEP TRAB LIKE 'E%'
UNION ALL
SELECT NUM_EMP
FROM ATIV_EMP
WHERE NUM_PROJ IN('MA2100', 'MA2110', 'MA2112');

-- Realizar a mesma consulta do exemplo anterior,
-- incluindo dois empregados adicionais que no
-- momento não se encontram em nenhuma tabela,
-- e marcar estas linhas como "nova".
SELECT NUM_EMP, 'emp'
FROM EMPREGADOS
WHERE DEP TRAB LIKE 'E%'
UNION
SELECT NUM_EMP, 'ativ_emp'
FROM ATIV_EMP
WHERE NUM_PROJ IN('MA2100', 'MA2110', 'MA2112')
UNION
VALUES ('NEWAAA', 'nova'), ('NEWBBB', 'nova');

```

RIGHT OUTER JOIN

RIGHT OUTER JOIN é uma [Operação JOIN](#) que permite especificar a cláusula de junção. Preserva as linhas sem correspondência da segunda tabela (direita), juntando-as com uma linha nula na forma da primeira tabela (esquerda). (A LEFT OUTER JOIN B) é equivalente a (B RIGHT OUTER JOIN A), com as colunas em uma ordem diferente.

Sintaxe

```
ExpressãoTabela RIGHT [ OUTER ] JOIN ExpressãoTabela
{
    ON ExpressãoBooleana
}
```

O escopo das expressões na cláusula ON inclui as tabelas correntes, e as tabelas nos blocos de consulta externos ao SELECT corrente. A cláusula ON pode fazer referência a tabelas que não estão sendo juntadas, e não é obrigada a fazer referência a nenhuma das tabelas sendo juntadas (embora tipicamente o faça).

```
-- obter todos os países e cidades correspondentes,
-- incluindo os países sem nenhuma cidade
SELECT NOME_CIDADE, CIDADES.PAÍS
FROM CIDADES RIGHT OUTER JOIN PAÍSES
    ON CIDADES.COD_ISO_PAÍS = PAÍSES.COD_ISO_PAÍS;

-- obter todos países da África e as cidades correspondentes,
-- incluindo os países sem cidades
SELECT NOME_CIDADE, CIDADES.PAÍS
FROM CIDADES RIGHT OUTER JOIN PAÍSES
    ON CIDADES.COD_ISO_PAÍS = PAÍSES.COD_ISO_PAÍS;
WHERE PAÍSES.REGIÃO = 'África';

-- uso da sintaxe sinônimo, RIGHT JOIN, para obter exatamente
-- os mesmos resultados do exemplo acima
SELECT NOME_CIDADE, CIDADES.PAÍS
FROM CIDADES RIGHT JOIN PAÍSES
    ON CIDADES.COD_ISO_PAÍS = PAÍSES.COD_ISO_PAÍS
WHERE PAÍSES.REGIÃO = 'África';

-- a ExpressãoTabela pode ser uma OperaçãoJunção. Portanto,
-- podem haver várias operações de junção na cláusula FROM
-- Listar todos os números e último nome dos empregados,
-- juntamente com os números e último nome de seus gerentes
SELECT E.NUM_EMP, E.ÚLTIMO_NOME, M.NUM_EMP, M.ÚLTIMO_NOME
FROM EMPREGADOS E RIGHT OUTER JOIN
    DEPARTAMENTOS RIGHT OUTER JOIN EMPREGADOS M
    ON NUM_GER = M.NUM_EMP
    ON E.DEP_TRAB = NUM_DEP;
```

SubconsultaEscalar

A *SubconsultaEscalar* pode ser colocada em qualquer lugar onde uma *Expressão* é permitida. A *SubconsultaEscalar* torna o resultado da *ExpressãoSeleção* um valor escalar, porque retorna apenas o valor de uma única linha e coluna.

A consulta deve produzir uma única linha com uma única coluna.

Algumas vezes também é chamada de expressão de subconsulta.

Sintaxe

(*Consulta*)

```
-- a média sempre retorna um único valor,
-- portanto a subconsulta é uma SubconsultaEscalar
SELECT NOME, COMISS
FROM EQUIPE
WHERE EXISTS
    (SELECT AVG(BÔNUS + 800)
    FROM EMPREGADOS
    WHERE COMISS < 5000
    AND EMPREGADOS.ÚLTIMO_NOME = UPPER(EQUIPES.NOME)
    );

-- Introduzir uma maneira de "gerar" novos valores de dados,
-- utilizando uma consulta que seleciona da cláusula VALUES
-- (que é uma forma alternativa de FULLSELECT).
-- Esta consulta mostra como pode ser criada uma tabela chamada "X",
-- possuindo duas colunas "R1" e "R2" e uma linha de dados.
SELECT R1,R2
```

```
FROM (VALUES('GRUPO 1','GRUPO 2')) AS X(R1,R2);
```

ExpressãoSeleção

A *ExpressãoSeleção* é a construção básica SELECT-FROM-WHERE utilizada para construir um valor tabela baseado na filtragem e projeção de valores de outras tabelas.

Sintaxe

```
SELECT [ DISTINCT | ALL ] ItemSeleção [ , ItemSeleção ]*
Cláusula FROM
[ Cláusula WHERE ]
[ Cláusula GROUP BY ]
[ Cláusula HAVING ]
```

ItemSeleção:

```
{
  * |
  { nome-da-tabela | nome-da-correlação } .* |
  Expressão [AS nome-de-coluna-simples]
}
```

A cláusula SELECT contém uma lista de expressões e um quantificador opcional que é aplicado aos resultados da *Cláusula FROM* e da *Cláusula WHERE*. Se for especificado DISTINCT, somente será incluída no resultado uma cópia de qualquer valor linha. Os nulos são considerados duplicados entre si para as finalidades do DISTINCT. Se não for especificado um quantificador, ou se for especificado ALL, nenhuma linha será removida do resultado na aplicação da cláusula SELECT (ALL é o padrão).

O *ItemSeleção* projeta um ou mais valores coluna de resultado na tabela resultado sendo construída na *ExpressãoSeleção*.

O resultado da *Cláusula FROM* é o produto cruzado dos itens do FROM. A *Cláusula WHERE* pode qualificar ainda mais este resultado.

A cláusula WHERE faz com que as linhas do resultado sejam filtradas com base em expressões booleanas. Somente as linhas para as quais a expressão booleana é avaliada como verdade são retornadas no resultado.

A cláusula GROUP BY agrupa as linhas do resultado em subconjuntos que possuem valores correspondentes em uma ou mais colunas. As cláusulas GROUP BY são utilizadas normalmente com agregações.

Caso haja uma cláusula GROUP BY, a cláusula SELECT deverá conter *apenas* agregações ou colunas de agrupamento. Se for desejado incluir uma coluna não agrupada na cláusula SELECT, esta coluna deverá ser incluída em uma expressão de agregação. Por exemplo:

```
-- Listar o chefe de cada departamento,
-- o número do departamento (DEP_TRAB),
-- e o salário médio do departamento (SALÁRIO)
-- para todos os departamentos na tabela EMPREGADOS.
-- Organizar a tabela de resultado na ordem ascendente
-- do salário médio do departamento
SELECT DEP_TRAB, AVG(SALÁRIO)
FROM EMPREGADOS
GROUP BY DEP_TRAB
ORDER BY 2;
```

Se não houver uma cláusula GROUP BY, mas *ItemSeleção* contiver uma agregação que não esteja em uma subconsulta, a consulta será agrupada implicitamente. Toda a tabela se torna um único grupo.

A cláusula HAVING restringe a tabela agrupada, especificando uma condição de procura (muito semelhante à cláusula WHERE) que pode fazer referência apenas às colunas de agrupamento ou agregações do escopo corrente. A cláusula HAVING é aplicada a cada grupo da tabela agrupada. Se a cláusula HAVING for avaliada como TRUE, a linha será retida para processamento adicional. Se a cláusula HAVING for avaliada como FALSE ou NULL, a linha será desprezada. Se houver uma cláusula HAVING mas não houver GROUP BY, a tabela será agrupada implicitamente em um grupo para toda a tabela.

O Derby processa a *ExpressãoSeleção* na seguinte ordem:

- Cláusula FROM
- Cláusula WHERE
- GROUP BY (ou GROUP BY implícito)
- Cláusula HAVING
- Cláusula SELECT

O resultado da *ExpressãoSeleção* é sempre uma tabela.

Quando a consulta não possui uma cláusula FROM (quando está sendo construído um valor, e não obtendo dados de uma tabela), é utilizada a instrução VALUES, e não a *ExpressãoSeleção*. Por exemplo:

```
VALUES CURRENT_TIMESTAMP
```

Consulte [Expressão VALUES](#).

O curinga *

O * é expandido como todas as colunas presentes nas tabelas da cláusula FROM associada.

*nome-da-tabela.** e *nome-da-correlação.** são expandidos como todas as colunas da tabela identificada. Esta tabela deve estar listada na cláusula FROM associada.

Atribuir nomes às colunas

Pode ser atribuído um nome a uma coluna do *ItemSeleção* utilizando a cláusula AS. Quando a *ExpressãoSeleção* aparece no operador UNION, INTERSECT ou EXCEPT, os nomes da primeira *ExpressãoSeleção* são usados como os nomes das colunas no resultado da operação. Se uma coluna do *ItemSeleção* não for uma expressão simples de *ReferênciaColuna*, ou não for dado um nome através da cláusula AS, será atribuído para a mesma um nome único gerado.

Estes nomes de colunas são úteis em vários casos:

- São tornados disponíveis no *ResultSetMetaData* do JDBC.
- São utilizados como nomes das colunas na tabela resultante, quando a *ExpressãoSeleção* é utilizada como subconsulta de tabela na cláusula FROM.
- São utilizados na cláusula ORDER BY como nomes de coluna disponíveis para classificação.

```
-- este exemplo mostra SELECT-FROM-WHERE
-- com uma cláusula ORDER BY
-- e nome-da-correlação para as tabelas
SELECT CONSTRAINTNAME, COLUMNNAME
FROM SYS.SYSTABLES t, SYS.SYSCOLUMNS col,
SYS.SYSCONSTRAINTS cons, SYS.SYSCHECKS checks
WHERE t.TABLENAME = 'VÓOS'
AND t.TABLEID = col.REFERENCEID
AND t.TABLEID = cons.TABLEID
AND cons.CONSTRAINTID = checks.CONSTRAINTID
ORDER BY CONSTRAINTNAME;

-- Este exemplo mostra a utilização da cláusula DISTINCT
SELECT DISTINCT NUM_ATIV
FROM ATIV_EMP;
```

```
-- Este exemplo mostra como mudar o nome de uma expressão.
-- Utilizando a tabela EMPREGADOS,
-- listar o número do departamento (DEP_TRAB) e
-- o maior salário do departamento (SALÁRIO) com nome mudado para BOSS
-- para todos os departamentos cujo salário máximo seja menor que o
-- salário médio de todos os outros departamentos.
SELECT DEP_TRAB AS DPT, MAX(SALÁRIO) AS BOSS
FROM EMPREGADOS EMP_COR
GROUP BY DEP_TRAB
HAVING MAX(SALÁRIO) < (SELECT AVG(SALÁRIO)
                       FROM EMPREGADOS
                       WHERE NOT DEP_TRAB = EMP_COR.DEP_TRAB)
ORDER BY BOSS;
```

Instrução SELECT

A instrução SELECT consiste de uma consulta com uma [Cláusula ORDER BY](#) opcional, e uma [Cláusula FOR UPDATE](#) opcional. A instrução SELECT possui este nome porque tipicamente a primeira palavra da construção da consulta é SELECT (A *consulta* inclui a expressão VALUES, as expressões UNION, INTERSECT e EXCEPT, além de expressões SELECT).

A [Cláusula ORDER BY](#) garante a ordem do *ResultSet*. A [Cláusula FOR UPDATE](#) torna o resultado um cursor atualizável. A instrução SELECT suporta a cláusula FOR FETCH ONLY. A cláusula FOR FETCH ONLY é sinônimo da cláusula FOR READ ONLY.

Lembre-se: Para se obter um *ResultSet* atualizável, deve ser incluída a cláusula FOR UPDATE com a cláusula SELECT.

Sintaxe

```
Consulta
[Cláusula ORDER BY]
[Cláusula FOR UPDATE]
WITH {RR|RS|CS|UR}
```

Pode ser definido o nível de isolamento da instrução SELECT utilizando a sintaxe WITH {RR|RS|CS|UR}.

```
-- listar o nome da expressão SALÁRIO+BÔNUS+COMISS
-- como PAGAMENTO_TOTAL,
-- e ordenar pelo novo nome PAGAMENTO_TOTAL
SELECT PRIMEIRO_NOME, SALÁRIO+BÔNUS+COMISS AS PAGAMENTO_TOTAL
FROM EMPREGADOS
ORDER BY PAGAMENTO_TOTAL;

-- criar um cursor atualizável através da cláusula FOR UPDATE
-- para atualizar as colunas data de início (PROJ_DATA_INÍCIO) e
-- data de término (PROJ_DATA_FIM) da tabela PROJETO
SELECT NUM_PROJ, PROJ_DATA_INÍCIO, PROJ_DATA_FIM
FROM PROJETO
FOR UPDATE OF PROJ_DATA_INÍCIO, PROJ_DATA_FIM;

-- definir o nível de isolamento como RR apenas para esta instrução
SELECT *
FROM VÔOS
WHERE ID_VÔO BETWEEN 'AA1111' AND 'AA1112'
WITH RR;
```

A instrução SELECT retorna um *ResultSet*. O *cursor* é um ponteiro para uma linha específica do *ResultSet*. Nos aplicativos Java, todos os *ResultSets* são cursores. O cursor é atualizável, ou seja, podem ser atualizadas e excluídas linhas ao se caminhar através do *ResultSet*, se a instrução SELECT que gerou o cursor e sua consulta subjacente atenderem aos requisitos de poder ser atualizável, conforme detalhado abaixo. Deve ser utilizada a cláusula FOR UPDATE quando se deseja gerar um cursor atualizável.

Note: A cláusula ORDER BY permite ordenar os resultados do SELECT. Sem a cláusula ORDER BY, os resultados são retornados em ordem aleatória.

Se a instrução SELECT atender aos requisitos listados abaixo, os cursores serão atualizáveis apenas se for especificado FOR UPDATE na cláusula FOR (consulte a [Cláusula FOR UPDATE](#)).

Requisitos para cursores atualizáveis e ResultSets atualizáveis

Somente os cursores simples, com SELECT em uma única tabela, e os *ResultSets* FORWARD_ONLY, podem ser atualizáveis. A instrução SELECT para *ResultSet* atualizável possui a mesma sintaxe que a instrução SELECT para cursor atualizável.

Para gerar cursores atualizáveis:

- A instrução SELECT não pode incluir a cláusula ORDER BY.
- A *Consulta* subjacente deve ser uma [ExpressãoSeleção](#).
- A [ExpressãoSeleção](#) da Consulta subjacente não pode incluir:
 - DISTINCT
 - Agregações
 - Cláusula GROUP BY
 - Cláusula HAVING
- A cláusula FROM na *Consulta* subjacente não pode ter:
 - mais de uma tabela
 - qualquer outra coisa além de um nome de tabela
 - [ExpressãoSeleção](#)
 - subconsultas

Não existe instrução na linguagem SQL para *atribuir* nome a um cursor. Em vez disso, deve ser utilizada a API do JDBC para atribuir nomes a cursores ou obter os nomes gerados pelo sistema. Para obter mais informações, deve ser consultado "Atribuir nome ou acessar o nome do cursor" no capítulo 5 do *Guia do Desenvolvedor do Derby*.

Os cursores são apenas-de-leitura por padrão. Para um cursor ser atualizável, deve ser especificado FOR UPDATE na cláusula FOR (consulte a [Cláusula FOR UPDATE](#)).

Sistema de dependência de instrução

A instrução SELECT depende de todas as tabelas e visões especificadas na consulta, e dos conglomerados (unidades de armazenamento, como *heaps* e índices) escolhidos como caminho de acesso para estas tabelas. A instrução CREATE INDEX não invalida a instrução SELECT preparada. A instrução DROP INDEX invalida a instrução SELECT preparada, se o índice for um caminho de acesso na instrução. Quando o SELECT inclui visões, também depende dos objetos do dicionário dos quais as visões dependem (consulte a [Instrução CREATE VIEW](#)).

Toda instrução UPDATE WHERE CURRENT ou DELETE WHERE CURRENT preparada sobre cursor de SELECT depende do SELECT. A remoção do SELECT através de uma instrução *java.sql.Statement.close* invalida UPDATE WHERE CURRENT e DELETE WHERE CURRENT.

O SELECT depende de todos os aliases utilizados na consulta. Remover um aliás invalida a instrução SELECT preparada, se a instrução utilizar o aliás.

ExpressãoTabela

A *ExpressãoTabela* especifica uma tabela ou visão na [Cláusula FROM](#). É a fonte de onde a [ExpressãoSeleção](#) seleciona o resultado.

Pode ser aplicado um nome de correlação para a tabela na *ExpressãoTabela*, para que suas colunas sejam qualificadas por este nome. Se não for especificado um nome de correlação, o nome da tabela qualificará o nome da coluna. Quando é especificado um nome de correlação para a tabela, não pode ser utilizado o nome da tabela para qualificar as colunas. Deve ser utilizado o nome da correlação quando se qualifica o

nome da coluna.

Não podem haver dois itens na cláusula FROM com o mesmo nome de correlação, e nenhum nome de correlação pode ser idêntico a um nome de tabela não qualificado especificado na cláusula FROM.

Além disso, podem ser dados novos nomes às colunas da tabela através da cláusula AS. Algumas situações onde é útil:

- Quando é utilizado [VALUES Expressão](#) na [SubconsultaTabela](#), uma vez que não há outra maneira de atribuir nomes às colunas de [VALUES Expressão](#).
- Quando os nomes das colunas de outra forma seriam idênticos aos das colunas de outra tabela; mudar os nomes significa que não é necessário qualificá-los.

A Consulta na [SubconsultaTabela](#) que aparece no *ItemFrom* pode conter várias colunas e retornar várias linhas. Consulte [SubconsultaTabela](#).

Para obter informações sobre as redefinições do otimizador que podem ser especificadas, consulte [Ajuste do Derby](#).

Sintaxe

```
{
ExpressãoTabelaOuVisão | Operação JOIN
}
```

```
-- selecionar de uma expressão de junção
SELECT E.NUM_EMP, E.ÚLTIMO_NOME, M.NUM_EMP, M.ÚLTIMO_NOME
FROM EMPREGADOS E LEFT OUTER JOIN
    DEPARTAMENTOS INNER JOIN EMPREGADOS M
    ON NUM_GER = M.NUM_EMP
    ON E.DEP_TRAB = NUM_DEP
```

ExpressãoTabelaOuVisão

```
{nome-da-tabela | nome-da-visão}
[ [ AS ] nome-da-correlação
  [ ( nome-de-coluna-simples [ , nome-de-coluna-simples ]* ) ] ] ]
```

SubconsultaTabela

A *SubconsultaTabela* é uma subconsulta que retorna várias linhas.

Diferentemente da [SubconsultaEscalar](#), a *SubconsultaTabela* é permitida apenas:

- como uma [ExpressãoTabela](#) em uma [Cláusula FROM](#)
- com EXISTS, IN, ou comparações quantificadas.

Quando utilizada como uma [ExpressãoTabela](#) em uma [Cláusula FROM](#), pode retornar várias colunas. Quando utilizada com EXISTS, pode retornar várias colunas somente se for utilizado o * para retornar várias colunas.

Quando utilizada com IN ou comparações quantificadas, deve retornar uma única coluna.

Sintaxe

```
(Consulta)
```

```
-- subconsulta utilizada como ExpressãoTabela na cláusula FROM
SELECT TABELA_VÔOS_VIRTUAL.ID_VÔO
FROM
    (SELECT ID_VÔO, AEROPORTO_ORIGEM, AEROPORTO_DESTINO
     FROM VÔOS
```

```

WHERE (AEROPORTO_ORIGEM = 'SFO' OR AEROPORTO_DESTINO = 'SCL') )
AS TABELA_VÔOS_VIRTUAL

-- subconsulta (VALUES expressão) utilizada como uma ExpressãoTabela
-- na cláusula FROM
SELECT MINHA_COLUNA1
FROM
    (VALUES (1, 2), (3, 4))
AS MINHA_TABELA (MINHA_COLUNA1, MINHA_COLUNA2)

-- subconsulta com EXISTS
SELECT *
FROM VÔOS
WHERE EXISTS
    (SELECT * FROM VÔOS WHERE AEROPORTO_DESTINO = 'SFO'
    AND AEROPORTO_ORIGEM = 'GRU')

-- subconsulta usada com IN
SELECT ID_VÔO, NÚMERO_SEGMENTO
FROM VÔOS
WHERE ID_VÔO IN
    (SELECT ID_VÔO
    FROM VÔOS WHERE AEROPORTO_ORIGEM = 'SFO'
    OR AEROPORTO_DESTINO = 'SCL')

-- subconsulta utilizada com uma comparação quantificada
SELECT NOME, COMISS
FROM EQUIPES
WHERE COMISS >
    (SELECT AVG(BÔNUS + 800)
    FROM EMPREGADOS
    WHERE COMISS < 5000);

```

Instrução UPDATE

A instrução UPDATE define o valor na coluna.

Pode ser atualizada a linha corrente de um cursor atualizável aberto. Se não houver linha corrente, ou a linha corrente não satisfizer mais a consulta do cursor, será lançada um exceção.

Sintaxe

```

{
    UPDATE nome-da-tabela
        SET nome-da-coluna = Valor
        [ , nome-da-coluna = Valor ] *
    [Cláusula WHERE] |
    UPDATE nome-da-tabela
        SET nome-da-coluna = Valor
        [ , nome-da-coluna = Valor ] *
        WHERE CURRENT OF
}

```

A primeira forma sintática é chamada de atualização procurada. A segunda forma sintática é chamada de atualização posicionada.

Nas atualizações procuradas, são atualizadas todas as linhas da tabela para as quais a cláusula WHERE é avaliada como TRUE.

Nas atualizações posicionadas, somente podem ser atualizadas as colunas incluídas na **Cláusula FOR UPDATE** da instrução SELECT que criou o cursor. Se a instrução SELECT não incluir a cláusula FOR UPDATE, o cursor será apenas-de-leitura, não podendo ser utilizado para atualizações.

A especificação de DEFAULT para o valor atualizado, define o valor da coluna como o valor padrão definido nesta tabela.

```

-- Todos os empregados, exceto o gerente,
-- do departamento (DEP_TRAB) 'E21' foram temporariamente

```

```
-- reatribuídos. Indique isto alterando seus cargos (CARGO)
-- para NULL, e os valores de seus pagamentos (SALÁRIO, BÔNUS, COMISS)
-- para zero, na tabela EMPREGADOS.
UPDATE EMPREGADOS
  SET CARGO=NULL, SALÁRIO=0, BÔNUS=0, COMISS=0
  WHERE DEP_TRAB = 'E21' AND CARGO <> 'GERENTE'

-- Promover o cargo (CARGO) de determinado empregado para GERENTE
UPDATE EMPREGADOS
  SET CARGO = 'GERENTE'
  WHERE CURRENT OF CURS1;

-- Multiplicar a equipe do projeto (PROJ_EQUIPE) por 1.5
stmt.executeUpdate("UPDATE PROJETO SET PROJ_EQUIPE = "
"PROJ_EQUIPE * 1.5" +
"WHERE CURRENT OF" + ResultSet.getCursorName());

-- Alterar o cargo (CARGO) do empregado número (NUM_EMP) '000290'
-- na tabela EMPREGADOS para o seu valor DEFAULT que é NULL
UPDATE EMPREGADOS
  SET CARGO = DEFAULT
  WHERE NUM_EMP = '000290';
```

Sistema de dependência de instrução

A instrução de atualização procurada depende da tabela sendo atualizada, de todos os seus conglomerados (unidades de armazenamento como *heaps* e índices), todas as suas restrições, e de todas as outras tabelas referenciadas na cláusula WHERE e nas expressões SET. A execução de uma instrução CREATE ou DROP INDEX, ou de uma instrução ALTER TABLE, na tabela de destino de uma instrução de atualização procurada preparada, invalida a instrução de atualização procurada preparada.

A instrução de atualização posicionada depende do cursor e de todas as tabelas que o cursor faz referência. A atualização posicionada pode ser compilada mesmo se o cursor ainda não tiver sido aberto. Entretanto, a remoção do cursor aberto através do método *close* do JDBC invalida a atualização posicionada.

A execução de uma instrução CREATE ou DROP INDEX, ou de uma instrução ALTER TABLE, na tabela de destino de uma instrução de atualização posicionada preparada, invalida a instrução de atualização posicionada preparada.

A remoção de um aliás invalida a instrução de atualização preparada, se a instrução utilizar o aliás.

Remover ou adicionar gatilhos na tabela de destino da atualização invalida a instrução de atualização.

Valor

Expressão | DEFAULT

VALUES Expressão

"VALUES Expressão" permite construir uma linha ou uma tabela a partir de valores. A instrução VALUES é utilizada quando não existe uma cláusula FROM. Esta construção pode ser utilizada em todos os locais onde uma consulta pode ser utilizada e, portanto, pode ser utilizada como uma instrução que retorna um *ResultSet*, dentro de expressões e instruções sempre que uma subconsulta for permitida, e como fonte de valores para a instrução INSERT.

Sintaxe

```
{
  VALUES ( Valor { , Valor }* )
  [ , ( Valor { , Valor }* ) ]* |
  VALUES Valor [ , Valor ]*
```

}

A primeira forma constrói linhas com várias colunas. A segunda forma constrói linhas com uma única coluna, com cada expressão sendo o valor da coluna da linha.

A palavra chave DEFAULT é permitida somente quando a expressão VALUES está em uma instrução INSERT. A especificação de DEFAULT para a coluna insere o valor padrão da coluna na coluna. Outra forma de inserir o valor padrão na coluna é omitir a coluna na lista de colunas, e somente inserir valores nas outras colunas da tabela.

```
-- 3 linhas de 1 coluna
VALUES (1),(2),(3);

-- 3 linhas de 1 coluna
VALUES 1, 2, 3;

-- 1 linha de 3 colunas
VALUES (1, 2, 3);

-- 3 linhas de 2 colunas
VALUES (1,21),(2,22),(3,23);

-- construção de uma tabela derivada
VALUES ('laranja', 'laranja'), ('maçã', 'vermelha'),
('banana', 'amarela')

-- Inserir dois novos departamentos na tabela DEPARTAMENTOS
-- utilizando uma instrução, mas não atribuir gerente
-- aos novos departamentos.
INSERT INTO DEPARTAMENTOS (NUM_DEP, NOME_DEP, ADMRDEPT)
VALUES ('B11', 'COMPRAS', 'B01'),
('E41', 'ADMINISTRAÇÃO DE BANCO DE DADOS', 'E01')

-- inserir uma linha com valor DEFAULT para a coluna PROJ_PRINC
INSERT INTO PROJETO (NUM_PROJ, NOME_PROJ, NUM_DEP, EMP_RESP,
PROJ_DATA_INÍCIO, PROJ_PRINC)
VALUES ('PL2101', 'ENSURE COMPAT PLAN', 'B01', '000020',
CURRENT_DATE, DEFAULT);

-- utilização de função nativa
VALUES CURRENT_DATE

-- obter o valor a partir de uma expressão arbitrária
VALUES (3*29, 26.0E0/3)

-- obter o valor retornado por uma função nativa
VALUES CHAR(1)
```

Valor

Expressão | DEFAULT

Cláusula WHERE

A cláusula WHERE é uma parte opcional da [Expressão Seleção](#), [Instrução DELETE](#) e da [Instrução UPDATE](#). A cláusula WHERE permite selecionar linhas baseado em uma expressão booleana. Somente as linhas para as quais a expressão é avaliada como TRUE são retornadas no resultado, ou no caso da instrução DELETE, excluídas, ou no caso da instrução UPDATE, atualizadas.

Sintaxe

WHERE [Expressão booleana](#)

São permitidas expressões booleanas na cláusula WHERE. A maioria das expressões gerais listadas na [Tabela de Expressões](#) podem resultar em um valor booleano.

Além disso, lá estão as expressões booleanas mais comuns. Os operadores booleanos específicos listados na Tabela 10 recebem um ou mais operandos; as expressões

retornam um valor booleano.

```
-- descobrir os vôos onde nenhum assento da
-- classe executiva foi reservado
SELECT *
FROM DISPONIBILIDADE_VÔO
WHERE ASSENTOS_EXECUTIVOS_OCUPADOS IS NULL
OR ASSENTOS_EXECUTIVOS_OCUPADOS = 0

-- Junção das tabelas ATIV_EMP e EMPREGADOS
-- selecionar todas as colunas da tabela ATIV_EMP, e adicionar o
-- sobrenome do empregado (ÚLTIMO_NOME) da tabela EMPREGADOS
-- a todas as linhas do resultado.
SELECT SAMP.ATIV_EMP.*, ÚLTIMO_NOME
FROM SAMP.ATIV_EMP, SAMP.EMPREGADOS
WHERE ATIV_EMP.NUM_EMP = EMPREGADOS.NUM_EMP;

-- Determinar o número do empregado e o salário dos representantes de
-- venda
-- junto com o salário médio e a conta principal de seus departamentos.
-- Esta consulta deverá criar primeiro um novo-nome-de-coluna
-- especificado
-- na cláusula AS, que está fora do FULLSELECT (DINFO),
-- para obter as colunas SALÁRIO_MÉDIO e CONTA_EMP,
-- assim como a coluna NUM_DEP utilizada na cláusula WHERE
SELECT THIS_EMP.NUM_EMP, THIS_EMP.SALÁRIO, DINFO.SALÁRIO_MÉDIO,
DINFO.CONTA_EMP
FROM EMPREGADOS THIS_EMP,
  (SELECT OUTROS.DEP_TRAB AS NUM_DEP,
    AVG(OUTROS.SALÁRIO) AS SALÁRIO_MÉDIO,
    COUNT(*) AS CONTA_EMP
   FROM EMPREGADOS OUTROS
  GROUP BY OUTROS.DEP_TRAB
  ) AS DINFO
WHERE THIS_EMP.CARGO = 'REP_VENDAS'
  AND THIS_EMP.DEP_TRAB = DINFO.NUM_DEP;
```

Cláusula WHERE CURRENT OF

A cláusula WHERE CURRENT OF é uma cláusula de algumas instruções UPDATE e DELETE. Permite realizar atualizações e exclusões posicionadas em cursores atualizáveis. Para obter mais informações sobre cursores atualizáveis deve ser consultada a [Instrução SELECT](#).

Sintaxe

```
WHERE CURRENT OF nome-do-cursor
```

```
Statement s = conn.createStatement();
s.setCursorName("ResultadosLinhasAereas");
ResultSet rs = conn.executeQuery(
    "SELECT LINHAS_AÉREA, TARIFA_BÁSICA " +
    "FROM LINHAS_AÉREAS FOR UPDATE OF TARIFA_BÁSICA");
Statement s2 = conn.createStatement();
s2.executeUpdate("UPDATE LINHAS_AÉREAS SET TARIFA_BÁSICA = TARIFA_BÁSICA
" +
    "+ .25 WHERE CURRENT OF ResultadosLinhasAereas");
```

Funções nativas

A função nativa é uma expressão na qual a palavra chave do SQL ou o operador especial executa alguma operação. As funções nativas utilizam palavras chave ou operadores nativos especiais. As funções nativas são IdentificadoresSQL92 e não diferenciam letras maiúsculas e minúsculas. Deve ser observado que as funções em escape, como TIMESTAMPADD e TIMESTAMPDIFF, são acessíveis apenas utilizando a sintaxe de escape do JDBC para funções, podendo ser encontradas em [Sintaxe de escape do JDBC](#).

Funções nativas padrão

- ABS ou ABSVAL
- BIGINT
- CAST
- CHAR
- Concatenação
- Expressões NULLIF e CASE
- CURRENT_DATE
- CURRENT ISOLATION
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER
- DATE
- DAY
- DOUBLE
- HOUR
- IDENTITY_VAL_LOCAL
- INTEGER
- LENGTH
- LOCATE
- LCASE ou LOWER
- LTRIM
- MINUTE
- MOD
- MONTH
- RTRIM
- SECOND
- SESSION_USER
- SMALLINT
- SQRT
- SUBSTR
- TIME
- TIMESTAMP
- UCASE ou UPPER
- USER
- VARCHAR
- YEAR

Agregações (funções de conjunto)

Esta seção descreve as agregações (também descritas como *funções de conjunto* no ANSI SQL-92 e como *funções de coluna* em algumas literaturas sobre banco de dados). As agregações fornecem meios de avaliar uma expressão para um conjunto de linhas. Enquanto as outras funções nativas operam sobre uma única expressão, as funções de agregação operam sobre conjuntos de valores reduzindo-os a um único valor escalar. As agregações nativas podem calcular o valor mínimo, o valor máximo, a soma e a média de uma expressão com relação a um conjunto de valores, assim como contar linhas. Também podem ser criadas agregações do usuário para executar outras funções de conjunto, como calcular o desvio padrão.

As agregações nativas podem operar nos tipos de dado mostrados em [Tipos de dado permitidos pelas agregações nativas](#).

Tabela 2. Tipos de dado permitidos pelas agregações nativas

	Todos os tipos	Tipos de dado numéricos nativos
COUNT	X	X
MIN		X

	Todos os tipos	Tipos de dado numéricos nativos
MAX	'	X
AVG	'	X
SUM	'	X

As agregações são permitidas apenas:

- No *ItemSeleção* da [ExpressãoSeleção](#).
- Na [Cláusula HAVING](#).
- Na [Cláusula ORDER BY](#) (utilizando um nome aliás) se a agregação aparecer no resultado do bloco de consulta relevante. Ou seja, é permitido um aliás para uma agregação na [Cláusula ORDER BY](#) se, e somente se, a agregação aparecer no *ItemSeleção* da [ExpressãoSeleção](#).

Todas as expressões nos *ItensSelecionados* da [ExpressãoSeleção](#) devem ser agregações ou colunas agrupadas (consulte a [Cláusula GROUP BY](#)) (O mesmo é verdade se houver uma cláusula HAVING sem uma cláusula GROUP BY). Isto se deve ao fato do *ResultSet* da [ExpressãoSeleção](#) dever ser um escalar (valor único) ou um vetor (vários valores), mas não a uma mistura dos dois (as agregações têm como resultado um valor escalar, e a referência a uma coluna pode ter como resultado um vetor). Por exemplo, a consulta a seguir mistura valor escalar e vetor e, portanto, não é válida:

```
-- inválido
SELECT MIN(TEMPO_VÔO), ID_VÔO
FROM VÔOS
```

Não são permitidas agregações em referências externas (correlações). Isto significa que se a subconsulta contiver uma agregação, esta agregação não poderá avaliar uma expressão que inclua uma referência a uma coluna do bloco de consulta externo. Por exemplo, a seguinte consulta não é válida porque SUM opera sobre uma coluna da consulta externa:

```
SELECT c1
FROM t1
GROUP BY c1
HAVING c2 >
  (SELECT t2.x
   FROM t2
   WHERE t2.y = SUM(t1.c3))
```

O cursor declarado em um *ResultSet* que inclui uma agregação no bloco de consulta externo não é atualizável.

Esta seção inclui as seguintes agregações:

- [AVG](#)
- [COUNT](#)
- [MAX](#)
- [MIN](#)
- [SUM](#)

ABS ou ABSVAL

ABS, ou ABSVAL, retorna o valor absoluto da expressão numérica. O tipo retornado é o tipo do parâmetro. São suportados todos os tipos numéricos nativos ([DECIMAL](#), [DOUBLE PRECISION](#), [FLOAT](#), [INTEGER](#), [BIGINT](#), [NUMERIC](#), [REAL](#) e [SMALLINT](#)).

Sintaxe

```
ABS(ExpressãoNumérica)
```

```
-- retorna 3
VALUES ABS(-3)
```

AVG

AVG é uma função de agregação que calcula a média de uma expressão sobre um conjunto de linhas (consulte [Agregações \(funções de conjunto\)](#)). AVG somente é permitida nas expressões que têm como resultado tipos de dado numéricos.

Sintaxe

```
AVG ( [ DISTINCT | ALL ] Expressão )
```

O qualificador DISTINCT elimina as linhas duplicadas. O qualificador ALL mantém as linhas duplicadas. ALL é o valor padrão, se nem ALL nem DISTINCT for especificado. Por exemplo, se uma coluna contiver os valores 1.0, 1.0, 1.0, 1.0 e 2.0, AVG(coluna) retornará um valor menor que AVG(DISTINCT coluna).

Somente é permitida uma expressão de agregação DISTINCT por [ExpressãoSeleção](#). Por exemplo, a seguinte consulta não é válida:

```
SELECT AVG (DISTINCT TEMPO_VÔO), SUM (DISTINCT MILHAS)
FROM VÔOS
```

A expressão pode conter várias referências a colunas ou expressões, mas não pode conter outra agregação ou subconsulta. Deve ter como resultado um tipo de dado numérico do SQL-92. Portanto, podem ser chamados métodos que têm como resultado tipos de dado SQL-92. Se uma expressão for avaliada como NULL, a agregação saltará este valor.

O tipo de dado do resultado é o mesmo da expressão em que opera (nunca estoura). Por exemplo, a seguinte consulta retorna o INTEGER 1, que pode não ser o esperado:

```
SELECT AVG(C1)
FROM (VALUES (1), (1), (1), (1), (2)) AS MINHA_TABELA (C1)
```

Se for desejada uma precisão maior, a expressão deverá ser convertida para outro tipo de dado:

```
SELECT AVG(CAST (C1 AS DOUBLE PRECISION))
FROM (VALUES (1), (1), (1), (1), (2)) AS MINHA_TABELA (c1)
```

BIGINT

A função BIGINT retorna a representação inteira de 64 bits do número ou da cadeia de caracteres na forma de uma constante inteira.

Sintaxe

```
BIGINT (ExpressãoCaractere | ExpressãoNumérica )
```

ExpressãoCaractere

Uma expressão que retorna um valor cadeia de caracteres, de comprimento não maior que o comprimento máximo da constante caractere. Os brancos iniciais e finais são eliminados, e a cadeia resultante deve estar em conformidade com as regras de formação de uma constante inteira do SQL. A cadeia de caracteres não pode ser uma cadeia longa. Se o argumento for uma *ExpressãoCaractere*, o resultado será o

mesmo número que seria produzido se a constante inteira correspondente fosse atribuída a uma coluna ou a uma variável inteira grande.

ExpressãoNumérica

Uma expressão que retorna um valor de qualquer tipo de dado numérico nativo. Se o argumento for uma *ExpressãoNumérica*, o resultado será o mesmo número que seria produzido se o argumento fosse atribuído a uma coluna ou a uma variável inteira grande. Se a parte inteira do argumento não estiver dentro da faixa dos inteiros grandes, ocorrerá um erro. A parte decimal do argumento será truncada, se estiver presente.

O resultado da função é um inteiro grande. Se o argumento puder ser nulo, o resultado poderá ser nulo; Se o argumento for nulo, o resultado será o valor nulo.

Utilizando a tabela EMPREGADOS selecionar a coluna NUM_EMP na forma de inteiro grande para processamento posterior no aplicativo:

```
SELECT BIGINT (NUM_EMP) FROM EMPREGADOS
```

CAST

CAST converte o valor de um tipo de dado para outro, e fornece tipo de dado para o parâmetro dinâmico (?) e para o valor NULL.

As expressões CAST são permitidas em qualquer lugar onde uma expressão é permitida.

Sintaxe

```
CAST ( [ Expressão | NULL | ? ]
      AS TipoDado)
```

O tipo de dado para o qual a expressão está sendo convertida é o *tipo de destino*. O tipo de dado da expressão a partir da qual está sendo feita a conversão é o *tipo de origem*.

Conversões CAST entre tipos de dado SQL-92

A tabela a seguir mostra as conversões explícitas válidas entre tipos de origem e tipos de destino para os tipos de dado do SQL.

Tabela 3. Conversões explícitas entre tipos de origem e tipos de destino para os tipos de dado do SQL

Esta tabela mostra quais conversões explícitas entre tipos de dado são válidas. A primeira coluna da tabela lista o tipo de origem, enquanto a primeira linha lista os tipos de destino. "S" indica que a conversão da origem para o destino é válida.

Tipos	S M A L L I N T	I N T E G E R	B I G I N T	D E C I M A L	R E A L	D O U B L E	F L O A T	C H A R	V A R C H A R	L O N G V A R C H A R	C H A R F O R B I T D A T A	V A R C H A R F O R B I T D A T A	L O N G V A R C H A R F O R B I T D A T A	C L O B	B L O B	D A T E	T I M E	T I M E S T A M P
SMALLINT	S	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-
INTEGER	S	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-
BIGINT	S	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-
DECIMAL	S	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-
REAL	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
DOUBLE	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
FLOAT	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
CHAR	S	S	S	S	-	-	-	S	S	S	-	-	-	S	-	S	S	S
VARCHAR	S	S	S	S	-	-	-	S	S	S	-	-	-	S	-	S	S	S
LONG VARCHAR	-	-	-	-	-	-	-	S	S	S	-	-	-	S	-	-	-	-
CHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	S	S	S	S	S	-	-	-
VARCHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	S	S	S	S	S	-	-	-
LONG VARCHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	S	S	S	S	S	-	-	-
CLOB	-	-	-	-	-	-	-	S	S	S	-	-	-	S	-	-	-	-
BLOB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	S	-	-	-
DATE	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	S	-	S
TIME	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	-	S	S
TIME STAMP	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	S	S	S

Se a conversão for válida, então CAST será permitida. Incompatibilidades de tamanhos entre os tipos de origem e de destino podem causar erros em tempo de execução.

Notas

Nesta discussão, os tipos de dado SQL-92 do Derby são categorizados da seguinte maneira:

- *numérico*
 - numérico exato (SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC)
 - numérico aproximado (FLOAT, REAL, DOUBLE PRECISION)
- *cadeia*
 - cadeia de caracteres (CLOB, CHAR, VARCHAR, [LONG VARCHAR](#))
 - cadeia de bits (BLOB, CHAR FOR BIT DATA, VARCHAR FOR BIT DATA, [LONG VARCHAR FOR BIT DATA](#))
- *data/hora*
 - [DATE](#)
 - [TIME](#)
 - [TIMESTAMP](#)

Conversões de tipos de dado numéricos

Um tipo numérico pode ser convertido para qualquer outro tipo numérico. Se o tipo de destino não puder representar o componente não-fracionário sem truncamento, será lançada uma exceção. Se o destino numérico não puder representar o componente fracionário (escala) da origem numérica, então a origem será truncada em silêncio para caber no destino. Por exemplo, converter 763.1234 para INTEGER tem como resultado 763.

Conversões de/para cadeias de bits

As cadeias de bits podem ser convertidas para outras cadeias de bits, mas não para cadeias de caracteres. As cadeias que são convertidas para cadeias de bits são preenchidas com zero no final, para se ajustarem ao tamanho da cadeia de bits do destino. O tipo BLOB é mais limitado, e requer conversão explícita. Na maioria dos casos o tipo BLOB não pode ser convertido de, ou para, outros tipos.

Conversões de valores de data/hora

Um valor data/hora sempre pode ser convertido de e para TIMESTAMP. Se DATE for convertido para TIMESTAMP, o componente TIME do TIMESTAMP resultante será sempre 00:00:00. Se um valor de dado TIME for convertido para TIMESTAMP, o componente DATE de TIMESTAMP será definido como o valor de CURRENT_DATE do momento da execução de CAST. Se TIMESTAMP for convertido para DATE, o componente TIME será truncado em silêncio. Se TIMESTAMP for convertido para TIME, o componente DATE será truncado em silêncio.

```
SELECT CAST (MILHAS AS INT)
FROM VÔOS

-- converter de carimbo do tempo para texto
INSERT INTO MINHA_TABELA (COLUNA_TEXTO)
VALUES (CAST (CURRENT_TIMESTAMP AS VARCHAR(100)))

-- NULL deve ser convertido para algum tipo de dado para poder ser usado
SELECT LINHA_AEREA
FROM LINHAS_AEREAS
UNION ALL
VALUES (CAST (NULL AS CHAR(2)))

-- conversão de precisão dupla para decimal
SELECT CAST (TEMPO_VOO AS DECIMAL(5,2))
FROM VÔOS

-- conversão de SMALLINT para BIGINT
VALUES CAST (CAST (12 AS SMALLINT) AS BIGINT)
```

CHAR

A função CHAR retorna a representação cadeia de caracteres de comprimento fixo de:

- uma cadeia de caracteres, se o primeiro argumento for algum tipo de cadeia de caracteres.
- um valor data/hora, se o primeiro argumento for DATE, TIME ou TIMESTAMP.
- um número decimal, se o primeiro argumento for um número decimal.
- um número de ponto flutuante de precisão dupla, se o primeiro argumento for DOUBLE ou REAL.
- um número inteiro, se o primeiro argumento for SMALLINT, INTEGER ou BIGINT.

O primeiro argumento deve ser de um tipo de dado nativo. O resultado da função é uma cadeia de caracteres de comprimento fixo. Se o primeiro argumento puder ser nulo, o resultado poderá ser nulo. Se o primeiro argumento for nulo, o resultado será o valor nulo.

Sintaxe caractere para caractere

```
CHAR (ExpressãoCaractere [, inteiro] )
```

ExpressãoCaractere

Uma expressão que retorna um valor do tipo de dado CHAR, VARCHAR, LONG VARCHAR ou CLOB.

inteiro

O atributo comprimento da cadeia de caracteres de comprimento fixo resultante. O valor deve estar entre 0 e 254.

Se o comprimento da expressão caractere for menor que o atributo comprimento do resultado, o resultado será preenchido com brancos até o comprimento do resultado. Se o comprimento da expressão caractere for maior que o atributo comprimento do resultado, será realizado um truncamento. Retorna uma advertência, a menos que os caracteres truncados sejam todos brancos e a expressão caractere não seja uma cadeia longa (LONG VARCHAR ou CLOB).

Sintaxe inteiro para caractere

```
CHAR (ExpressãoInteira)
```

ExpressãoInteira

Uma expressão que retorna um valor com tipo de dado inteiro (SMALLINT, INTEGER ou BIGINT).

O resultado é a representação cadeia de caracteres do argumento na forma de uma constante inteira do SQL. O resultado consiste de n caracteres, que são os dígitos significativos que representam o valor do argumento, precedidos pelo sinal menos se o argumento for negativo. É alinhado à esquerda.

- Se o primeiro argumento for um SMALLINT: O comprimento do resultado será igual a 6. Se o número de caracteres do resultado for menor que 6, então o resultado será preenchido à direita com brancos até o comprimento 6.
- Se o primeiro argumento for um INTEGER: O comprimento do resultado será igual 11. Se o número de caracteres no resultado for menor que 11, então o resultado será preenchido à direita com brancos até o comprimento 11.
- Se o primeiro argumento for um BIGINT: O comprimento do resultado será igual a 20. Se o número de caracteres no resultado for menor que 20, então o resultado será preenchido à direita com brancos até o comprimento 20.

Sintaxe de data/hora para caractere

```
CHAR (ExpressãoDataHora)
```

ExpressãoDataHora

Uma expressão de um dos três seguintes tipos de dado:

- **DATE:** O resultado é a representação caractere da data. O comprimento do

resultado é 10.

- **TIME**: O resultado é a representação caractere da hora. O comprimento do resultado é 8.
- **TIMESTAMP**: O resultado é a representação caractere do carimbo do tempo. O comprimento do resultado é 26.

Decimal para caractere

CHAR (*ExpressãoDecimal*)

ExpressãoDecimal

Uma expressão que retorna um valor com tipo de dado decimal. Se for desejada uma precisão e escala diferente, poderá ser utilizado primeiro a função escalar DECIMAL para fazer a alteração.

Sintaxe ponto flutuante para caractere

CHAR (*ExpressãoPontoFlutuante*)

ExpressãoPontoFlutuante

Uma expressão que retorna um valor como tipo de dado de ponto flutuante (DOUBLE ou REAL).

Utilizar a função CHAR para retornar os valores para NÍVEL_EDUC (definido como SMALLINT) como uma cadeia de caracteres de comprimento fixo:

```
SELECT CHAR(NÍVEL_EDUC) FROM EMPREGADOS
```

Um NÍVEL_EDUC igual a 18 retorna como o valor CHAR(6) igual a '18 ' (18 seguido por quatro brancos).

LENGTH

LENGTH é aplicado tanto a expressões cadeia de caracteres quanto a expressões cadeia de bits, e retorna o número de caracteres no resultado.

Como todos os tipos de dado nativos são convertidos implicitamente em cadeias, esta função pode atuar em todos os tipos de dados nativos.

Sintaxe

LENGTH ({ *ExpressãoCaractere* | *ExpressãoBit* })

```
-- retorna 20
VALUES LENGTH('supercalifragilistic')

-- retorna 1
VALUES LENGTH(X'FF')

-- retorna 4
VALUES LENGTH(1234567890)
```

Concatenação

O operador de concatenação, ||, concatena seu operando à direita ao final do seu operando à esquerda. Opera em expressão de caractere e de bit.

Como todos os tipos de dado nativos são convertidos implicitamente em cadeias, esta função pode atuar em todos os tipos de dado nativos.

Sintaxe

{

```
{ ExpressãoCaractere || ExpressãoCaractere } |
{ ExpressãoBit || ExpressãoBit }
```

Nas cadeias de caracteres, se os operandos esquerdo e direito forem do tipo CHAR, o tipo do resultado será CHAR; senão, será VARCHAR. As regras normais de preenchimento/corte para CHAR e VARCHAR se aplicam ao resultado deste operador.

O comprimento da cadeia resultante é a soma dos comprimentos dos dois operandos.

Para as cadeias de bits, se os operandos esquerdo e direito forem do tipo CHAR FOR BIT DATA, o tipo do resultado será CHAR FOR BIT DATA; senão, será VARCHAR FOR BIT DATA.

```
--retorna 'supercalifragilisticexbealidocious(sp?)'
VALUES 'supercalifragilistic' || 'exbealidocious' || '(sp?)'

-- retorna NULL
VALUES CAST (NULL AS VARCHAR(7)) || 'UmaCadeiaDeCaracteres'

-- retorna '130asdf'
VALUES '130' || 'asdf'
```

Expressões NULLIF e CASE

São utilizadas as expressões CASE e NULLIF para as expressões condicionais no Derby.

Sintaxe da expressão NULLIF

```
NULLIF(L,R)
```

A expressão NULLIF é muito semelhante à expressão CASE. Por exemplo:

```
NULLIF(V1,V2)
```

equivale à seguinte expressão CASE:

```
CASE WHEN V1=V2 THEN NULL ELSE V1 END
```

Sintaxe da expressão CASE

A expressão CASE pode ser colocada em qualquer lugar onde é permitida uma expressão. Escolhe a expressão a ser avaliada baseado em um teste booleano.

```
CASE WHEN ExpressãoBooleana THEN ExpressãoThen ELSE ExpressãoElse END
```

A *ExpressãoThen* e a *ExpressãoElse* são expressões que devem possuir tipos compatíveis. Para os tipos nativos, isto significa que estes tipos devem ser o mesmo, ou deve haver uma conversão nativa de alargamento entre os tipos.

Não é necessário utilizar a expressão CASE para evitar *NullPointerExceptions* quando uma coluna que pode ter nulo se torna receptora do método.

```
-- retorna 3
VALUES CASE WHEN 1=1 THEN 3 ELSE 4 END;
```

Se o valor da instância especificada na chamada do método de instância for nulo, o resultado da chamada será nulo (SQL NULL). Entretanto, ainda é necessário utilizar a

expressão CASE para uma coluna que pode ter nulo, quando a coluna for o parâmetro de um método primitivo.

COUNT

COUNT é uma função de agregação que conta o número de linhas acessadas pela expressão (consulte [Agregações \(funções de conjunto\)](#)). COUNT é permitida em todos os tipos de expressão.

Sintaxe

```
COUNT ( [ DISTINCT | ALL ] Expressão )
```

O qualificador DISTINCT elimina as linhas duplicadas. O qualificador ALL mantém as linhas duplicadas. É assumido ALL se não for especificado nem ALL nem DISTINCT. Por exemplo, se uma coluna contiver os valores 1, 1, 1, 1 e 2, COUNT(coluna) retornará um valor maior que COUNT(DISTINCT coluna).

Somente é permitida uma expressão de agregação DISTINCT por [ExpressãoSeleção](#). Por exemplo, a seguinte consulta não é permitida:

```
-- consulta não permitida
SELECT COUNT (DISTINCT TEMPO_VÔO), SUM (DISTINCT MILHAS)
FROM VÔOS
```

A expressão pode conter várias referências a colunas ou expressões, mas não pode conter outra agregação ou subconsulta. Se a *Expressão* resultar em NULL, a agregação não será processada para este valor.

O tipo de dado do resultado de COUNT é [BIGINT](#).

```
-- Contar o número de países em cada região,
-- mostrando apenas as regiões que possuem ao menos 2 países
SELECT COUNT (PAÍS), REGIÃO
FROM PAÍSES
GROUP BY REGIÃO
HAVING COUNT (PAÍS) > 1
```

COUNT(*)

COUNT(*) é uma função de agregação que conta o número de linhas acessadas. Não são eliminados nulos nem linhas duplicadas. COUNT(*) não opera em uma expressão.

Sintaxe

```
COUNT ( * )
```

O tipo de dado do resultado é [BIGINT](#).

```
-- Contar o número de linhas da tabela VÔOS
SELECT COUNT ( * )
FROM VÔOS
```

CURRENT DATE

CURRENT DATE é sinônimo de [CURRENT_DATE](#).

CURRENT_DATE

CURRENT_DATE retorna a data corrente; o valor retornado não muda se for executado mais de uma vez na mesma instrução. Isto significa que o valor é fixo, mesmo que haja uma longa demora entre as linhas trazidas pelo cursor.

Sintaxe

```
CURRENT_DATE
```

ou, como alternativa

```
CURRENT DATE
```

```
-- descobrir os vôos futuros disponíveis:  
SELECT * FROM VÔOS_DISPONIBILIDADE WHERE DATA_VÔO > CURRENT_DATE;
```

CURRENT ISOLATION

CURRENT ISOLATION retorna o nível de isolamento corrente como um valor CHAR(2) contendo ""(branco), "UR", "CS", "RS" ou "RR".

Sintaxe

```
CURRENT ISOLATION
```

```
VALUES CURRENT ISOLATION
```

CURRENT SCHEMA

CURRENT SCHEMA retorna o nome do esquema utilizado para qualificar as referências a objetos do banco de dados não qualificadas.

Note: CURRENT SCHEMA e CURRENT SQLID são sinônimos.

Estas funções retornam uma cadeia de até 128 caracteres.

Sintaxe

```
CURRENT SCHEMA
```

```
-- ou, como alternativa:
```

```
CURRENT SQLID
```

```
-- Fazer com que a coluna NOME tenha como valor padrão  
-- o esquema corrente:  
CREATE TABLE MINHA_TABELA (ID INT, NOME VARCHAR(128) DEFAULT CURRENT  
SQLID)  
  
-- Inserir o valor padrão igual ao valor do  
-- esquema corrente na tabela:  
INSERT INTO MINHA_TABELA(ID) VALUES (1)  
  
-- Retornar as linhas com o mesmo nome do esquema corrente:  
SELECT NOME FROM MINHA_TABELA WHERE NOME = CURRENT SCHEMA
```

CURRENT TIME

CURRENT TIME é sinônimo de [CURRENT_TIME](#).

CURRENT_TIME

`CURRENT_TIME` retorna a hora corrente; o valor retornado não muda se for executado mais de uma vez na mesma instrução. Isto significa que o valor é fixo, mesmo que haja uma longa demora entre as linhas trazidas pelo cursor.

Sintaxe

```
CURRENT_TIME
```

ou, como alternativa

```
CURRENT TIME
```

```
VALUES CURRENT_TIME
-- ou, como alternativa:
VALUES CURRENT TIME
```

CURRENT_TIMESTAMP

`CURRENT_TIMESTAMP` é sinônimo de `CURRENT_TIMESTAMP`.

CURRENT_TIMESTAMP

`CURRENT_TIMESTAMP` retorna o carimbo do tempo corrente; o valor retornado não muda se for executado mais de uma vez na mesma instrução. Isto significa que o valor é fixo, mesmo que haja uma longa demora entre as linhas trazidas pelo cursor.

Sintaxe

```
CURRENT_TIMESTAMP
```

ou, como alternativa

```
CURRENT TIMESTAMP
```

```
VALUES CURRENT_TIMESTAMP
-- ou, como alternativa:
VALUES CURRENT TIMESTAMP
```

CURRENT_USER

`CURRENT_USER` retorna o identificador de autorização do usuário corrente (o nome de usuário passado quando o usuário se conectou ao banco de dados). Se não houver usuário corrente, será retornado *APP*.

`USER` e `SESSION_USER` são sinônimos.

Estas funções retornam uma cadeia de até 128 caracteres.

Sintaxe

```
CURRENT_USER
```

```
VALUES CURRENT_USER
```

DATE

A função DATE retorna a data de um valor. O argumento deve ser uma data, carimbo do tempo, um número positivo menor ou igual a 3.652.059, uma representação cadeia de caracteres válida de uma data ou carimbo do tempo, ou uma cadeia de caracteres de comprimento 7, que não seja CLOB ou LONG VARCHAR. Se o argumento for uma cadeia de caracteres de comprimento 7, deverá representar uma data válida na forma `yyyynnn`, onde `yyyy` são os dígitos que representam o ano, e `nnn` são os dígitos entre 001 e 366 que representam o dia do ano. O resultado da função é uma data. Se o argumento puder ser nulo, o resultado poderá ser nulo; Se o argumento for nulo, o resultado será o valor nulo.

As outras regras dependem do tipo de dado do argumento especificado:

- Se o argumento for uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo: O resultado será a parte data do valor.
- Se o argumento for um número: O resultado será a data `n-1` dia após 1 de janeiro de 0001, onde `n` é a parte inteira do número.
- Se o argumento for uma cadeia de caracteres com comprimento 7: O resultado será a data representada pela cadeia de caracteres.

Sintaxe

```
DATE ( expressão )
```

Este exemplo resulta na representação interna de '1988-12-25'.

```
VALUES DATE('1988-12-25')
```

DAY

A função DAY retorna a parte relativa ao dia do valor. O argumento deve ser uma data, carimbo do tempo, ou uma representação cadeia de caracteres válida de uma data ou carimbo do tempo, que não seja CLOB ou LONG VARCHAR. O resultado da função é inteiro. Se o argumento puder ser nulo, o resultado poderá ser nulo; se o argumento for nulo, o resultado será o valor nulo.

As outras regras dependem do tipo de dado do argumento especificado:

- Se o argumento for uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo: O resultado será a parte relativa ao dia do valor, que é um inteiro entre 1 e 31.
- Se o argumento for uma duração de tempo ou duração de carimbo do tempo: O resultado será a parte relativa ao dia do valor, que é um inteiro entre -99 e 99. Um resultado diferente de zero possui o mesmo sinal do argumento.

Sintaxe

```
DAY ( expressão )
```

```
VALUES DAY('2005-08-02');
```

O valor do resultado é 2.

DOUBLE

A função DOUBLE retorna um número de ponto flutuante correspondente a:

- número, se o argumento for uma expressão numérica.
- representação cadeia de caracteres do número, se o argumento for uma expressão cadeia de caracteres.

Numérico para precisão dupla

```
DOUBLE [PRECISION] (ExpressãoNumérica )
```

ExpressãoNumérica

O argumento é uma expressão que retorna um valor de qualquer tipo de dado numérico nativo.

O resultado da função é um número de ponto flutuante de precisão dupla. Se o argumento puder ser nulo, o resultado poderá ser nulo; se o argumento for nulo, o resultado será o valor nulo. O resultado é o mesmo número que seria produzido se o argumento fosse atribuído a uma coluna ou variável de ponto flutuante de precisão dupla.

Cadeia de caracteres para precisão dupla

```
DOUBLE (ExpressãoCadeiaCaracteres )
```

ExpressãoCadeiaCaracteres

O argumento pode ser do tipo CHAR ou VARCHAR na forma de uma constante numérica. Os brancos iniciais e finais no argumento são ignorados.

O resultado da função é um número de ponto flutuante de precisão dupla. O resultado pode ser nulo; se o argumento for nulo, o resultado será o valor nulo. O resultado é o mesmo número que seria produzido se a cadeia de caracteres fosse considerada uma constante e atribuída a uma coluna ou variável de ponto flutuante de precisão dupla.

HOUR

A função HOUR retorna a parte relativa à hora do valor. O argumento deve ser uma hora, carimbo do tempo, ou a representação cadeia de caracteres válida de uma hora ou carimbo do tempo, que não seja CLOB nem LONG VARCHAR. O resultado da função é um inteiro. Se o argumento puder ser nulo, o resultado poderá ser nulo; se o argumento for nulo, o resultado será o valor nulo.

As outras regras dependem do tipo de dado do argumento especificado:

- Se o argumento for uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo: O resultado será a parte relativa à hora do valor, que é um inteiro entre 0 e 24.
- Se o argumento for uma duração de hora ou duração de carimbo do tempo: O resultado será a parte relativa à hora do valor, que é um inteiro entre -99 e 99. Um resultado diferente de zero possui o mesmo sinal do argumento.

Sintaxe

```
HOUR ( expressão )
```

Selecionar as aulas que começam no período da tarde na tabela chamada TABELA1.

```
SELECT * FROM TABELA1  
WHERE HOUR(INÍCIO) BETWEEN 12 AND 17
```

IDENTITY_VAL_LOCAL

O Derby suporta a função IDENTITY_VAL_LOCAL.

Sintaxe:

```
IDENTITY_VAL_LOCAL ( )
```

A função `IDENTITY_VAL_LOCAL` é uma função não determinística, que retorna o valor atribuído mais recentemente a uma coluna identidade para a conexão, onde a atribuição ocorreu como resultado de uma instrução `INSERT` de uma única linha utilizando uma cláusula `VALUES`.

A função `IDENTITY_VAL_LOCAL` não possui parâmetros de entrada. O resultado é um `DECIMAL (31,0)`, independentemente do tipo de dado real da coluna identidade correspondente.

O valor retornado pela função `IDENTITY_VAL_LOCAL`, para a conexão, é o valor atribuído à coluna identidade da tabela identificada na instrução `INSERT` de uma única linha mais recente. A instrução `INSERT` deve conter a cláusula `VALUES` na tabela que contém a coluna de identidade. O valor atribuído é um valor identidade gerado pelo Derby. A função retorna o valor nulo quando não tiver sido emitida uma instrução `INSERT` de uma única linha com a cláusula `VALUES` para uma tabela contendo coluna identidade.

O resultado da função não é afetado por:

- Uma instrução `INSERT` de uma única linha com a cláusula `VALUES` para uma tabela sem coluna identidade
- Uma instrução `INSERT` de várias linhas com a cláusula `VALUES`
- Uma instrução `INSERT` com um `FULLSELECT`

Se a tabela com a coluna identidade possuir um gatilho para `INSERT` definido, que insere em outra tabela com outra coluna identidade, então a função `IDENTITY_VAL_LOCAL()` retornará o valor gerado para a tabela da instrução, e não para a tabela modificada pelo gatilho.

Exemplos:

```
ij> CREATE TABLE T1(C1 INT GENERATED ALWAYS AS IDENTITY, C2 INT);
0 rows inserted/updated/deleted
ij> INSERT INTO T1(C2) VALUES (8);
1 row inserted/updated/deleted
ij> VALUES IDENTITY_VAL_LOCAL();
1
-----
1
1 row selected
ij> SELECT IDENTITY_VAL_LOCAL()+1, IDENTITY_VAL_LOCAL()-1 FROM T1;
1 | 2
-----
2 | 0
1 row selected
ij> INSERT INTO T1(C2) VALUES (IDENTITY_VAL_LOCAL());
1 row inserted/updated/deleted
ij> SELECT * FROM T1;
C1 | C2
-----
1 | 8
2 | 1
2 rows selected
ij> VALUES IDENTITY_VAL_LOCAL();
1
-----
2
1 row selected
ij> INSERT INTO T1(C2) VALUES (8), (9);
2 rows inserted/updated/deleted
ij> -- inserção de vários valores;
-- valor retornado pela função não deve mudar
VALUES IDENTITY_VAL_LOCAL();
1
```

```

-----
2
1 row selected
ij> SELECT * FROM T1;
C1          | C2
-----
1           | 8
2           | 1
3           | 8
4           | 9
4 rows selected
ij> INSERT INTO T1(C2) SELECT C1 FROM T1;
4 rows inserted/updated/deleted
ij> -- inserção som sub-seleção;
-- valor retornado pela função não deve mudar
VALUES IDENTITY_VAL_LOCAL();
1
-----
2
1 row selected
ij> SELECT * FROM T1;
C1          | C2
-----
1           | 8
2           | 1
3           | 8
4           | 9
5           | 1
6           | 2
7           | 3
8           | 4
8 rows selected

```

INTEGER

A função **INTEGER** retorna a representação inteira de um número, cadeia de caracteres, data ou hora na forma de uma constante inteira.

Sintaxe

```
INT[EGER] ( ExpressãoNumérica | ExpressãoCaractere )
```

ExpressãoNumérica

Uma expressão que retorna um valor de qualquer tipo de dado numérico nativo. Se argumento for uma *ExpressãoNumérica*, o resultado será o mesmo número que seria produzido se o argumento fosse atribuído a uma coluna ou variável inteira. Se a parte inteira do argumento não estiver dentro da faixa dos inteiros, ocorrerá um erro. A parte decimal do argumento será truncada, se estiver presente.

ExpressãoCaractere

Uma expressão que retorna um valor cadeia de caracteres, de comprimento não maior que o comprimento máximo de uma constante caractere. Os brancos iniciais e finais são eliminados, e a cadeia resultante deve estar em conformidade com as regras de formação de uma constante inteira do SQL. A cadeia de caracteres não pode ser uma cadeia longa. Se o argumento for uma *ExpressãoCaractere*, o resultado será o mesmo número que seria produzido se a constante inteira correspondente fosse atribuída a uma coluna ou variável inteira.

O resultado da função é um inteiro. Se o argumento puder ser nulo, o resultado poderá ser nulo; Se o argumento for nulo, o resultado será o valor nulo.

Utilizando a tabela EMPREGADOS selecionar uma lista contendo o salário (SALÁRIO) dividido pelo nível de educação (NÍVEL_EDUC). Truncar a parte decimal nos cálculos. A lista também deve conter os valores utilizados no cálculo e o número do empregado (NUM_EMP). A lista deve estar em ordem descendente do valor calculado:

```
SELECT INTEGER (SALÁRIO / NÍVEL_EDUC), SALÁRIO, NÍVEL_EDUC, NUM_EMP
FROM EMPREGADOS
ORDER BY 1 DESC
```

LOCATE

Se a subcadeia especificada for encontrada na cadeia de procura especificada, LOCATE retornará o índice onde a subcadeia foi encontrada na cadeia de procura. Quando a subcadeia não é encontrada, LOCATE retorna 0.

Sintaxe

```
LOCATE(ExpressãoCaractere, ExpressãoCaractere [, PosiçãoInicial] )
```

A segunda *ExpressãoCaractere* é a cadeia de procura, que é procurada desde o início, a menos que seja especificada a *PosiçãoInicial*, caso em que a procura começa nesta posição; o índice começa por 1. Retorna 0 se a cadeia não for encontrada.

O tipo retornado por LOCATE é um inteiro.

```
-- retorna 2  
VALUES LOCATE('love', 'clover')
```

LCASE ou LOWER

LCASE, ou LOWER, recebe uma expressão caractere como parâmetro, e retorna uma cadeia onde os caracteres alfabéticos foram convertidos para letras minúsculas.

Sintaxe

```
LCASE ou LOWER ( ExpressãoCaractere )
```

A *ExpressãoCaractere* é do tipo de dado CHAR, VARCHAR ou LONG VARCHAR, ou qualquer tipo de dado nativo convertido implicitamente para uma cadeia de caracteres (exceto uma expressão de bit).

Se o tipo de dado do parâmetro for CHAR ou LONG VARCHAR, o tipo retornado será CHAR ou LONG VARCHAR. Senão, o tipo retornado será VARCHAR.

O comprimento e o comprimento máximo do valor retornado são idênticos ao comprimento e comprimento máximo do parâmetro.

Se a *ExpressãoCaractere* for avaliada como nula, esta função retornará nulo.

```
-- retorna 'asdl#w'  
VALUES LOWER('aSDl#w')  
SELECT LOWER(ID_VÔO) FROM VÔOS
```

LTRIM

LTRIM remove brancos do início da expressão cadeia de caracteres.

Sintaxe

```
LTRIM(ExpressãoCaractere)
```

A *ExpressãoCaractere* é do tipo de dado CHAR, VARCHAR ou LONG VARCHAR, ou de qualquer tipo de dado nativo que seja convertido implicitamente em uma cadeia de caracteres.

LTRIM retorna NULL se a *ExpressãoCaractere* for avaliada como nula.

```
-- retorna 'asdf '
VALUES LTRIM(' asdf  ')
```

MAX

MAX é uma função de agregação que calcula o valor máximo de uma expressão para um conjunto de valores (consulte [Agregações \(funções de conjunto\)](#)). MAX é permitido apenas nas expressões que resultam em tipos de dado nativos (incluindo CHAR, VARCHAR, DATE, TIME, CHAR FOR BIT DATA, etc.).

Sintaxe

```
MAX ( [ DISTINCT | ALL ] Expressão )
```

O qualificador DISTINCT elimina as linhas duplicadas. O qualificador ALL mantém as linhas duplicadas. Estes qualificadores não produzem efeito na expressão MAX. Somente é permitida uma expressão de agregação DISTINCT por [ExpressãoSeleção](#). Por exemplo, a seguinte consulta não é permitida:

```
SELECT COUNT (DISTINCT TEMPO_VÔO), MAX (DISTINCT MILHAS)
FROM VÔOS
```

A *Expressão* pode conter várias referências a colunas ou expressões, mas não pode conter outra agregação ou subconsulta. Deve ter como resultado um tipo de dado nativo. Portanto, podem ser chamados métodos que têm como resultado tipos de dado nativos (Por exemplo, um método que retorna *java.lang.Integer* ou *int* tem como resultado um INTEGER.) Se uma expressão for avaliada como NULL, a agregação saltará este valor.

Para os tipos de dado CHAR, VARCHAR e [LONG VARCHAR](#), o número de espaços em branco no final do valor pode afetar como MAX é avaliada. Por exemplo, se 'z' e 'z ' estiverem armazenados em uma coluna, não pode ser controlado qual dos dois será retornado como máximo, porque o espaço em branco não possui valor.

O tipo de dado do resultado é idêntico ao da expressão em que opera (nunca estoura).

```
-- encontrar a data mais tarde da tabela DISPONIBILIDADE_VÔO
SELECT MAX (DATA_VÔO) FROM DISPONIBILIDADE_VÔO

-- encontrar o vôo mais longo a partir de cada aeroporto,
-- mas somente quando o vôo mais longo tiver mais de 10 horas
SELECT MAX(TEMPO_VÔO), AEROPORTO_ORIGEM
FROM VÔOS
GROUP BY AEROPORTO_ORIGEM
HAVING MAX(TEMPO_VÔO) > 10
```

MIN

MIN é uma função de agregação que calcula o valor mínimo de uma expressão para um conjunto de valores (consulte [Agregações \(funções de conjunto\)](#)). MIN é permitido apenas nas expressões que têm como resultado tipos de dado nativos (incluindo CHAR, VARCHAR, DATE, TIME, etc.).

Sintaxe

```
MIN ( [ DISTINCT | ALL ] Expressão )
```

Os qualificadores DISTINCT e ALL eliminam ou retêm linhas duplicadas, mas estes qualificadores não produzem efeito na função MIN. Somente é permitida uma expressão de agregação DISTINCT por [ExpressãoSeleção](#). Por exemplo, a seguinte consulta não é permitida:

```
SELECT COUNT (DISTINCT TEMPO_VÔO), MIN (DISTINCT MILHAS)
FROM VÔOS
```

A *Expressão* pode conter várias referências a colunas ou expressões, mas não pode conter outra agregação ou subconsulta. Deve ter como resultado um tipo de dado nativo. Portanto, podem ser chamados métodos que têm como resultado tipos de dado nativos (Por exemplo, um método que retorna *java.lang.Integer* ou *int* tem como resultado um *INTEGER*.) Se uma expressão for avaliada como *NULL*, a agregação saltará este valor.

As regras de comparação do tipo determinam o valor máximo. Para *CHAR*, *VARCHAR* e *LONG VARCHAR*, o número de espaços em branco no final do valor pode afetar o resultado.

O tipo de dado do resultado é idêntico ao da expressão em que opera (nunca estoura).

```
-- não é válido:
SELECT DISTINCT TEMPO_VÔO, MIN(DISTINCT MILHAS) FROM VÔOS

-- não é válido:
SELECT COUNT(DISTINCT TEMPO_VÔO), MIN(DISTINCT MILHAS) FROM VÔOS

-- encontrar a data mais próxima:
SELECT MIN (DATA_VÔO) FROM DISPONIBILIDADE_VÔO;
```

MINUTE

A função *MINUTE* retorna a parte relativa aos minutos do valor. O argumento deve ser uma hora, carimbo do tempo, ou a representação cadeia de caracteres válida de uma hora ou carimbo do tempo, que não seja *CLOB* nem *LONG VARCHAR*. O resultado da função é um inteiro. Se o argumento puder ser nulo, o resultado poderá ser nulo; se o argumento for nulo, o resultado será o valor nulo.

As outras regras dependem do tipo de dado do argumento especificado:

- Se o argumento for uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo: O resultado será a parte relativa aos minutos do valor, que é um inteiro entre 0 e 59.
- Se o argumento for uma duração de hora ou duração de carimbo do tempo: O resultado será a parte relativa aos minutos do valor, que é um inteiro entre -99 e 99. Um resultado diferente de zero possui o mesmo sinal do argumento.

Sintaxe

```
MINUTE ( expressão )
```

```
-- Selecionar todas as aulas que não terminam na hora exata;
SELECT * FROM TABELA1 WHERE MINUTE(FIM) < 60;
```

MOD

MOD retorna o resto (módulo) do primeiro argumento dividido pelo segundo argumento. O resultado será negativo somente se o primeiro argumento for negativo.

Sintaxe

```
mod(tipo-inteiro, tipo-inteiro)
```

O resultado da função é:

- SMALLINT se os dois argumentos forem SMALLINT.
- INTEGER se um dos argumentos for INTEGER e o outro INTEGER ou SMALLINT.
- BIGINT se um dos argumentos for BIGINT e o outro BIGINT, INTEGER ou SMALLINT.

O resultado pode ser nulo; se um dos argumentos for nulo, o resultado será nulo.

MONTH

A função MONTH retorna a parte relativa ao mês do valor. O argumento deve ser uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo, que não seja CLOB nem LONG VARCHAR. O resultado da função é um inteiro. Se o argumento puder ser nulo, o resultado poderá ser nulo; se o argumento for nulo, o resultado será o valor nulo.

As outras regras dependem do tipo de dado do argumento especificado:

- Se o argumento for uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo: O resultado será a parte relativa ao mês do valor, que é um inteiro entre 1 e 12.
- Se o argumento for uma duração de data ou duração de carimbo do tempo: O resultado será a parte relativa ao mês do valor, que é um inteiro entre -99 e 99. Um resultado diferente de zero possui o mesmo sinal do argumento.

Sintaxe

```
MONTH ( expressão )
```

Selecionar todas as linhas da tabela EMPREGADOS para as pessoas nascidas (DATA_NASC) em dezembro.

```
SELECT * FROM EMPREGADOS  
WHERE MONTH(DATA_NASC) = 12
```

RTRIM

RTRIM remove brancos do final da expressão cadeia de caracteres.

Sintaxe

```
RTRIM(ExpressãoCaractere)
```

A *ExpressãoCaractere* é do tipo de dado CHAR, VARCHAR ou LONG VARCHAR, ou de qualquer tipo de dado nativo convertido implicitamente em uma cadeia de caracteres.

RTRIM retorna NULL quando a *ExpressãoCaractere* é avaliada como nula.

```
-- retorna 'asdf'  
VALUES RTRIM(' asdf  ')  
  
-- retorna 'asdf'  
VALUES RTRIM('asdf  ')
```

SECOND

A função SECOND retorna a parte relativa aos segundos do valor. O argumento deve ser uma hora, carimbo do tempo, ou a representação cadeia de caracteres válida de uma hora ou do carimbo do tempo, que não seja CLOB nem LONG VARCHAR. O

resultado da função é um inteiro. Se o argumento puder ser nulo, o resultado poderá ser nulo; se o argumento for nulo, o resultado será o valor nulo.

As outras regras dependem do tipo de dado do argumento especificado:

- Se o argumento for uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo: O resultado será a parte relativa aos segundos do valor, que é um inteiro entre 0 e 59.
- Se o argumento for uma duração de hora ou duração de carimbo do tempo: O resultado será a parte relativa aos segundos do valor, que é um inteiro entre -99 e 99. Um resultado diferente de zero possui o mesmo sinal do argumento.

Sintaxe

```
SECOND ( expressão )
```

Assumindo que a coluna RECEBIDO (TIMESTAMP) possua um valor interno equivalente a 1988-12-25-17.12.30.000000.

```
SECOND(RECEBIDO)
```

Retorna o valor 30.

SESSION_USER

SESSION_USER retorna o identificador de autorização ou nome do usuário corrente. Se não houver usuário corrente, será retornado *APP*.

[USER](#), [CURRENT_USER](#), e [SESSION_USER](#) são sinônimos.

Sintaxe

```
SESSION_USER
```

```
VALUES SESSION_USER
```

SMALLINT

A função SMALLINT retorna a representação inteira pequena de um número ou cadeia de caracteres na forma de uma constante inteira pequena.

Sintaxe

```
SMALLINT (ExpressãoNumérica | ExpressãoCaractere )
```

ExpressãoNumérica

Uma expressão que retorna um valor de qualquer tipo de dado numérico nativo. Se argumento for uma *ExpressãoNumérica*, o resultado será o mesmo número que seria produzido se o argumento fosse atribuído a uma coluna ou variável inteira pequena. Se a parte inteira do argumento não estiver dentro da faixa dos inteiros pequenos, ocorrerá um erro. A parte decimal do argumento será truncada, se estiver presente.

ExpressãoCaractere

Uma expressão que retorna um valor cadeia de caracteres, de comprimento não maior que o comprimento máximo de uma constante caractere. Os brancos iniciais e finais são eliminados, e a cadeia resultante deve estar em conformidade com as regras de formação de uma constante inteira do SQL. Entretanto, o valor da constante deve estar na faixa dos inteiros pequenos. A cadeia de caracteres não pode ser uma cadeia longa. Se o argumento for uma *ExpressãoCaractere*, o resultado será o mesmo número que seria produzido se a constante inteira

correspondente fosse atribuída a uma coluna ou variável inteira pequena.

O resultado da função é um inteiro pequeno. Se o argumento puder ser nulo, o resultado poderá ser nulo; Se o argumento for nulo, o resultado será o valor nulo.

SQRT

Retorna a raiz quadrada de um número de ponto flutuante; somente são suportados os tipos de dado nativos [REAL](#), [FLOAT](#) e [DOUBLE PRECISION](#). O tipo retornado pela função SQRT é o mesmo tipo do parâmetro.

Note: Para executar SQRT em outros tipos de dado, estes devem ser convertidos em tipos de ponto flutuante.

Sintaxe

```
SQRT(ExpressãoPontoFlutuante)
```

```
VALUES SQRT(3421E+09)

-- lança uma exceção se alguma linha armazenar um número negativo:
SELECT SQRT(MINHA_COLUNA_DUPLA) FROM MINHA_TABELA

-- retorna a raiz quadrada de um INTEGER após convertê-lo
-- em um tipo de dado de ponto flutuante:
VALUES SQRT (CAST(25 AS FLOAT));
```

SUBSTR

A função SUBSTR atua sobre uma expressão cadeia de caracteres ou uma expressão cadeia de bits. No primeiro caso o tipo do resultado é [VARCHAR](#), e no segundo caso é [VARCHAR FOR BIT DATA](#). O comprimento do resultado é o comprimento máximo do tipo de origem.

Sintaxe

```
SUBSTR({ ExpressãoCaractere },
       PosiçãoInicial [, ComprimentoCadeia ] )
```

A *PosiçãoInicial* e o parâmetro opcional *ComprimentoCadeia* são ambas expressões inteiras (O primeiro caractere ou bit possui *PosiçãoInicial* igual a 1; se for especificado 0, o Derby assume que é 1).

A *ExpressãoCaractere* é do tipo de dado CHAR, VARCHAR ou LONG VARCHAR, ou de qualquer tipo de dado nativo que seja convertido implicitamente em uma cadeia (exceto a expressão de bit).

Para as expressões caractere, tanto *PosiçãoInicial* quanto *ComprimentoCadeia* se referem a caracteres. Para expressões de bit, tanto *PosiçãoInicial* quanto *ComprimentoCadeia* se referem a bits.

SUBSTR retorna NULL se for especificado *ComprimentoCadeia*, e este for menor que zero.

Se *PosiçãoInicial* for positiva, então se refere a posição a partir do início da expressão de origem (contando o primeiro caractere como 1). Se *PosiçãoInicial* for negativa, então é a posição a partir do fim da origem.

Se não for especificado *ComprimentoCadeia*, então SUBSTR retornará a subcadeia da expressão desde a *PosiçãoInicial* até o fim da expressão de origem. Se for especificado *ComprimentoCadeia*, então SUBSTR retornará um VARCHAR ou VARBIT com

comprimento *ComprimentoCadeia* a partir da *PosiçãoInicial*.

SUM

SUM é uma expressão de agregação que calcula a soma da expressão para um conjunto de linhas (consulte [Agregações \(funções de conjunto\)](#)). SUM é permitida apenas nas expressões que resultam em tipos de dado numéricos.

Sintaxe

```
SUM ( [ DISTINCT | ALL ] Expressão )
```

Os qualificadores DISTINCT e ALL eliminam ou mantêm as linhas duplicadas. É assumido ALL se não for especificado nem ALL nem DISTINCT. Por exemplo, se a coluna contiver os valores 1, 1, 1, 1 e 2, SUM(coluna) retorna um valor maior que SUM(DISTINCT coluna).

Somente é permitida uma expressão de agregação DISTINCT por [ExpressãoSeleção](#). Por exemplo, a seguinte consulta não é permitida:

```
SELECT AVG (DISTINCT TEMPO_VÔO), SUM (DISTINCT MILHAS)
FROM VÔOS
```

A expressão pode conter várias referências a colunas ou expressões, mas não pode conter outra agregação ou subconsulta. Deve ter como resultado um tipo de dado numérico nativo. Se uma expressão for avaliada como NULL, a agregação saltará este valor.

O tipo de dado do resultado é idêntico ao da expressão em que opera (pode estourar).

```
-- descobrir a quantidade de assentos econômicos disponíveis:
SELECT SUM (ASSENTOS_ECONÔMICOS) FROM LINHAS_AÉREAS;

-- utilizar SUM fazendo referência a várias colunas
-- (descobrir o número total de todos os assentos ocupados):
SELECT SUM (ASSENTOS_ECONÔMICOS_OCUPADOS +
            ASSENTOS_EXECUTIVOS_OCUPADOS +
            ASSENTOS_PRIMEIRA_CLASSE_OCUPADOS) AS ASSENTOS_OCUPADOS
FROM VÔOS_DISPONIBILIDADE;
```

TIME

A função TIME retorna a parte relativa à hora do valor. O argumento deve ser uma hora, carimbo do tempo, ou a representação cadeia de caracteres válida de uma hora ou carimbo do tempo, que não seja CLOB nem LONG VARCHAR. O resultado da função é uma hora. Se o argumento puder ser nulo, o resultado poderá ser nulo; se o argumento for nulo, o resultado será o valor nulo.

As outras regras dependem do tipo de dado do argumento especificado:

- Se o argumento for uma hora: O resultado será esta hora.
- Se o argumento for um carimbo do tempo: O resultado será a parte relativa à hora do carimbo do tempo
- Se o argumento for uma cadeia de caracteres: O resultado será a hora representada pela cadeia de caracteres

Sintaxe

```
TIME ( expressão )
```

```
VALUES TIME(CURRENT_TIMESTAMP)
```

Se a hora corrente for 5:03 da tarde, o valor retornado será 17:03:00.

TIMESTAMP

A função **TIMESTAMP** retorna o carimbo do tempo a partir de um valor, ou de um par de valores.

As regras para os argumentos dependem do segundo argumento ser especificado:

- Se for especificado apenas um argumento: Deverá ser um carimbo do tempo, a representação cadeia de caracteres válida de um carimbo do tempo, ou uma cadeia de caracteres de comprimento 14, que não seja CLOB nem LONG VARCHAR. A cadeia de caracteres de comprimento 14 deverá ser uma cadeia de dígitos representando uma data e hora válida no formato *yyyxxddhmmss*, onde *yyyy* é o ano, *xx* é o mês, *dd* é o dia, *hh* é a hora, *mm* é o minuto e *ss* são os segundos.
- Se os dois argumentos forem especificados: O primeiro argumento deverá ser uma data, ou a representação cadeia de caracteres válida de uma data, e o segundo argumento deverá ser uma hora, ou a representação cadeia de caracteres válida de uma hora.

As outras regras dependem do segundo argumento ser especificado:

- Se os dois argumentos forem especificados: O resultado será um carimbo do tempo com a data especificada pelo primeiro argumento e a hora especificada pelo segundo argumento. A parte relativa aos microssegundos será zero.
- Se for especificado apenas um argumento, e este for um carimbo do tempo: O resultado será este carimbo do tempo.
- Se for especificado apenas um argumento, e este for uma cadeia de caracteres: O resultado será o carimbo do tempo representado pela cadeia de caracteres. Se o argumento for uma cadeia de caracteres de comprimento 14, o carimbo do tempo terá a parte relativa aos microssegundos igual a zero.

Sintaxe

```
TIMESTAMP ( expressão [, expressão ] )
```

Assumindo que a coluna **DATA_INÍCIO** (DATA) possua um valor equivalente a 1988-12-25, e a coluna **HORA_INÍCIO** (HORA) possua um valor equivalente a 17.12.30.

```
TIMESTAMP(DATA_INÍCIO, HORA_INÍCIO)
```

Retorna o valor '1988-12-25-17.12.30.000000'.

UCASE ou UPPER

UCASE, ou **UPPER**, recebe uma expressão caractere como parâmetro, e retorna uma cadeia onde os caracteres alfabéticos foram convertidos para letras maiúsculas.

Sintaxe

```
UCASE ou UPPER ( ExpressãoCaractere )
```

Se o tipo do parâmetro for **CHAR**, o tipo retornado será **CHAR**. Senão, o tipo retornado será **VARCHAR**.

Note: As funções **UPPER** e **LOWER** seguem o idioma do banco de dados. Consulte [territory=II_CC](#) para obter mais informações sobre como especificar o idioma.

O comprimento e o comprimento máximo do valor retornado são idênticos ao comprimento e comprimento máximo do parâmetro.

```
-- retorna 'ASD1#W'
VALUES UPPER('aSD1#w')
```

USER

USER retorna o identificador de autorização ou nome do usuário corrente. Se não houver usuário corrente, será retornado *APP*.

USER, [CURRENT_USER](#) e [SESSION_USER](#) são sinônimos.

Sintaxe

```
USER
```

```
VALUES USER
```

VARCHAR

A função VARCHAR retorna a representação cadeia de caracteres de comprimento variável da cadeia de caracteres.

Sintaxe caractere para VARCHAR

```
VARCHAR (ExpressãoCadeiaCaracteres )
```

ExpressãoCadeiaCaracteres

Uma expressão cujo valor deve ser do tipo de dado cadeia de caracteres com comprimento máximo de 32.672 bytes.

Sintaxe data/hora para VARCHAR

```
VARCHAR (ExpressãoDataHora )
```

ExpressãoDataHora

Uma expressão cujo valor deve ser do tipo de dado data, hora ou carimbo do tempo.

Utilizando a tabela EMPREGADOS selecionar a descrição do cargo (CARGO definido como CHAR(8)) para 'Dolores Quintana' como o VARCHAR equivalente:

```
SELECT VARCHAR(CARGO)
FROM EMPREGADOS
WHERE ÚLTIMO_NOME = 'QUINTANA'
```

YEAR

A função YEAR retorna a parte relativa ao ano do valor. O argumento deve ser uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo. O resultado da função é um inteiro. Se o argumento puder ser nulo, o resultado poderá ser nulo; se o argumento for nulo, o resultado será o valor nulo.

As outras regras dependem do tipo de dado do argumento especificado:

- Se o argumento for uma data, carimbo do tempo, ou a representação cadeia de caracteres válida de uma data ou carimbo do tempo: O resultado será a parte relativa ao ano do valor, que é um inteiro entre 1 e 9 999.
- Se o argumento for uma duração de data ou duração de carimbo do tempo: O resultado será a parte relativa ao ano do valor, que é um inteiro entre -9 999 e 9 999. Um resultado diferente de zero possui o mesmo sinal do argumento.

Sintaxe

```
YEAR ( expressão )
```

Selecionar todos os projetos da tabela PROJETO agendados para iniciar (PROJ_DATA_INÍCIO) e terminar (PROJ_DATA_FIM) no mesmo ano.

```
SELECT * FROM PROJETO
WHERE YEAR(PROJ_DATA_INÍCIO) = YEAR(PROJ_DATA_FIM)
```

Funções do sistema nativas

Esta seção descreve as diferentes funções do sistema nativas disponíveis no Derby.

SYSCS_UTIL.SYSCS_CHECK_TABLE

A função SYSCS_UTIL.SYSCS_CHECK_TABLE verifica a tabela especificada, para assegurar que todos os seus índices estão consistentes com a tabela base. Quando os índices estão consistentes, este método retorna um SMALLINT com valor igual a 1. Quando os índices estão inconsistentes, a função lança uma exceção.

Sintaxe

```
SMALLINT SYSCS_UTIL.SYSCS_CHECK_TABLE( IN SCHEMANAME VARCHAR(128),
IN TABLENAME VARCHAR(128) )
```

Ocorre um erro se SCHEMANAME ou TABLENAME for nulo.

Exemplo

```
VALUES SYSCS_UTIL.SYSCS_CHECK_TABLE( 'VENDAS', 'PEDIDOS' );
```

SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS

A FUNÇÃO SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS retorna um valor VARCHAR(32762), que representa o plano de execução da instrução e estatísticas de tempo de execução para um java.sql.ResultSet. O plano de execução da instrução é uma árvore de nós de execução. Existe um número de tipos de nós possíveis. A estatísticas são acumuladas durante a execução de cada nó. Os tipos das estatísticas incluem a quantidade de tempo gasto em operações específicas, o número de linhas passadas para o nó por seus descendentes, e o número de linhas retornadas pelo nó para seu ancestral (As estatísticas exatas são específicas de cada tipo de nó). A função SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS é mais significativa para as instruções da DML, como SELECT, INSERT, DELETE e UPDATE.

Sintaxe

```
VARCHAR(32762) SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS( )
```

Exemplo

```
VALUES SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS( )
```

SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY

A função SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY retorna o valor da propriedade especificada por KEY, do banco de dados na conexão corrente.

Sintaxe

```
VARCHAR(32762) SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY(IN KEY
VARCHAR(128))
```

Retorna erro se KEY for nulo.

Exemplo

```
VALUES
SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY('cadeia_com_valor_da_chave');
```

Procedimentos do sistema nativos

Alguns procedimentos nativos não são compatíveis com a sintaxe SQL utilizada por outros bancos de dados relacionais. Estes procedimentos somente podem ser utilizados no Derby.

SYSCS_UTIL.SYSCS_COMPRESS_TABLE

O procedimento do sistema SYSCS_UTIL.SYSCS_COMPRESS_TABLE é utilizado para recuperar o espaço alocado, mas não utilizado, de uma tabela e seus índices. Normalmente, existe espaço alocado mas não utilizado quando uma grande quantidade de dados é excluída da tabela, ou os índices são atualizados. Por padrão, o Derby não retorna para o sistema operacional o espaço não utilizado. Por exemplo, uma vez que uma página tenha sido alocada para uma tabela ou um índice, esta não é devolvida automaticamente para o sistema operacional até que a tabela ou o índice tenha sido removido. O procedimento do sistema SYSCS_UTIL.SYSCS_COMPRESS_TABLE permite devolver para o sistema operacional o espaço não utilizado.

Sintaxe

```
SYSCS_UTIL.SYSCS_COMPRESS_TABLE (IN SCHEMANAME VARCHAR(128),
IN TABLENAME VARCHAR(128), IN SEQUENTIAL SMALLINT)
```

SCHEMANAME

Um argumento de entrada do tipo VARCHAR(128) que especifica o esquema da tabela. Passar um valor nulo resulta em erro.

TABLENAME

Um argumento de entrada do tipo VARCHAR(128) que especifica o nome da tabela. A cadeia de caracteres deve corresponder exatamente às letras maiúsculas e minúsculas do nome da tabela: um argumento igual a "Fred" é passado para o SQL como o identificador delimitado 'Fred'. Passar um argumento nulo resulta em erro.

SEQUENTIAL

Um argumento de entrada diferente de zero do tipo SMALLINT força a operação executar no modo seqüencial, enquanto um argumento igual a zero força a operação a não executar no modo seqüencial. Passar um argumento nulo resulta em um erro.

Exemplo SQL

Para comprimir a tabela chamada CLIENTE no esquema chamado US, utilizando a opção SEQUENTIAL:

```
CALL SYSCS_UTIL.SYSCS_COMPRESS_TABLE('US', 'CLIENTE', 1)
```

Exemplo Java

Para comprimir a tabela chamada CLIENTE no esquema chamado US, utilizando a opção SEQUENTIAL:

```
CallableStatement cs = conn.prepareCall
```



```
( "CALL SYSCS_UTIL.SYSCS_COMPRESS_TABLE(?, ?, ?)" );
cs.setString(1, "US");
cs.setString(2, "CLIENTE");
cs.setShort(3, (short) 1);
cs.execute();
```

Se o parâmetro SEQUENTIAL não for especificado, o Derby reconstruirá todos os índices simultaneamente com a tabela base. Se não for especificado o argumento SEQUENTIAL, este procedimento poderá fazer uso intenso da memória e utilizar muito espaço temporário em disco (uma quantidade aproximadamente igual a duas vezes o espaço utilizado, mais o espaço alocado mas não utilizado). Isto acontece porque o Derby comprime a tabela copiando as linhas ativas para o novo espaço alocado (em vez de embaralhar e truncar o espaço existente). O espaço extra utilizado é retornado ao sistema operacional no COMMIT.

Quando é especificado SEQUENTIAL, o Derby comprime a tabela base e depois comprime cada um dos índices seqüencialmente. Utilizar SEQUENTIAL faz com que seja usado menos memória e espaço em disco, mas demora mais tempo. Deve ser utilizado o argumento SEQUENTIAL para reduzir a utilização de memória e espaço em disco.

O procedimento do sistema SYSCS_UTIL.SYSCS_COMPRESS_TABLE não pode devolver espaço em disco para o sistema operacional enquanto o COMMIT não for executado. Isto significa que o espaço ocupado pela tabela base e seus índices não pode ser retornado para o sistema operacional enquanto COMMIT não for executado (somente o espaço em disco alocado temporariamente pela classificação externa pode ser retornado para o sistema operacional antes de COMMIT). Recomenda-se utilizar o procedimento SYSCS_UTIL.SYSCS_COMPRESS_TABLE no modo de auto-efetivação. **Note:** Este procedimento obtém um bloqueio exclusivo na tabela sendo comprimida. Todos os planos de instrução dependentes da tabela e de seus índices são invalidados. Para obter informações sobre como identificar espaço não utilizado deve ser consultado o *Guia do Servidor e Administração do Derby*.

SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE

O procedimento do sistema SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE é utilizado para recuperar o espaço alocado, mas não utilizado, de uma tabela e seus índices. Normalmente, existe espaço alocado mas não utilizado quando uma grande quantidade de dados é excluída da tabela, e não ocorrem inserções subseqüentes para utilizar o espaço criado pelas exclusões. Por padrão, o Derby não retorna para o sistema operacional o espaço não utilizado. Por exemplo, uma vez que uma página tenha sido alocada para uma tabela ou um índice, esta não é devolvida automaticamente para o sistema operacional até que a tabela ou o índice tenha sido removido. O procedimento do sistema SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE permite devolver para o sistema operacional o espaço não utilizado.

Este procedimento do sistema pode ser utilizado para ocasionar três níveis de compressão *in loco* de uma tabela SQL: PURGE_ROWS, DEFRAGMENT_ROWS e TRUNCATE_END. Diferentemente de SYSCS_UTIL.SYSCS_COMPRESS_TABLE(), todo o trabalho é feito *in loco* na tabela ou índice existente.

Sintaxe

```
SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE(
    IN SCHEMANAME VARCHAR(128),
    IN TABLENAME VARCHAR(128),
    IN PURGE_ROWS SMALLINT,
    IN DEFRAGMENT_ROWS SMALLINT,
    IN TRUNCATE_END SMALLINT )
```

SCHEMANAME

Um argumento de entrada do tipo VARCHAR(128) que especifica o esquema da tabela. Passar um argumento nulo resulta em erro.

TABlename

Um argumento de entrada do tipo VARCHAR(128) que especifica o nome da tabela. A cadeia de caracteres deve corresponder exatamente às letras maiúsculas e minúsculas do nome da tabela: um argumento igual a "Fred" é passado para o SQL como o identificador delimitado 'Fred'. Passar um argumento nulo resulta em erro.

PURGE_ROWS

Se PURGE_ROWS for definido com um valor diferente de zero, então será feita uma única passagem através da tabela para remover as linhas com a exclusão efetivada. Este espaço se torna disponível para as próximas linhas inseridas, mas permanece alocado para a tabela. Como esta opção varre todas as páginas da tabela, seu desempenho é linearmente proporcional ao tamanho da tabela.

DEFRAGMENT_ROWS

Se DEFRAGMENT_ROWS for definido com um valor diferente de zero, então será feita uma única passagem de desfragmentação para mover as linhas existentes no final da tabela em direção ao início da tabela. O objetivo da desfragmentação é esvaziar um conjunto de páginas no final da tabela, que poderão depois ser devolvidas ao sistema operacional pela opção TRUNCATE_END. Recomenda-se somente utilizar a opção DEFRAGMENT_ROWS quando a opção TRUNCATE_END também é utilizada. A opção DEFRAGMENT_ROWS varre toda a tabela, e precisa atualizar as entradas dos índices de todas as linhas da tabela base movidas, portanto o tempo de execução é linearmente proporcional ao tamanho da tabela.

TRUNCATE_END

Se TRUNCATE_END for definido com um valor diferente de zero, então todas as páginas contíguas no final da tabela serão devolvidas para o sistema operacional. A execução das opções PURGE_ROWS e/ou DEFRAGMENT_ROWS pode aumentar o número de páginas afetadas. Por si só, esta opção não realiza uma varredura na tabela.

Exemplo SQL

Para comprimir a tabela chamada CLIENTE no esquema chamado US, utilizando todas as opções de compressão disponíveis:

```
CALL SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE('US', 'CLIENTE', 1, 1, 1);
```

Para devolver o espaço livre no final da mesma tabela, pode se feita a seguinte chamada, que é muito mais rápida do que executar todas as opções, mas provavelmente devolve muito menos espaço:

```
CALL SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE('US', 'CLIENTE', 0, 0, 1);
```

Exemplo Java

Para comprimir a tabela chamada CLIENTE no esquema chamado US, utilizando todas as opções de compressão disponíveis:

```
CallableStatement cs = conn.prepareCall(
    ("CALL SYSCS_UTIL.SYSCS_COMPRESS_TABLE(?, ?, ?, ?, ?)");
cs.setString(1, "US");
cs.setString(2, "CLIENTE");
cs.setShort(3, (short) 1);
cs.setShort(4, (short) 1);
cs.setShort(5, (short) 1);
cs.execute();
```

Para devolver o espaço livre no final da mesma tabela, pode se feita a seguinte chamada, que é muito mais rápida do que executar todas as opções, mas provavelmente devolve muito menos espaço:

```
CallableStatement cs = conn.prepareCall(
    ("CALL SYSCS_UTIL.SYSCS_COMPRESS_TABLE(?, ?, ?, ?, ?)");
cs.setString(1, "US");
```

```
cs.setString(2, "CLIENTE");
cs.setShort(3, (short) 0);
cs.setShort(4, (short) 0);
cs.setShort(5, (short) 1);
cs.execute();
```

Recomenda-se que o procedimento SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE seja executado no modo de auto-efetivação.

Note: Este procedimento obtém um bloqueio exclusivo na tabela sendo comprimida. Todos os planos de instrução dependentes da tabela ou de seus índices são invalidados. Para obter informações sobre como identificar espaço não utilizado deve ser consultado o *Guia do Servidor e Administração do Derby*.

SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS

O procedimento do sistema SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS() torna as estatísticas em tempo de execução da conexão ligadas ou desligadas. Por padrão, as estatísticas em tempo de execução estão desligadas. Quando o atributo runtimestatistics está ligado, o Derby mantém informações sobre o plano de execução de cada instrução executada pela conexão (exceto COMMIT), até que este atributo seja desligado. Para desligar o atributo runtimestatistics, este procedimento deve ser chamado com um argumento igual a zero. Para ligar o atributo runtimestatistics, este procedimento deve ser chamado com um argumento diferente de zero.

Nas instruções que não retornam linhas, o objeto é criado quando todo o processamento interno está completo, antes de retornar para o programa cliente. Nas instruções que retornam linhas, o objeto é criado quando a primeira chamada a next() retorna zero linhas, ou se for encontrada uma chamada a close(), o que ocorrer primeiro.

Sintaxe

```
SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(IN SMALLINT ENABLE)
```

Exemplo

```
-- estabelecer a conexão
-- ligar RUNTIMESTATISTIC para a conexão:
CALL SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(1);
-- executar uma consulta complexa
-- caminhar pelo conjunto de resultados
-- acessar as informações de estatísticas em tempo de execução:
CALL SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(0);
```

SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING

A estatística dos tempos é um atributo associado à conexão, que é ligado e desligado utilizando o procedimento do sistema SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING. A estatística dos tempos está desligada por padrão. A estatística dos tempos deve ser ligada apenas quando o atributo runtimestatistics já estiver ligado. Ligar a estatística dos tempos quando o atributo runtimestatistics está desligado não causa nenhum efeito.

A estatística dos tempos é ligada chamando este procedimento com um argumento diferente de zero. A estatística dos tempos é desligada chamando este procedimento com um argumento igual a zero.

Quando a estatística dos tempos está ligada, o Derby registra os tempos de vários aspectos da execução da instrução. Esta informação é incluída nas informações retornadas pela função do sistema [SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS](#). Quando a estatística dos tempos está desligada, a função do sistema [SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS](#) mostra todos os valores de tempo como zero.

Sintaxe

```
SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING(IN SMALLINT ENABLE)
```

Exemplo

Ligar o atributo `runtimestatistics` e depois ligar o atributo estatísticas dos tempos:

```
CALL SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(1);
CALL SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING(1);
```

SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY

O procedimento do sistema `SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY` é utilizado para definir ou remover o valor de uma propriedade do banco de dados na conexão corrente.

Se o parâmetro "VALUE" não for nulo, então a propriedade com valor chave igual ao valor do parâmetro "KEY" será definida com o valor do parâmetro "VALUE". Se o parâmetro "VALUE" for nulo, então a propriedade com o valor chave igual ao valor do parâmetro "KEY" será removida do conjunto de propriedades do banco de dados.

Sintaxe

```
SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY(IN KEY VARCHAR(128),
IN VALUE VARCHAR(32672))
```

Este procedimento não retorna resultado.

Exemplo JDBC

Definir a propriedade `derby.locks.deadlockTimeout` com o valor 10:

```
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY(?, ?)");
cs.setString(1, "derby.locks.deadlockTimeout");
cs.setString(2, "10");
cs.execute();
cs.close();
```

Exemplo SQL

Definir a propriedade `derby.locks.deadlockTimeout` com o valor 10:

```
CALL SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY
('derby.locks.deadlockTimeout', '10');
```

SYSCS_UTIL.SYSCS_FREEZE_DATABASE

O procedimento do sistema `SYSCS_UTIL.SYSCS_FREEZE_DATABASE` congela temporariamente o banco de dados para fazer cópia de segurança.

Sintaxe

```
SYSCS_UTIL.SYSCS_FREEZE_DATABASE()
```

Este procedimento não retorna resultado.

Exemplo

```
String diretorio_copias = "c:/minhas_copias/" + JCalendar.getToday();
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_FREEZE_DATABASE()");
cs.execute();
cs.close();
// código fornecido pelo usuário para fazer a cópia de segurança
// completa do "diretorio_copias".
// descongelar o banco de dados após a cópia de segurança terminar:
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE()");
cs.execute();
cs.close();
```

SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE

O procedimento do sistema SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE descongela o banco de dados após o término da cópia de segurança.

Sintaxe

```
SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE()
```

Este procedimento não retorna resultado.

Exemplo

```
String diretorio_copias = "c:/minhas_copias/" + JCalendar.getToday();
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_FREEZE_DATABASE()");
cs.execute();
cs.close();
// código fornecido pelo usuário para fazer a cópia de segurança
// completa do "diretorio_copias".
// descongelar o banco de dados após a cópia de segurança terminar:
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE()");
cs.execute();
cs.close();
```

SYSCS_UTIL.SYSCS_CHECKPOINT_DATABASE

O procedimento do sistema SYSCS_UTIL.SYSCS_CHECKPOINT_DATABASE cria um ponto de controle no banco de dados, descarregando todos os dados no *cache* para o disco.

Sintaxe

```
SYSCS_UTIL.SYSCS_CHECKPOINT_DATABASE()
```

Este procedimento não retorna resultado.

Exemplo JDBC

```
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_CHECKPOINT_DATABASE()");
cs.execute();
cs.close();
```

Exemplo SQL

```
CALL SYSCS_UTIL.SYSCS_CHECKPOINT_DATABASE ( );
```

SYSCS_UTIL.SYSCS_BACKUP_DATABASE

O procedimento do sistema SYSCS_UTIL.SYSCS_BACKUP_DATABASE faz a cópia de segurança do banco de dados no diretório de cópia de segurança especificado.

Sintaxe

```
SYSCS_UTIL.SYSCS_BACKUP_DATABASE (IN BACKUPDIR VARCHAR ( ))
```

Este procedimento não retorna resultado.

BACKUPDIR

Um argumento de entrada do tipo VARCHAR(32672), que especifica o caminho de sistema completo do diretório onde será armazenada a cópia de segurança do banco de dados.

Exemplo JDBC

O exemplo a seguir faz a cópia de segurança do banco de dados no diretório c:/copia_seguranca:

```
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE(?)");
cs.setString(1, "c:/copia_seguranca");
cs.execute();
cs.close();
```

Exemplo SQL

O exemplo a seguir faz a cópia de segurança do banco de dados no diretório c:/copia_seguranca:

```
CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE('c:/copia_seguranca');
```

SYSCS_UTIL.SYSCS_EXPORT_TABLE

O procedimento do sistema SYSCS_UTIL.SYSCS_EXPORT_TABLE exporta todos os dados de uma tabela para um arquivo do sistema operacional no formato delimitado.

Sintaxe

```
SYSCS_UTIL.SYSCS_EXPORT_TABLE (IN SCHEMANAME VARCHAR(128),
IN TABLENAME VARCHAR(128), IN FILENAME VARCHAR(32672),
IN COLUMNDELIMITER CHAR(1), IN CHARACTERDELIMITER CHAR(1),
IN CODESET VARCHAR(128))
```

Este procedimento não retorna resultado.

SCHEMANAME

Um argumento de entrada do tipo VARCHAR(128), que especifica o nome do esquema da tabela. Passar um valor nulo resulta na utilização do nome de esquema padrão.

TABLENAME

Um argumento de entrada do tipo VARCHAR(128), que especifica o nome da tabela/visão da qual os dados serão exportados. Passar um valor nulo resulta em erro.

FILENAME

Um argumento de entrada do tipo VARCHAR(32672), que especifica o nome do arquivo para onde os dados serão exportados. Se não for especificado o caminho

completo para o arquivo, o procedimento de exportação utilizará o diretório corrente e a unidade padrão como destino. Se for especificado o nome de um arquivo que já existe, o procedimento de exportação sobrescreverá o conteúdo do arquivo; a informação não será anexada. Passar um valor nulo resulta em erro.

COLUMNDELIMITER

Um argumento de entrada do tipo CHAR(1), que especifica o delimitador de coluna. O caractere especificado é utilizado no lugar da vírgula para sinalizar o fim da coluna. Passar um valor nulo resulta na utilização do valor padrão; o valor padrão é a vírgula (,).

CHARACTERDELIMITER

Um argumento de entrada do tipo CHAR(1), que especifica o delimitador de caracteres. O caractere especificado é utilizado no lugar das aspas para envolver a cadeia de caracteres. Passar um valor nulo resulta na utilização do valor padrão; o valor padrão é aspas (").

CODESET

Um argumento de entrada do tipo VARCHAR(128), que especifica o código de conjunto de caracteres dos dados no arquivo exportado. O nome do código de conjunto de caracteres deve ser igual ao de uma das codificações de caracteres suportadas pelo Java. Antes de escrever no arquivo, os dados são convertidos do código de conjunto de caracteres do banco de dados para o código de conjunto de caracteres especificado. Passar um valor nulo faz com que os dados sejam escritos no mesmo código de conjunto de caracteres da JVM em que está sendo executado. Se o nome do esquema ou da tabela foi criado usando um identificador não delimitado, o nome deverá ser passado para o procedimento de exportação usando todas as letras em maiúsculo. Se o nome do esquema, da tabela ou da coluna foi criado usando um identificador delimitado, o nome deverá ser passado para o procedimento de exportação com letras maiúsculas e minúsculas, conforme utilizado na criação.

Exemplo

O exemplo a seguir mostra como exportar as informações da tabela EQUIPE do banco de dados SAMPLE para o arquivo `meu_arquivo.csv`.

```
CALL SYSCS_UTIL.SYSCS_EXPORT_TABLE (null, 'EQUIPE', 'meu_arquivo.csv',
null, null, null);
```

Para obter mais informações sobre exportação deve ser consultado o *Guia das Ferramentas e Utilitários do Derby*.

SYSCS_UTIL.SYSCS_EXPORT_QUERY

O procedimento do sistema SYSCS_UTIL.SYSCS_EXPORT_QUERY exporta os resultados de uma instrução SELECT para um arquivo do sistema operacional no formato delimitado.

Sintaxe

```
SYSCS_UTIL.SYSCS_EXPORT_QUERY(IN SELECTSTATEMENT VARCHAR(32672),
IN FILENAME VARCHAR(32672), IN COLUMNDELIMITER CHAR(1),
IN CHARACTERDELIMITER CHAR(1), IN CODESET VARCHAR(128))
```

Este procedimento não retorna resultado.

SELECTSTATEMENT

Um argumento de entrada do tipo VARCHAR(32672), que especifica a instrução SELECT (consulta) que retorna os dados a serem exportados. Passar um valor nulo resulta em erro.

FILENAME

Um argumento de entrada do tipo VARCHAR(32672), que especifica o nome do arquivo para onde os dados serão exportados. Se não for especificado o caminho

completo para o arquivo, o procedimento de exportação utilizará o diretório corrente e a unidade padrão como destino. Se for especificado o nome de um arquivo que já existe, o procedimento de exportação sobrescreverá o conteúdo do arquivo; a informação não será anexada. Passar um valor nulo resulta em erro.

COLUMNDELIMITER

Um argumento de entrada do tipo CHAR(1), que especifica o delimitador de coluna. O caractere especificado é utilizado no lugar da vírgula para sinalizar o fim da coluna. Passar um valor nulo resulta na utilização do valor padrão; o valor padrão é a vírgula (,).

CHARACTERDELIMITER

Um argumento de entrada do tipo CHAR(1), que especifica o delimitador de caracteres. O caractere especificado é utilizado no lugar das aspas para envolver a cadeia de caracteres. Passar um valor nulo resulta na utilização do valor padrão; o valor padrão é aspas (").

CODESET

Um argumento de entrada do tipo VARCHAR(128), que especifica o código de conjunto de caracteres dos dados no arquivo exportado. O nome do código de conjunto de caracteres deve ser igual ao de uma das codificações de caracteres suportadas pelo Java. Antes de escrever no arquivo, os dados são convertidos do código de conjunto de caracteres do banco de dados para o código de conjunto de caracteres especificado. Passar um argumento nulo faz com que os dados sejam escritos no mesmo código de conjunto de caracteres da JVM em que está sendo executado.

Exemplo

O exemplo a seguir mostra como exportar as informações dos empregados do departamento 20 da tabela EQUIPE do banco de dados SAMPLE para o arquivo meu_arquivo.csv.

```
CALL SYSCS_UTIL.SYSCS_EXPORT_QUERY('SELECT * FROM EQUIPE WHERE NUM_DEP
=20',
'c:/saida/meu_arquivo.csv', null, null, null);
```

Para obter mais informações sobre exportação deve ser consultado o *Guia das Ferramentas e Utilitários do Derby*.

SYSCS_UTIL.SYSCS_IMPORT_TABLE

O procedimento do sistema SYSCS_UTIL.SYSCS_IMPORT_TABLE importa dados de um arquivo de entrada para todas as colunas de uma tabela. Se a tabela que vai receber os dados já contiver dados, os dados importados poderão substituir ou ser anexado aos dados existentes.

Sintaxe

```
SYSCS_UTIL.SYSCS_IMPORT_TABLE (IN SCHEMANAME VARCHAR(128),
IN TABLENAME VARCHAR(128), IN FILENAME VARCHAR(32672),
IN COLUMNDELIMITER CHAR(1), IN CHARACTERDELIMITER CHAR(1),
IN CODESET VARCHAR(128), IN REPLACE SMALLINT)
```

Este procedimento não retorna resultado.

SCHEMANAME

Um argumento de entrada do tipo VARCHAR(128), que especifica o nome do esquema da tabela. Passar um valor nulo resulta na utilização do nome de esquema padrão

TABLENAME

Um argumento de entrada do tipo VARCHAR(128), que especifica o nome da tabela para onde os dados serão importados. Não pode ser uma tabela do sistema ou uma tabela temporária declarada. Passar um valor nulo resulta em erro.

FILENAME

Um argumento de entrada do tipo VARCHAR(32672), que especifica o nome do arquivo que contém os dados a serem importados. Se não for especificado o caminho, será utilizado o diretório de trabalho corrente. Passar um valor nulo resulta em erro.

COLUMNDELIMITER

Um argumento de entrada do tipo CHAR(1), que especifica o delimitador de coluna. O caractere especificado é utilizado no lugar da vírgula para sinalizar o fim da coluna. Passar um valor nulo resulta na utilização do valor padrão; o valor padrão é a vírgula (,).

CHARACTERDELIMITER

Um argumento de entrada do tipo CHAR(1), que especifica o delimitador de caracteres. O caractere especificado é utilizado no lugar das aspas para envolver a cadeia de caracteres. Passar um valor nulo resulta na utilização do valor padrão; o valor padrão é aspas (").

CODESET

Um argumento de entrada do tipo VARCHAR(128), que especifica o código de conjunto de caracteres dos dados no arquivo de entrada. O nome do código de conjunto de caracteres deve ser igual ao de uma das codificações de caracteres suportadas pelo Java. Os dados são convertidos do código de conjunto de caracteres especificado para o código de conjunto de caracteres do banco de dados (UTF-8). Passar um valor nulo faz interpretar os dados do arquivo no mesmo código de conjunto de caracteres da JVM em que está sendo executado.

REPLACE

Um argumento de entrada do tipo SMALLINT. Um valor diferente de zero faz com que execute no modo de substituição, enquanto um valor igual a zero faz com que execute no modo de inserção. O modo de substituição exclui todos os dados existentes na tabela truncando o objeto de dados, e insere os dados importados. A definição da tabela e as definições dos índices não são alteradas. O modo de inserção adiciona os dados importados à tabela, sem alterar os dados existentes na tabela. Passar um valor nulo resulta em erro.

Se o nome do esquema, da tabela ou da coluna foi criado usando um identificador não delimitado, o nome deverá ser passado para o procedimento de importação usando todas as letras em maiúsculo. Se o nome do esquema, da tabela ou da coluna foi criado usando um identificador delimitado, o nome deverá ser passado para o procedimento de importação com letras maiúsculas e minúsculas, conforme utilizado na criação.

Exemplo

O exemplo a seguir importa dados para a tabela EQUIPE a partir de um arquivo de dados delimitado chamado `meu_arquivo.csv`, que utiliza o caractere de percentagem (%) como delimitador de cadeia de caracteres, e o caractere ponto-e-vírgula (;) como delimitador de coluna:

```
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE
(null, 'EQUIPE', 'c:/output/meu_arquivo.csv', ';', '%', NULL, 0);
```

Para obter mais informações sobre importação deve ser consultado o *Guia das Ferramentas e Utilitários do Derby*.

SYSCS_UTIL.SYSCS_IMPORT_DATA

O procedimento do sistema `SYSCS_UTIL.SYSCS_IMPORT_DATA` importa dados para um subconjunto de colunas da tabela. O subconjunto de colunas é escolhido especificando as colunas para inserção. Este procedimento também é utilizado para importar um subconjunto de colunas de dados do arquivo especificando os índices das colunas.

Sintaxe

```
SYSCS_UTIL.SYSCS_IMPORT_DATA (IN SCHEMANAME VARCHAR(128),  
IN TABLENAME VARCHAR(128), IN INSERTCOLUMNS VARCHAR(32672),  
IN COLUMNINDEXES VARCHAR(32672), IN FILENAME VARCHAR(32672),  
IN COLUMNDELIMITER CHAR(1), IN CHARACTERDELIMITER CHAR(1),  
IN CODESET VARCHAR(128), IN REPLACE SMALLINT)
```

Este procedimento não retorna resultado.

SCHEMANAME

Um argumento de entrada do tipo VARCHAR(128), que especifica o nome do esquema da tabela. Passar um valor nulo resulta na utilização do nome de esquema padrão

TABLENAME

Um argumento de entrada do tipo VARCHAR(128), que especifica o nome da tabela para onde os dados serão importados. Não pode ser uma tabela do sistema ou uma tabela temporária declarada. Passar um valor nulo resulta em erro.

INSERTCOLUMNS

Um argumento de entrada do tipo VARCHAR (32762), que especifica os nomes das colunas (separados por vírgula) da tabela para onde os dados serão importados. Passar um valor nulo resulta em erro.

COLUMNINDEXES

Um argumento de entrada do tipo VARCHAR (32762), que especifica os índices (numerados a partir de 1 e separados por vírgulas) dos campos de dado da entrada a serem importados. Passar um valor nulo faz com que sejam utilizados todos os campos de dado de entrada presentes no arquivo.

FILENAME

Um argumento de entrada do tipo VARCHAR(32672), que especifica o nome do arquivo que contém os dados a serem importados. Se não for especificado o caminho, será utilizado o diretório de trabalho corrente. Passar um valor nulo resulta em erro.

COLUMNDELIMITER

Um argumento de entrada do tipo CHAR(1), que especifica o delimitador de coluna. O caractere especificado é utilizado no lugar da vírgula para sinalizar o fim da coluna. Passar um valor nulo resulta na utilização do valor padrão; o valor padrão é a vírgula (,).

CHARACTERDELIMITER

Um argumento de entrada do tipo CHAR(1), que especifica o delimitador de caracteres. O caractere especificado é utilizado no lugar das aspas para envolver a cadeia de caracteres. Passar um valor nulo resulta na utilização do valor padrão; o valor padrão é aspas (").

CODESET

Um argumento de entrada do tipo VARCHAR(128), que especifica o código de conjunto de caracteres dos dados no arquivo de entrada. O nome do código de conjunto de caracteres deve ser igual ao de uma das codificações de caracteres suportadas pelo Java. Os dados são convertidos do código de conjunto de caracteres especificado para o código de conjunto de caracteres do banco de dados (UTF-8). Passar um valor nulo faz interpretar os dados do arquivo no mesmo código de conjunto de caracteres da JVM em que está sendo executado.

REPLACE

Um argumento de entrada do tipo SMALLINT. Um valor diferente de zero faz com que execute no modo de substituição, enquanto um valor igual a zero faz com que execute no modo de inserção. O modo de substituição exclui todos os dados existentes da tabela truncando o objeto de dados, e insere os dados importados. A definição da tabela e as definições dos índices não são alteradas. O modo de inserção adiciona os dados importados à tabela, sem alterar os dados existentes na tabela. Passar um valor nulo resulta em erro.

Se o nome do esquema, da tabela ou da coluna foi criado usando um identificador não delimitado, o nome deverá ser passado para o procedimento de importação usando todas as letras em maiúsculo. Se o nome do esquema, da tabela ou da coluna foi criado usando um identificador delimitado, o nome deverá ser passado para o procedimento de

importação com letras maiúsculas e minúsculas, conforme utilizado na criação.

Exemplo

O exemplo a seguir importa alguns dos campos de dados do arquivo de dados delimitado chamado `meu_arquivo.csv` para a tabela `EQUIPE`:

```
CALL SYSCS_UTIL.SYSCS_IMPORT_DATA
(NULL, 'EQUIPE', null, '1,3,4', 'meu_arquivo.csv', null, null, null,0)
```

Para obter mais informações sobre importação deve ser consultado o *Guia das Ferramentas e Utilitários do Derby*.

Tipos de dado

Esta seção descreve os tipos de dado utilizados no Derby.

Visão geral dos tipos de dado nativos

O sistema de tipos de dado do SQL é utilizado pelo compilador da linguagem para determinar o tipo da expressão em tempo de compilação, e pelo sistema de execução da linguagem para determinar o tipo da expressão em tempo de execução, que pode ser um subtipo ou uma implementação do tipo em tempo de compilação.

Cada tipo possui, associado ao mesmo, valores deste tipo. Além disso, os valores no banco de dados e os resultados das expressões podem ser nulos, o que significa que o valor está faltando ou é desconhecido. Embora existam alguns lugares onde a palavra chave `NULL` pode ser utilizada explicitamente, esta não é em si própria um valor, porque necessita de um tipo de dado associado à mesma.

A sintaxe apresentada nesta seção é a sintaxe utilizada ao especificar o tipo de dado da coluna na instrução `CREATE TABLE`.

Tipos numéricos

Tipos numéricos utilizados no Derby.

Visão geral dos tipos numéricos

Os tipos numéricos incluem os seguintes tipos, que fornecem armazenamento de tamanhos diferentes:

- Numéricos inteiros
 - `SMALLINT` (2 bytes)
 - `INTEGER` (4 bytes)
 - `BIGINT` (8 bytes)
- Numéricos aproximados, ou de ponto flutuante
 - `REAL` (4 bytes)
 - `DOUBLE PRECISION` (8 bytes)
 - `FLOAT` (outro nome para `DOUBLE PRECISION` ou `REAL`)
- Numéricos exatos
 - `DECIMAL` (armazenamento baseado na precisão)
 - `NUMERIC` (outro nome para `DECIMAL`)

Promoção de tipo numérico nas expressões

Nas expressões que utilizam apenas os tipos inteiros, o Derby promove o tipo do resultado para pelo menos `INTEGER`. Nas expressões que misturam tipos inteiros e não inteiros, o Derby promove o resultado da expressão para o tipo mais alto da expressão. A tabela [Promoção de tipo nas expressões](#) mostra a promoção dos tipos de dado nas expressões.

Tabela 4. Promoção de tipo nas expressões

Maior tipo que aparece na expressão	Tipo do resultado da expressão
DOUBLE PRECISION	DOUBLE PRECISION
REAL	DOUBLE PRECISION
DECIMAL	DECIMAL
BIGINT	BIGINT
INTEGER	INTEGER
SMALLINT	INTEGER

Por exemplo:

```
-- retorna precisão dupla
VALUES 1 + 1.0e0
-- retorna decimal
VALUES 1 + 1.0
-- retorna inteiro
VALUES CAST (1 AS INT) + CAST (1 AS INT)
```

Armazenamento de valores de um tipo de dado numérico em colunas de outro tipo de dado numérico

A tentativa de colocar um tipo de ponto flutuante com tamanho de armazenamento maior em um local com tamanho de armazenamento menor falha apenas se o valor não puder ser armazenado no local com tamanho menor. Por exemplo:

```
CREATE TABLE MINHA_TABELA (R REAL, D DOUBLE PRECISION);
0 rows inserted/updated/deleted
INSERT INTO MINHA_TABELA (R, D) VALUES (3.4028236E38, 3.4028235E38);
ERRO X0X41: O número '3.4028236E38' está fora da faixa para o tipo de
dado REAL.
```

Pode ser armazenado um tipo de ponto flutuante em uma coluna INTEGER; a parte fracionária do número é truncada. Por exemplo:

```
INSERT INTO MINHA_TABELA(COLUNA_INTEIRA) VALUES (1.09e0);
1 row inserted/updated/deleted
SELECT coluna_inteira
FROM MINHA_TABELA;
I
-----
1
```

Os tipos inteiros sempre podem ser colocados com sucesso em valores numéricos aproximados, embora com uma possível perda de precisão.

Os inteiros podem ser armazenados em decimais, se a precisão do DECIMAL for grande o suficiente para o valor. Por exemplo:

```
ij> INSERT INTO MINHA_TABELA (COLUNA_DECIMAL)
VALUES (5555555555566666666666);
ERRO X0Y21: O número '5555555555566666666666' está fora da faixa do tipo de
dado
de destino DECIMAL/NUMERIC(5,2).
```

A tentativa de colocar um valor inteiro com tamanho de armazenamento maior em um local com tamanho de armazenamento menor falha apenas se o valor não puder ser armazenado no local com tamanho menor. Por exemplo:

```
INSERT INTO MINHA_TABELA (COLUNA_INTEIRA) VALUES 2147483648;
ERRO 22003: O valor resultante está fora da faixa para o tipo de dado
INTEGER.
```

Note: Ao truncar os dígitos no final do valor NUMERIC, o Derby arredonda para baixo.
Escala na aritmética decimal

As instruções SQL podem envolver expressões aritméticas que utilizam tipos de dado decimais com *precisão* (o número total de dígitos, tanto à esquerda quanto à direita do ponto decimal) e *escala* (o número de dígitos da parte fracionária) diferentes. A precisão e a escala do tipo decimal resultante dependem da precisão e da escala dos operandos.

Dada uma expressão aritmética envolvendo dois operandos decimais:

- *lp* significa a precisão do operando à esquerda
- *rp* significa a precisão do operando à direita
- *ls* significa a escala do operando à esquerda
- *rs* significa a escala do operando à direita

São utilizadas as seguintes fórmulas para determinar a escala do tipo de dado resultante para os seguintes tipos de expressões aritméticas:

- *multiplicação*

$ls + rs$

- *divisão*

$31 - lp + ls - rs$

- *AVG()*

$\max(\max(ls, rs), 4)$

- *todas as outras*

$\max(ls, rs)$

Por exemplo, a escala do tipo de dado resultante da expressão a seguir é 27:

```
11.0/1111.33
// 31 - 3 + 1 - 2 = 27
```

São utilizadas as seguintes fórmulas para determinar a precisão do tipo de dado resultante para os seguintes tipos de expressões aritméticas:

- *multiplicação*

$lp + rp$

- *adição*

$2 * (p - s) + s$

- *divisão*

$lp - ls + rp + \max(ls + rp - rs + 1, 4)$

- *todas as outras*

$\max(lp - ls, rp - rs) + 1 + \max(ls, rs)$

Tipo de dado - atribuições e comparação, classificação e ordenação

Tabela 5. Atribuições permitidas pelo Derby

Esta tabela mostra as atribuições válidas entre tipos de dado no Derby. "S" indica que a atribuição é válida.

Tipos	S M A L L I N T	I N T E G E R	B I G I N T	D E C I M A L	R E A L	D O U B L E	F L O A T	C H A R	V A R C H A R	L O N G V A R C H A R	C H A R F O R B I T D A T A	V A R C H A R F O R B I T D A T A	L O N G V A R C H A R F O R B I T D A T A	C L O B	B L O B	D A T E	T I M E	T I M E S T A M P
SMALL INT	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
INTEGER	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
BIGINT	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
DECIMAL	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
REAL	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
DOUBLE	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
FLOAT	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
CHAR	-	-	-	-	-	-	-	S	S	S	-	-	-	S	-	S	S	S
VARCHAR	-	-	-	-	-	-	-	S	S	S	-	-	-	S	-	S	S	S
LONG VARCHAR	-	-	-	-	-	-	-	S	S	S	-	-	-	S	-	-	-	-
CHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	S	S	S	-	-	-	-	-
VARCHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	S	S	S	-	-	-	-	-
LONG VARCHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	S	S	S	-	-	-	-	-
CLOB	-	-	-	-	-	-	-	S	S	S	-	-	-	S	-	-	-	-
BLOB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	S	-	-	-
DATE	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	S	-	-
TIME	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	-	S	-
TIME STAMP	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	-	-	S

Tabela 6. Comparações permitidas pelo Derby

Esta tabela mostra as comparações válidas entre tipos de dado no Derby. "S" indica que a comparação é permitida.

Tipos	S M A L L I N T	I N T E G E R	B I G I N T	D E C I M A L	R E A L	D O U B L E	F L O A T	C H A R	V A R C H A R	L O N G V A R C H A R	C H A R F O R B I T D A T A	V A R C H A R F O R B I T D A T A	L O N G V A R C H A R F O R B I T D A T A	C L O B	B L O B	D A T E	T I M E	T I M E S T A M P
SMALL INT	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
INTEGER	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
BIGINT	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
DECIMAL	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
REAL	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
DOUBLE	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
FLOAT	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-
CHAR	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	S	S	S
VARCHAR	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	S	S	S
LONG VARCHAR	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-
VARCHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-
LONG VARCHAR FOR BIT DATA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CLOB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
BLOB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DATE	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	S	-	-
TIME	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	-	S	-
TIME STAMP	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	-	-	S

BIGINT

BIGINT fornece 8 bytes de armazenamento para valores inteiros.

Sintaxe

BIGINT

Tipo Java correspondente em tempo de compilação

java.lang.Long

Tipo do metadado do JDBC (java.sql.Types)

BIGINT

Valor mínimo

-9223372036854775808 (*java.lang.Long.MIN_VALUE*)

Valor máximo

9223372036854775807 (*java.lang.Long.MAX_VALUE*)

Quando misturado com outros tipos de dado nas expressões, o tipo de dado do resultado segue as regras mostradas em [Promoção de tipo numérico nas expressões](#).

A tentativa de colocar um valor inteiro com tamanho de armazenamento maior em um local com tamanho de armazenamento menor falha apenas se o valor não puder ser armazenado no local com tamanho menor. Os tipos inteiros sempre podem ser colocados com sucesso em valores numéricos aproximados, embora com uma possível perda de precisão. Os valores BIGINT podem ser armazenados em decimais, se a precisão do DECIMAL for grande o suficiente para o valor.

9223372036854775807

BLOB

O BLOB (objeto grande binário) é uma cadeia binária de comprimento variável, que pode ter comprimento de até 2.147.483.647 bytes. Como os outros tipos binários, as cadeias BLOB não estão associadas a uma página de código. Além disso, as cadeias BLOB não armazenam dados caracteres.

O comprimento do BLOB é especificado em bytes, a menos que seja especificado um dos sufixos K, M ou G, relacionados aos múltiplos 1024, 1024*1024 e 1024*1024*1024, respectivamente.

Note: O comprimento do BLOB é especificado em bytes.

Sintaxe

{ BLOB | BINARY LARGE OBJECT } (comprimento [{K | M | G }]))

Tipo Java correspondente em tempo de compilação

java.sql.Blob

Tipo do metadado do JDBC (java.sql.Types)

BLOB

No *java.sql.ResultSet* é utilizado o método *getBlob* para obter um tratador de BLOB para os dados subjacentes.

Informações relacionadas

Consulte [java.sql.Blob](#) e [java.sql.Clob](#)

```
CREATE TABLE IMAGENS(
    NOME VARCHAR(32) NOT NULL PRIMARY KEY,
    IMAGEM BLOB(16M));
-- Encontrar todas as imagens de logotipos
SELECT LENGTH(IMAGEM), NOME FROM IMAGENS WHERE NOME LIKE '%logo%';
-- Encontrar todas as imagens duplicadas (comparação de BLOBs)
SELECT A.NOME AS DUPLICADA_1, B.NOME AS DUPLICADA_2
FROM IMAGENS AS A, IMAGENS AS B
WHERE A.NOME < B.NOME
AND A.IMAGEM = B.IMAGEM
ORDER BY 1,2;
```

CHAR

CHAR fornece armazenamento para cadeias de comprimento fixo.

Sintaxe

```
CHAR[ACTER] [(comprimento)]
```

O *comprimento* é uma constante inteira sem sinal. O comprimento padrão de CHAR é 1.

Tipo Java correspondente em tempo de compilação

java.lang.String

Tipo do metadado do JDBC (*java.sql.Types*)

CHAR

O Derby insere espaços para completar o valor cadeia menor que o comprimento esperado. O Derby trunca espaços do valor cadeia maior que o comprimento esperado. Caracteres diferentes de espaço fazem com que seja lançada uma exceção. Quando são aplicados operadores de comparação binária em tipos CHAR, a cadeia mais curta é completada com espaços até o comprimento da cadeia mais longa.

Quando se mistura CHAR e VARCHAR em expressões, o valor mais curto é completado com espaços até o comprimento do valor mais longo.

O tipo de uma constante cadeia é CHAR.

Aspectos definidos pela implementação

O único limite de comprimento para os tipos de dado CHAR é o valor *java.lang.Integer.MAX_VALUE*.

```
-- Dentro de uma constante cadeia utilizar dois
-- apóstrofos para representar um único apóstrofo
VALUES 'Olá, esta é a cadeia da Maria D''Almeida'
```

CHAR FOR BIT DATA

O tipo de dado CHAR FOR BIT DATA permite armazenar cadeias de bytes de

comprimento fixo. É útil para dados não estruturados, onde as cadeias de caracteres não são apropriadas.

Sintaxe

```
{ CHAR | CHARACTER }[(comprimento)] FOR BIT DATA
```

O *comprimento* é um literal inteiro sem sinal designando o comprimento em bytes.

O *comprimento* padrão para o tipo de dado CHAR FOR BIT DATA é 1, e o tamanho máximo do *comprimento* são 254 bytes.

Tipo do metadado do JDBC (java.sql.Types)

BINARY

CHAR FOR BIT DATA armazena cadeias de bytes de comprimento variável. Se o valor CHAR FOR BIT DATA for menor que CHAR FOR BIT DATA de destino, este será completado com o valor de byte 0x20.

As comparações entre valores CHAR FOR BIT DATA e VARCHAR FOR BIT DATA são precisas. Para duas cadeias de bits serem iguais, devem ser *exatamente* do mesmo comprimento (isto é diferente da maneira como outros SGBDs tratam os valores binários, mas funciona conforme especificado pelo padrão SQL-92).

Uma operação com valores VARCHAR FOR BIT DATA e CHAR FOR BIT DATA (por exemplo, uma concatenação), tem como resultado um valor VARCHAR FOR BIT DATA.

```
CREATE TABLE T (B CHAR(2) FOR BIT DATA);
INSERT INTO T VALUES (X'DE');
SELECT *
FROM T;
-- produz o seguinte resultado
B
-----
de20
```

CLOB

O valor CLOB (objeto grande caractere) pode ter comprimento de até 2.147.483.647 caracteres. O CLOB é utilizado para armazenar dados baseados em caractere UNICODE, como documentos grandes em qualquer conjunto de caracteres.

O comprimento do CLOB é especificado em caracteres, a menos que seja especificado um dos sufixos K, M ou G, relacionados aos múltiplos 1024, 1024*1024 e 1024*1024*1024, respectivamente.

Note: O comprimento do CLOB é especificado em caracteres (UNICODE).

Sintaxe

```
{CLOB | CHARACTER LARGE OBJECT}(comprimento [{K | M | G}])
```

Tipo Java correspondente em tempo de compilação

java.sql.Clob

Tipo do metadado do JDBC (java.sql.Types)

CLOB

Tipo do metadado do JDBC*java.sql.Clob***Tipo do metadado do JDBC (java.sql.Types)**

CLOB

No *java.sql.ResultSet* é utilizado o método *getClob* para obter um tratador de CLOB para os dados subjacentes.

Informação relacionada

Consulte [java.sql.Blob](#) e [java.sql.Clob](#).

```
import java.sql.*;

public class clob
{
    public static void main(String[] args) {
        try {
            String url =
"jdbc:derby:clobberyclob;create=true";
            Class.forName("org.apache.derby.jdbc.EmbeddedDriver").newInstance();
            Connection conn =
            DriverManager.getConnection(url);

            Statement s = conn.createStatement();
            s.executeUpdate("CREATE TABLE documents (id INT, text CLOB(64
K))");
            conn.commit();

            // --- adicionar um arquivo
            java.io.File file = new java.io.File("asciifile.txt");
            int fileLength = (int) file.length();

            // - primeiro, criar um fluxo de entrada
            java.io.InputStream fin = new java.io.FileInputStream(file);
            PreparedStatement ps = conn.prepareStatement("INSERT
            INTO documents VALUES (?, ?)");
            ps.setInt(1, 1477);

            // - definir o valor do parâmetro de entrada para o fluxo de
            entrada
            ps.setAsciiStream(2, fin, fileLength);
            ps.execute();
            conn.commit();

            // --- ler as colunas
            ResultSet rs = s.executeQuery("SELECT text FROM documents
            WHERE id = 1477");
            while (rs.next()) {
                java.sql.Clob aclob = rs.getClob(1);
                java.io.InputStream ip = rs.getAsciiStream(1);
                int c = ip.read();
                while (c > 0) {
                    System.out.print((char)c);
                    c = ip.read();
                }
                System.out.print("\n");
                // ...
            }
        } catch (Exception e) {
            System.out.println("Erro! "+e);
        }
    }
}
```

DATE

DATE fornece armazenamento para ano-mês-dia na faixa suportada por *java.sql.Date*.

Sintaxe**DATE**

Tipo Java correspondente em tempo de compilação*java.sql.Date***Tipo do metadado do JDBC (java.sql.Types)**

DATE

As expressões não devem misturar datas, horas e carimbos do tempo.

É permitido qualquer valor reconhecido pelo método *java.sql.Date* na coluna com tipo de dado data/hora SQL correspondente. O Derby suporta os seguintes formatos para DATE:

```
yyyy-mm-dd
mm/dd/yyyy
dd.mm.yyyy
```

O primeiro dos três formatos acima é o formato de *java.sql.Date*.

O ano deve ser sempre expresso com quatro dígitos, enquanto os meses e os dias podem ter um ou dois dígitos.

O Derby também aceita cadeias no formato de data/hora específico do idioma, utilizando idioma do servidor de banco de dados. No caso de haver ambiguidade, os formatos nativos acima têm precedência.

Exemplos

```
VALUES DATE('1994-02-23')
VALUES '1993-09-01'
```

DECIMAL

DECIMAL fornece um numérico exato onde a precisão e a escala podem ter tamanhos arbitrários. Pode ser especificada a precisão (o número total de dígitos, tanto à esquerda quanto à direita do ponto decimal), e a escala (o número de dígitos da parte fracionária). A quantidade de armazenamento necessária é baseada na precisão.

Sintaxe

```
{ DECIMAL | DEC } [(precisão [, escala ])]
```

A *precisão* deve estar entre 1 e 31. A *escala* deve ser menor ou igual a precisão.

Quando a escala não é especificada, é utilizado o valor padrão igual a 0. Quando a precisão não é especificada, é utilizada a precisão padrão igual a 5.

A tentativa de colocar um valor numérico em um DECIMAL é permitida, desde que a precisão não fracionária não seja perdida. Ao remover os dígitos finais do valor DECIMAL, o Derby arredonda para baixo.

Por exemplo:

```
-- esta conversão perde apenas a precisão fracionária
VALUES CAST (1.798765 AS DECIMAL(5,2));
1
-----
1.79
-- esta conversão não cabe
VALUES CAST (1798765 AS DECIMAL(5,2));
1
-----
ERRO 22003: O valor resultante está fora da faixa para o tipo de dado
DECIMAL/NUMERIC(5,2).
```

Quando misturado com outros tipos de dado nas expressões, o tipo de dado do resultado segue as regras mostradas em [Promoção de tipo numérico nas expressões](#).

Consulte também [Armazenamento de valores de um tipo de dado numérico em colunas de outro tipo de dado numérico](#).

Quando são misturados dois valores decimais em uma expressão, a escala e a precisão do valor resultante seguem as regras mostradas em [Escala na aritmética decimal](#).

Tipo Java correspondente em tempo de compilação

java.math.BigDecimal

Tipo do metadado do JDBC (java.sql.Types)

DECIMAL

```
VALUES 123.456
```

```
VALUES 0.001
```

As constantes inteiras muito grandes para BIGINT são tornadas constantes DECIMAL.

DOUBLE

O tipo de dado DOUBLE é um sinônimo para o tipo de dado [DOUBLE PRECISION](#).

Sintaxe

```
DOUBLE
```

DOUBLE PRECISION

O tipo de dado DOUBLE PRECISION fornece de 8 bytes de armazenamento para números, utilizando a notação de ponto flutuante do IEEE.

Sintaxe

```
DOUBLE PRECISION
```

ou, como alternativa

```
DOUBLE
```

DOUBLE pode ser utilizado como sinônimo para DOUBLE PRECISION.

Limitações

Faixa dos valores DOUBLE:

- Menor valor DOUBLE: -1.79769E+308
- Maior valor DOUBLE: 1.79769E+308
- Menor valor DOUBLE positivo: 2.225E-307
- Maior valor DOUBLE negativo: -2.225E-307

Estes limites são diferentes dos limites do tipo Java `java.lang.Double`.

É lançada uma exceção quando é calculado ou entrado um valor de precisão dupla fora desta faixa de valores. As operações aritméticas **não** arredondam o valor de seus

resultados para zero. Se o valor for muito pequeno, será lançada uma exceção.

As constantes numéricas de ponto flutuante estão limitadas a um comprimento de 30 caracteres.

```
-- este exemplo falha porque a constante é muito longa:  
VALUES 01234567890123456789012345678901e0;
```

Tipo Java correspondente em tempo de compilação

java.lang.Double

Tipo do metadado do JDBC (java.sql.Types)

DOUBLE

Quando misturado com outros tipos de dado nas expressões, o tipo de dado do resultado segue as regras mostradas em [Promoção de tipo numérico nas expressões](#).

Consulte também [Armazenamento de valores de um tipo de dado numérico em colunas de outro tipo de dado numérico](#).

Exemplos

```
3421E+09  
425.43E9  
9E-10  
4356267544.32333E+30
```

FLOAT

O tipo de dado FLOAT é um aliás para o tipo de dado REAL ou DOUBLE PRECISION, dependendo da precisão especificada.

Sintaxe

```
FLOAT [ (precisão) ]
```

A *precisão* padrão para FLOAT é 53, e é equivalente a DOUBLE PRECISION. A precisão igual a 23, ou menor, torna FLOAT equivalente a REAL. A precisão igual a 24, ou maior, torna FLOAT equivalente a DOUBLE PRECISION. Se for especificada uma precisão igual a 0, ocorrerá um erro. Se for especificada uma precisão negativa, ocorrerá um erro de sintaxe.

Tipo do metadado do JDBC (java.sql.Types)

REAL ou DOUBLE

Limitações

Se estiver sendo utilizada uma precisão igual a 24, ou maior, os limites para FLOAT serão semelhantes aos de DOUBLE.

Se estiver sendo utilizada uma precisão igual a 23, ou menor, os limites para FLOAT serão semelhantes aos de REAL.

INTEGER

INTEGER fornece 4 bytes de armazenamento para valores inteiros.

Sintaxe

```
{ INTEGER | INT }
```

Tipo Java correspondente em tempo de compilação

java.lang.Integer

Tipo do metadado do JDBC (java.sql.Types)

INTEGER

Valor mínimo

-2.147.483.648 (*java.lang.Integer.MIN_VALUE*)

Valor máximo

2.147.483.647 (*java.lang.Integer.MAX_VALUE*)

Quando misturado com outros tipos de dado nas expressões, o tipo de dado do resultado segue as regras mostradas em [Promoção de tipo numérico nas expressões](#).

Consulte também [Armazenamento de valores de um tipo de dado numérico em colunas de outro tipo de dado numérico](#).

```
3453
425
```

LONG VARCHAR

O tipo de dado LONG VARCHAR permite armazenar cadeias de caracteres com comprimento máximo de 32.700 caracteres. É idêntico a VARCHAR, exceto por não ser necessário especificar o comprimento máximo ao criar colunas deste tipo.

Sintaxe

```
LONG VARCHAR
```

Tipo Java correspondente em tempo de compilação

java.lang.String

Tipo do metadado do JDBC (java.sql.Types)

LONGVARCHAR

Ao converter valores Java para valores SQL, nenhum tipo da linguagem Java corresponde a LONG VARCHAR.

LONG VARCHAR FOR BIT DATA

O tipo de dado LONG VARCHAR FOR BIT DATA permite armazenar cadeias de bits de até 32.700 bytes. É idêntico a [VARCHAR FOR BIT DATA](#), exceto por não ser necessário especificar o comprimento máximo ao criar colunas deste tipo.

Sintaxe

```
LONG VARCHAR FOR BIT DATA
```

NUMERIC

NUMERIC é sinônimo de [DECIMAL](#), e se comporta da mesma maneira. Consulte [DECIMAL](#).

Sintaxe

```
NUMERIC [(precisão [, escala ])]
```

Tipo Java correspondente em tempo de compilação

java.math.BigDecimal

JDBC metadata Ttype (java.sql.Types)

NUMERIC

```
123.456
.001
```

REAL

O tipo de dado REAL fornece 4 bytes de armazenamento para números, utilizando a notação de ponto flutuante do IEEE.

Sintaxe

```
REAL
```

Tipo Java correspondente em tempo de compilação

java.lang.Float

Tipo do metadado do JDBC (java.sql.Types)

REAL

Limitações

Faixa de valores de REAL:

- Menor valor REAL: -3.402E+38
- Maior valor REAL: 3.402E+38
- Menor valor REAL positivo: 1.175E-37
- Maior valor REAL negativo: -1.175E-37

Estes limites são diferentes dos limites do tipo Java *java.lang.Float*.

É lançada uma exceção quando é calculado ou entrado um valor de precisão dupla fora desta faixa de valores. As operações aritméticas **não** arredondam o valor de seus resultados para zero. Se o valor for muito pequeno, será lançada uma exceção. As operações aritméticas são realizadas em aritmética de precisão dupla, com a finalidade de detectar números muito pequenos (underflow).

As constantes numéricas de ponto flutuante estão limitadas a um comprimento de 30 caracteres.

```
-- este exemplo falha porque a constante é muito longa:
values 012345678901234567890123456789012345678901e0;
```

Quando misturado com outros tipos de dado nas expressões, o tipo de dado do resultado segue as regras mostradas em [Promoção de tipo numérico nas expressões](#).

Consulte também [Armazenamento de valores de um tipo de dado numérico em colunas de outro tipo de dado numérico](#).

As constantes são sempre mapeadas para DOUBLE PRECISION; deve ser utilizado CAST para converter para REAL.

SMALLINT

SMALLINT fornece 2 bytes de armazenamento.

Sintaxe

SMALLINT

Tipo Java correspondente em tempo de compilação

java.lang.Short

Tipo do metadado do JDBC (java.sql.Types)

SMALLINT

Valor mínimo

-32.768 (*java.lang.Short.MIN_VALUE*)

Valor máximo

32.767 (*java.lang.Short.MAX_VALUE*)

Quando misturado com outros tipos de dado nas expressões, o tipo de dado do resultado segue as regras mostradas em [Promoção de tipo numérico nas expressões](#).

Consulte também [Armazenamento de valores de um tipo de dado numérico em colunas de outro tipo de dado numérico](#).

As constantes no formato apropriado sempre são mapeadas para INTEGER ou BIGINT, dependendo de seus comprimento.

TIME

TIME fornece armazenamento para valor da hora do dia.

Sintaxe

TIME

Tipo Java correspondente em tempo de compilação

java.sql.Time

Tipo do metadado do JDBC (java.sql.Types)

TIME

Datas, horas e carimbos do tempo não podem ser misturados em expressões, exceto com CAST.

É permitido qualquer valor reconhecido pelo método *java.sql.Time* na coluna com tipo de dado data/hora SQL correspondente. O Derby suporta os seguintes formatos para TIME:

```
hh:mm[:ss]
hh:mm[.ss]
hh[:mm] {AM | PM}
```

O primeiro dos três formatos acima é o formato de *java.sql.Time*.

As horas podem ter um ou dois dígitos. Os minutos e os segundos, se estiverem presentes, devem ter dois dígitos.

O Derby também aceita cadeias no formato de data/hora específico do idioma, utilizando o idioma do servidor de banco de dados. No caso de haver ambiguidade, os formatos nativos acima têm precedência.

Exemplos

```
VALUES TIME('15:09:02')
VALUES '15:09:02'
```

TIMESTAMP

TIMESTAMP armazena o valor combinado de DATE e TIME. Permite um valor de segundos fracionário com até nove dígitos.

Sintaxe

```
TIMESTAMP
```

Tipo Java correspondente em tempo de compilação

java.sql.Timestamp

Tipo do metadado do JDBC (java.sql.Types)

TIMESTAMP

Datas, horas e carimbos do tempo não podem ser misturados em expressões, exceto com CAST.

O Derby suporta os seguintes formatos para TIMESTAMP:

```
yyyy-mm-dd hh[:mm[:ss[.nnnnnn]]]
yyyy-mm-dd-hh[.mm[.ss[.nnnnnn]]]
```

O primeiro dos dois formatos acima é o formato de *java.sql.Timestamp*.

O ano sempre deve ter quatro dígitos. Os meses, dias e horas podem ter um ou dois dígitos. Os minutos e os segundos, se estiverem presentes, devem ter dois dígitos. Os nanossegundos, se estiverem presentes, podem ter de um a seis dígitos.

O Derby também aceita cadeias no formato de data/hora específico do idioma, utilizando o idioma do servidor de banco de dados. No caso de haver ambiguidade, os formatos nativos acima têm precedência.

Exemplos

```
VALUES '1960-01-01 23:03:20'
VALUES TIMESTAMP('1962-09-23 03:23:34.234')
VALUES TIMESTAMP('1960-01-01 23:03:20')
```

VARCHAR

VARCHAR fornece armazenamento para cadeias de comprimento variável.

Sintaxe

```
{ VARCHAR | CHAR VARYING | CHARACTER VARYING }(comprimento)
```

O *comprimento* é uma constante inteira sem sinal, não devendo ser maior que a restrição para o inteiro usado para especificar o comprimento, que é o valor *java.lang.Integer.MAX_VALUE*.

O comprimento máximo para uma cadeia VARCHAR é 32672 caracteres.

Tipo Java correspondente em tempo de compilação

java.lang.String

Tipo do metadado do JDBC (java.sql.Types)

VARCHAR

O Derby não completa o valor VARCHAR com comprimento menor que o especificado. O Derby trunca espaços do valor cadeia, quando é fornecida uma cadeia com comprimento maior que o esperado pelo VARCHAR. Os caracteres diferentes de espaço não são truncados; em vez disto faz com que seja lançada uma exceção. Quando são aplicados operadores de comparação binários a VARCHAR os comprimentos dos operandos não são alterados, e os espaços no final dos valores são ignorados.

Quando são misturados CHAR e VARCHAR em expressões, o valor mais curto é completado com espaços até o comprimento do valor mais longo.

O tipo da constante cadeia é CHAR, e não VARCHAR.

VARCHAR FOR BIT DATA

O tipo de dado VARCHAR FOR BIT DATA permite armazenar cadeias binárias menores ou iguais ao comprimento especificado. É útil para dados não estruturados, onde as cadeias de caracteres não são apropriadas (por exemplo, imagens).

Sintaxe

```
{ VARCHAR | CHAR VARYING | CHARACTER VARYING }(comprimento) FOR BIT DATA
```

O *comprimento* é um literal inteiro sem sinal designando o comprimento em bytes.

Diferentemente do tipo de dado CHAR FOR BIT DATA, não existe *comprimento* padrão para o tipo de dado VARCHAR FOR BIT DATA. O tamanho máximo do *comprimento* é de 32672 bytes.

Tipo do metadado do JDBC (java.sql.Types)

VARBINARY

VARCHAR FOR BIT DATA armazena cadeias de bytes de comprimento variável. Diferentemente dos valores CHAR FOR BIT DATA, os valores VARCHAR FOR BIT DATA não são completados até o comprimento do destino.

Uma operação envolvendo valores VARCHAR FOR BIT DATA e CHAR FOR BIT DATA (por exemplo, uma concatenação) produz um valor VARCHAR FOR BIT DATA.

O tipo do literal byte é sempre VARCHAR FOR BIT DATA, e não CHAR FOR BIT DATA.

Expressões SQL

A sintaxe de muitas instruções e expressões inclui o termo *Expressão*, ou um termo para um tipo específico de expressão como *SubconsultaTabela*. Dentro das instruções, são permitidas expressões nestes pontos especificados. Alguns locais permitem apenas um tipo específico de expressão, ou uma expressão com uma propriedade específica. A [Tabela de Expressões](#) lista todas as expressões SQL possíveis, e indica onde são permitidas.

Quando não é especificado o contrário, é permitida uma expressão em qualquer lugar onde aparece *Expressão* na sintaxe. Isto inclui:

- [ExpressãoSeleção](#)
- [Instrução UPDATE](#) (porção SET)
- [VALUES Expressão](#)
- [Cláusula WHERE](#)

Obviamente, muitas outras instruções incluem estes elementos como blocos de construção e, portanto, permitem expressões como parte destes elementos.

Tabela 7. Tabela de Expressões

	Tipo da expressão	Explicação
<i>Expressões gerais</i>		
'	Referência a coluna Permitida na ExpressãoSeleção , na instrução UPDATE e na cláusula WHERE das instruções de manipulação de dados.	O nome-da-coluna que faz referência ao valor da coluna tornado visível para a expressão que contém a referência a coluna. O nome-da-coluna deve ser qualificado pelo nome da tabela ou pelo nome da correlação se houver ambiguidade. O qualificador do nome-da-coluna deve ser o nome da correlação, se for atribuído nome de correlação à tabela na Cláusula FROM . O nome da tabela não é mais visível como qualificador para nome-da-coluna após a tabela ter recebido um aliás pelo nome da correlação.
'	Constante	A maioria dos tipos de dado nativos normalmente possuem constantes associadas aos mesmos (conforme mostrado em Tipos de dado).
'	NULL Permitido nas expressões CAST, na lista de valores do INSERT, e na cláusula SET do UPDATE. Utilizado na expressão CAST o nulo recebe um tipo de dado específico.	NULL é uma constante sem tipo que representa um valor desconhecido.
'	Parâmetro dinâmico Permitido em qualquer lugar na expressão onde o tipo de dado pode ser facilmente deduzido. Consulte Parâmetros dinâmicos .	Parâmetro dinâmico é o parâmetro de uma instrução SQL para o qual não é especificado valor quando a instrução é criada. Em vez disto, a instrução possui um ponto de interrogação (?) marcando a posição de cada um dos parâmetros dinâmicos. Consulte Parâmetros dinâmicos .

ID	Tipo da expressão	Explicação
		Os parâmetros dinâmicos são permitidos apenas em instruções preparadas. Devem ser especificados valores para os mesmos antes da instrução preparada ser executada. Os valores especificados devem corresponder aos tipos esperados.
'	Expressão CAST	Permite especificar o tipo de dado de NULL ou de um parâmetro dinâmico, ou converter um valor para outro tipo de dado. Consulte CAST .
'	Subconsulta escalar	Uma subconsulta que retorna uma única linha com uma única coluna. Consulte SubconsultaEscalar .
'	Subconsulta tabela Permitida como <i>ExpressãoTabela</i> na cláusula FROM, e com EXISTS, IN e comparações quantificadas.	Uma subconsulta que retorna mais de uma coluna e mais de uma linha. Consulte SubconsultaTabela .
'	Expressão condicional	A expressão condicional escolhe a expressão a ser avaliada baseado em um teste booleano.
Expressões booleanas		
Expressões numéricas		
'	Expressões +, -, *, /, e + - unários	<p>+ -, *, /, e + - unários</p> <p>Avaliam a operação matemática esperada nos operandos. Se os dois operandos forem do mesmo tipo, o tipo do resultado não será promovido, e portanto o operador de divisão em operandos inteiros resulta em um inteiro que é o truncamento do resultado numérico real. Quando são misturados tipos de dado diferentes, estes são promovidos conforme descrito em Tipos de dado.</p> <p>O + unário é um não-operador (ou seja, +4 é o mesmo que 4). O - unário é o mesmo que multiplicar o valor por -1, mudando efetivamente o sinal.</p>
'	AVG	Retorna a média de um conjunto de valores numéricos. AVG
'	SUM	Retorna a soma de um conjunto de valores numéricos. SUM
'	LENGTH	Retorna o número de caracteres em uma cadeia de caracteres ou de bits. Consulte LENGTH .
'	LOWER	Consulte LCASE ou LOWER .
'	COUNT	Retorna o contagem de um conjunto de valores. Consulte COUNT e COUNT(*) .
Expressões de caractere		

	Tipo da expressão	Explicação
'	Um valor CHAR ou VARCHAR que utiliza curingas. Usadas no padrão do LIKE.	Os caracteres curinga % e _ transformam a cadeia de caracteres em um padrão com relação ao qual o operador LIKE pode procurar por uma correspondência.
'	Expressão de concatenação	Em uma operação de concatenação, o operador de concatenação, " ", concatena seu operando à direita ao seu operando à esquerda. Opera em cadeias de caracteres e de bits. Consulte Concatenação .
'	Funções de cadeia nativas	As funções de cadeia nativas atuam em uma cadeia e retornam uma cadeia. Consulte LTRIM , LCASE ou LOWER , RTRIM , SUBSTR e UCASE ou UPPER
'	Funções USER	As funções USER retornam informações sobre o usuário corrente na forma de uma cadeia de caracteres. Consulte CURRENT_USER , SESSION_USER , e USER
Expressões de data e hora		
'	CURRENT_DATE	Retorna a data corrente. Consulte CURRENT_DATE .
'	CURRENT_TIME	Retorna a hora corrente. Consulte CURRENT_TIME .
'	CURRENT_TIMESTAMP	Retorna o carimbo do tempo corrente. Consulte CURRENT_TIMESTAMP .

Precedência das expressões

A precedência das operações, da mais alta para a mais baixa, é:

- (), ?, Constante (incluindo sinal), NULL, *ReferênciaColuna*, *SubconsultaEscalar*, CAST
- LENGTH, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, e outras operações nativas
- + e - unários
- *, /, || (concatenação)
- + e - binários
- comparações, comparações quantificadas, EXISTS, IN, IS NULL, LIKE, BETWEEN, IS
- NOT
- AND
- OR

A precedência pode ser especificada explicitamente colocando as expressões entre parênteses. A expressão entre parênteses é avaliada antes de qualquer operação fora dos parênteses ser aplicada a mesma.

Exemplo

```
(3+4)*9
(idade < 16 OR idade > 65) AND empregado = TRUE
```

Expressão booleana

As expressões booleanas são permitidas na cláusula WHERE e nas restrições de verificação. As expressões booleanas nas restrições de verificação possuem limitações não descritas aqui; para obter mais informações deve ser consultada a [Cláusula CONSTRAINT](#). As expressões booleanas nas cláusulas WHERE possuem uma sintaxe bem liberal; consulte, por exemplo, a [Cláusula WHERE](#).

As expressões booleanas podem incluir um ou mais operadores booleanos, listados em [Operadores booleanos SQL](#).

Tabela 8. Operadores booleanos SQL

Operador	Explicação e exemplo	Sintaxe
AND, OR, NOT	Avalia os operandos, que são expressões booleanas <pre>(AEROPORTO_ORIGEM = 'SFO') OR (AEROPORTO_DESTINO = 'GRU') -- retorna verdade</pre>	<pre>{ Expressão AND Expressão Expressão OR Expressão NOT Expressão }</pre>
Comparações	<, =, >, <=, >=, <> são aplicáveis a todos os tipos nativos. <pre>DATE('1998-02-26') < DATE('1998-03-01') -- retorna verdade</pre>	<pre>Expressão { < = > <= >= <> } Expressão</pre>
IS NULL, IS NOT NULL	Testa se o resultado de uma expressão é nulo ou não. <pre>WHERE NOME_DO_MEIO IS NULL</pre>	<pre>Expressão IS [NOT] NULL</pre>
LIKE	Tenta fazer a correspondência entre uma expressão de caractere e um padrão de caractere, que é uma cadeia de caracteres que inclui um ou mais caracteres curinga. O caractere % corresponde a qualquer número (zero ou mais) de caracteres na posição correspondente na primeira expressão de caractere. O caractere _ corresponde a um caractere na posição correspondente na expressão de caractere. Qualquer outro caractere corresponde apenas a este caractere na posição correspondente na expressão de caractere. <pre>cidade LIKE 'Sant_'</pre> Para tratar os caracteres % e _ como caracteres constante, o caractere deve ser precedido por um caractere de escape opcional, especificado na cláusula ESCAPE.	<pre>ExpressãoCaractere [NOT] LIKE ExpressãoCaractere ComCaractereCuringa [ESCAPE 'CaractereDeEscape']</pre>

Operador	Explicação e exemplo	Sintaxe
	<pre>SELECT a FROM tabA WHERE a LIKE '%=_ ' ESCAPE '='</pre>	
BETWEEN	<p>Testa se o primeiro operando está entre o segundo e o terceiro operandos. O segundo operando deve ser menor que o terceiro operando. Aplicável apenas aos tipos de dado onde se pode aplicar <= e >=.</p> <pre>WHERE DATA_DE_RESERVA BETWEEN DATE('1998-02-26') AND DATE('1998-03-01')</pre>	<pre>Expressão [NOT] BETWEEN Expressão AND Expressão</pre>
IN	<p>Opera sobre uma subconsulta a tabela ou uma lista de valores. Retorna TRUE se o valor da expressão à esquerda está presente no resultado da subconsulta a tabela ou na lista de valores. A subconsulta a tabela pode retornar várias linhas, mas deve retornar uma única coluna.</p> <pre>WHERE DATA_DE_RESERVA NOT IN (SELECT DATA_DE_RESERVA FROM RESERVAS_EM_HOTEL WHERE QUARTOS_DISPONÍVEIS = 0)</pre>	<pre>{ Expressão [NOT] IN SubconsultaTabela Expressão [NOT] IN (Expressão [, Expressão]*) }</pre>
EXISTS	<p>Opera sobre uma subconsulta a tabela. Retorna TRUE se a subconsulta a tabela retornar alguma linha, e FALSE se não retornar nenhuma linha. A subconsulta a tabela pode retornar várias colunas (somente se for utilizado * para indicar várias colunas) e linhas.</p> <pre>WHERE EXISTS (SELECT * FROM VÔOS WHERE AEROPORTO_DESTINO = 'SFO' AND AEROPORTO_ORIGEM = 'GRU')</pre>	<pre>[NOT] EXISTS SubconsultaTabela</pre>
Comparação quantificada	<p>A comparação quantificada é um operador de comparação (<, =, >, <=, >=, <>) com ALL, ANY ou SOME aplicado.</p> <p>Opera sobre subconsultas a tabela, que podem retornar várias linhas, mas devem retornar uma única coluna.</p> <p>Se for utilizado ALL, a comparação deverá ser verdade para todos os valores retornados pela subconsulta a tabela. Se for utilizado ANY ou SOME, a comparação deverá ser verdade para pelo menos um valor da subconsulta a tabela. ANY e SOME são equivalentes.</p> <pre>WHERE TAXA_NORMAL < ALL (SELECT ORÇAMENTO/550 FROM GRUPOS)</pre>	<pre>Expressão OperadorComparação { ALL ANY SOME } SubconsultaTabela</pre>

Parâmetros dinâmicos

Podem ser preparadas instruções onde é permitido haver parâmetros para os quais o valor não é especificado quando a instrução é preparada utilizando os métodos *PreparedStatement* da API do JDBC. Estes parâmetros são chamados de parâmetros dinâmicos, e são representados por uma ?.

Os documentos da API do JDBC fazem referência aos parâmetros dinâmicos como parâmetros IN, INOUT ou OUT. Na linguagem SQL, estes são sempre parâmetros IN.

Novo: O Derby suporta a interface *ParameterMetaData*, nova no JDBC 3.0. Esta interface descreve o número, tipo e propriedades dos parâmetros da instrução preparada. Para obter mais informações deve ser consultado o *Guia do Desenvolvedor do Derby*.

Devem ser especificados valores para os parâmetros dinâmicos antes de executar a instrução. Os valores especificados devem corresponder aos tipos esperados.

Exemplo de parâmetros dinâmicos

```
PreparedStatement ps2 = conn.prepareStatement(
    "UPDATE DISPONIBILIDADE_HOTEL SET QUARTOS_DISPONÍVEIS = " +
    "(QUARTOS_DISPONÍVEIS - ?) WHERE ID_HOTEL = ? " +
    "AND DATA_DE_RESERVA BETWEEN ? AND ?");
-- este código de exemplo define os valores dos parâmetros dinâmicos
-- como sendo iguais aos valores de variáveis do programa
ps2.setInt(1, numeroQuartos);
ps2.setInt(2, oHotel.idHotel);
ps2.setDate(3, chegada);
ps2.setDate(4, saida);
updateCount = ps2.executeUpdate();
```

Onde são permitidos parâmetros dinâmicos

Os parâmetros dinâmicos podem ser utilizados em qualquer lugar da expressão onde seu tipo de dado pode ser facilmente deduzido.

1. É permitido o uso como primeiro operando do BETWEEN, se o segundo ou o terceiro operando não for um parâmetro dinâmico. O tipo de dado do primeiro operando é assumido como sendo o tipo de dado do parâmetro que não é dinâmico, ou o resultado da união dos tipos do segundo e do terceiro operandos, se estes dois não forem parâmetros dinâmicos.

```
WHERE ? BETWEEN DATE('1996-01-01') AND ?
-- tipos assumidos como sendo DATE
```

2. É permitido o uso como segundo ou terceiro operando do BETWEEN. O tipo de dado é assumido como sendo o tipo de dado do operando à esquerda.

```
WHERE DATE('1996-01-01') BETWEEN ? AND ?
-- tipos assumidos como sendo DATE
```

3. É permitido o uso como operando à esquerda da lista IN se pelo menos um item da lista não for um parâmetro dinâmico. O tipo de dado do operando à esquerda é assumido como sendo o resultado da união dos tipos de dado dos parâmetros não dinâmicos da lista.

```
WHERE ? NOT IN (?, ?, 'Santiago')
-- tipos assumidos como sendo CHAR
```

4. É permitido o uso na lista de valores do predicado IN, se o primeiro operando não for um parâmetro dinâmico, ou seu tipo foi determinado pela regra anterior. O tipo dos parâmetros dinâmicos que aparecem na lista de valores é assumido como sendo o tipo do operando à esquerda.

```
WHERE ColunaPontoFlutuante IN (?, ?, ?)
-- tipos assumidos como sendo FLOAT
```

5. Para os operadores binários +, -, *, /, AND, OR, <, >, =, <>, <= e >=, é permitido o uso do parâmetro dinâmico como um dos operandos, mas não os dois. Seu tipo de dado é obtido a partir do tipo do outro lado.

```
WHERE ? < CURRENT_TIMESTAMP
-- tipo assumido como sendo TIMESTAMP
```

6. A utilização em CAST é sempre permitida, porque CAST fornece tipo de dado ao parâmetro dinâmico.

```
CALL valueOf(CAST (? AS VARCHAR(10)))
```

7. É permitido o uso nos dois lados do operador LIKE. Quando utilizado no lado esquerdo, o tipo de dado do parâmetro dinâmico é definido como sendo o tipo de dado do operando à direita, mas com o comprimento máximo permitido para o tipo de dado. Quando utilizado no lado direito, o tipo de dado é assumido como sendo do mesmo comprimento e tipo de dado do operando à esquerda (LIKE é permitido nos tipos de dado CHAR e VARCHAR; para obter mais informações deve ser consultado [Concatenação](#)).

```
WHERE ? LIKE 'Santi%'
-- tipo assumido como sendo CHAR com comprimento igual a
-- java.lang.Integer.MAX_VALUE
```

8. O parâmetro ?, sem mais nada, é permitido em apenas um dos lados do operador ||, ou seja, "? || ?" não é permitido. O tipo do parâmetro ? de um dos lados do operador || é determinado pelo tipo da expressão do outro lado do operador ||. Se a expressão do outro lado for do tipo de dado CHAR ou VARCHAR, o tipo do parâmetro será VARCHAR com o comprimento máximo permitido para o tipo de dado. Se a expressão do outro lado for do tipo de dado CHAR FOR BIT DATA ou VARCHAR FOR BIT DATA, o tipo de dado do parâmetro será VARCHAR FOR BIT DATA com o comprimento máximo permitido para o tipo de dado.

```
SELECT coluna_bit || ?
FROM tabela_usuario
-- tipo assumido como sendo CHAR FOR BIT DATA
-- com o comprimento especificado para coluna_bit
```

9. Em uma expressão condicional, que utiliza ?, também é permitida a utilização do parâmetro dinâmico (que também é representado por ?). O tipo do parâmetro dinâmico como primeiro operando é assumido como sendo booleano. Com relação ao segundo e terceiro operandos, somente um dos dois pode ser um parâmetro dinâmico, e seu tipo é assumido como sendo o mesmo do outro operando (ou seja, o terceiro e segundo operando, respectivamente).

```
SELECT c1 IS NULL ?? : c1
-- permite especificar o valor "padrão" em tempo de execução
-- o parâmetro dinâmico é assumido como tendo o tipo de c1
-- não podem haver parâmetros dinâmicos dos dois lados dos :
```

10. É permitido usar parâmetro dinâmico como item da lista de valores ou da lista de seleção da instrução INSERT. O tipo do parâmetro dinâmico é assumido como sendo o tipo da coluna de destino. O parâmetro ?, sem mais nada, não é permitido na lista de seleção, inclusive na lista de seleção da subconsulta, a menos que exista uma coluna correspondente em UNION, INTERSECT ou EXCEPT (consulte o nº 16, abaixo) que não seja dinâmica.

```
INSERT INTO t VALUES (?)
-- o parâmetro dinâmico é assumido como sendo
-- do tipo da única coluna da tabela t
INSERT INTO t SELECT ?
FROM t2
-- não é permitido
```

11. O parâmetro ? na comparação com uma subconsulta recebe seu tipo a partir da expressão sendo selecionada na subconsulta. Por exemplo:

```
SELECT *
FROM tab1
WHERE ? = (SELECT x FROM tab2)

SELECT *
FROM tab1
```

```
WHERE ? = ANY (SELECT x FROM tab2)
-- Nos dois casos, o tipo do parâmetro dinâmico é
-- assumido como sendo o mesmo tipo de tab2.x.
```

12. É permitido usar parâmetro dinâmico como o valor em uma instrução UPDATE. O tipo do parâmetro dinâmico é assumido como sendo o tipo da coluna na tabela de destino.

```
UPDATE t2 SET c2 =? -- o tipo é assumido como sendo o tipo de c2
```

13. Não é permitido parâmetro dinâmico como operando dos operadores unários - e +.
14. LENGTH permite parâmetro dinâmico. O tipo é assumido como sendo o comprimento máximo do tipo VARCHAR.

```
SELECT LENGTH(?)
```

15. Comparações qualificadas.

```
? = SOME (SELECT 1 FROM t)
-- é válido. O parâmetro dinâmico é assumido como sendo do tipo
INTEGER
1 = SOME (SELECT ? FROM t)
-- é válido. O parâmetro dinâmico é assumido como sendo do tipo
INTEGER
```

16. É permitido usar parâmetro dinâmico para representar uma coluna se aparecer em uma expressão UNION, INTERSECT ou EXCEPT; O Derby pode inferir o tipo de dado a partir da coluna correspondente na expressão.

```
SELECT ?
FROM t
UNION SELECT 1
FROM t
-- o parâmetro dinâmico é assumido como sendo INT
VALUES 1 UNION VALUES ?
-- o parâmetro dinâmico é assumido como sendo INT
```

17. É permitido parâmetro dinâmico como operando à esquerda de uma expressão IS, sendo assumido como booleano.

Uma vez que o tipo de dado do parâmetro dinâmico tenha sido determinado baseado na expressão onde se encontra, esta expressão pode estar em qualquer lugar onde normalmente seria permitida se não incluísse o parâmetro dinâmico. Por exemplo, acima foi dito que o parâmetro dinâmico não poderia ser utilizado como operando do - unário. Entretanto, pode aparecer em expressões que são operandos do menos unário, como:

```
- (1+?)
```

O parâmetro dinâmico é assumido como sendo do tipo INTEGER (porque o outro operando operador binário + é do tipo INT). Como se sabe seu tipo, é permitido como operando do - unário.

Palavras reservadas do SQL

Esta seção lista todas as palavras reservadas do Derby, incluindo as do padrão SQL-92. O Derby retorna erro quando se utiliza uma destas palavras chave como nome de identificador, a menos que o nome do identificador seja envolto por aspas ("). Consulte [Regras para identificadores SQL92](#).

ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BEGIN
BETWEEN
BIT
BOOLEAN
BOTH
BY
CALL
CASCADE
CASCADED
CASE
CAST
CHAR
CHARACTER
CHECK
CLOSE
COLLATE
COLLATION
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
CONVERT
CORRESPONDING
COUNT
CREATE
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DEALLOCATE
DEC
DECIMAL
DECLARE

DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DIAGNOSTICS
DISCONNECT
DISTINCT
DOUBLE
DROP
ELSE
END
ENDEXEC
ESCAPE
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXPLAIN
EXTERNAL
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FOUND
FROM
FULL
FUNCTION
GET
GET_CURRENT_CONNECTION
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INDICATOR
INITIALLY
INNER
INOUT
INPUT
INSENSITIVE
INSERT
INT
INTEGER
INTERSECT
INTO
IS
ISOLATION
JOIN
KEY
LAST

LEFT
LIKE
LONGINT
LOWER
LTRIM
MATCH
MAX
MIN
MINUTE
NAÇÃOAL
NATURAL
NCHAR
NVARCHAR
NEXT
NO
NOT
NULL
NULLIF
NUMERIC
OF
ON
ONLY
OPEN
OPTION
OR
ORDER
OUT
OUTER
OUTPUT
OVERLAPS
PAD
PARTIAL
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
READ
REAL
REFERENCES
RELATIVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
RTRIM
SCHEMA
SCROLL
SECOND
SELECT
SESSION_USER
SET
SMALLINT
SOME
SPACE
SQL

SQLCODE
SQLERROR
SQLSTATE
SUBSTR
SUBSTRING
SUM
SYSTEM_USER
TABLE
TEMPORARY
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TRAILING
TRANSACTION
TRANSLATE
TRANSLATION
TRUE
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USER
USING
VALUES
VARCHAR
VARYING
VIEW
WHenever
WHERE
WITH
WORK
WRITE
XML
XMLEXISTS
XMLPARSE
XMLSERIALIZE
YEAR

Suporte do Derby às funcionalidades do SQL-92

Existem quatro níveis de suporte ao SQL-92:

- SQL92E
Entrada (*entry*)
- SQL92T
Transicional (*transitional*), um nível definido pelo NIST em uma publicação chamada FIPS 127-2
- SQL92I
Intermediário (*intermediate*)
- SQL92F
Completo (*full*)

Tabela 9. Suporte às funcionalidades do SQL-92: Tipos básicos

Funcionalidade	Origem	Derby
SMALLINT	SQL92E	sim
INTEGER	SQL92E	sim
DECIMAL(p,s)	SQL92E	sim
NUMERIC(p,s)	SQL92E	sim
REAL	SQL92E	sim
FLOAT(p)	SQL92E	sim
DOUBLE PRECISION	SQL92E	sim
CHAR(n)	SQL92E	sim

Tabela 10. Suporte às funcionalidades do SQL-92: Operações matemáticas básicas

Funcionalidade	Origem	Derby
+, *, -, /, + unário, - unário	SQL92E	sim

Tabela 11. Suporte às funcionalidades do SQL-92: Comparações básicas

Funcionalidade	Origem	Derby
<, >, <=, >=, <>, =	SQL92E	sim

Tabela 12. Suporte às funcionalidades do SQL-92: Predicados básicos

Funcionalidade	Origem	Derby
BETWEEN, LIKE, NULL	SQL92E	sim

Tabela 13. Suporte às funcionalidades do SQL-92: Predicados quantificados

Funcionalidade	Origem	Derby
IN, ALL/SOME, EXISTS	SQL92E	sim

Tabela 14. Suporte às funcionalidades do SQL-92: Definição do esquema

Funcionalidade	Origem	Derby
tabelas	SQL92E	sim
visões	SQL92E	sim
privilégios	SQL92E	sim

Tabela 15. Suporte às funcionalidades do SQL-92: Atributos das colunas

Funcionalidade	Origem	Derby
valor padrão	SQL92E	sim
nulo	SQL92E	sim

Tabela 16. Suporte às funcionalidades do SQL-92: Restrições (não-postergáveis)

Funcionalidade	Origem	Derby
NOT NULL	SQL92E	sim (não armazenada em SYSCONSTRAINTS)
UNIQUE/PRIMARY KEY	SQL92E	sim
FOREIGN KEY	SQL92E	sim
CHECK	SQL92E	sim
Visão com WITH CHECK OPTION	SQL92E	não, uma vez que as visões não são atualizáveis

Tabela 17. Suporte às funcionalidades do SQL-92: Cursores

Funcionalidade	Origem	Derby
DECLARE, OPEN, FETCH, CLOSE	SQL92E	realizadas através do JDBC
UPDATE, DELETE CURRENT	SQL92E	sim

Tabela 18. Suporte às funcionalidades do SQL-92: SQL dinâmico 1

Funcionalidade	Origem	Derby
ALLOCATE / DEALLOCATE / GET / SET DESCRIPTOR	SQL92T	realizadas através do JDBC
PREPARE / EXECUTE / EXECUTE IMMEDIATE	SQL92T	realizadas através do JDBC
Cursor dinâmico DECLARE, OPEN, FETCH, CLOSE, UPDATE, DELETE	SQL92T	realizadas através do JDBC
Saída DESCRIBE	SQL92T	realizadas através do JDBC

Tabela 19. Suporte às funcionalidades do SQL-92: Esquema de informações básico

Funcionalidade	Origem	Derby
TABLES	SQL92T	SYS.SYSTABLES, SYS.SYSVIEWS, SYS.SYSCOLUMNS

Funcionalidade	Origem	Derby
VIEWS	SQL92T	<i>SYS.SYSTABLES, SYS.SYSVIEWS, SYS.SYSCOLUMNS</i>
COLUMNS	SQL92T	<i>SYS.SYSTABLES, SYS.SYSVIEWS, SYS.SYSCOLUMNS</i>

Tabela 20. Suporte às funcionalidades do SQL-92: Manipulação básica de esquema

Funcionalidade	Origem	Derby
CREATE / DROP TABLE	SQL92T	sim
CREATE / DROP VIEW	SQL92T	sim
GRANT / REVOKE	SQL92T	não
ALTER TABLE ADD COLUMN	SQL92T	sim
ALTER TABLE DROP COLUMN	SQL92T	não

Tabela 21. Suporte às funcionalidades do SQL-92: Tabela juntada

Funcionalidade	Origem	Derby
INNER JOIN	SQL92T	sim
junção natural	SQL92T	não
LEFT, RIGHT OUTER JOIN	SQL92T	sim
condição de junção	SQL92T	sim
junção de colunas nomeada	SQL92T	sim

Tabela 22. Suporte às funcionalidades do SQL-92: Tabela juntada

Funcionalidade	Origem	Derby
DATE, TIME, TIMESTAMP, INTERVAL simples	SQL92T	sim, menos INTERVAL
constantes data/hora	SQL92T	sim
matemática data/hora	SQL92T	pode ser feito com métodos Java
comparações de data/hora	SQL92T	sim
predicados: OVERLAPS	SQL92T	pode ser feito com métodos Java

Tabela 23. Suporte às funcionalidades do SQL-92: VARCHAR

Funcionalidade	Origem	Derby
LENGTH	SQL92T	sim
concatenação ()	SQL92T	sim

Tabela 24. Suporte às funcionalidades do SQL-92: Isolamento de transação

Funcionalidade	Origem	Derby
READ WRITE / READ ONLY	SQL92T	através do JDBC, propriedades do banco de dado e meio de

Funcionalidade	Origem	Derby
		armazenamento.
RU, RC, RR, SER	SQL92T	sim

Tabela 25. Suporte às funcionalidades do SQL-92: Vários esquemas por usuário

Funcionalidade	Origem	Derby
Visão SCHEMATA	SQL92T	<i>SYS.SYSSCHEMAS</i>

Tabela 26. Suporte às funcionalidades do SQL-92: Privilégios em tabelas

Funcionalidade	Origem	Derby
TABLE_PRIVILEGES	SQL92T	não
COLUMNS_PRIVILEGES	SQL92T	não
USAGE_PRIVILEGES	SQL92T	não

Tabela 27. Suporte às funcionalidades do SQL-92: Operações de tabela

Funcionalidade	Origem	Derby
Relaxamento de UNION	SQL92I	sim
EXCEPT	SQL92I	sim
INTERSECT	SQL92I	sim
CORRESPONDING	SQL92I	não

Tabela 28. Suporte às funcionalidades do SQL-92: Instrução de definição de esquema

Funcionalidade	Origem	Derby
CREATE SCHEMA	SQL92I	sim, parcialmente

Tabela 29. Suporte às funcionalidades do SQL-92: Autorização de usuário

Funcionalidade	Origem	Derby
SET SESSION AUTHORIZATION	SQL92I	usado SET SCHEMA
CURRENT_USER	SQL92I	sim
SESSION_USER	SQL92I	sim
SYSTEM_USER	SQL92I	não

Tabela 30. Suporte às funcionalidades do SQL-92: Restrições de tabela

Funcionalidade	Origem	Derby
TABLE CONSTRAINTS	SQL92I	<i>SYS.SYSCONSTRAINTS</i>
REFERENTIAL CONSTRAINTS	SQL92I	<i>SYS.SYSFOREIGNKEYS</i>
CHECK CONSTRAINTS	SQL92I	<i>SYS.SYSCHECKS</i>

Tabela 31. Suporte às funcionalidades do SQL-92: Esquema de documentação

Funcionalidade	Origem	Derby
SQL_FEATURES	SQL92I/FIPS 127-2	usado JDBC <i>DatabaseMetaData</i>
SQL_SIZING	SQL92I/FIPS 127-2	usado JDBC <i>DatabaseMetaData</i>

Tabela 32. Suporte às funcionalidades do SQL-92: DATETIME completo

Funcionalidade	Origem	Derby
precisão para TIME e TIMESTAMP	SQL92F	sim

Tabela 33. Suporte às funcionalidades do SQL-92: Funções de caractere completa

Funcionalidade	Origem	Derby
Expressão POSITION	SQL92F	usado métodos Java ou LOCATE
Funções UPPER/LOWER	SQL92F	sim

Tabela 34. Suporte às funcionalidades do SQL-92: Diversas

Funcionalidade	Origem	Derby
Identificadores delimitados	SQL92E	sim
Subconsultas correlacionadas	SQL92E	sim
Instruções de inserção, atualização e exclusão	SQL92E	sim
Junções	SQL92E	sim
Qualificações no WHERE	SQL92E	sim
GROUP BY	SQL92E	sim
HAVING	SQL92E	sim
Funções de agregação	SQL92E	sim
ORDER BY	SQL92E	sim
Expressões de seleção	SQL92E	sim
SELECT *	SQL92E	sim
SQLCODE	SQL92E	não, em obsolescência no SQL-92
SQLSTATE	SQL92E	sim
UNION, INTERSECT e EXCEPT em visões	SQL92T	sim
Conversão numérica implícita	SQL92T	sim
Conversão de caractere implícita	SQL92T	sim
Obter diagnóstico	SQL92T	usado JDBC <i>SQLException</i>
Operações agrupadas	SQL92T	sim
* qualificado na lista de seleção	SQL92T	sim
Identificadores em minúsculas	SQL92T	sim
PRIMARY KEY com nulo	SQL92T	não
Suporte a vários módulos	SQL92T	não (não é requerido e não faz

Funcionalidade	Origem	Derby
		parte do JDBC)
Ações de exclusão referencial	SQL92T	CASCADE, SET NULL, RESTRICT e NO ACTION.
Funções CAST	SQL92T	sim
Expressões INSERT	SQL92T	sim
Padrões explícitos	SQL92T	sim
Relaxamento de palavra chave	SQL92T	sim
Definição de domínio	SQL92I	não
Expressão CASE	SQL92I	suporte parcial
Constantes cadeia de caracteres compostas	SQL92I	usado concatenação
Melhorias no LIKE	SQL92I	sim
Predicado UNIQUE	SQL92I	não
Utilização de tabelas	SQL92I	<i>SYS.SYSDEPENDS</i>
Esquema de informação intermediário	SQL92I	usado JDBC <i>DatabaseMetaData</i> e tabelas do sistema do Derby
Suporte a subprograma	SQL92I	não é relevante para o JDBC, que é muito mais rico
Sinalização SQL intermediária	SQL92I	não
Manipulação de esquema	SQL92I	sim
Identificadores longos	SQL92I	sim
Junção externa completa	SQL92I	não
Especificação de zona horária	SQL92I	não
Cursors rolados	SQL92I	parcial (rolagem não sensível ao conjunto de resultados através do JDBC 2.0)
Suporte a funções de conjunto intermediário	SQL92I	parcial
Definição de conjunto de caracteres	SQL92I	suporta idiomas do Java
Conjuntos de caracteres nomeados	SQL92I	suporta idiomas do Java
Valores de subconsulta escalar	SQL92I	sim
Predicado nulo expandido	SQL92I	sim
Gerenciamento de restrições	SQL92I	sim (ADD/DROP CONSTRAINT)
Tipos FOR BIT DATA	SQL92F	sim
Restrições de asserção	SQL92F	não
Tabelas temporárias	SQL92F	somente sintaxe específica da IBM
SQL dinâmico completo	SQL92F	não
Expressões de valor completa	SQL92F	sim
Testes de valor verdade	SQL92F	sim
tabelas derivadas no FROM	SQL92F	sim

Funcionalidade	Origem	Derby
Sublinhado no final	SQL92F	sim
Tipos de dado indicador	SQL92F	não é relevante para o JDBC
Ordem de nome referencial	SQL92F	não
Sinalização SQL completa	SQL92F	não
Construtores de linha e de tabela	SQL92F	sim
Qualificadores de nome de catálogo	SQL92F	não
Tabelas simples	SQL92F	não
Subconsultas no CHECK	SQL92F	não, mas pode ser feito com métodos Java
Junção de união	SQL92F	não
Agrupamento e tradução	SQL92F	Suportados os idiomas do Java
Ações de atualização referenciais	SQL92F	RESTRICT e NO ACTION. As demais podem ser feitas com gatilhos.
ALTER domínio	SQL92F	não
INSERT privilégios em colunas	SQL92F	não
Tipos MATCH referenciais	SQL92F	não
Melhorias de CHECK em visões	SQL92F	não, as visões não são atualizáveis
Gerenciamento de sessão	SQL92F	usado JDBC
Gerenciamento de conexão	SQL92F	usado JDBC
Operações de auto-referenciamento	SQL92F	sim
Cursors não sensíveis	SQL92F	Sim, através do JDBC 2.0
Função de conjunto completa	SQL92F	parcialmente
Sinalização de catálogo	SQL92F	não
Referências a tabela local	SQL92F	não
Atualização de cursor completa	SQL92F	não

Tabelas do sistema Derby

O Derby inclui tabelas do sistema.

As tabelas do sistema podem ser consultadas, mas não podem ser alteradas.

Todas as tabelas do sistema residem no esquema SYS. Como este não é o esquema padrão, todas as consultas que acessam tabelas do sistema devem ser qualificadas pelo nome de esquema SYS.

A forma recomendada para obter mais informações sobre estas tabelas é utilizar uma instância da interface Java *java.sql.DatabaseMetaData*.

SYSALIASES

Descreve os procedimentos e funções no banco de dados.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
ALIASID	CHAR	36	falso	identificador único para o aliás
ALIAS	VARCHAR	128	falso	aliás
SCHEMAID	CHAR	36	verdade	reservado para uso futuro
JAVACLASSNAME	LONGVARCHAR	255	falso	o nome da classe Java
ALIASTYPE	CHAR	1	falso	'F' (função), 'P' (procedimento)
NAMESPACE	CHAR	1	falso	'F' (função), 'P' (procedimento)
SYSTEMALIAS	BOOLEAN	'	falso	<i>verdade</i> (fornecido pelo sistema ou aliás nativo) <i>falso</i> (aliás criado pelo usuário)
ALIASINFO	org.apache.derby.catalog.AliasInfo: Esta classe não faz parte da API pública	'	verdade	interface Java que encapsula informações adicionais específicas de um aliás
SPECIFICNAME	VARCHAR	128	falso	identificador gerado pelo sistema

SYSCHECKS

Descreve as restrições de verificação no banco de dados corrente.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
CONSTRAINTID	CHAR	36	falso	identificador único para a

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
				restrição
CHECKDEFINITION	LONG VARCHAR	'	falso	texto da definição da restrição de verificação
REFERENCEDCOLUMNS	org.apache.derby.catalog.ReferencedColumns: Esta classe não faz parte da API pública.	'	falso	descrição das colunas referenciadas pela restrição de verificação

SYSCOLUMNS

Descreve as colunas de todas as tabelas do banco de dados corrente:

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
REFERENCEID	CHAR	36	falso	identificador da tabela (junção com <i>SYSTABLES.TABLEID</i>)
COLUMNNAME	CHAR	128	falso	nome da coluna ou do parâmetro
COLUMNNUMBER	INT	4	falso	posição da coluna na tabela
COLUMNDATATYPE	org.apache.derby.catalog.TypeDescriptor Esta classe não faz parte da API pública.	'	falso	tipo do sistema que descreve a precisão, comprimento, escala, se pode ser nulo, nome do tipo, e tipo de armazenamento do dado
COLUMNDEFAULT	<i>java.io.Serializable</i>	'	verdade	para as tabelas descreve o valor padrão da coluna. O método <i>toString()</i> do objeto armazenado na tabela retorna o texto do valor padrão conforme especificado na instrução CREATE TABLE ou ALTER TABLE.
COLUMNDEFAULTID	CHAR	36	verdade	identificador único para o valor padrão
AUTOINCREMENT COLUMNVALUE	BIGINT	'	verdade	qual será o próximo valor para a coluna, se a coluna for uma coluna identidade
AUTOINCREMENT COLUMNSTART	BIGINT	'	verdade	valor inicial da coluna (se especificado), se a coluna for uma coluna de identidade
AUTOINCREMENT COLUMNINC	BIGINT	'	verdade	quantidade incrementada automaticamente no valor da coluna, se a coluna for

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
				uma coluna identidade

SYSCONGLOMERATES

Descreve os conglomerados do banco de dados corrente. O conglomerado é uma unidade de armazenamento, sendo uma tabela ou um índice.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
SCHEMAID	CHAR	36	falso	ID de esquema para o conglomerado
TABLEID	CHAR	36	falso	identificador da tabela (junção com <i>SYSTABLES.TABLEID</i>)
CONGLOMERATENUMBER	BIGINT	8	falso	ID de conglomerado para o conglomerado (<i>heap</i> ou índice)
CONGLOMERATENAME	VARCHAR	128	verdade	nome do índice, se o conglomerado for um índice, senão o ID da tabela
ISINDEX	BOOLEAN	1	falso	indica se o conglomerado é um índice, ou não
DESCRIPTOR	org.apache.derby.catalog.IndexDescriptor Esta classe não faz parte da API pública.	'	verdade	tipo do sistema descrevendo o índice
ISCONSTRAINT	BOOLEAN	1	verdade	indica se o conglomerado é um índice gerado pelo sistema para impor uma restrição, ou não
CONGLOMERATEID	CHAR	36	falso	identificador único para o conglomerado

SYSCONSTRAINTS

Descreve as informações comuns a todos os tipos de restrição no banco de dados corrente (no momento inclui restrições de chave primária, unicidade, chave estrangeira e verificação).

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
CONSTRAINTID	CHAR	36	falso	identificador único para a restrição
TABLEID	CHAR	36	falso	identificador para a tabela (junção com <i>SYSTABLES.TABLEID</i>)

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
CONSTRAINTNAME	VARCHAR	128	falso	nome da restrição (gerado internamente se não for especificado pelo usuário)
TYPE	CHAR	1	falso	<i>P</i> (chave primária), <i>U</i> (unicidade), <i>C</i> (verificação) ou <i>F</i> (chave estrangeira)
SCHEMAID	CHAR	36	falso	identificador do esquema ao qual a restrição pertence (junção com <i>SYSSCHEMAS.SCHEMAID</i>)
STATE	CHAR	1	falso	<i>E</i> para habilitado (enabled), <i>D</i> para desabilitado (disabled)
REFERENCECOUNT	INTEGER	1	falso	o contador do número de restrições de chave estrangeira que fazem referência a esta restrição; este número só pode ser maior que zero para as restrições PRIMARY KEY e UNIQUE

SYSDEPENDS

Descreve os relacionamentos de dependência entre objetos persistentes no banco de dados. Os objetos persistentes podem ser dependentes (dependem de outros objetos) e/ou fornecedores (outros objetos dependem destes).

Os fornecedores são tabelas, conglomerados e restrições. Os dependentes são visões.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
DEPENDENTID	CHAR	36	falso	identificador único para o dependente
DEPENDENTFINDER	org.apache.derby.catalog.DependableFinder: Esta classe não faz parte da API pública.	1	falso	tipo do sistema descrevendo a visão
PROVIDERID	CHAR	36	falso	identificador único para o fornecedor
PROVIDERFINDER	org.apache.derby.catalog.DependableFinder Esta classe não faz parte da API pública.	1	falso	tipo do sistema descrevendo as tabelas, conglomerados e restrições que são fornecedores

SYSFILES

Descreve os arquivos *jar* armazenados no banco de dados.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
FILEID	CHAR	36	falso	identificador único para o arquivo <i>jar</i>
SCHEMAID	CHAR	36	falso	ID do esquema do arquivo <i>jar</i> (junção com <i>SYSSCHEMAS.SCHEMAID</i>)
FILENAME	VARCHAR	128	falso	nome SQL do arquivo <i>jar</i>
GENERATIONID	BIGINT		falso	Número de geração do arquivo. Quando os arquivos <i>jar</i> são substituídos, seus identificadores de geração são alterados.

SYSFOREIGNKEYS

Descreve as informações específicas das restrições de chave estrangeira no banco de dados corrente.

O Derby gera um índice de apoio para cada restrição de chave estrangeira; o nome deste índice é o mesmo de SYSFOREIGNKEYS.CONGLOMERATEID.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
CONSTRAINTID	CHAR	36	falso	identificador único para a restrição de chave estrangeira (junção com <i>SYSCONSTRAINTS.CONSTRAINTID</i>)
CONGLOMERATEID	CHAR	36	falso	identificador único para o índice de apoio da restrição de chave estrangeira (junção com <i>SYSCONGLOMERATES.CONGLOMERATEID</i>)
KEYCONSTRAINTID	CHAR	36	falso	identificador único para a restrição de chave primária ou de unicidade referenciada por esta chave estrangeira (<i>SYSKEYS.CONSTRAINTID</i> ou <i>SYSCONSTRAINTS.CONSTRAINTID</i>)
DELETERULE	CHAR	1	falso	<i>R</i> para NO ACTION (padrão), <i>S</i> para RESTRICT, <i>C</i> para CASCADE, <i>U</i> para SET NULL
UPDATERULE	CHAR	1	falso	<i>R</i> para NO ACTION (padrão), <i>S</i> para RESTRICT

SYSKEYS

Descreve as informações específicas para as restrições de chave primária e de

unicidade no banco de dados corrente. O Derby gera um índice na tabela para apoiar cada restrição destes tipos. O nome do índice é o mesmo de *SYSKEYS.CONGLOMERATEID*.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
CONSTRAINTID	CHAR	36	falso	identificador único para a restrição
CONGLOMERATEID	CHAR	36	falso	identificador único para o índice de apoio

SYSSCHEMAS

Descreve os esquemas no banco de dados corrente.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
SCHEMAID	CHAR	36	falso	identificador único para o esquema
SCHEMANAME	VARCHAR	128	falso	nome do esquema
AUTHORIZATIONID	VARCHAR	128	falso	o identificador de autorização do dono do esquema

SYSSTATISTICS

Descreve as estatísticas no banco de dados corrente.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
STATID	CHAR	36	falso	identificador único para a estatística
REFERENCEID	CHAR	36	falso	o conglomerado para o qual a estatística foi criada (junção com SYSCONGLOMERATES.CONGLOMERATEID)
TABLEID	CHAR	36	falso	a tabela para a qual a informação foi coletada
CREATIONTIMESTAMP	TIMESTAMP	'	falso	momento em que a estatística foi criada ou atualizada
TYPE	CHAR	1	falso	tipo da estatística
VALID	BOOLEAN	'	falso	se a estatística ainda é válida
COLCOUNT	INTEGER	'	falso	número de colunas na estatística
STATISTICS	org.apache.derby. Statistics: Esta classe não faz parte da API pública.	'	verdade	informação estatística

SYSSTATEMENTS

Contém uma linha por instrução preparada.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
STMTID	CHAR	36	falso	identificador único para a instrução
STMTNAME	VARCHAR	128	falso	nome da instrução
SCHEMAID	CHAR	36	falso	o esquema onde a instrução reside
TYPE	CHAR	1	falso	sempre 'S'
VALID	BOOLEAN	'	falso	TRUE se válida, FALSE se inválida
TEXT	LONG VARCHAR	'	falso	texto da instrução
LASTCOMPILED	TIMESTAMP	'	verdade	momento em que a instrução foi compilada
COMPILATION SCHEMAID	CHAR	36	falso	ID do esquema contendo a instrução
USINGTEXT	LONG VARCHAR	'	verdade	texto da cláusula USING da instrução CREATE STATEMENT e ALTER STATEMENT

SYSTABLES

Descreve as tabelas e visões no banco de dados corrente.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
TABLEID	CHAR	36	falso	identificador único para a tabela ou visão
TABLENAME	VARCHAR	128	falso	nome da tabela ou da visão
TABLETYPE	CHAR	1	falso	'S' (tabela do sistema), 'T' (tabela do usuário), ou 'V' (visão)
SCHEMAID	CHAR	36	falso	ID do esquema da tabela ou da visão
LOCK GRANULARITY	CHAR	1	falso	Indica a granularidade do bloqueio da tabela 'T' (bloqueio no nível de tabela) 'R' (bloqueio no nível de linha, o padrão)

SYSTRIGGERS

Descreve os gatilhos do banco de dados.

Nome da coluna	Tipo	Compr	Aceita nulo	Conteúdo
TRIGGERID	CHAR	36	falso	identificador único para o gatilho
TRIGGERNAME	VARCHAR	128	falso	nome do gatilho
SCHEMAID	CHAR	36	falso	ID do esquema do gatilho (junção com SYSSCHEMAS.SCHEMAID)
CREATIONTIMESTAMP	TIMESTAMP	'	falso	momento em que o gatilho foi criado
EVENT	CHAR	1	falso	'U' para atualização (update), 'D' para exclusão (delete), 'I' para inserção (insert)
FIRINGTIME	CHAR	1	falso	'B' para antes (before) e 'A' para após (after)
TYPE	CHAR	1	falso	'R' para linha (row), 'S' para instrução (statement)
STATE	CHAR	1	falso	'E' para habilitado (enabled), 'D' para desabilitado (disabled)
TABLEID	CHAR	36	falso	ID da tabela para a qual o gatilho foi definido
WHENSTMTID	CHAR	36	verdade	usado somente quando existe a cláusula WHEN (ainda não suportado)
ACTIONSTMTID	CHAR	36	verdade	ID da instrução preparada armazenada para a instrução-SQL-engatilhada (junção com SYSSTATEMENTS.STMTID)
REFERENCEDCOLUMNS	org.apache.derby.catalog.ReferencedColumns: Esta classe não faz parte da API pública.	'	verdade	descriptor das colunas referenciadas pelos gatilhos de UPDATE
TRIGGERDEFINITION	LONG VARCHAR	'	verdade	texto da instrução SQL de ação
REFERENCINGOLD	BOOLEAN	'	verdade	indica se OLDREFERENCINGNAME, se não for nulo, faz referência à linha ou tabela OLD, ou não
REFERENCINGNEW	BOOLEAN	'	verdade	indica se NEWREFERENCINGNAME, se não for nulo, faz referência à linha ou tabela NEW, ou não
OLDREFERENCINGNAME	VARCHAR	128	verdade	pseudonimo definido utilizando a cláusula REFERENCING OLD AS
NEWREFERENCINGNAME	VARCHAR	128	verdade	pseudonimo definido utilizando a cláusula

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
				REFERENCING NEW AS

Todo texto SQL que faz parte da instrução-SQL-engatilhada é compilado e armazenado em *SYSSTATEMENTS*. *ACTIONSTMTID* e *WHENSTMTID* são chaves estrangeiras que fazem referência a *SYSSTATEMENTS.STMTID*. As instruções para o gatilho estão sempre no mesmo esquema do gatilho.

SYSVIEWS

Descreve as definições das visões no banco de dados corrente.

Nome da coluna	Tipo	Comprimento	Aceita nulo	Conteúdo
TABLEID	CHAR	36	falso	identificador único para a visão (chamada TABLEID uma vez que é feita a junção com uma coluna com este nome em SYSTABLES)
VIEWDEFINITION	LONG VARCHAR		falso	texto da definição da visão
CHECKOPTION	CHAR	1	falso	'N' (opção de verificação, ainda não suportada)
COMPILATION SCHEMAID	CHAR	36	falso	ID do esquema que contém a visão

Mensagens de exceção e estados SQL do Derby

O *driver* de JDBC retorna *SQLException* para todos os erros do Derby. Se a exceção for originária de um tipo do usuário, mas não for em si mesma uma *SQLException*, a exceção será envolta por uma *SQLException*. As *SQLException* específicas do Derby utilizam códigos de classe *SQLState* começados por X. É retornado o valor padrão de *SQLState* para a exceção quando apropriado.

Aspectos não implementados do *driver* de JDBC retornam uma *SQLException* com uma mensagem declarando "Funcionalidade não implementada", e um *SQLState* igual a XJZZZ. Estas partes não implementadas são de funcionalidades não suportadas pelo Derby.

O Derby fornece valores para a mensagem e para os campos *SQLState*. Além disso, algumas vezes o Derby retorna várias *SQLException* utilizando o encadeamento *nextException*. A primeira exceção é sempre a de maior severidade, com as exceções do padrão SQL-92 precedendo as exceções específicas do Derby.

Para obter informações sobre como processar *SQLException* deve ser consultado o *Guia do Desenvolvedor do Derby*.

Referência de SQLState e mensagem de erro

As tabelas a seguir listam os *SQLState* para as exceções. As exceções que começam por X são específicas do Derby. Deve ser observado que alguns *SQLState* específicos do cliente da rede poderão mudar nas versões futuras.

Tabela 35. Código de classe 01: Advertência

SQLSTATE	Texto da mensagem
01003	Null values were eliminated from the argument of a column function.
0100E	XX Attempt to return too many result sets.
01500	The constraint <constraintName> on table <tableName> has been dropped.
01501	The view <viewName> has been dropped.
01502	The trigger <triggerName> on table <tableName> has been dropped.
01503	The column <columnName> on table <tableName> has been modified by adding a not null constraint.
01504	The new index is a duplicate of an existing index: <index>.
01505	The value <valueName> may be truncated.
01522	The newly defined synonym '<synonymName>' resolved to the object '<objectName>' which is currently undefined.
01J01	Database '<databaseName>' not created, connection made to existing database instead.
01J02	Scroll sensitive cursors are not currently implemented.
01J03	Scroll sensitive and scroll insensitive updatable ResultSets are not currently implemented.
01J04	The class '<className>' for column '<columnName>' does not implement java.io.Serializable or java.sql.SQLData. Instances must implement one of these interfaces to allow them to be stored.

SQLSTATE	Texto da mensagem
01J05	Database upgrade succeeded. The upgraded database is now ready for use. Revalidating stored prepared statements failed. See next exception for details about failure.
01J06	ResultSet not updatable. Query does not qualify to generate an updatable ResultSet.

Tabela 36. Código de classe 04: Autenticação no banco de dados

SQLSTATE	Texto da mensagem
04501	Database connection refused.

Tabela 37. Código de classe 07: Erro de SQL dinâmico

SQLSTATE	Texto da mensagem
07000	At least one parameter to the current statement is uninitialized.
07004	Parameter <i><parameterName></i> is a <i><procedureName></i> procedure parameter and must be registered with CallableStatement.registerOutParameter before execution.
07009	No input parameters.

Tabela 38. Código de classe 08: Exceção de conexão

SQLSTATE	Texto da mensagem
08000	Connection closed by unknown interrupt.
08003	No current connection.
08004	Connection refused : <i><connectionName></i>
08006	Database ' <i><databaseName></i> '

Tabela 39. Código de classe 0A: Funcionalidade não suportada

SQLSTATE	Texto da mensagem
0A000	Feature not implemented: <i><featureName></i> .

Tabela 40. Código de classe 21: Violação de cardinalidade

SQLSTATE	Texto da mensagem
21000	Scalar subquery is only allowed to return a single row.

Tabela 41. Código de classe 22: Exceção de dado

SQLSTATE	Texto da mensagem
2200L	XMLPARSE operand is not an XML document; see next exception for details.
22001	A truncation error was encountered trying to shrink <i><value></i> ' <i><value></i> ' to length <i><value></i>
22003	The resulting value is outside the range for the data type <i><datatypeName></i> .
22005	An attempt was made to get a data value of type ' <i><typeName></i> ' from a data value of type ' <i><typeName></i> '.
22007	The string representation of a datetime value is out of range.
22007	The syntax of the string representation of a datetime value is incorrect.

SQLSTATE	Texto da mensagem
22008	'<argument>' is an invalid argument to the <functionName> function.
22011	The second or third argument of the SUBSTR function is out of range.
22012	Attempt to divide by zero.
22013	Attempt to take the square root of a negative number, '<number>'.
22014	The start position for LOCATE is invalid; it must be a positive integer. The index to start the search from is '<index>'. The string to search for is '<index>'. The string to search from is '<index>'.
22015	The '<functionName>' function is not allowed on the following set of types. First operand is of type '<typeName>'. Second operand is of type '<typeName>'. Third operand (start position) is of type '<typeName>'.
22018	Invalid character string format for type <typeName>.
22019	Invalid escape sequence, '<sequenceName>'. The escape string must be exactly one character. It cannot be a null or more than one character.
22025	Escape character must be followed by escape character, _, or %. It cannot be followed by any other character or be at the end of the pattern.
22027	The built-in TRIM() function only supports a single trim character. The LTRIM() and RTRIM() built-in functions support multiple trim characters.
22501	An ESCAPE clause of NULL returns undefined results and is not allowed.

Tabela 42. Código de classe 23: Violação de restrição

SQLSTATE	Texto da mensagem
23502	Column '<columnName>' cannot accept a NULL value.
23503	<value> on table '<tableName>' caused a violation of foreign key constraint '<constraintName>' for key <keyName>. The statement has been rolled back.
23505	The statement was aborted because it would have caused a duplicate key value in a unique or primary key constraint or unique index identified by '<value>' defined on '<value>'.
23513	The check constraint '<constraintName>' was violated while performing an INSERT or UPDATE on table '<tableName>'.

Tabela 43. Código de classe 24: Estado do cursor inválido

SQLSTATE	Texto da mensagem
24000	Invalid cursor state - no current row.

Tabela 44. Código de classe 25: Estado da transação inválido

SQLSTATE	Texto da mensagem
25000	Invalid transaction state.
25501	Unable to set the connection read-only property in an active transaction.
25502	An SQL data change is not permitted for a read-only connection, user or database.
25503	DDL is not permitted for a read-only connection, user or database.
25505	A read-only user or a user in a read-only database is not permitted to disable read-only mode on a connection.

Tabela 45. Código de classe 28: Especificação de autorização inválida

SQLSTATE	Texto da mensagem
28501	Invalid database authorization property '<value>=<value>'.
28502	The user name '<userName>' is not valid.
28503	User(s) '<userName>' must not be in both read-only and full-access authorization lists.
28504	Repeated user(s) '<userName>' in access list '<listName>'.

Tabela 46. Código de classe 38: Exceção de função externa

SQLSTATE	Texto da mensagem
38000	The exception '<exception>' was thrown while evaluating an expression.
38001	The external routine is not allowed to execute SQL statements.
38002	The routine attempted to modify data, but the routine was not defined as MODIFIES SQL DATA.
38004	The routine attempted to read data, but the routine was not defined as READS SQL DATA.

Tabela 47. Código de classe 39: Exceção de função externa

SQLSTATE	Texto da mensagem
39004	A NULL value cannot be passed to a method which takes a parameter of primitive type '<type>'.

Tabela 48. Código de classe 3B: SAVEPOINT inválido

SQLSTATE	Texto da mensagem
3B001	SAVEPOINT, <savepoint> does not exist or is not active in the current transaction.
3B002	The maximum number of savepoints has been reached.
3B501	A SAVEPOINT with the passed name already exists in the current transaction.
3B502	A RELEASE or ROLLBACK TO SAVEPOINT was specified, but the savepoint does not exist.

Tabela 49. Código de classe 40: Transação desfeita

SQLSTATE	Texto da mensagem
40001	A lock could not be obtained due to a deadlock, cycle of locks and waiters is: <value>. The selected victim is XID : <value>.
40XC0	Dead statement. This may be caused by catching a transaction severity error inside this statement.
40XD0	Container has been closed.
40XD1	Container was opened in read-only mode.
40XD2	Container <containerName> cannot be opened; it either has been dropped or does not exist.
40XL1	A lock could not be obtained within the time requested.
40XL2	A lock could not be obtained within the time requested. The lockTable dump is: <tableDump>.
40XT0	An internal error was identified by RawStore module.

SQLSTATE	Texto da mensagem
40XT1	An exception was thrown during transaction commit.
40XT2	An exception was thrown during rollback of a SAVEPOINT.
40XT4	An attempt was made to close a transaction that was still active. The transaction has been aborted.
40XT5	Exception thrown during an internal transaction.
40XT6	Database is in quiescent state, cannot activate transaction. Please wait for a moment until it exits the inactive state.
40XT7	Operation is not supported in an internal transaction.

Tabela 50. Código de classe 42: Erro de sintaxe ou violação de regra de acesso

SQLSTATE	Texto da mensagem
42000	Syntax error or access rule violation; see additional errors for details.
42601	ALTER TABLE statement cannot add an IDENTITY column to a table.
42601	In an ALTER TABLE statement, the column '<columnName>' has been specified as NOT NULL and either the DEFAULT clause was not specified or was specified as DEFAULT NULL.
42605	The number of arguments for function '<functionName>' is incorrect.
42606	An invalid hexadecimal constant starting with '<number>' has been detected.
42610	All the arguments to the COALESCE/VALUE function cannot be parameters. The function needs at least one argument that is not a parameter.
42611	The length, precision, or scale attribute for column, or type mapping '<value>' is not valid.
42613	Multiple or conflicting keywords involving the '<clause>' clause are present.
42621	A check constraint or generated column that is defined with '<value>' is invalid.
42622	The name '<name>' is too long. The maximum length is '<maxLength>'.
42734	Name '<name>' specified in context '<context>' is not unique.
42802	The number of values assigned is not the same as the number of specified or implied columns.
42803	An expression containing the column '<columnName>' appears in the SELECT list and is not part of a GROUP BY clause.
42815	The replacement value for '<value>' is invalid.
42815	The data type, length or value of arguments '<firstArgument>' and '<secondArgument>' is incompatible.
42818	Comparisons between '<value>' and '<value>' are not supported.
42820	The floating point literal '<string>' contains more than 30 characters.
42821	Columns of type '<type>' cannot hold values of type '<type>'.
42824	An operand of LIKE is not a string, or the first operand is not a column.
42831	'<columnName>' cannot be a column of a primary key or unique key because it can contain null values.
42834	SET NULL cannot be specified because FOREIGN KEY '<key>' cannot contain null values.
42837	ALTER TABLE '<tableName>' specified attributes for column '<columnName>' that are not compatible with the existing column.
42846	Cannot convert types '<type>' to '<type>'.

SQLSTATE	Texto da mensagem
42877	A qualified column name '<columnName>' is not allowed in the ORDER BY clause.
42884	No authorized routine named '<routineName>' of type '<type>' having compatible arguments was found.
42886	'<value>' parameter '<value>' requires a parameter marker '<parameter>'.
42894	DEFAULT value or IDENTITY attribute value is not valid for column '<columnName>'.
428C1	Only one identity column is allowed in a table.
428EK	The qualifier for a declared global temporary table name must be SESSION.
42903	Invalid use of an aggregate function.
42908	The CREATE VIEW statement does not include a column list.
42915	Foreign Key '<key>' is invalid because '<value>'.
42916	Synonym '<synonym2>' cannot be created for '<synonym1>' as it would result in a circular synonym chain.
42939	An object cannot be created with the schema name '<nome-do-esquema>'.
42962	Long column type column or parameter '<columnName>' not permitted in declared global temporary tables or procedure definitions.
42972	An ON clause associated with a JOIN operator is not valid.
42995	The requested function does not apply to global temporary tables.
42X01	Syntax error: <error>.
42X02	<value>.
42X03	Column name '<columnName>' is in more than one table in the FROM list.
42X04	Column '<columnName>' is either not in any table in the FROM list or appears within a join specification and is outside the scope of the join specification or appears in a HAVING clause and is not in the GROUP BY list. If this is a CREATE or ALTER TABLE statement then '<columnName>' is not a column in the target table.
42X05	Table '<tableName>' does not exist.
42X06	Too many result columns specified for table '<tableName>'.
42X07	Null is only allowed in a VALUES clause within an INSERT statement.
42X08	The constructor for class '<className>' cannot be used as an external virtual table because the class does not implement '<constructorName>'.
42X09	The table or alias name '<tableName>' is used more than once in the FROM list.
42X10	'<tableName>' is not an exposed table name in the scope in which it appears.
42X12	Column name '<tableName>' appears more than once in the CREATE TABLE statement.
42X13	Column name '<columnName>' appears more than once in the column list of an INSERT statement.
42X14	'<columnName>' is not a column in table or VTI '<value>'.
42X15	Column name '<columnName>' appears in a statement without a FROM list.
42X16	Column name '<columnName>' appears multiple times in the SET clause of an UPDATE statement.
42X17	In the Properties list of a FROM clause, the value '<value>' is not valid as a joinOrder specification. Only the values FIXED and UNFIXED are valid.
42X19	The WHERE or HAVING clause or CHECK CONSTRAINT definition is a '<value>' expression. It must be a BOOLEAN expression.
42X23	Cursor <cursorName> is not updatable.

SQLSTATE	Texto da mensagem
42X25	The '<functionName>' function is not allowed on the '<type>' type.
42X26	The class '<className>' for column '<columnName>' does not exist or is inaccessible. This can happen if the class is not public.
42X28	Delete table '<tableName>' is not target of cursor '<cursorName>'.
42X29	Update table '<tableName>' is not the target of cursor '<cursorName>'.
42X30	Cursor '<cursorName>' not found. Verify that autocommit is OFF.
42X31	Column '<columnName>' is not in the FOR UPDATE list of cursor '<cursorName>'.
42X32	The number of columns in the derived column list must match the number of columns in table '<tableName>'.
42X33	The derived column list contains a duplicate column name '<columnName>'.
42X34	There is a ? parameter in the select list. This is not allowed.
42X35	It is not allowed for both operands of '<value>' to be ? parameters.
42X36	The '<operator>' operator is not allowed to take a ? parameter as an operand.
42X37	The unary '<operator>' operator is not allowed on the '<type>' type.
42X38	'SELECT *' only allowed in EXISTS and NOT EXISTS subqueries.
42X39	Subquery is only allowed to return a single column.
42X40	A NOT statement has an operand that is not boolean . The operand of NOT must evaluate to TRUE, FALSE, or UNKNOWN.
42X41	In the Properties clause of a FROM list, the property '<propertyName>' is not valid (the property was being set to '<value>').
42X42	Correlation name not allowed for column '<columnName>' because it is part of the FOR UPDATE list.
42X43	The ResultSetMetaData returned for the class/object '<className>' was null. In order to use this class as an external virtual table, the ResultSetMetaData cannot be null.
42X44	Invalid length '<length>' in column specification.
42X45	<type> is an invalid type for argument number <value> of <value>.
42X48	Value '<value>' is not a valid precision for <value>.
42X49	Value '<value>' is not a valid integer literal.
42X50	No method was found that matched the method call <methodName>.<value>(<value>), tried all combinations of object and primitive types and any possible type conversion for any parameters the method call may have. The method might exist but it is not public and/or static, or the parameter types are not method invocation convertible.
42X51	The class '<className>' does not exist or is inaccessible. This can happen if the class is not public.
42X52	Calling method ('<methodName>') using a receiver of the Java primitive type '<type>' is not allowed.
42X53	The LIKE predicate can only have 'CHAR' or 'VARCHAR' operands. Type '<type>' is not permitted.
42X54	The Java method '<methodName>' has a ? as a receiver. This is not allowed.
42X55	Table name '<tableName>' should be the same as '<value>'.
42X56	The number of columns in the view column list does not match the number of columns in the underlying query expression in the view definition for '<value>'.
42X57	The getColumnCount() for external virtual table '<tableName>' returned an invalid value '<value>'. Valid values are greater than or equal to 1.

SQLSTATE	Texto da mensagem
42X58	The number of columns on the left and right sides of the <code><tableName></code> must be the same.
42X59	The number of columns in each VALUES constructor must be the same.
42X60	Invalid value ' <code><value></code> ' for insertMode property specified for table ' <code><tableName></code> '.
42X61	Types ' <code><type></code> ' and ' <code><type></code> ' are not <code><value></code> compatible.
42X62	' <code><value></code> ' is not allowed in the ' <code><nome-do-esquema></code> ' schema.
42X63	The USING clause did not return any results. No parameters can be set.
42X64	In the Properties list, the invalid value ' <code><value></code> ' was specified for the useStatistics property. The only valid values are TRUE or FALSE .
42X65	Index ' <code><index></code> ' does not exist.
42X66	Column name ' <code><columnName></code> ' appears more than once in the CREATE INDEX statement.
42X68	No field ' <code><fieldName></code> ' was found belonging to class ' <code><className></code> '. The field might exist, but it is not public, or the class does not exist or is not public.
42X69	It is not allowed to reference a field (' <code><fieldName></code> ') using a referencing expression of the Java primitive type ' <code><type></code> '.
42X72	No static field ' <code><fieldName></code> ' was found belonging to class ' <code><className></code> '. The field might exist, but it is not public and/or static, or the class does not exist or the class is not public.
42X73	Method resolution for signature <code><value>.<value>(<value>)</code> was ambiguous. (No single maximally specific method.)
42X74	Invalid CALL statement syntax.
42X75	No constructor was found with the signature <code><value>(<value>)</code> . It may be that the parameter types are not method invocation convertible.
42X76	At least one column, ' <code><columnName></code> ', in the primary key being added is nullable. All columns in a primary key must be non-nullable.
42X77	Column position ' <code><columnPosition></code> ' is out of range for the query expression.
42X78	Column ' <code><columnName></code> ' is not in the result of the query expression.
42X79	Column name ' <code><columnName></code> ' appears more than once in the result of the query expression.
42X80	VALUES clause must contain at least one element. Empty elements are not allowed.
42X82	The USING clause returned more than one row. Only single-row ResultSets are permissible.
42X83	The constraints on column ' <code><columnName></code> ' require that it be both nullable and not nullable.
42X84	Index ' <code><index></code> ' was created to enforce constraint ' <code><constraintName></code> '. It can only be dropped by dropping the constraint.
42X85	Constraint ' <code><constraintName></code> ' is required to be in the same schema as table ' <code><tableName></code> '.
42X86	ALTER TABLE failed. There is no constraint ' <code><constraintName></code> ' on table ' <code><tableName></code> '.
42X87	At least one result expression (THEN or ELSE) of the ' <code><expressão></code> ' expression must not be a '?'. .
42X88	A conditional has a non-boolean operand. The operand of a conditional must evaluate to TRUE, FALSE, or UNKNOWN.
42X89	Types ' <code><type></code> ' and ' <code><type></code> ' are not type compatible. Neither type is assignable to the

SQLSTATE	Texto da mensagem
	other type.
42X90	More than one primary key constraint specified for table '<tableName>'.
42X91	Constraint name '<constraintName>' appears more than once in the CREATE TABLE statement.
42X92	Column name '<columnName>' appears more than once in a constraint's column list.
42X93	Table '<tableName>' contains a constraint definition with column '<columnName>' which is not in the table.
42X94	<value> '<value>' does not exist.
42X96	The database classpath contains an unknown jar file '<fileName>'.
42X98	Parameters are not allowed in a VIEW definition.
42Y00	Class '<className>' does not implement org.apache.derby.iapi.db.AggregateDefinition and thus cannot be used as an aggregate expression.
42Y01	Constraint '<constraintName>' is invalid.
42Y03	'<statement>' is not recognized as a function or procedure.
42Y04	Cannot create a procedure or function with EXTERNAL NAME '<name>' because it is not a list separated by periods. The expected format is <fulljavapath>.<method name>.
42Y05	There is no Foreign Key named '<key>'.
42Y07	Schema '<nome-do-esquema>' does not exist.
42Y08	Foreign key constraints are not allowed on system tables.
42Y09	Void methods are only allowed within a CALL statement.
42Y10	A table constructor that is not in an INSERT statement has all ? parameters in one of its columns. For each column, at least one of the rows must have a non-parameter.
42Y11	A join specification is required with the '<clauseName>' clause.
42Y12	The ON clause of a JOIN is a '<expressionType>' expression. It must be a BOOLEAN expression.
42Y13	Column name '<columnName>' appears more than once in the CREATE VIEW statement.
42Y16	No public static method '<methodName>' was found in class '<className>'. The method might exist, but it is not public, or it is not static.
42Y19	'<columnName>' appears multiple times in the GROUP BY list. Columns in the GROUP BY list must be unambiguous.
42Y22	Aggregate <aggregateType> cannot operate on type <type>.
42Y23	Incorrect JDBC type info returned for column <columnName>.
42Y24	View '<viewName>' is not updatable. (Views are currently not updatable.)
42Y25	'<tableName>' is a system table. Users are not allowed to modify the contents of this table.
42Y27	Parameters are not allowed in the trigger action.
42Y29	The SELECT list of a non-grouped query contains at least one invalid expression. When the SELECT list contains at least one aggregate then all entries must be valid aggregate expressions.
42Y30	The SELECT list of a grouped query contains at least one invalid expression. If a SELECT list has a GROUP BY, the list may only contain grouping columns and valid aggregate expressions.
42Y32	Aggregator class '<className>' aggregate '<aggregateName>' on type <type> does

SQLSTATE	Texto da mensagem
	not implement com.ibm.db2j.aggregates.Aggregator.
42Y33	Aggregate <aggregateName> contains one or more aggregates.
42Y34	Column name '<columnName>' matches more than one result column in table '<tableName>'.
42Y35	Column reference '<reference>' is invalid. When the SELECT list contains at least one aggregate then all entries must be valid aggregate expressions.
42Y36	Column reference '<value>' is invalid. For a SELECT list with a GROUP BY, the list may only contain grouping columns and valid aggregate expressions.
42Y37	'<value>' is a Java primitive and cannot be used with this operator.
42Y38	<pre>insertMode = replace</pre> is not permitted on an insert where the target table, '<tableName>', is referenced in the SELECT.
42Y39	'<value>' may not appear in a CHECK CONSTRAINT definition because it may return non-deterministic results.
42Y40	'<value>' appears multiple times in the UPDATE OF column list for trigger '<triggerName>'.
42Y41	'<value>' cannot be directly invoked via EXECUTE STATEMENT because it is part of a trigger.
42Y42	Scale '<scaleValue>' is not a valid scale for a <value>.
42Y43	Scale '<scaleValue>' is not a valid scale with precision of '<precision>'.
42Y44	Invalid key '<key>' specified in the Properties list of a FROM list. The case-sensitive keys that are currently supported are '<key>'.
42Y45	VTI '<value>' cannot be bound because it is a special trigger VTI and this statement is not part of a trigger action or WHEN clause.
42Y46	Invalid Properties list in FROM list. There is no index '<index>' on table '<tableName>'.
42Y48	Invalid Properties list in FROM list. Either there is no named constraint '<constraintName>' on table '<tableName>' or the constraint does not have a backing index.
42Y49	Multiple values specified for property key '<key>'.
42Y50	Properties list for table '<tableName>' may contain values for index or for constraint but not both.
42Y55	'<value>' cannot be performed on '<value>' because it does not exist.
42Y56	Invalid join strategy '<strategyValue>' specified in Properties list on table '<tableName>'. The currently supported values for a join strategy are: <pre>hash</pre> and <pre>nestedloop</pre> .
42Y58	NumberFormatException occurred when converting value '<value>' for optimizer override '<value>'.
42Y59	Invalid value, '<value>', specified for hashInitialCapacity override. Value must be greater than 0.
42Y60	Invalid value, '<value>', specified for hashLoadFactor override. Value must be greater than 0.0 and less than or equal to 1.0.

SQLSTATE	Texto da mensagem
42Y61	Invalid value, '<value>' specified for hashMaxCapacity override. Value must be greater than 0.
42Y62	'<value>' is not allowed on '<value>' because it is a view.
42Y63	Hash join requires an optimizable equijoin predicate on a column in the selected index or heap. An optimizable equijoin predicate does not exist on any column in table or index '<index>'. Use the 'index' optimizer override to specify such an index or the heap on table '<tableName>'.
42Y64	bulkFetch value of '<value>' is invalid. The minimum value for bulkFetch is 1.
42Y65	bulkFetch is not permitted on '<joinTypes>' joins.
42Y66	bulkFetch is not permitted on updatable cursors.
42Y67	Schema '<nome-do-esquema>' cannot be dropped.
42Y69	No valid execution plan was found for this statement. This may have one of two causes: either you specified a hash join strategy when hash join is not allowed (no optimizable equijoin) or you are attempting to join two external virtual tables, each of which references the other, and so the statement cannot be evaluated.
42Y70	The user specified an illegal join order. This could be caused by a join column from an inner table being passed as a parameter to an external virtual table.
42Y71	System function or procedure '<procedureName>' cannot be dropped.
42Y82	System generated stored prepared statement '<statement>' that cannot be dropped using DROP STATEMENT. It is part of a trigger.
42Y83	An untyped null is not permitted as an argument to aggregate <aggregateName>. Please cast the null to a suitable type.
42Y84	'<value>' may not appear in a DEFAULT definition.
42Y85	The DEFAULT keyword is only allowed in a VALUES clause when the VALUES clause appears within an INSERT statement.
42Y90	FOR UPDATE is not permitted in this type of statement.
42Y91	The USING clause is not permitted in an EXECUTE STATEMENT for a trigger action.
42Y92	<triggerName> triggers may only reference <value> transition variables/tables.
42Y93	Illegal REFERENCING clause: only one name is permitted for each type of transition variable/table.
42Y94	An AND or OR has a non-boolean operand. The operands of AND and OR must evaluate to TRUE, FALSE, or UNKNOWN.
42Y95	The '<operatorName>' operator with a left operand type of '<operandType>' and a right operand type of '<operandType>' is not supported.
42Y97	Invalid escape character at line '<lineNumber>', column '<columnName>'.
42Z02	Multiple DISTINCT aggregates are not supported at this time.
42Z07	Aggregates are not permitted in the ON clause.
42Z08	Bulk insert replace is not permitted on '<value>' because it has an enabled trigger (<value>).
42Z15	Invalid type specified for column '<columnName>'. The type of a column may not be changed.
42Z16	Only columns of type VARCHAR may have their length altered.
42Z17	Invalid length specified for column '<columnName>'. Length must be greater than the current column length.
42Z18	Column '<columnName>' is part of a foreign key constraint '<constraintName>'. To alter the length of this column, you should drop the constraint first, perform the ALTER

SQLSTATE	Texto da mensagem
	TABLE, and then recreate the constraint.
42Z19	Column '<columnName>' is being referenced by at least one foreign key constraint '<constraintName>'. To alter the length of this column, you should drop referencing constraints, perform the ALTER TABLE, and then recreate the constraints.
42Z20	Column '<columnName>' cannot be made nullable. It is part of a primary key, which cannot have any nullable columns.
42Z21	Invalid increment specified for identity for column '<columnName>'. Increment cannot be zero.
42Z22	Invalid type specified for identity column '<columnName>'. The only valid types for identity columns are BIGINT, INT and SMALLINT.
42Z23	Attempt to modify an identity column '<columnName>'.
42Z24	Overflow occurred in identity value for column '<columnName>' in table '<tableName>'.
42Z25	INTERNAL ERROR identity counter. Update was called without arguments with current value = NULL .
42Z26	A column, '<columnName>', with an identity default cannot be made nullable.
42Z27	A nullable column, '<columnName>', cannot be modified to have identity default.
42Z50	INTERNAL ERROR: Unable to generate code for <value>.
42Z53	INTERNAL ERROR: Do not know what type of activation to generate for node choice <value>.
42Z90	Class '<className>' does not return an updatable ResultSet.
42Z91	subquery
42Z92	repeatable read
42Z93	Constraints '<constraintName>' and '<constraintName>' have the same set of columns, which is not allowed.
42Z97	Renaming column '<columnName>' will cause check constraint '<constraintName>' to break.
42Z99	String or Hex literal cannot exceed 64K.
42Z9A	read uncommitted
42Z9B	The external virtual table interface does not support BLOB or CLOB columns. '<value>' column '<value>'.
42Z9D	'<statement>' statements are not allowed in '<triggerName>' triggers.
42Z9E	Constraint '<constraintName>' is not a <value> constraint.
42Z9F	Too many indexes (<index>) on the table <tableName>. The limit is <number>.

Tabela 51. Código de classe XOX: Exceção de execução

SQLState	Texto da mensagem
X0X02	Table '<tableName>' cannot be locked in '<mode>' mode.
X0X03	Invalid transaction state - held cursor requires same isolation level
X0X05	Table '<tableName>' does not exist.
X0X07	Cannot drop jar file '<fileName>' because its on your db2j.database.classpath '<classpath>'.
X0X0E	The column position '<columnPosition>' listed in the auto-generated column selection

SQLState	Texto da mensagem
	array was not found in the insert table.
X0X0F	Column name '<columnName>' listed in auto-generated column selection array not found in the insert table.
X0X10	The USING clause returned more than one row; only single-row ResultSets are permissible.
X0X11	The USING clause did not return any results so no parameters can be set.
X0X13	Jar file '<fileName>' does not exist in schema '<nome-do-esquema>'.
X0X14	Binding directly to an XML value is not allowed; try using XMLPARSE.
X0X15	XML values are not allowed in top-level result sets; try using XMLSERIALIZE.
X0X16	XML syntax error; missing keyword(s): '<keywords>'.
X0X17	Invalid target type for XMLSERIALIZE: '<value>'.
X0X18	XML feature not supported: '<featureName>'.
X0X57	An attempt was made to put a Java value of type '<type>' into a SQL value, but there is no corresponding SQL type. The Java value is probably the result of a method call or field access.
X0X60	A cursor with name '<cursorName>' already exists.
X0X61	The values for column '<value>' in index '<value>' and table '<value>.<value>' do not match for row location '<value>'. The value in the index is '<value>', while the value in the base table is '<value>'. The full index key, including the row location, is '<value>'. The suggested corrective action is to recreate the index.
X0X62	Inconsistency found between table '<tableName>' and index '<index>'. Error when trying to retrieve row location '<rowLocation>' from the table. The full index key, including the row location, is '<index>'. The suggested corrective action is to recreate the index.
X0X63	Got IOException '<value>'.
X0X67	Columns of type '<type>' may not be used in CREATE INDEX, ORDER BY, GROUP BY, UNION, INTERSECT, EXCEPT or DISTINCT statements because comparisons are not supported for that type.
X0X81	<value> '<value>' does not exist.
X0X85	Index '<index>' was not created because '<type>' is not a valid index type.
X0X86	0 is an invalid parameter value for <pre>ResultSet.absolute(int row)</pre> .
X0X87	<pre>ResultSet.relative(int row)</pre> cannot be called when the cursor is not positioned on a row.
X0X95	Operation '<operation>' cannot be performed on object '<object>' because there is an open ResultSet dependent on that object.
X0X99	Index '<index>' does not exist.
X0XML	Encountered unexpected error while processing XML; see next exception for details.

Tabela 52. Código de classe X0Y: Exceções de execução

SQLSTATE	Texto da mensagem
X0Y16	'<value>' is not a view. If it is a table, then use DROP TABLE instead.

SQLSTATE	Texto da mensagem
X0Y23	Operation '<operation>' cannot be performed on object '<object>' because VIEW '<viewName>' is dependent on that object.
X0Y24	Operation '<operation>' cannot be performed on object '<object>' because STATEMENT '<statement>' is dependent on that object.
X0Y25	Operation '<value>' cannot be performed on object '<value>' because <value> '<value>' is dependent on that object.
X0Y26	Index '<index>' is required to be in the same schema as table '<tableName>'.
X0Y28	Index '<index>' cannot be created on system table '<tableName>'. Users cannot create indexes on system tables.
X0Y32	<value> '<value>' already exists in <value> '<value>'.
X0Y38	Cannot create index '<index>' because table '<tableName>' does not exist.
X0Y41	Constraint '<constraintName>' is invalid because the referenced table <tableName> has no primary key. Either add a primary key to <tableName> or explicitly specify the columns of a unique constraint that this foreign key references.
X0Y42	Constraint '<constraintName>' is invalid: the types of the foreign key columns do not match the types of the referenced columns
X0Y43	Constraint '<value>' is invalid: the number of columns in <value> (<value>) does not match the number of columns in the referenced key (<value>).
X0Y44	Constraint '<constraintName>' is invalid: there is no unique or primary key constraint on table '<tableName>' that matches the number and types of the columns in the foreign key.
X0Y45	Foreign key constraint '<constraintName>' cannot be added to or enabled on table <tableName> because one or more foreign keys do not have matching referenced keys.
X0Y46	Constraint '<constraintName>' is invalid: referenced table <tableName> does not exist.
X0Y54	Schema '<nome-do-esquema>' cannot be dropped because it is not empty.
X0Y55	The number of rows in the base table does not match the number of rows in at least 1 of the indexes on the table. Index '<value>' on table '<value>.<value>' has <value> rows, but the base table has <value> rows. The suggested corrective action is to recreate the index.
X0Y56	'<value>' is not allowed on the System table '<value>'.
X0Y57	A non-nullable column cannot be added to table '<tableName>' because the table contains at least one row. Non-nullable columns can only be added to empty tables.
X0Y58	Attempt to add a primary key constraint to table '<tableName>' failed because the table already has a constraint of that type. A table can only have a single primary key constraint.
X0Y59	Attempt to add or enable constraint(s) on table '<tableName>' failed because the table contains <rowName> row(s) that violate the following check constraint(s): <constraintName>.
X0Y63	The command on table '<tableName>' failed because null data was found in the primary key or unique constraint/index column(s). All columns in a primary or unique index key must not be null.
X0Y66	Cannot issue commit in a nested connection when there is a pending operation in the parent connection.
X0Y67	Cannot issue rollback in a nested connection when there is a pending operation in the parent connection.
X0Y68	<value> '<value>' already exists.
X0Y69	<value> is not permitted because trigger <triggerName> is active on <value>.

SQLSTATE	Texto da mensagem
X0Y70	INSERT, UPDATE and DELETE are not permitted on table <i><tableName></i> because trigger <i><triggerName></i> is active.
X0Y71	Transaction manipulation such as SET ISOLATION is not permitted because trigger <i><triggerName></i> is active.
X0Y72	Bulk insert replace is not permitted on ' <i><value></i> ' because it has an enabled trigger (<i><value></i>).
X0Y77	Cannot issue set transaction isolation statement on a global transaction that is in progress because it would have implicitly committed the global transaction.
X0Y78	Statement.executeQuery() cannot be called with a statement that returns a row count.
X0Y79	Statement.executeUpdate() cannot be called with a statement that returns a ResultSet.
X0Y80	ALTER table ' <i><tableName></i> ' failed. Null data found in column ' <i><columnName></i> '.
X0Y83	WARNING: While deleting a row from a table the index row for base table row <i><rowName></i> was not found in index with conglomerate id <i><id></i> . This problem has automatically been corrected as part of the delete operation.

Tabela 53. Código de classe XBCA: CacheService

SQLSTATE	Texto da mensagem
XBCA0	Cannot create new object with key <i><key></i> in <i><cache></i> cache. The object already exists in the cache.

Tabela 54. Código de classe XBCM: ClassManager

SQLSTATE	Texto da mensagem
XBCM1	Java linkage error thrown during load of generated class <i><className></i> .
XBCM2	Cannot create an instance of generated class <i><className></i> .
XBCM3	Method <i><methodName></i> () does not exist in generated class <i><className></i> .

Tabela 55. Código de classe XBCX: Criptografia

SQLSTATE	Texto da mensagem
XBCX0	Exception from Cryptography provider. See next exception for details.
XBCX1	Initializing cipher with illegal mode, must be either ENCRYPT or DECRYPT.
XBCX2	Initializing cipher with a boot password that is too short. The password must be at least <i><number></i> characters long.
XBCX5	Cannot change boot password to null.
XBCX6	Cannot change boot password to a non-string serializable type.
XBCX7	Wrong format for changing boot password. Format must be : old_boot_password, new_boot_password.
XBCX8	Cannot change boot password for a non-encrypted database.
XBCX9	Cannot change boot password for a read-only database.
XBCXA	Wrong boot password.
XBCXB	Bad encryption padding ' <i><value></i> ' or padding not specified. 'NoPadding' must be used.
XBCXC	Encryption algorithm ' <i><algorithmName></i> ' does not exist. Please check that the chosen provider ' <i><providerName></i> ' supports this algorithm.
XBCXD	The encryption algorithm cannot be changed after the database is created.

SQLSTATE	Texto da mensagem
XBCXE	The encryption provider cannot be changed after the database is created.
XBCXF	The class '<className>' representing the encryption provider cannot be found.
XBCXG	The encryption provider '<providerName>' does not exist.
XBCXH	The encryptionAlgorithm '<algorithmName>' is not in the correct format. The correct format is algorithm/feedbackMode/NoPadding.
XBCXI	The feedback mode '<mode>' is not supported. Supported feedback modes are CBC, CFB, OFB and ECB.
XBCXJ	The application is using a version of the Java Cryptography Extension (JCE) earlier than 1.2.1. Please upgrade to JCE 1.2.1 and try the operation again.
XBCXK	The given encryption key does not match the encryption key used when creating the database. Please ensure that you are using the correct encryption key and try again.
XBCXL	The verification process for the encryption key was not successful. This could have been caused by an error when accessing the appropriate file to do the verification process. See next exception for details.

Tabela 56. Código de classe XBM: Monitor

SQLSTATE	Texto da mensagem
XBM01	Startup failed due to an exception. See next exception for details.
XBM02	Startup failed due to missing functionality for <value>. Please ensure your classpath includes the correct Derby Derby software.
XBM05	Startup failed due to missing product version information for <value>.
XBM06	Startup failed. An encrypted database cannot be accessed without the correct boot password.
XBM07	Startup failed. Boot password must be at least 8 bytes long.
XBM08	Could not instantiate <value> StorageFactory class <value>.
XBM0G	Failed to start encryption engine. Please make sure you are running Java 2 and have downloaded an encryption provider such as jce and put it in your classpath.
XBM0H	Directory <directoryName> cannot be created.
XBM0I	Directory <directoryName> cannot be removed.
XBM0J	Directory <directoryName> already exists.
XBM0K	Unknown sub-protocol for database name <databaseName>.
XBM0L	Specified authentication scheme class <className> does implement the authentication interface <interfaceName>.
XBM0M	Error creating instance of authentication scheme class <className>.
XBM0N	JDBC Driver registration with java.sql.DriverManager failed. See next exception for details.
XBM0P	Service provider is read-only. Operation not permitted.
XBM0Q	File <fileName> not found. Please make sure that backup copy is the correct one and it is not corrupted.
XBM0R	Unable to remove file <fileName>.
XBM0S	Unable to rename file '<fileName>' to '<fileName>'
XBM0T	Ambiguous sub-protocol for database name <databaseName>.
XBM0U	No class was registered for identifier <identifierName>.
XBM0V	An exception was thrown while loading class <className> registered for identifier

SQLSTATE	Texto da mensagem
	<i>identifierName</i> >.
XBM0W	An exception was thrown while creating an instance of class <className> registered for identifier <identifierName>.
XBM0X	Supplied territory description '<value>' is invalid, expecting In[_CO[_variant]] In=lower-case two-letter ISO-639 language code, CO=upper-case two-letter ISO-3166 country codes, see java.util.Locale.
XBM0Y	Backup database directory <directoryName> not found. Please make sure that the specified backup path is right.
XBM0Z	Unable to copy file '<value>' to '<value>'. Please make sure that there is enough space and permissions are correct.

Tabela 57. Código de classe XCL: Non-SQLSTATE

SQLSTATE	Texto da mensagem
XCL01	ResultSet does not return rows. Operation <operationName> not permitted.
XCL05	Activation closed. Operation <operationName> not permitted.
XCL07	Cursor '<cursorName>' is closed. Verify that autocommit is OFF.
XCL08	Cursor '<cursorName>' is not on a row.
XCL09	An Activation was passed to the '<methodName>' method that does not match the PreparedStatement.
XCL10	A PreparedStatement has been recompiled and the parameters have changed. If you are using JDBC you must prepare the statement again.
XCL12	An attempt was made to put a data value of type '<type>' into a data value of type '<type>'.
XCL13	The parameter position '<parameterPosition>' is out of range. The number of parameters for this prepared statement is '<number>'.
XCL14	The column position '<value>' is out of range. The number of columns for this ResultSet is '<number>'.
XCL15	A ClassCastException occurred when calling the compareTo() method on an object '<object>'. The parameter to compareTo() is of class '<className>'.
XCL16	ResultSet not open. Operation '<operation>' not permitted. Verify that autocommit is OFF.
XCL17	Statement not allowed in this database.
XCL19	Missing row in table '<tableName>' for key '<key>'.
XCL20	Catalogs at version level '<versionLevel>' cannot be upgraded to version level '<versionLevel>'.
XCL21	You are trying to execute a Data Definition statement (CREATE, DROP, or ALTER) while preparing a different statement. This is not allowed. It can happen if you execute a Data Definition statement from within a static initializer of a Java class that is being used from within a SQL statement.
XCL22	Parameter <parameterName> cannot be registered as an OUT parameter because it is an IN parameter.
XCL23	SQL type number '<type>' is not a supported type by registerOutParameter().
XCL24	Parameter <parameterName> appears to be an output parameter, but it has not been so designated by registerOutParameter(). If it is not an output parameter, then it has to be set to type <type>.
XCL25	Parameter <parameterName> cannot be registered to be of type <type> because it

SQLSTATE	Texto da mensagem
	maps to type <i><type></i> and they are incompatible.
XCL26	Parameter <i><parameterName></i> is not an output parameter.
XCL27	Return output parameters cannot be set.
XCL30	An IOException was thrown when reading a ' <i><value></i> ' from an InputStream.
XCL31	Statement closed.
XCL33	The table cannot be defined as a dependent of table <i><tableName></i> because of delete rule restrictions. (The relationship is self-referencing and a self-referencing relationship already exists with the SET NULL delete rule.)
XCL34	The table cannot be defined as a dependent of table <i><tableName></i> because of delete rule restrictions. (The relationship forms a cycle of two or more tables that cause the table to be delete-connected to itself (all other delete rules in the cycle would be CASCADE)).
XCL35	The table cannot be defined as a dependent of table <i><tableName></i> because of delete rule restrictions. (The relationship causes the table to be delete-connected to the indicated table through multiple relationships and the delete rule of the existing relationship is SET NULL).
XCL36	The delete rule of foreign key must be <i><ruleName></i> . (The referential constraint is self-referencing and an existing self-referencing constraint has the indicated delete rule (NO ACTION, RESTRICT or CASCADE).)
XCL37	The delete rule of foreign key must be <i><ruleName></i> . (The referential constraint is self-referencing and the table is dependent in a relationship with a delete rule of CASCADE.)
XCL38	The delete rule of foreign key must be <i><ruleName></i> . (The relationship would cause the table to be delete-connected to the same table through multiple relationships and such relationships must have the same delete rule (NO ACTION, RESTRICT or CASCADE).)
XCL39	The delete rule of foreign key cannot be CASCADE. (A self-referencing constraint exists with a delete rule of SET NULL, NO ACTION or RESTRICT.)
XCL40	The delete rule of foreign key cannot be CASCADE. (The relationship would form a cycle that would cause a table to be delete-connected to itself. One of the existing delete rules in the cycle is not CASCADE, so this relationship may be definable if the delete rule is not CASCADE.)
XCL41	the delete rule of foreign key can not be CASCADE. (The relationship would cause another table to be delete-connected to the same table through multiple paths with different delete rules or with delete rule equal to SET NULL.)
XCL42	CASCADE
XCL43	SET NULL
XCL44	RESTRICT
XCL45	NO ACTION
XCL46	SET DEFAULT
XCL47	Use of ' <i><value></i> ' requires database to be upgraded from version <i><versionNumber></i> to version <i><versionNumber></i> or later.
XCL48	TRUNCATE TABLE is not permitted on ' <i><value></i> ' because unique/primary key constraints on this table are referenced by enabled foreign key constraints from other tables.
XCL49	TRUNCATE TABLE is not permitted on ' <i><value></i> ' because it has an enabled DELETE trigger (<i><value></i>).
XCL50	Upgrading the database from a previous version is not supported. The database being

SQLSTATE	Texto da mensagem
	accessed is at version level ' <i>versionNumber</i> ', this software is at version level ' <i>versionNumber</i> '.
XCL51	The requested function cannot reference tables in SESSION schema.

Tabela 58. Código de classe XCW: Atualização não suportada

SQLSTATE	Texto da mensagem
XCW00	Unsupported upgrade from ' <i>value</i> ' to ' <i>value</i> '.

Tabela 59. Código de classe XCXA: Erro de análise de ID

SQLSTATE	Texto da mensagem
XCXA0	Invalid identifier.

Tabela 60. Código de classe XCXB: DB_Class_Path_Parse_Error

SQLSTATE	Texto da mensagem
XCXB0	Invalid database classpath: ' <i>classpath</i> '.

Tabela 61. Código de classe XCXC: Erro de análise de lista de ID

SQLSTATE	Texto da mensagem
XCXC0	Invalid id list.

Tabela 62. Código de classe XCXE: Sem idioma

SQLSTATE	Texto da mensagem
XCXE0	You are trying to do an operation that uses the territory of the database, but the database does not have a territory.

Tabela 63. Código de classe XCY: Propriedades

SQLSTATE	Texto da mensagem
XCY00	Invalid value for property ' <i>value</i> '=' <i>value</i> '.
XCY02	The requested property change is not supported ' <i>value</i> '=' <i>value</i> '.
XCY03	Required property ' <i>propertyName</i> ' has not been set.

Tabela 64. Código de classe XCZ: org.apache.derby.database.UserUtility

SQLSTATE	Texto da mensagem
XCZ00	Unknown permission ' <i>permissionName</i> '.
XCZ01	Unknown user ' <i>userName</i> '.
XCZ02	Invalid parameter ' <i>value</i> '=' <i>value</i> '.

Tabela 65. Código de classe XD00x: Gerenciador de dependências

SQLSTATE	Texto da mensagem
XD003	Unable to restore dependency from disk. DependableFinder = '<value>'. Further information: '<value>'.
XD004	Unable to store dependencies.

Tabela 66. Código de classe XIE: Importação/Exportação

SQLSTATE	Texto da mensagem
XIE01	Connection was null.
XIE03	Data found on line <lineNumber> for column <columnName> after the stop delimiter.
XIE04	Data file not found: <fileName>.
XIE05	Data file cannot be null.
XIE06	Entity name was null.
XIE07	Field and record separators cannot be substrings of each other.
XIE08	There is no column named: <columnName>.
XIE09	The total number of columns in the row is: <number>.
XIE0B	Column '<columnName>' in the table is of type <type>, it is not supported by the import/export feature.
XIE0D	Cannot find the record separator on line <lineNumber>.
XIE0E	Read endOfFile at unexpected place on line <lineNumber>.
XIE0I	An IOException occurred while writing data to the file.
XIE0J	A delimiter is not valid or is used more than once.
XIE0K	The period was specified as a character string delimiter.
XIE0M	Table '<tableName>' does not exist.

Tabela 67. Código de classe XJ: Erros de conectividade

SQLSTATE	Texto da mensagem
XJ004	Database '<databaseName>' not found.
XJ009	Use of CallableStatement required for stored procedure call or use of output parameters: <value>
XJ010	Cannot issue savepoint when autoCommit is on.
XJ011	Cannot pass null for savepoint name.
XJ012	'<value>' already closed.
XJ013	No ID for named savepoints.
XJ014	No name for un-named savepoints.
XJ015	Derby system shutdown.
XJ016	Method '<methodName>' not allowed on prepared statement.
XJ017	No savepoint command allowed inside the trigger code.
XJ018	Column name cannot be null.
XJ020	Object type not convertible to TYPE '<typeName>', invalid java.sql.Types value, or object was null.
XJ022	Unable to set stream: '<name>'.

SQLSTATE	Texto da mensagem
XJ023	Input stream held less data than requested length.
XJ025	Input stream cannot have negative length.
XJ028	The URL '<urlValue>' is not properly formed.
XJ030	Cannot set AUTOCOMMIT ON when in a nested connection.
XJ040	Failed to start database '<databaseName>', see the next exception for details.
XJ041	Failed to create database '<databaseName>', see the next exception for details.
XJ042	'<value>' is not a valid value for property '<propertyName>'.
XJ044	'<value>' is an invalid scale.
XJ045	Invalid or (currently) unsupported isolation level, '<value>', passed to Connection.setTransactionIsolationLevel(). The currently supported values are java.sql.Connection.TRANSACTION_SERIALIZABLE, java.sql.Connection.TRANSACTION_REPEATABLE_READ, java.sql.Connection.TRANSACTION_READ_COMMITTED, and java.sql.Connection.TRANSACTION_READ_UNCOMMITTED.
XJ049	Conflicting create attributes specified.
XJ04B	Batch cannot contain a command that attempts to return a result set.
XJ04C	CallableStatement batch cannot contain output parameters.
XJ056	Cannot set AUTOCOMMIT ON when in an XA connection.
XJ057	Cannot commit a global transaction using the Connection. Commit processing must go through XAResource interface.
XJ058	Cannot rollback a global transaction using the Connection, commit processing must go through XAResource interface.
XJ059	Cannot close a connection while a global transaction is still active.
XJ05B	JDBC attribute '<attributeName>' has an invalid value '<value>', Valid values are '<value>'.
XJ05C	Cannot set holdability ResultSet.HOLD_CURSORS_OVER_COMMIT for a global transaction.
XJ061	The '<valueName>' method is only allowed on scroll cursors.
XJ062	Invalid parameter value '<value>' for ResultSet.setFetchSize(int rows).
XJ063	Invalid parameter value '<value>' for Statement.setMaxRows(int maxRows). Parameter value must be >= 0.
XJ064	Invalid parameter value '<value>' for setFetchDirection(int direction).
XJ065	Invalid parameter value '<value>' for Statement.setFetchSize(int rows).
XJ066	Invalid parameter value '<value>' for Statement.setMaxFieldSize(int max).
XJ067	SQL text pointer is null.
XJ068	Only executeBatch and clearBatch allowed in the middle of a batch.
XJ069	No SetXXX methods allowed in case of USING execute statement.
XJ070	Negative or zero position argument '<argument>' passed in a Blob or Clob method.
XJ071	Zero or negative length argument '<argument>' passed in a BLOB or CLOB method.
XJ072	Null pattern or searchStr passed in to a BLOB or CLOB position method.
XJ073	The data in this BLOB or CLOB is no longer available. The BLOB or CLOB's transaction may be committed, or its connection is closed.
XJ076	The position argument '<argument>' exceeds the size of the BLOB/CLOB.
XJ077	Got an exception when trying to read the first byte/character of the BLOB/CLOB

SQLSTATE	Texto da mensagem
	pattern using getBytes/getSubString.
XJ080	USING execute statement passed <numparameters> parameters rather than <numparameters>.
XJ081	Conflicting create/restore/recovery attributes specified.
XJ081	Invalid value '<value>' passed as parameter '<parameterName>' to method '<moduleName>'.

Tabela 68. Código de classe XSAIx: Armazenamento - access.protocol.interface statement exceptions

SQLSTATE	Texto da mensagem
XSAI2	The conglomerate (<value>) requested does not exist.
XSAI3	Feature not implemented.

Tabela 69. Código de classe XSAMx: Armazenamento - AccessManager

SQLSTATE	Texto da mensagem
XSAM0	Exception encountered while trying to boot module for '<value>'.
XSAM2	There is no index or conglomerate with conglom id '<conglomID>' to drop.
XSAM3	There is no index or conglomerate with conglom id '<conglomID>'.
XSAM4	There is no sort called '<sortName>'.
XSAM5	Scan must be opened and positioned by calling next() before making other calls.
XSAM6	Record <recordnumber> on page <page> in container <containerName> not found.

Tabela 70. Código de classe XSASx: Armazenamento - Sort

SQLSTATE	Texto da mensagem
XSAS0	A scan controller interface method was called which is not appropriate for a scan on a sort.
XSAS1	An attempt was made to fetch a row before the beginning of a sort or after the end of a sort.
XSAS3	The type of a row inserted into a sort does not match the sorts template.
XSAS6	Could not acquire resources for sort.

Tabela 71. Código de classe XSAXx: Armazenamento - access.protocol.XA statement exception

SQLSTATE	Texto da mensagem
XSAX0	XA protocol violation.
XSAX1	An attempt was made to start a global transaction with an Xid of an existing global transaction.

Tabela 72. Código de classe XSCBx: Armazenamento - BTree

SQLSTATE	Texto da mensagem
XSCB0	Could not create container.

SQLSTATE	Texto da mensagem
XSCB1	Container <containerName> not found.
XSCB2	The required property <propertyName> not found in the property list given to createConglomerate() for a btree secondary index.
XSCB3	Unimplemented feature.
XSCB4	A method on a btree open scan was called prior to positioning the scan on the first row (that is, no next() call has been made yet). The current state of the scan is (<value>).
XSCB5	During logical undo of a btree insert or delete, the row could not be found in the tree.
XSCB6	Limitation: Record of a btree secondary index cannot be updated or inserted due to lack of space on the page. Use the parameters derby.storage.pageSize and/or derby.storage.pageReservedSpace to work around this limitation.
XSCB7	An internal error was encountered during a btree scan - current_rh is null = <value>, position key is null = <value>.
XSCB8	The btree conglomerate <value> is closed.
XSCB9	Reserved for testing.

Tabela 73. Código de classe XSCG0: Conglomerado

SQLSTATE	Texto da mensagem
XSCG0	Could not create a template.

Tabela 74. Código de classe XSCHx: Heap

SQLSTATE	Texto da mensagem
XSCH0	Could not create container.
XSCH1	Container <containerName> not found.
XSCH4	Conglomerate could not be created.
XSCH5	In a base table there was a mismatch between the requested column number <columnnumber> and the maximum number of columns <maxcol>.
XSCH6	The heap container with container id <containerID> is closed.
XSCH7	The scan is not positioned.
XSCH8	The feature is not implemented.

Tabela 75. Código de classe XSDAx: RawStore - Data.Generic statement exceptions

SQLSTATE	Texto da mensagem
XSDA1	An attempt was made to access an out of range slot on a page.
XSDA2	An attempt was made to update a deleted record.
XSDA3	Limitation: Record cannot be updated or inserted due to lack of space on the page. Use the parameters derby.storage.pageSize and/or derby.storage.pageReservedSpace to work around this limitation.
XSDA4	An unexpected exception was thrown.
XSDA5	An attempt was made to undelete a record that was not deleted.
XSDA6	Column <column> of row is null, it needs to be set to point to an object.
XSDA7	Restore of a serializable or SQLData object of class <className>, attempted to read more data than was originally stored.

SQLSTATE	Texto da mensagem
XSDA8	Exception during restore of a serializable or SQLData object of class <i><className></i> .
XSDA9	Class not found during restore of a serializable or SQLData object of class <i><className></i> .
XSDAA	Illegal time stamp <i><timestamp></i> , either time stamp is from a different page or of incompatible implementation.
XSDAB	Cannot set a null time stamp.
XSDAC	Attempt to move either rows or pages from one container to another.
XSDAD	Attempt to move zero rows from one page to another.
XSDAE	Can only make a record handle for special record handle IDs.
XSDAF	Using special record handle as if it were a normal record handle.
XSDAG	The allocation nested top transaction cannot open the container
XSDAI	Page <i><page></i> being removed is already locked for deallocation.
XSDAJ	Exception during write of a serializable or SQLData object.
XSDAK	The wrong page was retrieved for record handle <i><value></i> .
XSDAL	Record handle <i><value></i> unexpectedly points to overflow page.

Tabela 76. Código de classe XSDBx: RawStore - Data.Generic transaction exceptions

SQLSTATE	Texto da mensagem
XSDB0	Unexpected exception on in-memory page <i><page></i> .
XSDB1	Unknown page format at page <i><page></i> .
XSDB2	Unknown container format at container <i><containerName></i> : <i><value></i> .
XSDB3	Container information cannot change once written: was <i><value></i> , now <i><value></i> .
XSDB4	Page <i><page></i> is at version <i><value></i> but the log file contains change version <i><value></i> . Either the log records for this page are missing or this page was not written to disk properly.
XSDB5	Log has change record on page <i><page></i> , which is beyond the end of the container.
XSDB6	Another instance of Derby may have already booted the database <i><value></i> .
XSDB7	WARNING: Derby (instance <i><value></i>) is attempting to boot the database <i><value></i> even though Derby (instance <i><value></i>) might still be active. Only one instance of Derby should boot a database at a time. Severe and non-recoverable corruption can result and might have already occurred.
XSDB8	WARNING: Derby (instance <i><value></i>) is attempting to boot the database <i><value></i> even though Derby (instance <i><value></i>) might still be active. Only one instance of Derby should boot a database at a time. Severe and non-recoverable corruption can result if two instances of Derby boot on the same database at the same time. The db2j.database.forceDatabaseLock=true property is set so the database will not boot until the db.lck is no longer present. Normally this file is removed when the first instance of Derby to boot on the database exits. However, it is not removed in some shutdowns. If the file is not removed, you must remove it manually. It is important to verify that no other VM is accessing the database before manually deleting the db.lck file.
XSDB9	Stream container <i><containerName></i> is corrupt.
XSDBA	Attempt to allocate object <i><object></i> failed.

Tabela 77. Código de classe XSDFx: RawStore - Data.Filesystem statement exceptions

SQLSTATE	Texto da mensagem
XSDF0	Could not create file <fileName> as it already exists.
XSDF1	Exception during creation of file <fileName> for container.
XSDF2	Exception during creation of file <fileName> for container, file could not be removed. The exception was: <value>.
XSDF3	Cannot create segment <segmentName>.
XSDF4	Exception during remove of file <fileName> for dropped container, file could not be removed <value>.
XSDF6	Cannot find the allocation page <page>.
XSDF7	Newly created page failed to be latched <value>.
XSDF8	Cannot find page <page> to reuse.
XSDFB	Operation not supported by a read only database.
XSDFD	Different page image read on two I/Os on Page <page>. The first image has an incorrect checksum, the second image has a correct checksum. Page images follows: <value> <value>.
XSDFE	The requested operation failed due to an unexpected exception.

Tabela 78. Código de classe XSDGx: RawStore - Data.Filesystem database exceptions

SQLSTATE	Texto da mensagem
XSDG0	Page <page> could not be read from disk.
XSDG1	Page <page> could not be written to disk, please check if disk is full.
XSDG2	Invalid checksum on Page <page>, expected=<value>, on-disk version=<value>, page dump follows: <value>.
XSDG3	Meta-data for Container <containerName> could not be accessed.
XSDG5	Database is not in create mode when createFinished is called.
XSDG6	Data segment directory not found in <value> backup during restore. Please make sure that backup copy is the right one and it is not corrupted.
XSDG7	Directory <directoryName> could not be removed during restore. Please make sure that permissions are correct.
XSDG8	Unable to copy directory '<directoryName>' to '<value>' during restore. Please make sure that there is enough space and permissions are correct.

Tabela 79. Código de classe XSLAx: RawStore - Log.Generic database exceptions

SQLSTATE	Texto da mensagem
XSLA0	Cannot flush the log file to disk <value>.
XSLA1	Log Record has been sent to the stream, but it cannot be applied to the store (Object <object>). This may cause recovery problems also.
XSLA2	An IOException occurred while accessing the log file. The system will shut down.
XSLA3	The log file is corrupt. The log stream contains invalid data.
XSLA4	Unable to write to the log, most likely because the log is full. It is also possible that the file system is read-only, the disk failed, or another problem occurred with the media.

SQLSTATE	Texto da mensagem
	Delete unnecessary files.
XSLA5	Cannot read log stream for some reason to rollback transaction <value>.
XSLA6	Cannot recover the database.
XSLA7	Cannot redo operation <operation> in the log.
XSLA8	Cannot rollback transaction <value>, trying to compensate <value> operation with <value>.
XSLAA	The store has been marked for shutdown by an earlier exception.
XSLAB	Cannot find log file <logfileName>. Verify that the logDevice property is set with the correct path separator for your platform.
XSLAC	Database at <value> have incompatible format with the current version of software, it may have been created by or upgraded by a later version.
XSLAD	Log record at instance <value> in log file <logfileName> is corrupted. Expected log record length <value>, actual length <value>.
XSLAE	Control file at <value> cannot be written or updated.
XSLAF	A read-only database was created with dirty data buffers.
XSLAH	A read-only database is being updated.
XSLAI	Cannot log the checkpoint log record.
XSLAJ	The log record size <value> exceeds the maximum allowable log file size <maxSize>. An error was encountered in log file <fileName>, position <value>.
XSLAK	Database has exceeded largest log file number <value>.
XSLAL	The log record size <value> exceeds the maximum allowable log file size <maxSize>. An error was encountered in log file <fileName>, position <value>.
XSLAM	Cannot verify database format at <value> due to IOException.
XSLAN	Database at <value> has an incompatible format with the current version of the software. The database was created by or upgraded by version <version>.
XSLAO	Recovery failed. Unexpected problem <value>.
XSLAP	Database at <value> is at version <version>. Beta databases cannot be upgraded.
XSLAQ	Cannot create log file at directory <directory>.
XSLAR	Unable to copy log file '<logfileName>' to '<value>' during restore. Please make sure that there is enough space and permissions are correct.
XSLAS	Log directory <directory> not found in backup during restore. Please make sure that backup copy is the correct one and it is not corrupted.
XSLAT	Log directory <directory> exists. Please make sure specified logDevice location is correct.

Tabela 80. Código de classe XSLBx: RawStore - Log.Generic statement exceptions

SQLSTATE	Texto da mensagem
XSLB1	Log operation <logoperation> encounters error writing itself out to the log stream, this could be caused by an errant log operation or internal log buffer full due to excessively large log operation.
XSLB2	Log operation <logoperation> logging excessive data, it filled up the internal log buffer.
XSLB4	Cannot find truncationLWM <value>.
XSLB5	Illegal truncationLWM instance <value> for truncation point <value>. Legal range is from <value> to <value>.

SQLSTATE	Texto da mensagem
XSLB6	Trying to log a 0 or -ve length log Record.
XSLB8	Trying to reset a scan to <value>, beyond its limit of <value>.
XSLB9	Unable to issue any more changes. Log factory has been stopped.

Tabela 81. Código de classe XSRSx: RawStore - protocol.Interface statement exceptions

SQLSTATE	Texto da mensagem
XSRS0	Cannot freeze the database after it is already frozen.
XSRS1	Cannot backup the database to <value>, which is not a directory.
XSRS4	Error renaming file (during backup) from <value> to <value>.
XSRS5	Error copying file (during backup) from <path> to <path>.
XSRS6	Cannot create backup directory <directoryName>.
XSRS7	Backup caught unexpected exception.
XSRS8	Log device can only be set during database creation time, it cannot be changed after the database is created.
XSRS9	Record <recordName> no longer exists.

Tabela 82. Código de classe XSTA2: XACT_TRANSACTION_ACTIVE

SQLSTATE	Texto da mensagem
XSTA2	A transaction was already active when an attempt was made to activate another transaction.

Tabela 83. Código de classe XSTBx: RawStore - Transactions.Basic system exceptions

SQLSTATE	Texto da mensagem
XSTB0	An exception was thrown during transaction abort.
XSTB2	Unable to log transaction changes, possibly because the database is read-only.
XSTB3	Cannot abort transaction because the log manager is null, probably due to an earlier error.
XSTB5	Creating database with logging disabled encountered unexpected problem.
XSTB6	Cannot substitute a transaction table with another while one is already in use.

Tabela 84. Código de classe XXXXX : No SQLSTATE

SQLSTATE	Texto da mensagem
XXXXX	Normal database session close.

Referência do JDBC

O Derby vem com um driver de JDBC nativo. Isto torna a API do JDBC a única API para trabalhar com os bancos de dados do Derby. O *driver* é um *driver* de protocolo nativo totalmente Java (tipo número quatro, entre os tipos definidos pela Sun).

Este capítulo fornece informações de referência sobre a implementação da API do JDBC do Derby, e documenta sua conformidade com as APIs 2.0 e 3.0 do JDBC.

Consulte o *Guia do Desenvolvedor do Derby* para obter instruções orientadas a tarefa sobre como trabalhar com este driver.

Este *driver* de JDBC implementa a interface padrão de JDBC definida pela Sun. Ao ser chamado a partir de um aplicativo executando na mesma JVM do Derby, o *driver* de JDBC suporta conexões com o banco de dados do Derby no modo incorporado (embedded). Não é requerido nenhum transporte de rede para acessar o banco de dados. No modo cliente/servidor, o aplicativo cliente envia as requisições JDBC para o servidor JDBC através da rede; o servidor, por sua vez, que executa na mesma JVM do Derby, envia requisições para o Derby através do *driver* de JDBC incorporado.

A implementação de JDBC do Derby fornece acesso aos bancos de dados do Derby, e fornece todas as interfaces JDBC requeridas. Os aspectos não implementados do *driver* de JDBC retornam uma *SQLException* com uma mensagem declarando "Funcionalidade não implementada", e *SQLState* igual a XJZZZ. As partes não implementadas são de funcionalidades não suportadas pelo Derby.

Classes, interfaces e métodos java.sql do núcleo do JDBC

Esta seção detalha a implementação no Derby das seguintes classes, interfaces e métodos *java.sql*:

- [java.sql.Driver](#)
- [java.sql.DriverManager.getConnection](#)
- [java.sql.Driver.getPropertyInfo](#)
- [java.sql.Connection](#)
- [java.sql.DatabaseMetaData](#)
- [java.sql.Statement](#)
- [java.sql.PreparedStatement](#)
- [java.sql.CallableStatement](#)
- [java.sql.ResultSet](#)
- [java.sql.ResultSetMetaData](#)
- [java.sql.SQLException](#)
- [java.sql.SQLWarning](#)
- [Mapeamento de java.sql.Types em tipos SQL](#)

java.sql.Driver

A classe que carrega o *driver* de JDBC local do Derby é a classe *org.apache.derby.jdbc.EmbeddedDriver*. Abaixo estão listadas algumas maneiras de criar instâncias desta classe. A classe não deve ser utilizada diretamente através da interface *java.sql.Driver*. Deve ser utilizada a classe *DriverManager* para criar as conexões.

- *Class.forName("org.apache.derby.jdbc.EmbeddedDriver")*

Esta é a maneira recomendada, porque assegura que a classe é carregada em todas as JVMs criando uma instância ao mesmo tempo.

- *new org.apache.derby.jdbc.EmbeddedDriver()*

O mesmo que `Class.forName("org.apache.derby.jdbc.EmbeddedDriver")`, exceto por requerer que a classe seja encontrada quando o código é compilado.

- `Class c = org.apache.derby.jdbc.EmbeddedDriver.class`

Também é o mesmo que

`Class.forName("org.apache.derby.jdbc.EmbeddedDriver")`, exceto por requerer que a classe seja encontrada quando o código é compilado. O campo pseudo-estático `class` resulta na classe nomeada.

- *Definição da propriedade do sistema `jdbc.drivers`*

Para definir uma propriedade do sistema, deve ser alterada a linha de comando da chamada, ou as propriedades do sistema no aplicativo. Não é possível alterar as propriedades do sistema em uma *applet*.

```
java -Djdbc.drivers=org.apache.derby.jdbc.EmbeddedDriver  
classeAplicativo
```

O *driver* real que fica registrado em *DriverManager* para tratar o protocolo *jdbc:derby:* não é a classe `org.apache.derby.jdbc.EmbeddedDriver`, esta classe simplesmente detecta o tipo de *driver* do Derby necessário, e faz com que o *driver* apropriado do Derby seja carregado.

A única maneira suportada para conectar ao sistema Derby através do protocolo *jdbc:derby:* é utilizando *DriverManager* para obter o *driver* (`java.sql.Driver`) ou a conexão (`java.sql.Connection`), através das chamadas de método *getDriver* e *getConnection*.

java.sql.DriverManager.getConnection

Um aplicativo Java utilizando a API do JDBC estabelece a conexão com o banco de dados obtendo um objeto *Connection*. A forma padrão para obter um objeto *Connection* é chamando o método *DriverManager.getConnection*, que recebe uma cadeia de caracteres contendo a URL de conexão com o banco de dados. A URL (*uniform resource locator*) de conexão com o banco de dados JDBC fornece a maneira de identificar o banco de dados.

DriverManager.getConnection pode receber um argumento, além da URL de conexão com o banco de dados, que é um objeto *Properties*. O objeto *Properties* pode ser utilizado para definir atributos da URL de conexão com o banco de dados.

Também podem ser fornecidas cadeias de caracteres representando nomes de usuários e senhas. Quando são fornecidos, o Derby verifica se são válidos para o sistema corrente, se a autenticação de usuário estiver habilitada. Os nomes de usuário são passados para o Derby como identificadores de autorização, utilizados para determinar se o usuário está autorizado a acessar o banco de dados, e para determinar o esquema padrão. Quando a conexão é estabelecida, se não for fornecido um usuário, o Derby define o usuário padrão como *APP*, que o Derby utiliza para dar nome ao esquema padrão. Se for fornecido um usuário, o nome de esquema padrão será o mesmo nome do usuário.

Sintaxe da URL de conexão com banco de dados Derby

A URL de conexão com o banco de dados Derby consiste da URL de conexão com o banco de dados básica, seguida por um subsubprotocolo opcional e atributos opcionais.

Esta seção fornece apenas informações de referência. Para obter uma descrição mais completa, incluindo exemplos, deve ser consultado "Conexão com bancos de dados", no capítulo 1 do *Guia do Desenvolvedor do Derby*.

Sintaxe da URL de conexão com banco de dados para aplicativos com bancos de dados incorporados

Para aplicativos com bancos de dados incorporados, a sintaxe da URL de conexão com banco de dados é

```
jdbc:derby: [subsubprotocolo:] [nomeBancoDados] [;atributos]*
```

- *jdbc:derby:*

No jargão do JDBC, *derby* é o *subprotocolo* para conectar a um banco de dados Derby. O subprotocolo é sempre *derby*, não variando.

- O *subsubprotocolo*:

O *subsubprotocolo*, que normalmente não é especificado, especifica onde o Derby procura pelo banco de dados: em um diretório, em um caminho de classe, ou em um arquivo jar. É usado raramente, geralmente em bancos de dados apenas-de-leitura. O *subsubprotocolo* é um dos seguintes:

- *directory*
- *classpath*: Os bancos de dados são tratados como apenas-de-leitura, e todos os *nomeBancoDados* devem começar com pelo menos uma barra, porque são especificados como "relativos" ao diretório do caminho de classes ou arquivo (Não é necessário especificar *classpath* como sendo o *subsubprotocolo*; está implícito).
- Os bancos de dados *jar* são tratados como bancos de dados apenas-de-leitura.

jar: requer um elemento adicional, imediatamente antes do *nomeBancoDados*:

```
(caminhoArquivo)
```

O *caminhoArquivo* é o caminho para o arquivo *jar* ou *zip* que contém o banco de dados, e inclui o nome do arquivo *jar* ou *zip*.

Para obter exemplos de URLs de conexão com bancos de dados apenas-de-leitura deve ser consultado o *Guia do Desenvolvedor do Derby*.

- *nomeBancoDados*

Deve ser especificado o *nomeBancoDados* para se conectar a um banco de dados existente ou a um banco de dados novo.

Pode ser especificado apenas o nome do banco de dados, ou junto com um caminho absoluto ou relativo. Consulte "Conexões padrão - Conectar a bancos de dados no sistema de arquivos", no capítulo 1 do *Guia do Desenvolvedor do Derby*.

- *atributos*

São especificados zero ou mais atributos da URL de conexão com banco de dados, conforme detalhado em [Atributos da URL de conexão com o banco de dados Derby](#).

Sintaxe SQL adicional

O Derby também suporta a seguinte sintaxe do padrão SQL para obter uma referência à conexão corrente em um procedimento ou método JDBC do lado do servidor:

```
jdbc:default:connection
```

Atributos da URL de conexão com o banco de dados Derby

Pode ser fornecida uma lista de atributos, opcional, para a URL de conexão com o banco de dados. O Derby traduz estes atributos em propriedades, portanto também podem ser definidos atributos no objeto *Properties* passado para *DriverManager.getConnection* (Estes atributos não podem ser definidos como propriedades do sistema, somente em um objeto passado para o método *DriverManager.getConnection*).

Estes atributos são específicos do Derby, sendo listados em [Definição de atributos para a URL de conexão com o banco de dados](#).

Os pares nome/valor do atributo são convertidos em propriedades, e adicionados às propriedades fornecidas na chamada de conexão. Se não for fornecida nenhuma propriedade na chamada da conexão, será criado um conjunto de propriedades contendo apenas as propriedades obtidas da URL de conexão com o banco de dados.

```
import java.util.Properties;

Connection conn = DriverManager.getConnection(
    "jdbc:derby:bancoAmostra;create=true");

-- definição de um atributo no objeto Properties
Properties minhasPropriedades = new Properties();
minhasPropriedades.put("create", "true");
Connection conn = DriverManager.getConnection(
    "jdbc:derby:bancoAmostra", minhasPropriedades);

-- passagem do nome do usuário e da senha
Connection conn = DriverManager.getConnection(
    "jdbc:derby:bancoAmostra", "dba", "senha");
```

Note: Os atributos não são analisados quanto à correção. Se for passado um atributo, ou seu valor correspondente, incorreto, este será simplesmente ignorado (O Derby fornece uma ferramenta para analisar a correção dos atributos. Para obter mais informações deve ser consultado o *Guia das Ferramentas e Utilitários do Derby*).

java.sql.Driver.getPropertyInfo

Para obter um objeto *DriverPropertyInfo*, este deve ser requisitado ao gerenciador de *drivers* do *driver* de JDBC:

```
java.sql.DriverManager.getDriver("jdbc:derby:").
    getPropertyInfo(URL, Prop)
```

Não deve ser requisitado a *org.apache.derby.jdbc.EmbeddedDriver*, que é somente uma classe intermediária que carrega o *driver* real.

Este método deve retornar um objeto *DriverPropertyInfo*. No sistema Derby, este objeto consiste de uma matriz de atributos da URL de conexão com o banco de dados. O atributo mais útil é *databaseName=nomeBancoDados*, significando que o objeto consiste de uma lista dos bancos de dados inicializados no sistema corrente.

Por exemplo, se o sistema Derby possuir os bancos de dados *bancoTurismo* e *bancoVoos* no seu diretório de sistema, estiver configurado para inicializar automaticamente todos os bancos de dados quando o sistema é inicializado, e o usuário

também tiver conectado ao banco de dados `A:/dbs/turismo94`, a matriz retornada por `getPropertyInfo` conterá um objeto correspondente ao atributo `databaseName`. O campo `choices` do objeto `DriverPropertyInfo` conterá uma matriz de três cadeias de caracteres com os valores `bancoTurismo`, `bancoVoos` e `A:/dbs/turismo94`. Deve ser observado que este objeto será retornado apenas se os objetos de conexão propostos não incluírem o nome do banco de dados (em qualquer forma) ou o atributo `shutdown` com o valor `true`.

Para obter mais informações sobre `java.sql.Driver.getPropertyInfo` deve ser consultado "Oferecimento de escolhas de conexão para o usuário", no capítulo 8 do *Guia do Desenvolvedor do Derby*.

java.sql.Connection

O objeto `Connection` do Derby não é coletado pelo coletor de lixo até que todos os outros objetos JDBC criados a partir desta conexão sejam fechados explicitamente, ou sejam coletados pelo coletor de lixo. Uma vez fechada a conexão, não poderá ser feita nenhuma requisição JDBC adicional aos objetos criados a partir da conexão. O objeto `Connection` não deverá ser fechado explicitamente, até que este não seja mais necessário para a execução das instruções.

Um exceção com severidade de sessão, ou maior, faz a conexão ser fechada e todos os demais objetos JDBC associados a esta conexão serem fechados. Exceções com severidade de sistema causam a parada do sistema Derby, o que não apenas fecha a conexão, mas significa também que não poderá ser aberta nenhuma nova conexão na JVM corrente.

java.sql.Connection.setTransactionIsolation

Estão disponíveis os níveis de isolamento de transação `java.sql.Connection.TRANSACTION_SERIALIZABLE`, `java.sql.Connection.TRANSACTION_REPEATABLE_READ`, `java.sql.Connection.TRANSACTION_READ_COMMITTED` e `java.sql.Connection.TRANSACTION_READ_UNCOMMITTED` nos bancos de dados do Derby.

`TRANSACTION_READ_COMMITTED` é o nível de isolamento padrão.

Pelo padrão JDBC, alterar o nível isolamento corrente para a conexão através de `setConnection` efetiva a transação corrente e inicia uma nova transação.

java.sql.Connection.setReadOnly

É suportado `java.sql.Connection.setReadOnly`.

java.sql.Connection.isReadOnly

Se for feita uma conexão com um banco de dados apenas-de-leitura, será retornado o valor apropriado pelo método `isReadOnly` de `DatabaseMetaData`. Por exemplo, as conexões definidas como apenas-de-leitura utilizando o método `setReadOnly`, as conexões para as quais o usuário foi definido com `readOnlyAccess` (através de uma das propriedades do Derby), e as conexões com bancos de dados em mídias apenas-de-leitura, retornam verdade.

Funcionalidades de conexão não suportadas

O Derby não utiliza nomes de catálogo; os métodos *getCatalog* e *setCatalog* resultam na *SQLException* "Funcionalidade não implementada", com *SQLState* igual a XJZZZ.

java.sql.DatabaseMetaData

Esta sessão discute a funcionalidade de *java.sql.DatabaseMetaData* no Derby.

Conjuntos de resultados DatabaseMetaData

Os conjuntos de resultados *DatabaseMetaData* não fecham os conjuntos de resultados das outras instruções, mesmo quanto a auto-efetivação (*auto-commit*) está ativa.

Os conjuntos de resultados *DatabaseMetaData* são fechados se o usuário realizar qualquer outra ação em um objeto JDBC que faça com que ocorra um *COMMIT* automático. Se houver necessidade dos conjuntos de resultados *DatabaseMetaData* ficarem acessíveis enquanto se executa outras ações que causam *COMMIT* automático, deverá ser desativada a auto-efetivação utilizando *setAutoCommit(false)*.

getProcedureColumns

O Derby suporta procedimentos Java. O Derby permite chamar procedimentos Java a partir de instruções SQL. O Derby retorna informações sobre os parâmetros na chamada a *getProcedureColumns*. Se o método Java correspondente estiver sobrecarregado, serão retornadas informações sobre cada assinatura separadamente. O Derby retorna informações sobre todos os procedimentos Java definidos por *CREATE PROCEDURE*.

getProcedureColumns retorna um *ResultSet*. Cada linha descreve um único parâmetro ou o valor retornado.

Parâmetros para getProcedureColumns

A API do JDBC define os seguintes parâmetros para a chamada deste método:

- *catalog*
deve ser sempre utilizado *NULL* para este parâmetro no Derby.
- *schemaPattern*
procedimentos Java possuem esquema.
- *procedureNamePattern*
um objeto *String* representando o padrão de nome do procedimento.
- *columnNamePattern*
um objeto *String* representando o padrão de nome dos nomes dos parâmetros, ou dos nomes do valor retornado. Os procedimentos Java possuem nomes de parâmetro correspondentes aos definidos na instrução *CREATE PROCEDURE*. Deve ser utilizado "%" para encontrar todos os nomes de parâmetros.

Colunas do ResultSet retornado por getProcedureColumns

As colunas do *ResultSet* retornado por *getProcedureColumns* estão descritas na API. Abaixo são mostrados detalhes adicionais para algumas colunas específicas:

- **PROCEDURE_CAT**
sempre "NULL" no Derby
- **PROCEDURE_SCHEM**
esquema do procedimento Java
- **PROCEDURE_NAME**
nome do procedimento
- **COLUMN_NAME**
nome do parâmetro (consulte [columnNamePattern](#))
- **COLUMN_TYPE**
indicação do que a linha descreve. É sempre *DatabaseMetaData.procedureColumnIn* para parâmetros de método, a menos que o parâmetro seja uma matriz. Neste caso é *DatabaseMetaData.procedureColumnInOut*. Retorna sempre *DatabaseMetaData.procedureColumnReturn* para os valores retornados.
- **TYPE_NAME**
Nome específico do Derby para o tipo.
- **NULLABLE**
sempre retorna *DatabaseMetaData.procedureNoNulls* para parâmetros primitivos, e *DatabaseMetaData.procedureNullable* para parâmetros de objeto
- **REMARKS**
uma cadeia de caracteres descrevendo o tipo Java do parâmetro do método
- **METHOD_ID**
coluna específica do Derby.
- **PARAMETER_ID**
coluna específica do Derby.

Funcionalidades de DatabaseMetaData não suportadas

Na versão corrente, o Derby não fornece todas as funcionalidades de *DatabaseMetaData*. As seguintes requisições de JDBC resultam em conjuntos de resultados vazios, no formato requerido pela API do JDBC:

- *getColumnPrivileges*
- *getTablePrivileges*

O Derby não implementa privilégios e, portanto, não possui informações para fornecer para estas chamadas.

getBestRowIdentifier procura pelos identificadores nesta ordem:

- chave primária da tabela
- restrição de unicidade ou índice único na tabela
- todas as colunas da tabela

Devido a esta última escolha, sempre encontra um conjunto de colunas que identifica uma linha. Entretanto, havendo linhas duplicadas na tabela, a utilização de todas as colunas não identifica necessariamente uma única linha da tabela.

java.sql.Statement

O Derby não implementa os seguintes métodos do JDBC 1.2 de *java.sql.Statement*:

- *cancel*
- *setEscapeProcessing*
- *setQueryTimeout*

Objetos ResultSet

No caso de ocorrer um erro quando a instrução SELECT é executada pela primeira vez, isto impede que seja aberto um objeto *ResultSet* para a mesma. O mesmo erro não fecha o *ResultSet* quando ocorre após o *ResultSet* ter sido aberto.

Por exemplo, se ocorrer um erro de divisão-por-zero quando o método *executeQuery* for chamado por *java.sql.Statement* ou *java.sql.PreparedStatement*, será lançada uma exceção e não será retornado nenhum conjunto de resultados, enquanto que, se ocorrer o mesmo erro quando o método *next* for chamado no objeto *ResultSet*, isto não fará com que o conjunto de resultados seja fechado.

Podem ocorrer erros durante a criação do *ResultSet*, se o sistema executar parcialmente a consulta antes da primeira linha ser trazida. Isto pode acontecer em qualquer consulta que utiliza mais de uma tabela, e em consultas que utilizam agregações, GROUP BY, ORDER BY, DISTINCT, INTERSECT, EXCEPT e UNION.

Fechar um *Statement* faz com que todos os objetos *ResultSet* abertos para esta instrução sejam fechados também.

O nome do cursor para o cursor do *ResultSet* pode ser definido antes das instrução ser executada. Entretanto, uma vez executada, o nome do cursor não poderá mais ser alterado.

java.sql.PreparedStatement

O Derby fornece todas as conversões de tipo requeridas pelo JDBC 1.2, e além disso permite o uso de métodos *setXXX* individuais para cada tipo, como se a chamada *setObject(Value, JDBCTypeCode)* tivesse sido feita.

Isto significa que pode ser utilizado *setString* em todos os tipos de dado de destino nativos.

Pode ser utilizado o método *setCursorName* no *PreparedStatement*, antes da requisição de execução, para controlar o nome do cursor usado quando o cursor é criado.

Instruções preparadas e colunas de fluxo

As requisições *setXXXStream* realizam um fluxo de dados entre o aplicativo e o banco de dados.

O JDBC permite que o parâmetro IN seja definido como um fluxo de entrada do Java para passagem de uma grande quantidade de dados em frações menores. Quando a instrução é executada, o *driver* de JDBC faz chamadas repetidas a este fluxo de entrada,

lendo seu conteúdo e transmitindo este conteúdo como dados do parâmetro.

O Derby suporta os três tipos de fluxo fornecidos pelo JDBC 1.2. Estes três fluxos são:

- *setBinaryStream*
para fluxos contendo bytes não interpretados
- *setAsciiStream*
para fluxos contendo caracteres ASCII
- *setUnicodeStream*
para fluxos contendo caracteres Unicode

O JDBC requer que seja especificado o comprimento do fluxo. O objeto de fluxo passado para estes três métodos pode ser tanto um objeto de fluxo Java padrão, quanto uma subclasse própria do usuário que implementa a interface *java.io.InputStream* padrão.

De acordo com o padrão JDBC, os fluxos somente podem ser armazenados em colunas dos tipos de dado mostrados na tabela [Tipos de dado JDBC que permitem fluxo](#). Os fluxos não podem ser armazenados em colunas de outros tipos de dado nativos, ou em tipos de dado definidos pelo usuário.

Tabela 85. Tipos de dado JDBC que permitem fluxo

Valores da coluna	Tipo correspondente	AsciiStream	UnicodeStream	BinaryStream
CLOB	java.sql.Clob	x	x	'
CHAR	'	x	x	'
VARCHAR	'	x	x	'
LONGVARCHAR	'	X	X	'
BINARY	'	x	x	x
BLOB	java.sql.Blob	x	x	x
VARBINARY	'	x	x	x
LONGVARBINARY	'	x	x	X

O X maiúsculo indica o tipo de dado de destino preferido para o tipo de fluxo (Consulte [Mapeamento de java.sql.Types em tipos SQL.](#))

Note: Se o fluxo for armazenado em uma coluna de um tipo que não seja LONG VARCHAR ou LONG VARCHAR FOR BIT DATA, deverá ser possível que todo o fluxo caiba na memória de uma só vez. Os fluxos armazenados em colunas LONG VARCHAR e LONG VARCHAR FOR BIT DATA não possuem esta limitação.

O exemplo a seguir mostra como o usuário pode armazenar *java.io.File* em uma coluna LONG VARCHAR usando fluxo:

```
Statement s = conn.createStatement();
s.executeUpdate("CREATE TABLE minha_tabela (a INT, b LONG VARCHAR)");
conn.commit();
java.io.File arquivo = new java.io.File("derby.txt");
int comprArquivo = (int) arquivo.length();
// primeiro, criar o fluxo de entrada
java.io.InputStream fluxoEntrada = new java.io.FileInputStream(arquivo);
PreparedStatement ps = conn.prepareStatement(
    "INSERT INTO minha_tabela VALUES (?, ?)");
ps.setInt(1, 1);
// definir o valor do parâmetro de entrada como sendo o fluxo de entrada
ps.setAsciiStream(2, fluxoEntrada, comprArquivo);
ps.execute();
```

```
conn.commit();
```

java.sql.CallableStatement

O Derby suporta todos os métodos de *CallableStatement* do JDBC 1.2:

- *getBoolean()*
- *getByte()*
- *getBytes()*
- *getDate()*
- *getDouble()*
- *getFloat()*
- *getInt()*
- *getLong()*
- *getObject()*
- *getShort()*
- *getString()*
- *getTime()*
- *getTimestamp()*
- *registerOutParameter()*
- *wasNull()*

CallableStatement e parâmetros OUT

O Derby suporta todos os parâmetros OUT e instruções CALL que retornam valores, conforme mostrado no exemplo a seguir:

```
CallableStatement cs = conn.prepareCall(
    "? = CALL getDriverType(cast (? as INT))"
    cs.registerOutParameter(1, Types.INTEGER);
    cs.setInt(2, 35);
    cs.executeUpdate();
```

Note: A utilização da instrução CALL com um procedimento que retorna valor somente é suportada através da sintaxe ? =.

Deve ser registrado o tipo de dado de saída do parâmetro antes de executar a chamada.

CallableStatement e parâmetros INOUT

Os parâmetros INOUT são mapeados em uma *matriz* do tipo do parâmetro no Java (O método deve receber uma matriz como seu parâmetro). Isto está em conformidade com as recomendações do padrão SQL.

Dado o seguinte exemplo:

```
CallableStatement call = conn.prepareCall(
    "{CALL dobreMeuInteiro(?)}");
// para os parâmetros INOUT, é boa prática registrar
// o parâmetro de saída antes de definir o valor de entrada
call.registerOutParameter(1, Types.INTEGER);
call.setInt(1, 10);
call.execute();
int retval = call.getInt(1);
```

O método *doubleInt* deve receber uma matriz unidimensional de inteiros. Abaixo está um código fonte de amostra para este método:

```
public static void dobreMeuInteiro(int[] i) {
    i[0] *= 2;
    /* O Derby retorna o primeiro elemento da matriz. */
}
```

Note: O valor retornado *não* é empacotado em uma matriz, mesmo que o parâmetro para o método seja.

Tabela 86. Correspondência de tipo do parâmetro INOUT

Tipo do JDBC	Tipo da matriz para o parâmetro do método	Valor e tipo retornado
BIGINT	long[]	long
BINARY	byte[][]	byte[]
BIT	boolean[]	boolean
DATE	<i>java.sql.Date[]</i>	<i>java.sql.Date</i>
DOUBLE	double[]	double
FLOAT	double[]	double
INTEGER	int[]	int
LONGVARBINARY	byte[][]	byte[]
REAL	float[]	float
SMALLINT	short[]	short
TIME	<i>java.sql.Time[]</i>	<i>java.sql.Time</i>
TIMESTAMP	<i>java.sql.Timestamp[]</i>	<i>java.sql.Timestamp</i>
VARBINARY	byte[][]	byte[]
OTHER	<i>seuTipo[]</i>	<i>seuTipo</i>
JAVA_OBJECT (válido apenas nos ambientes Java2/JDBC 2.0)	<i>seuTipo[]</i>	<i>seuTipo</i>

Deve ser registrado o tipo de dado de saída do parâmetro antes de executar a chamada. Para os parâmetros INOUT, é boa prática registrar o parâmetro de saída antes de definir seu valor de entrada.

java.sql.ResultSet

Uma atualização ou exclusão posicionada, emitida contra um cursor sendo acessado através do objeto *ResultSet*, modifica ou exclui a linha corrente do objeto *ResultSet*.

Alguns protocolos intermediários podem trazer linhas antecipadamente. Isto faz com que as atualizações e exclusões posicionadas operem na linha sobre a qual se encontra o cursor subjacente, e não na linha corrente do *ResultSet*.

O Derby fornece todas as conversões de tipo requeridas pelo JDBC 1.2 para os métodos *getXXX*.

O JDBC não define o tipo de arredondamento a ser utilizado por *ResultSet.getBigDecimal*. O Derby utiliza *java.math.BigDecimal.ROUND_HALF_DOWN*.

ResultSets e colunas de fluxo

Se o objeto subjacente também for da classe *OutputStream*, *getBinaryStream* retornará o objeto diretamente.

Para obter um campo do *ResultSet* utilizando colunas de fluxo, podem ser utilizados os métodos *getXXXStream* se o tipo suportá-los. Para obter uma lista dos tipos que suportam vários fluxos deve ser consultado [Tipos de dado JDBC que permitem fluxo](#) (Consulte também [Mapeamento de java.sql.Types em tipos SQL.](#))

Os dados de uma coluna com tipo de dado suportado podem ser trazidos na forma de fluxo, independentemente de terem sido armazenados como um fluxo.

O exemplo a seguir mostra como o usuário pode trazer uma coluna LONG VARCHAR na forma de fluxo:

```
// trazer os dados como um fluxo
ResultSet rs = s.executeQuery("SELECT b FROM minha_tabela");
while (rs.next()) {
    // utilizar java.io.InputStream para trazer os dados
    java.io.InputStream ip = rs.getAsciiStream(1);
    // processar o fluxo -- esta é apenas uma forma genérica// de se
    mostrar os dados
    int c;
    int tamanhoColuna = 0;
    byte[] buff = new byte[128];
    for (;;) {
        int size = ip.read(buff);
        if (size == -1)
            break;
        tamanhoColuna += size;
        String chunk = new String(buff, 0, size);
        System.out.print(chunk);
    }
}
rs.close();
s.close();
conn.commit();
```

java.sql.ResultSetMetaData

O Derby não verifica a fonte ou possibilidade de atualização das colunas dos *ResultSets*, e portanto sempre retorna as seguintes constantes para os seguintes métodos:

Nome do método	Valor
<i>isDefinitelyWritable</i>	falso
<i>isReadOnly</i>	falso
<i>isWritable</i>	falso

java.sql.SQLException

O Derby fornece valores para as chamadas *getMessage()*, *getSQLState()* e *getErrorCode()* de *SQLException*. Além disso, algumas vezes o Derby retorna várias *SQLException* utilizando a cadeia *nextException*. A primeira exceção é sempre a exceção de maior severidade, com as exceções do padrão SQL-92 precedendo as exceções específicas do Derby. Para obter informações sobre como processar *SQLException* deve ser consultado "Trabalhando com *SQLException* do Derby em aplicativos", no capítulo 5 do *Guia do Desenvolvedor do Derby*.

java.sql.SQLWarning

O Derby pode gerar advertências sob certas circunstâncias. É gerada uma advertência, por exemplo, quando se tenta conectar a um banco de dados com o atributo *create* definido como *true*, e o banco de dados já existe. As agregações, como *sum()*, também

lançam advertências quando são encontrados valores nulos durante o processamento.

Todas as outras mensagens informativas são escritas no arquivo *derby.log* do sistema Derby.

Mapeamento de `java.sql.Types` em tipos SQL

A tabela [Mapeamento de `java.sql.Types` em tipos SQL](#) mostra o mapeamento de `java.sql.Types` em tipos SQL.

Tabela 87. Mapeamento de `java.sql.Types` em tipos SQL

<code>java.sql.Types</code>	Tipos SQL
BIGINT	BIGINT
BINARY	CHAR FOR BIT DATA
BIT ¹	CHAR FOR BIT DATA
BLOB	BLOB (JDBC 2.0 ou mais recente)
CHAR	CHAR
CLOB	CLOB (JDBC 2.0 ou mais recente)
DATE	DATE
DECIMAL	DECIMAL
DOUBLE	DOUBLE PRECISION
FLOAT	DOUBLE PRECISION ²
INTEGER	INTEGER
LONGVARBINARY	LONG VARCHAR FOR BIT DATA
LONGVARCHAR	LONG VARCHAR
NULL	Não é um tipo de dado; é sempre um valor de um determinado tipo
NUMERIC	DECIMAL
REAL	REAL
SMALLINT	SMALLINT
TIME	TIME
TIMESTAMP	TIMESTAMP
VARBINARY	VARCHAR FOR BIT DATA
VARCHAR	VARCHAR

Notas:

1. BIT é válido somente no JDBC 2.0 e ambientes mais recentes.
2. Os valores podem ser passados utilizando o código de tipo FLOAT; entretanto, estes valores são armazenados como valores DOUBLE PRECISION, e portanto sempre possuem o código de tipo DOUBLE quando trazidos.

`java.sql.Blob` e `java.sql.Clob`

No JDBC 2.0, `java.sql.Blob` é o mapeamento para o tipo BLOB (objeto grande binário) do SQL; `java.sql.Clob` é o mapeamento para o tipo CLOB (objeto grande caractere) do SQL.

`java.sql.Blob` e `java.sql.Clob` fornecem um ponteiro lógico para um objeto grande, em vez

de uma cópia completa do objeto. O Derby processa somente uma página de dados na memória por vez. Não há necessidade de processar e armazenar todo o BLOB em memória apenas para acessar alguns poucos bytes iniciais do objeto LOB

O Derby agora suporta os tipos de dado nativos BLOB e CLOB. O Derby também fornece o seguinte suporte para estes tipos de dado:

- **Funcionalidades do BLOB** O Derby suporta a interface *java.sql.Blob*, e os métodos relacionados ao BLOB em *java.sql.PreparedStatement* e *java.sql.ResultSet*. Os métodos *getBlob* de *CallableStatement* não estão implementados.
- **Funcionalidades do CLOB** O Derby suporta a interface *java.sql.Clob*, e os métodos relacionados ao CLOB em *java.sql.PreparedStatement* e *java.sql.ResultSet*. Os métodos *getClob* dos procedimentos de *CallableStatement* não estão implementados.

Para utilizar as funcionalidades *java.sql.Blob* e *java.sql.Clob*:

- Utilizar o tipo BLOB do SQL para armazenamento; Os tipos LONG VARCHAR FOR BIT DATA, BINARY e VARCHAR FOR BIT DATA também funcionam.
- Utilizar o tipo CLOB do SQL para armazenamento; Os tipos LONG VARCHAR, CHAR e VARCHAR também funcionam.
- Utilizar os métodos *getBlob* ou *getClob* da interface *java.sql.ResultSet*, para obter um tratador de *BLOB* ou *CLOB* para os dados subjacentes.
- Não podem ser chamados métodos estáticos (Extensão SQL sobre SQL) em qualquer coluna LOB.

Além disso, a conversão entre cadeias e BLOBs não é recomendada, porque a conversão é dependente da plataforma e do banco de dados.

O Derby utiliza cadeias UNICODE (caracteres com 2 bytes), enquanto outros produtos de banco de dados podem utilizar caracteres ASCII (1 byte por caractere). Se forem utilizadas várias páginas de código, cada caractere poderá necessitar de vários bytes. Poderá ser necessário um tipo BLOB maior para acomodar uma cadeia normal no Derby. Devem ser utilizados tipos CLOB para armazenar cadeias.

Restrições de BLOB, CLOB, (tipos-LOB):

- Os tipos-LOB não podem ser comparados com relação a igualdade (=) e desigualdade (!=, <>).
- Os valores dos tipos-LOB não são ordenáveis, portanto não são suportados os testes <, <=, >, >=.
- Os tipos-LOB não podem ser utilizados em índices, ou como colunas de chave primária.
- Também são proibidas as cláusulas DISTINCT, GROUP BY e ORDER BY nos tipos-LOB.
- Os tipos-LOB não podem estar envolvidos em conversões implícitas como outros tipos base.

O Derby implementa todos os métodos para estas interfaces do JDBC 2.0, exceto pelos métodos *set* e *get* da interface *CallableStatement*.

Recomendações: Como o tempo de vida de *java.sql.Blob* e *java.sql.Clob* termina quando a transação é efetivada, deve ser desativada a auto-efetivação (auto-commit) quando se usa as funcionalidades *java.sql.Blob* e *java.sql.Clob*.

Tabela 88. Métodos java.sql.Blob do JDBC 2.0 suportados

Retorna	Assinatura	Notas de implementação
<i>InputStream</i>	<i>getBinaryStream()</i>	'
<i>byte[]</i>	<i>getBytes(long pos, int length)</i>	São lançadas exceções se <i>pos</i> < 1, se <i>pos</i> for maior que o comprimento, ou se <i>length</i> <= 0.
<i>long</i>	<i>length()</i>	'
<i>long</i>	<i>position(byte[] pattern, long start)</i>	São lançadas exceções se <i>pattern</i> == null, se <i>start</i> < 1, ou se <i>pattern</i> for uma matriz de comprimento 0.
<i>long</i>	<i>position(Blob pattern, long start)</i>	São lançadas exceções se <i>pattern</i> == null, se <i>start</i> < 1, se <i>pattern</i> possuir comprimento 0, ou se for lançada uma exceção ao tentar ler o primeiro byte de <i>pattern</i> .

Tabela 89. Métodos java.sql.Clob do JDBC 2.0 suportados

Retorna	Assinatura	Notas de implementação
<i>InputStream</i>	<i>getAsciiStream()</i>	'
<i>Reader</i>	<i>getCharacterStream()</i>	NÃO SUPORTADO
<i>String</i>	<i>getSubString(long pos, int length)</i>	São lançadas exceções se <i>pos</i> < 1, se <i>pos</i> for maior que o comprimento de <i>Clob</i> , ou se <i>length</i> <= 0.
<i>long</i>	<i>length()</i>	'
<i>long</i>	<i>position(Clob searchstr, long start)</i>	São lançadas exceções se <i>searchStr</i> == null, se <i>start</i> < 1, se <i>searchStr</i> possuir comprimento 0, ou se for lançada uma exceção ao tentar ler o primeiro caractere de <i>searchStr</i> .
<i>long</i>	<i>position(String searchstr, long start)</i>	São lançadas exceções se <i>searchStr</i> == null, se <i>start</i> < 1, ou se o padrão for uma cadeia vazia.

Notas

O mecanismo normal de bloqueio do Derby (bloqueios compartilhados) impede que outras transações atualizem ou excluam um item do banco de dados para o qual o objeto *java.sql.Blob* ou *java.sql.Clob* seja um ponteiro. Entretanto, em alguns casos, o mecanismo de bloqueio instantâneo do Derby pode admitir um período de tempo onde a coluna subjacente a *java.sql.Blob* ou *java.sql.Clob* fique desprotegida. Uma chamada subsequente ao método *getBlob/getClob*, ou *java.sql.Blob/java.sql.Clob*, pode causar um comportamento indefinido.

Além disso, não há nada que impeça a transação que mantém *java.sql.Blob/java.sql.Clob* (ao contrário das outras transações) atualizar a linha subjacente (O mesmo problema existe com os métodos *getXXXStream*). Os aplicativos devem ser programados para impedir atualizações no objeto subjacente, enquanto houver um *java.sql.Blob/java.sql.Clob* aberto para o objeto; se isto não for feito, pode resultar em um comportamento indefinido.

Não deve ser chamado mais de um método *getXXX* de *ResultSet* para a mesma coluna, se um dos métodos for um dos seguintes:

- *getBlob*
- *getClob*
- *getAsciiStream*
- *getBinaryStream*
- *getUnicodeStream*

Estes métodos compartilham o mesmo fluxo subjacente; chamar mais de um destes métodos para a mesma coluna pode resultar em um comportamento indefinido. Por exemplo:

```
ResultSet rs = s.executeQuery("SELECT text FROM CLOBS WHERE i = 1");
while (rs.next()) {
    aclob=rs.getClob(1);
    ip = rs.getAsciiStream(1);
}
```

Os fluxos que tratam colunas longas não são seguros quanto a *thread*. Isto significa que se for decidido abrir várias *threads*, e acessar o fluxo a partir de cada uma das *threads*, o comportamento resultante será indefinido.

OS Clobs não são afetados pelo idioma.

java.sql.Connection

Tabela 90. Métodos de conexão do JDBC 2.0 suportados

Retorna	Assinatura
<i>Statement</i>	<i>createStatement(int resultSetType, int resultSetConcurrency)</i>
<i>PreparedStatement</i>	<i>prepareStatement(String sql, int resultSetType, int resultSetConcurrency)</i>
<i>CallableStatement</i>	<i>prepareCall(String sql, int resultSetType, int resultSetConcurrency)</i>

Notas da implementação

ResultSet.TYPE_FORWARD_ONLY e *ResultSet.TYPE_SCROLL_INSENSITIVE* são os únicos tipos de rolagem suportados. Se for requisitado *TYPE_SCROLL_SENSITIVE*, o Derby emitirá uma *SQLWarning* e retornará um *ResultSetTYPE_SCROLL_INSENSITIVE*.

Estes métodos suportam as simultaneidades *ResultSet.CONCUR_READ_ONLY* e *ResultSet.CONCUR_UPDATABLE*. Entretanto, somente pode ser requisitado um *ResultSet* atualizável que possua o tipo de rolagem *TYPE_FORWARD_ONLY*. Se for requisitado um *ResultSet* atualizável com tipo *SCROLL_SENSITIVE* ou *SCROLL_INSENSITIVE*, o Derby emitirá uma *SQLWarning* e retornará um *ResultSetTYPE_SCROLL_INSENSITIVE_READ_ONLY*.

(Para ver as advertências deve ser utilizado *Connection.getWarnings*)

java.sql.ResultSet

Tabela 91. Métodos de *ResultSet* do JDBC 2.0 suportados

Retorna	Assinatura	Notas da implementação
<i>void</i>	<i>afterLast()</i>	'
<i>void</i>	<i>beforeFirst()</i>	'
<i>void</i>	<i>beforeFirst()</i>	'
<i>void</i>	<i>deleteRow()</i>	Após a linha ser atualizada, o objeto <i>ResultSet</i> será posicionado antes da próxima linha. Antes de executar qualquer método diferente de <i>close</i> no objeto <i>ResultSet</i> , o programa precisará reposicionar o objeto <i>ResultSet</i> utilizando o método <i>next()</i> .

Retorna	Assinatura	Notas da implementação
<i>boolean</i>	<i>first()</i>	'
<i>Blob</i>	<i>getBlob(int columnIndex)</i>	Consulte java.sql.Blob e java.sql.Clob
<i>Blob</i>	<i>getBlob(String columnName)</i>	
<i>Clob</i>	<i>getClob(int columnIndex)</i>	
<i>Clob</i>	<i>getClob(String columnName)</i>	
<i>int</i>	<i>getConcurrency()</i>	Se o objeto <i>Statement</i> possuir a simultaneidade <i>CONCUR_READ_ONLY</i> , então este método retornará <i>ResultSet.CONCUR_READ_ONLY</i> . Mas se o objeto <i>Statement</i> possuir a simultaneidade <i>CONCUR_UPDATABLE</i> , então o valor retornado dependerá do <i>ResultSet</i> da linguagem subjacente ser atualizável ou não. Se o <i>ResultSet</i> da linguagem for atualizável, então <i>getConcurrency()</i> retornará <i>ResultSet.CONCUR_UPDATABLE</i> . Se o <i>ResultSet</i> da linguagem não for atualizável, então <i>getConcurrency()</i> retornará <i>ResultSet.CONCUR_READ_ONLY</i> .
<i>int</i>	<i>getFetchDirection()</i>	'
<i>int</i>	<i>getFetchSize()</i>	Sempre retorna 1.
<i>int</i>	<i>getRow()</i>	'
<i>boolean</i>	<i>isAfterLast()</i>	'
<i>boolean</i>	<i>isBeforeFirst</i>	'
<i>boolean</i>	<i>isFirst()</i>	'
<i>boolean</i>	<i>isLast()</i>	'
<i>boolean</i>	<i>last()</i>	'
<i>boolean</i>	<i>previous()</i>	'
<i>boolean</i>	<i>relative(int rows)</i>	'
<i>void</i>	<i>setFetchDirection(int direction)</i>	'
<i>void</i>	<i>setFetchSize(int rows)</i>	O tamanho de busca igual a 1 é o único tamanho suportado.
<i>void</i>	<i>updateRow()</i>	Após a linha ser atualizada, o objeto <i>ResultSet</i> será posicionado antes da próxima linha. Antes de executar qualquer método diferente de <i>close</i> no objeto <i>ResultSet</i> , o programa precisará reposicionar o objeto <i>ResultSet</i> utilizando o método <i>next()</i> .

Note: Não é requerido que o JDBC esteja com auto-efetivação desabilitada ao se utilizar *ResultSet* atualizável.

No momento, o Derby não suporta o método *insertRow()* em *ResultSet* atualizável.

java.sql.Statement

Tabela 92. Métodos *java.sql.Statement* do JDBC2.0 suportados

Retorna	Assinatura	Notas da implementação
<i>void</i>	<i>addBatch(String sql)</i>	'

Retorna	Assinatura	Notas da implementação
<i>void</i>	<i>clearBatch()</i>	'
<i>int[]</i>	<i>executeBatch()</i>	'
<i>int</i>	<i>getFetchDirection()</i>	Chamar este método não lança uma exceção, mas a chamada é ignorada.
<i>int</i>	<i>getFetchSize()</i>	Chamar este método não lança uma exceção, mas a chamada é ignorada.
<i>int</i>	<i>getMaxFieldSize()</i>	'
<i>void</i>	<i>getMaxRows()</i>	'
<i>void</i>	<i>setEscapeProcessing(boolean enable)</i>	'
<i>void</i>	<i>setFetchDirection(int direction)</i>	Chamar este método não lança uma exceção, mas a chamada é ignorada.
<i>void</i>	<i>setFetchSize(int rows)</i>	Chamar este método não lança uma exceção, mas a chamada é ignorada.
<i>void</i>	<i>setMaxFieldSize(int max)</i>	Sem efeito em <i>Blobs</i> e <i>Clobs</i> .
<i>void</i>	<i>setMaxRows()</i>	'

java.sql.PreparedStatement

Tabela 93. Métodos java.sql.PreparedStatement do JDBC 2.0 suportados

Retorna	Assinatura	Notas da implementação
<i>void</i>	<i>addBatch()</i>	'
<i>ResultSetMetaData</i>	<i>getMetaData()</i>	'
<i>void</i>	<i>setBlob(int i, Blob x)</i>	'
<i>void</i>	<i>setClob(int i, Clob x)</i>	'

java.sql.CallableStatement

Tabela 94. Métodos java.sql.CallableStatements do JDBC 2.0 suportados

Retorna	Assinatura	Notas da implementação
<i>BigDecimal</i>	<i>getBigDecimal</i>	'
<i>Date</i>	<i>getDate(int, Calendar)</i>	'
<i>Time</i>	<i>getTime(int, Calendar)</i>	'
<i>Timestamp</i>	<i>getTimestamp(int, Calendar)</i>	'

java.sql.DatabaseMetaData

O Derby implementa todos os métodos do JDBC 2.0 para esta interface.

java.sql.ResultSetMetaData

O Derby implementa todos os métodos do JDBC 2.0 para esta interface.

java.sql.BatchUpdateException

Lançada quando ocorre problema com atualização em lote.

Pacote JDBC para Connected Device Configuration/Foundation Profile (JSR169)

O Derby suporta a API do JDBC definida para a *Connected Device Configuration/Foundation Profile*, também conhecida por JSR169. As funcionalidades suportadas são um subconjunto da especificação JDBC 3.0. O suporte a JSR169 está limitado ao *driver* incorporado. O Derby não suporta a utilização do *Network Server* sob a JSR169.

Para obter uma conexão sob as especificações JSR169 deve ser utilizada a classe `org.apache.derby.jdbc.EmbeddedSimpleDataSource`. Esta classe é idêntica, em implementação, à classe `org.apache.derby.jdbc.EmbeddedDataSource`. Para obter informações sobre a utilização das propriedades da classe `org.apache.derby.jdbc.EmbeddedDataSource` deve ser consultado o *Guia do Desenvolvedor do Derby*.

Algumas outras funcionalidades a serem notadas relativas à implementação de JSR169 utilizando o Derby:

- Os aplicativos devem obter e definir valores DECIMAL utilizando métodos `getXXX` e `setXXX` alternativos do JDBC, como `getString()` e `setString()`. Qualquer método alternativo que funcione com o tipo DECIMAL no JDBC 2.0 ou 3.0 irá funcionar no JSR169.
- As funções e procedimentos Java que utilizam parâmetros do JDBC do lado servidor, como `CONTAINS SQL`, `READS SQL DATA` e `MODIFIES SQL DATA`, não são suportadas no JSR169.
- A API padrão utilizada para obter a conexão (`jdbc:default:connection`) não é suportada no JSR169. Ocorre um erro em tempo de execução quando a rotina tenta obter a conexão utilizando `jdbc:default:connection`.
- Não são suportadas tabelas de diagnóstico.
- Não são suportados gatilhos.
- Não são suportados bancos de dados criptografados.
- Não é suportado *DriverManager*. Não pode ser utilizado `DriverManager.getConnection()` para obter a conexão.

Funcionalidades apenas do JDBC 3.0

O JDBC 3.0 adiciona algumas funcionalidades à API núcleo. Esta seção documenta as funcionalidades suportadas pelo Derby.

Note: Estas funcionalidades estão presentes apenas no Java 2 versão 1.4 ou mais recente.

As funcionalidades são:

- Novos métodos `DatabaseMetaData`. Consulte [java.sql.DatabaseMetaData](#).
- Acesso a metadados de parâmetro. Consulte [java.sql.ParameterMetaData](#) e [java.sql.PreparedStatement](#).
- Acesso a chaves auto-geradas. Consulte [java.sql.Statement](#) e [java.sql.DatabaseMetaData](#).
- Pontos de salvamento. Consulte [java.sql.Connection](#).
- Cursores HOLD. Consulte [java.sql.DatabaseMetaData](#).

Lista completa:

- [java.sql.Connection](#)
- [java.sql.DatabaseMetaData](#)
- [java.sql.ParameterMetaData](#)
- [java.sql.PreparedStatement](#)
- [java.sql.Savepoint](#)
- [java.sql.Statement](#)

java.sql.Connection

Tabela 95. Métodos de conexão do JDBC 3.0 suportados

Retorna	Assinatura	Notas da implementação
<i>Savepoint</i>	<i>setSavepoint (String name)</i>	Cria um ponto de salvamento na transação corrente com o nome fornecido, e retorna o novo objeto <i>Savepoint</i> que representa o mesmo.
<i>Savepoint</i>	<i>setSavepoint ()</i>	Cria um ponto de salvamento sem nome na transação corrente, e retorna o novo objeto <i>Savepoint</i> que representa o mesmo.
<i>void</i>	<i>releaseSavepoint (Savepoint savepoint)</i>	Remove da transação corrente o objeto <i>Savepoint</i> especificado.
<i>void</i>	<i>rollback(Savepoint savepoint)</i>	Desfaz todas as alterações realizadas após o objeto <i>Savepoint</i> especificado ter sido definido.

java.sql.DatabaseMetaData

Tabela 96. Métodos DatabaseMetaData do JDBC 3.0 suportados

Retorna	Assinatura	Notas da implementação
<i>boolean</i>	<i>supportsSavepoints()</i>	'
<i>int</i>	<i>getDatabaseMajorVersion()</i>	'
<i>int</i>	<i>getDatabaseMinorVersion()</i>	'
<i>int</i>	<i>getJDBCMajorVersion()</i>	'
<i>int</i>	<i>getJDBCMinorVersion()</i>	'
<i>int</i>	<i>getSQLStateType()</i>	'
<i>boolean</i>	<i>supportsNamedParameters()</i>	'
<i>boolean</i>	<i>supportsMultipleOpenResults()</i>	'
<i>boolean</i>	<i>supportsGetGeneratedKeys()</i>	'
<i>boolean</i>	<i>supportsResultSetHoldability(int holdability)</i>	'
<i>int</i>	<i>getResultSetHoldability()</i>	retorna <i>ResultSet.HOLD_CURSORS_OVER_COMMIT</i>

java.sql.ParameterMetaData

ParameterMetaData é uma novidade do JDBC 3.0. Descreve o número, tipo e propriedades dos parâmetros das instruções preparadas. O método *PreparedStatement.getParameterMetaData* retorna o objeto *ParameterMetaData* que descreve os marcadores de parâmetro que aparecem no objeto *PreparedStatement*. Para obter mais informações deve ser consultado [java.sql.PreparedStatement](#).

Os métodos da interface *ParameterMetaData* estão listados abaixo.

Tabela 97. Métodos de ParameterMetaData do JDBC 3.0

Retorna	Assinatura	Notas da implementação
<i>int</i>	<i>getParameterCount()</i>	'
<i>int</i>	<i>isNullable(int param)</i>	'
<i>boolean</i>	<i>isSigned(int param)</i>	'
<i>int</i>	<i>getPrecision(int param)</i>	'
<i>int</i>	<i>getScale(int param)</i>	'
<i>int</i>	<i>getParameterType(int param)</i>	'
<i>String</i>	<i>getParameterTypeName (int param)</i>	'
<i>String</i>	<i>getParameterClassName (int param)</i>	'
<i>int</i>	<i>getParameterMode (int param)</i>	'

java.sql.PreparedStatement

O método *PreparedStatement.getParameterMetaData* retorna um objeto *ParameterMetaData* que descreve os marcadores de parâmetro que aparecem no objeto *PreparedStatement*. Para obter mais informações deve ser consultado [java.sql.ParameterMetaData](#).

Tabela 98. Métodos de PreparedStatement do JDBC 3.0

Retorna	Assinatura	Notas da implementação
<i>ParameterMetaData</i>	<i>getParameterMetaData()</i>	'

java.sql.Savepoint

A interface *Savepoint* é uma novidade do JDBC 3.0. Contém novos métodos para definir, liberar e desfazer transações até os pontos de salvamento designados. Uma vez definido um ponto de salvamento, a transação pode ser desfeita até este ponto de salvamento sem afetar o trabalho precedente. Os pontos de salvamento fornecem um controle das transações com granularidade mais fina, marcando pontos intermediários nas transações.

Definir e desfazer até um ponto de salvamento

A API do JDBC 3.0 adiciona o método *Connection.setSavepoint*, que define um ponto de salvamento na transação corrente. O método *Connection.rollback* foi sobrecarregado para receber o argumento ponto de salvamento. Para obter mais informações deve ser consultado [java.sql.Connection](#).

O código do exemplo abaixo insere uma linha na tabela, define o ponto de salvamento *svpt1*, e insere uma segunda linha. Quando mais tarde a transação é desfeita até *svpt1*, a segunda inserção é desfeita, mas a primeira inserção permanece intacta. Em outras palavras, quando a transação é efetivada somente a linha contendo '1' é adicionada à TABELA1.

```
// A auto-efetivação deve estar desativada
// para utilizar pontos de salvamento.
conn.setAutoCommit(false);
Statement stmt = conn.createStatement();
int rows = stmt.executeUpdate("INSERT INTO TABELA1 (COL1) VALUES(1)");
```



```
// definir o ponto de salvamento
Savepoint svpt1 = conn.setSavepoint("S1");
rows = stmt.executeUpdate("INSERT INTO TABELA1 (COL1) VALUES (2)");
...
conn.rollback(svpt1);
...
conn.commit();
```

Liberação de ponto de salvamento

O método *Connection.releaseSavepoint* recebe como parâmetro um objeto *Savepoint*, e o remove da transação corrente. Uma vez que o ponto de salvamento tenha sido liberado, uma tentativa de referenciá-lo em uma operação de desfazer faz com que seja lançada uma *SQLException*.

Todos os pontos de salvamento criados na transação são automaticamente liberados e tornam-se inválidos, quando a transação é efetivada ou quando toda a transação é desfeita.

Desfazer uma transação até um determinado ponto de salvamento libera automaticamente, e torna inválido, todos os outros pontos de salvamento criados após o ponto de salvamento em questão.

Regras para pontos de salvamento

O ponto de salvamento não pode ser definido dentro de lotes de instruções para habilitar a recuperação parcial. Se for definido um ponto de salvamento em qualquer momento anterior ao método *executeBatch* ser chamado, este será definido antes de qualquer instrução adicionada ao lote ser executada.

O nome do ponto de salvamento poderá ser reutilizado após ter sido liberado explicitamente (emitindo uma liberação do ponto de salvamento), ou implicitamente (emitindo um COMMIT/ROLLBACK para a conexão).

Restrições dos pontos de salvamento

O Derby não suporta pontos de salvamento em gatilhos.

O Derby não libera bloqueios como parte de desfazer até o ponto de salvamento.

Tabela 99. Métodos de Savepoint do JDBC 3.0

Retorna	Assinatura	Notas da implementação
<i>int</i>	<i>getSavepointId()</i>	Lança uma <i>SQLException</i> se for um ponto de salvamento com nome. Obtém o ID gerado para o ponto de salvamento que este objeto <i>Savepoint</i> representa.
<i>String</i>	<i>getSavepointName()</i>	Lança uma <i>SQLException</i> se for um ponto de salvamento sem nome. Obtém o nome do ponto de salvamento que este objeto <i>Savepoint</i> representa.

java.sql.Statement

Tabela 100. Métodos de Statement do JDBC 3.0

Retorna	Assinatura	Notas da implementação
<i>ResultSet</i>	<i>getGeneratedKeys()</i>	'

Chaves autogeradas

A funcionalidade de chaves autogeradas do JDBC 3.0 fornece uma maneira de obter valores de colunas que fazem parte de um índice ou possuem um valor padrão atribuído. O Derby suporta a funcionalidade de auto-incremento, que permite aos usuários criarem colunas em tabelas para as quais o sistema de banco de dados atribui automaticamente valores inteiros incrementais. No JDBC 3.0, pode ser chamado o método *Statement.getGeneratedKeys* para obter o valor de uma coluna como esta. Este método retorna um objeto *ResultSet* com uma coluna para a chave gerada automaticamente. Chamando-se *ResultSet.getMetaData* no objeto *ResultSet* retornado por *getGeneratedKeys* produz um objeto *ResultSetMetaData* semelhante ao retornado por *IDENTITY_VAL_LOCAL*. Quando a instrução é executada ou preparada, deve ser passado para os métodos *execute*, *executeUpdate* ou *prepareStatement* um sinalizador indicando que devem ser retornadas todas as colunas autogeradas.

Abaixo segue um exemplo que retorna um *ResultSet* com valores para as colunas autogeradas da TABELA1:

```
Statement stmt = conn.createStatement();
int rows = stmt.executeUpdate("INSERT INTO TABELA1 (C11, C12) VALUES
(1,1)",
Statement.RETURN_GENERATED_KEYS);
ResultSet rs = stmt.getGeneratedKeys();
```

Para utilizar chaves autogeradas nas instruções INSERT deve ser passado o sinalizador *Statement.RETURN_GENERATED_KEYS* para o método *execute* ou *executeUpdate*. O Derby não suporta a passagem de nomes de colunas ou índices de colunas para os métodos *execute*, *executeUpdate* e *prepareStatement*.

Sintaxe de escape do JDBC

O JDBC fornece uma maneira de suavizar algumas diferenças entre as maneiras como os diferentes fornecedores de SGBDs implementam o SQL. Isto é chamado de sintaxe de escape. A sintaxe de escape sinaliza que o *driver* de JDBC, fornecido por um determinado fornecedor, procura pela sintaxe de escape e a converte em um código que este determinado banco de dados compreende. Isto torna a sintaxe de escape independente do SGBD.

A cláusula de escape do JDBC começa e termina por chaves (*{}*). O caractere abre-chaves sempre seguido por uma palavra chave:

```
{palavra_chave }
```

O Derby suporta as seguintes palavras chave de escape do JDBC, sem diferenciar letras maiúsculas e minúsculas:

- *Palavra chave de escape do JDBC para instruções call*

A palavra chave de escape para uso em *CallableStatement*.

- *Sintaxe de escape do JDBC*

A palavra chave de escape para os formatos de data.

- *Sintaxe de escape do JDBC para cláusulas LIKE*

A palavra chave para especificar caracteres de escape para as cláusulas LIKE.

- *Sintaxe de escape do JDBC para a palavra chave fn*

A palavra chave de escape para as funções escalares.

- [Sintaxe de escape do JDBC para junções externas](#)

A palavra chave de escape para as junções externas.

- [Sintaxe de escape do JDBC para formatos de hora](#)

A palavra chave de escape para formatos de hora.

- [Sintaxe de escape do JDBC para formatos de carimbo do tempo](#)

A palavra chave de escape para formatos de carimbo do tempo.

As demais palavras chave de escape do JDBC não são suportadas.

Note: O Derby retorna o SQL inalterado na chamada *Connection.nativeSQL*, uma vez que a sintaxe de escape é nativa do SQL. Além disso, por este motivo não é necessário chamar *Statement.setEscapeProcessing*.

Palavra chave de escape do JDBC para instruções call

Esta sintaxe é suportada para *java.sql.Statement* e *java.sql.PreparedStatement* adicionalmente a *CallableStatement*.

Sintaxe

```
{call instrução }
```

```
-- Chamar um procedimento Java  
{ call TOURS.BOOK_TOUR(?, ?) }
```

Sintaxe de escape do JDBC

O Derby interpreta a sintaxe de escape do JDBC para datas como como equivalente a sintaxe SQL para datas.

Sintaxe

```
{d 'yyyy-mm-dd' }
```

Equivalente a

```
DATE('yyyy-mm-dd')
```

```
VALUES {d '1999-01-09' }
```

Sintaxe de escape do JDBC para cláusulas LIKE

O sinal de percentagem (%) e sublinhado (__) são metacaracteres nas cláusulas LIKE do SQL. O JDBC fornece uma sintaxe para forçar a interpretação literal destes caracteres. A cláusula JDBC imediatamente após a expressão LIKE permite especificar um caractere de escape:

Sintaxe

```
WHERE ExpressãoCaractere [ NOT ] LIKE  
ExpressãoCaractereComCaractereCuringa
```

```
{ ESCAPE 'CaractereDeEscape' }
```

```
-- descobrir todas as linhas que começam pelo caractere "%"
SELECT a FROM tabA WHERE a LIKE '$%',{escape '$'}
-- descobrir todas as linhas que terminam pelo caractere "_"
SELECT a FROM tabA WHERE a LIKE '%_',{escape '='}
```

Note: Não é permitido utilizar ? como caractere de escape se o padrão do LIKE também for um parâmetro dinâmico (?).

Em alguns idiomas, um único caractere é formado por mais de uma unidade de agrupamento (caractere de 16 bits). O *CaractereDeEscape* utilizado na cláusula de escape deve ser uma unidade de agrupamento única para que funcione adequadamente.

Também pode ser utilizada a sequência de caractere de escape para o LIKE sem utilizar as chaves do JDBC; consulte [Expressão booleana](#).

Sintaxe de escape do JDBC para a palavra chave fn

A palavra chave *fn* permite utilizar várias funções escalares. O nome da função vem imediatamente após a palavra chave *fn*.

Sintaxe

```
{fn chamadaFunção}
```

onde *chamadaFunção* é uma das seguintes funções:

```
concat (ExpressãoCaractere,
ExpressãoCaractere)
```

Cadeia de caracteres formada anexando a segunda cadeia à primeira; se uma das cadeias for nula, o resultado será nulo. {fn concat (*ExpressãoCaractere*, *ExpressãoCaractere*)} é equivalente à sintaxe nativa { *ExpressãoCaractere* || *ExpressãoCaractere* }. Para obter mais detalhes deve ser consultado [Concatenação](#).

```
sqrt (ExpressãoPontoFlutuante)
```

Raiz quadrada do número de ponto flutuante.

{fn sqrt (*ExpressãoPontoFlutuante*)} é equivalente à sintaxe nativa [SQRT](#)(*ExpressãoPontoFlutuante*). Para obter mais detalhes deve ser consultado [SQRT](#).

```
abs (ExpressãoNumérica)
```

Valor absoluto do número. {fn abs(*ExpressãoNumérica*)} é equivalente à sintaxe nativa [ABSOLUTE](#)(*ExpressãoNumérica*). Para obter mais detalhes deve ser consultado [ABS](#) ou [ABSVAL](#).

```
locate(ExpressãoCaractere, ExpressãoCaractere [, posiçãoInício] )
```

Posição na segunda *ExpressãoCaractere* da primeira ocorrência da primeira *ExpressãoCaractere*, procurando a partir do início da segunda expressão caractere, a menos que esteja especificada a *posiçãoInício*. {fn locate(*ExpressãoCaractere*,*ExpressãoCaractere* [, *posiçãoInício*])} é equivalente à sintaxe nativa [LOCATE](#)(*ExpressãoCaractere*, *ExpressãoCaractere* [, *PosiçãoInício*]). Para obter mais detalhes deve ser consultado [LOCATE](#).

```
substring(ExpressãoCaractere, posiçãoInício, comprimento)
```

A cadeia de caracteres formada pela extração de *comprimento* caracteres da *ExpressãoCaractere* a partir da *posiçãoInício*; a posição começa em 1.

```
mod(tipo_inteiro, tipo_inteiro)
```

MOD retorna o resto (módulo) do argumento 1 dividido pelo argumento 2. O resultado será negativo apenas se o argumento 1 for negativo. Para obter mais detalhes deve ser consultado [MOD](#).

Note: Esta sintaxe permite o uso de qualquer função nativa do Derby, e não apenas as listadas nesta seção.

```
TIMESTAMPADD( intervalo, expressãoInteira, expressãoCarimboTempo )
```

Utiliza a função `TIMESTAMPADD` para adicionar o valor de um intervalo a um carimbo do tempo. A função aplica o inteiro ao carimbo do tempo especificado baseado no tipo de intervalo, e retorna a soma como um novo carimbo do tempo. Pode ser realizada subtração no carimbo do tempo utilizando inteiros negativos.

Deve ser observado que `TIMESTAMPADD` é uma função do JDBC com escape, sendo acessível apenas utilizando a sintaxe de função com escape do JDBC.

Para realizar `TIMESTAMPADD` em datas e horas, é necessário convertê-los para carimbo do tempo. As datas são convertidas em carimbos do tempo colocando 00:00:00.0 no campo das horas. As horas são convertidas em carimbos do tempo colocando a data corrente no campo da data.

Deve ser observado que não deve ser colocada uma coluna data/hora dentro de uma função aritmética de carimbo do tempo na cláusula `WHERE`, porque o otimizador não usará nenhum índice na coluna.

```
TIMESTAMPDIFF( intervalo, expressãoCarimboTempo1, expressãoCarimboTempo2 )
```

Utiliza a função `TIMESTAMPDIFF` para descobrir a diferença entre dois valores de carimbo do tempo no intervalo especificado. Por exemplo, a função pode retornar o número de minutos entre dois carimbos do tempo especificados.

Deve ser observado que `TIMESTAMPDIFF` é uma função do JDBC com escape, sendo acessível apenas utilizando a sintaxe de função com escape do JDBC.

Para realizar `TIMESTAMPDIFF` em datas e horas, é necessário convertê-los em carimbo do tempo. As datas são convertidas em carimbos do tempo colocando 00:00:00.0 no campo das horas. As horas são convertidas em carimbos do tempo colocando a data corrente no campo da data.

Deve ser observado que não deve ser colocada uma coluna data/hora dentro de uma função aritmética de carimbo do tempo na cláusula `WHERE`, porque o otimizador não usará nenhum índice para a coluna.

Intervalos válidos para `TIMESTAMPADD` e `TIMESTAMPDIFF`

As funções `TIMESTAMPADD` e `TIMESTAMPDIFF` podem ser utilizadas para realizar aritmética com carimbos do tempo. Estas duas funções utilizam os seguintes intervalos válidos para as operações aritméticas:

- `SQL_TSI_DAY`
- `SQL_TSI_FRAC_SECOND`

- SQL_TSI_HOUR
- SQL_TSI_MINUTE
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_SECOND
- SQL_TSI_WEEK
- SQL_TSI_YEAR

Exemplo de TIMESTAMPADD e TIMESTAMPDIFF

```
{fn TIMESTAMPADD( SQL_TSI_MONTH, 1, CURRENT_TIMESTAMP)}
```

Retorna um valor carimbo do tempo um mês após o carimbo do tempo corrente.

```
{fn TIMESTAMPDIFF(SQL_TSI_WEEK, CURRENT_TIMESTAMP,
timestamp('2001-01-01-12.00.00.000000'))}
```

Retorna o número de semanas entre agora e a hora especificada em 1 de janeiro de 2001.

Sintaxe de escape do JDBC para junções externas

O Derby interpreta a sintaxe de escape do JDBC para as junções externas (e para todas as operações de junção) como equivalente à sintaxe SQL correta para junções externas ou para a operação de junção apropriada.

Para obter informações sobre operações de junção deve ser consultada a [Operação JOIN](#).

Sintaxe

```
{oj Operação JOIN [Operação JOIN ]* }
```

Equivalente a

```
Operação JOIN [Operação JOIN ]*
```

```
-- junção externa
SELECT *
FROM
{oj PAÍSES LEFT OUTER JOIN CIDADES ON
(PAÍSES.COD_ISO_PAÍS=CIDADES.COD_ISO_PAÍS)}
-- outra operação de junção
SELECT *
FROM
{oj PAÍSES JOIN CIDADES ON (PAÍSES.COD_ISO_PAÍS=CIDADES.COD_ISO_PAÍS)}
-- a ExpressãoTabela pode ser a OperaçãoJunção.
-- Portanto podem haver várias operações de
-- junção na cláusula FROM
SELECT E.NUM_EMP, E.ÚLTIMO_NOME, M.NUM_EMP, M.ÚLTIMO_NOME
FROM {oj EMPREGADOS E INNER JOIN DEPARTAMENTOS
INNER JOIN EMPREGADOS M ON NUM_GER = M.NUM_EMP ON E.DEP_TRAB = NUM_DEP};
```

Sintaxe de escape do JDBC para formatos de hora

O Derby interpreta a sintaxe de escape do JDBC para hora como equivalente à sintaxe SQL correta para hora. O Derby também suporta o formato ISO de 8 caracteres (6 dígitos e 2 pontos decimais).

Sintaxe

```
{t 'hh:mm:ss'}
```

Equivalente a

```
TIME 'hh:mm:ss'
```

```
VALUES {t '20:00:03'}
```

Sintaxe de escape do JDBC para formatos de carimbo do tempo

O Derby interpreta a sintaxe de escape do JDBC para carimbos do tempo como equivalente à sintaxe SQL correta para carimbos do tempo. O Derby também suporta o formato ISO de 23 caracteres (17 dígitos, 3 hífen e 3 pontos decimais).

Sintaxe

```
{ts 'yyyy-mm-dd hh:mm:ss.f...'}
```

Equivalente a

```
TIMESTAMP 'yyyy-mm-dd hh:mm:ss.f...'
```

Pode ser omitida a parte fracionária das constantes carimbo do tempo (.f...).

```
VALUES {ts '1999-01-09 20:11:11.123455'}
```

Definição de atributos para a URL de conexão com o banco de dados

O Derby permite que seja fornecida uma lista de atributos para sua URL de conexão com o banco de dados, o que é uma funcionalidade do JDBC.

Os atributos são específicos do Derby.

Normalmente, os atributos são definidos em uma lista separada por ponto-e-vírgula colocada após o protocolo e o subprotocolo. Para obter informações sobre como definir os atributos deve ser consultado [Atributos da URL de conexão com o banco de dados Derby](#). Este capítulo fornece apenas informações de referência.

Note: Os atributos não são analisados quanto à correção. Se for passado um atributo, ou seu valor correspondente, incorreto, este será simplesmente ignorado.

bootPassword=chave

Função

Especifica a chave a ser utilizada para criptografar o novo banco de dados, ou para carregar um banco de dados criptografado existente. Deve ser especificada uma cadeia alfanumérica com comprimento de pelo menos oito caracteres.

Combinação com outros atributos

Ao criar um novo banco de dados, este atributo deve ser combinado com [create=true](#) e [dataEncryption=true](#). Ao carregar um banco de dados criptografado existente, não há necessidade de nenhum outro atributo.

```
-- inicialização de um banco de dados criptografado
jdbc:derby:bancoCriptografado;bootPassword=cseveryPlace

-- criação de um banco de dados criptografado
jdbc:derby:bancoNovo;create=true;dataEncryption=true;
bootPassword=cseveryPlace
```

create=true

Função

Cria o banco de dados padrão do sistema Derby especificado na URL de conexão com o banco de dados, e conecta ao banco de dados. Se o banco de dados não puder ser criado, o erro aparecerá no *log* de erro, e a tentativa de conexão falhará com uma *SQLException* indicando que o banco de dados não pôde ser encontrado.

Se o banco de dados já existir, será criada uma conexão com o banco de dados existente e emitida uma *SQLWarning*.

O JDBC não remove o banco de dados devido à falha na conexão em tempo de criação, se a falha ocorrer depois que ocorre a chamada ao banco de dados. Se a URL de conexão com o banco de dados usar *create=true*, e a conexão não for criada, deverá ser verificado o diretório do banco de dados. Se o diretório existir, deverá ser removido junto com seu conteúdo antes da próxima tentativa de criar o banco de dados.

Combinação com outros atributos

Deve ser especificado o *nomeBancoDados* (após o subprotocolo na URL de conexão com o banco de dados), ou o atributo [databaseName=nomeBancoDados](#) com este

atributo.

Este atributo pode ser combinado com outros atributos. Para especificar o território ao criar o banco de dados é utilizado o atributo [territory=IL_CC](#).

Note: Quando é especificado *create=true* e o banco de dados já existe, é lançada uma *SQLWarning*.

```
jdbc:derby:bancoAmostra;create=true  
jdbc:derby:;databaseName=bancoNovo;create=true;
```

databaseName=nomeBancoDados

Função

Especifica o nome do banco de dados para a conexão; pode ser utilizado em vez de especificar o nome do banco de dados após o subprotocolo.

Por exemplo, estas combinações de URL (e objeto *Properties*) são equivalentes:

- *jdbc:derby:bancoTurismo*
- *jdbc:derby:;databaseName=bancoTurismo*
- *jdbc:derby:(com a propriedade *databaseName* e seu valor *bancoTurismo* definida no objeto *Properties* passado na requisição de conexão)*

Se o nome do banco de dados for especificado tanto na URL (como um subnome) quanto como atributo, o nome do banco de dados definido como subnome terá prioridade. Por exemplo, a seguinte URL de conexão com banco de dados conecta ao *bancoTurismo*:

```
jdbc:derby:bancoTurismo;databaseName=bancoVoos
```

Permitir que o nome do banco de dados seja definido como um atributo, permite que o método *getPropertyInfo* retorne uma lista de escolhas para o nome do banco de dados, baseado no conjunto de bancos de dados conhecidos pelo Derby. Para obter mais informações deve ser consultado [java.sql.Driver.getPropertyInfo](#).

Combinação com outros atributos

Este atributo pode ser combinado com todos os outros atributos.

```
jdbc:derby:;databaseName=bancoNovo;create=true
```

dataEncryption=true

Função

Especifica a criptografia dos dados em disco para o novo banco de dados (Para obter informações sobre criptografia dos dados, deve ser consultado "Criptografia do banco de dados no disco" no *Guia do Desenvolvedor do Derby*.)

Combinação com outros atributos

Deve ser combinado com [create=true](#) e [bootPassword=chave](#). Também pode ser especificado [encryptionProvider=nomeProvedor](#) e [encryptionAlgorithm=algoritmo](#).

```
jdbc:derby:bancoCriptografado;create=true;dataEncryption=true;  
bootPassword=cLo4u922sc23aPe
```


encryptionProvider=nomeProvedor

Função

Especifica o provedor para criptografia dos dados (Para obter informações sobre criptografia de dados, deve ser consultado "Criptografia do banco de dados no disco" no *Guia do Desenvolvedor do Derby*.)

Se este atributo não for especificado, o provedor de criptografia padrão será o incluído na JVM sendo utilizada.

Combinação com outros atributos

Deve ser combinado com [create=true](#), [bootPassword=chave](#) e [dataEncryption=true](#). Também pode ser especificado [encryptionAlgorithm=algoritmo](#).

```
jdbc:derby:bancoCriptografado;create=true;dataEncryption=true;
encryptionProvider=com.sun.crypto.provider.SunJCE;
encryptionAlgorithm=DESede/CBC/NoPadding;
bootPassword=cLo4u922sc23aPe
```

encryptionAlgorithm=algoritmo

Função

Especifica o algoritmo para criptografar os dados.

O algoritmo é especificado de acordo com as convenções da linguagem Java:

algoritmo/modo/preenchimento

O único tipo de preenchimento (*padding*) permitido no Derby é *NoPadding*.

Se não for especificado um algoritmo de criptografia, será utilizado o valor padrão *DES/CBC/NoPadding*.

(Para obter informações sobre criptografia de dados, deve ser consultado "Criptografia do banco de dados no disco" no capítulo 7 do *Guia do Desenvolvedor do Derby*.)

Combinação com outros atributos

Deve ser combinado com [create=true](#), [bootPassword=chave](#), [dataEncryption=true](#) e [encryptionProvider=nomeProvedor](#).

```
jdbc:derby:bancoCriptografado;create=true;dataEncryption=true;
encryptionProvider=com.sun.crypto.provider.SunJCE;
encryptionAlgorithm=DESede/CBC/NoPadding;
bootPassword=cLo4u922sc23aPe
```

Note: Se o provedor especificado não suportar o algoritmo especificado, o Derby lançará uma exceção.

territory=ll_CC

Função

Ao criar ou atualizar um banco de dados, este atributo é utilizado para associar um território diferente do padrão ao banco de dados. Definir o atributo *territory* sobrepõe o território padrão do sistema para o banco de dados. O território padrão do sistema é encontrado utilizando *java.util.Locale.getDefault()*.

O território é especificado na forma *LL_CC*, onde *LL* é o código de duas letras do idioma, e *CC* é o código de duas letras do país.

O código do idioma é formado por um par de letras minúsculas, em conformidade com o padrão ISO-639.

Tabela 101. Amostra de códigos de idioma

Código do idioma	Descrição
de	Alemão
en	Inglês
es	Espanhol
ja	Japonês
pt	Português

Para consultar a lista completa dos códigos ISO-639 deve ser vista a página <http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>.

O código do país é formado por um par de letras maiúsculas, em conformidade com o padrão ISO-3166.

Tabela 102. Amostra de códigos de país

Código do país	Descrição
BR	Brasil
DE	Alemanha
US	Estados Unidos
ES	Espanha
MX	México
JP	Japão

Pode ser obtida uma cópia do padrão ISO-3166 na página http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html.

Combinação com outros atributos

O atributo *territory* é utilizado apenas ao criar o banco de dados.

No exemplo a seguir, o novo banco de dados possui um território com o idioma Português e a nacionalidade Brasileira.

```
jdbc:derby:bancoBrasil;create=true;territory=pt_BR
```

logDevice=caminhoDiretorioLog

Função

O *caminhoDiretorioLog* especifica o caminho para o diretório onde fica armazenado o *log* do banco de dados durante a criação ou restauração do banco de dados. Mesmo que seja especificado como um caminho relativo, o *caminhoDiretorioLog* é armazenado internamente como um caminho absoluto.

Combinação com outros atributos

Utilizado junto com *create=true*, *createFrom*, *restoreFrom* ou *rollForwardRecoveryFrom*.

```
jdbc:derby:bancoNovo;create=true;logDevice=d:/logBancoNovo
```

password=senhaUsuario

Função

Uma senha válida para o nome de usuário fornecido.

Combinação com outros atributos

Utilizado junto com *user=nomeUsuário*.

```
jdbc:derby:bancoTurismo;user=jack;password=upTheHill
```

rollForwardRecoveryFrom=Caminho

Função

O atributo *rollForwardRecoveryFrom=Caminho* pode ser especificado na URL de conexão em tempo de inicialização, para restaurar o banco de dados utilizando uma cópia de segurança, e realizar a recuperação de rolagem para frente (*rollforward*) utilizando os *logs* arquivados e ativos.

Para restaurar um banco de dados utilizando uma recuperação de rolagem para frente, deve haver uma cópia de segurança do banco de dados, todos os *logs* arquivados desde então, e os arquivos de *log* ativos. Todos os arquivos de *log* devem estar no diretório de *log* do banco de dados.

Após o banco ser restaurado a partir da cópia de segurança completa, são refeitas as transações dos *logs* arquivados *online* e dos *logs* ativos.

Combinação com outros atributos

Este atributo não pode ser combinado com *createFrom*, *restoreFrom* ou *create*.

```
URL: jdbc:derby:wombat;rollForwardRecoveryFrom=d:/backup/wombat
```

createFrom=Caminho

Função

O atributo *createFrom=Caminho* pode ser especificado na URL de conexão em tempo de inicialização, para criar o banco de dados utilizando uma cópia de segurança completa em um local especificado. Se existir em *derby.system.home* um banco de dados com o mesmo nome, ocorrerá um erro e o banco de dados existente será deixado intacto. Se não existir um banco de dados com o mesmo nome no local *derby.system.home* corrente, todo o banco de dados será copiado do local da cópia de segurança para o local *derby.system.home*, e inicializado.

Os arquivos de *log* são copiados para o local padrão. O atributo *logDevice* pode ser utilizado junto com *createFrom=Caminho* para armazenar os *logs* em um local diferente. Com *createFrom=Caminho* não há necessidade de copiar os arquivos de *log* individualmente para o diretório de *log*.

Combinação com outros atributos

Este atributo não deve ser combinado com *rollforwardrecoveryFrom*, *restoreFrom* ou *create*.

```
URL: jdbc:derby:wombat;createFrom=d:/backup/wombat
```

restoreFrom=Caminho

Função

O atributo *restoreFrom=Caminho* pode ser especificado na URL de conexão em tempo de inicialização, para restaurar o banco de dados utilizando uma cópia de segurança completa no local especificado. Se existir no local `derby.system.home` um banco de dados com o mesmo nome, o banco de dados será excluído por completo, copiado do local da cópia de segurança, e depois reinicializado.

Os arquivos de *log* são copiados para o mesmo local onde estavam quando a cópia de segurança foi pega. O atributo *logDevice* pode ser utilizado junto com *restoreFrom=Caminho*, para armazenar os *logs* em um local diferente.

Combinação com outros atributos

Este atributo não pode ser combinado com *createFrom*, *rollforwardrecoveryFrom* ou *create*.

```
URL: jdbc:derby:wombat;restoreFrom=d:/backup/wombat
```

shutdown=true

Função

Pára o banco de dados especificado, se for especificado o atributo *databaseName* (A reconexão com o banco de dados reinicializa o banco de dados.)

Pára todo o sistema Derby se, e somente se, não for especificado o atributo *databaseName*.

Quando está sendo parado um único banco de dados, permite que o Derby realize um ponto de controle (checkpoint) final no banco de dados.

Quando está sendo parado todo o sistema, permite que o Derby realize um ponto de controle final em todos os bancos de dados do sistema, remova o registro do *driver* de JDBC, e termine na JVM antes que esta encerre a execução. Uma parada bem sucedida sempre resulta em uma *SQLException* indicando que o Derby parou, e não há mais conexão. Uma vez que o Derby tenha parado, este pode ser reinicializado recarregando o driver. Para obter detalhes sobre a reinicialização do Derby, deve ser consultado "Parar o sistema", no capítulo 1 do *Guia do Desenvolvedor do Derby*.

Criar um *ponto de controle* significa escrever todos os dados e informações de transação no disco, para não ser necessário realizar uma recuperação na próxima conexão.

Utilizado para parar todo o sistema apenas quando está incorporado ao aplicativo.

Note: Toda requisição a *DriverManager* com o atributo *shutdown=true* lança uma exceção.

```
-- parar todo o sistema
jdbc:derby:;shutdown=true
-- parar bancoVendas
```

```
jdbc:derby:bancoVendas;shutdown=true
```

user=nomeUsuário

Especifica um nome de usuário válido para o sistema, especificado com uma senha. Quando a autenticação de usuário está habilitada, é requerido um nome de usuário e senha válidos.

Combinação com outros atributos

Deve ser utilizado junto com *password=senhaUsuario*.

A seguinte URL de conexão com o banco de dados conecta o usuário *jill* ao banco de dados *bancoTurismo*:

```
jdbc:derby:bancoTurismo;user=jill;password=toFetchAPail
```

(nenhum atributo)

Se não for especificado nenhum atributo, o *nomeBancoDados* deverá ser especificado na URL de conexão.

O Derby estabelece uma conexão com o banco de dados existente com este nome no diretório de sistema corrente. Se o banco de dados não existir, a tentativa de conexão retornará uma *SQLException* indicando que o banco de dados não pôde ser encontrado.

```
jdbc:derby:meuBanco
```

Conformidade com o J2EE: API de transação Java e extensões javax.sql

A J2EE, ou *Java 2 Platform, Enterprise Edition*, é um padrão para desenvolvimento de aplicativos empresariais, baseado em componentes reutilizáveis em um ambiente de várias camadas (*multi-tier*). Além das funcionalidades da *Java 2 Platform, Standard Edition (J2SE)*, a J2EE adiciona suporte a *Enterprise Java Beans (EJBs)*, *Java Server Pages (JSPs)*, *Servlets*, *XML*, e muito mais. A arquitetura J2EE é utilizada para unir tecnologias existentes e aplicativos empresariais em um único ambiente gerenciável.

O Derby é um componente em conformidade com a J2EE em um sistema J2EE distribuído. Como tal, é parte de um sistema maior que inclui, entre outras coisas, um servidor JNDI, um módulo de *pool* de conexões, um gerenciador de transações, um gerenciador de recursos, e aplicativos dos usuários. Dentro deste sistema, o Derby pode servir como gerenciador de recursos.

Para obter mais informações sobre a J2EE, deve ser vista a especificação da J2EE disponível na página <http://java.sun.com/j2ee/docs.html>.

Para se qualificar como um gerenciador de recursos em um sistema J2EE, é requerido pela J2EE suporte a estas áreas básicas:

- Suporte JNDI.

Permite aos aplicativos que fazem a chamada registrar nomes para os bancos de dados, e acessá-los através destes nomes em vez de através de URL de conexão com o banco de dados. A implementação de uma das extensões do JDBC, [javax.sql.DataSource](#), fornece este suporte.

- Pool de conexões.

Mecanismo pelo qual um servidor de *pool* de conexões mantém um conjunto de conexões abertas para o gerenciador de recursos (Derby). Um usuário requisitando uma conexão pode obtê-la entre as conexões disponíveis no *pool*. Este *pool* de conexões é útil em ambientes cliente/servidor, porque o estabelecimento de uma conexão é relativamente dispendioso. Em um ambiente incorporado as conexões são muito menos dispendiosas, tornando a melhoria de desempenho do *pool* de conexões insignificante. A implementação de duas extensões do JDBC, [javax.sql.ConnectionPoolDataSource](#) e [javax.sql.PooledConnection](#), fornecem este suporte.

- Suporte ao XA.

O XA é um dos diversos padrões para gerenciamento de transações distribuídas. É baseado na efetivação de duas fases (*two-phase commit*). As interfaces [javax.sql.XAxxx](#), junto com o pacote [java.transaction.xa](#), são uma implementação abstrata do XA. Para obter mais informações sobre o XA deve ser consultado *X/Open CAE Specification-Distributed Transaction Processing: The XA Specification*, X/Open Document No. XO/CAE/91/300 ou ISBN 1 872630 24 3. A implementação da API do JTA, as interfaces do pacote [java.transaction.xa](#) ([javax.sql.XAConnection](#), [javax.sql.XADataSource](#), [javax.transaction.xa.XAResource](#), [javax.transaction.xa.XAException](#)), fornecem este suporte.

Exceto pelas interfaces JDBC do núcleo, estas interfaces não são visíveis pelos aplicativos do usuário final; em vez disso, são utilizadas no sistema somente pelos outros componentes do lado servidor.

Note: Para obter informações sobre as classes que implementam estas interfaces, e como utilizar o Derby como gerenciador de recursos, deve ser visto o capítulo 6,

"Utilização do Derby como gerenciador de recursos J2EE", no *Guia do Desenvolvedor do Derby*.

JVM e bibliotecas para as funcionalidades do J2EE

Estas funcionalidades requerem o seguinte:

- Ambiente *Java 2 Platform, Standard Edition* v 1.2 (J2SE), ou superior
- Bibliotecas *javax.sql*

Os binários de extensão do padrão JDBC 2.0 estão disponíveis na página <http://java.sun.com/products/jdbc/download.html>. Estas bibliotecas fazem parte do ambiente padrão da *Java 2 Platform, Standard Edition* v 1.4, ou mais recente.

- Bibliotecas *javax.transaction.xa*

Estas bibliotecas fazem parte do ambiente padrão da *Java 2 Platform, Standard Edition* v 1.4, ou mais recente.

Para as bibliotecas do JTA deve ser aberta a página

<http://java.sun.com/products/jta/>, e baixada a especificação e os arquivos de ajuda *javadoc* para as interfaces do JTA.

- O Derby (*derby.jar*)

A API do JTA

A API do JTA é composta por duas interfaces e uma exceção que fazem parte do pacote *java.transaction.xa*. O Derby implementa esta API em sua totalidade.

- *javax.transaction.xa.XAResource*
- *javax.transaction.xa.Xid*
- *javax.transaction.xa.XAException*

Notas sobre o comportamento do produto

Transações globais recuperadas

A utilização da chamada *XAResource.prepare* faz com que uma transação global entre no estado preparado, permitindo que esta seja persistente. Normalmente, o estado preparado é apenas um estado de transição antes do resultado da transação ser determinado. Entretanto, se o sistema cair, a recuperação coloca as transações que estão no estado preparado de volta neste estado, e aguarda instruções do gerenciador de transações.

XAConnections, nomes de usuários e senhas

Se o usuário abrir uma *XAConnection* com nome de usuário e senha, a transação criada não poderá ser anexada a uma *XAConnection* aberta com um nome de usuário e senha diferente. A transação criada por *XAConnection* sem nome de usuário e senha pode ser anexada a qualquer *XAConnection*.

Entretanto, o nome de usuário e senha das transações globais recuperadas são perdidos; qualquer *XAConnection* pode efetivar ou desfazer esta transação duvidosa.

Note: Quando for requerido suporte ao XA em um ambiente remoto (cliente/servidor), deve ser utilizada a interface *DataSource* do XA do *driver* cliente da rede (*org.apache.derby.jdbc.ClientXADataSource*).

javax.sql: Extensões JDBC

Esta seção documenta as extensões do JDBC que o Derby implementa para conformidade com o J2EE (Para obter mais detalhes sobre estas extensões deve ser

consultada a página

<http://java.sun.com/products/jdbc/jdbc20.stdxext.javadoc/javax/sql/package-summary.html>).

- *javax.sql.DataSource*

A implementação de *DataSource* no Derby significa que este suporta JNDI; como um gerenciador de recursos, permite que o banco de dados receba nome e seja registrado no servidor JNDI. Permite ao aplicativo que faz a chamada acessar o banco de dados pelo nome (como uma fonte de dados), em vez de através de uma URL de conexão com o banco de dados.

- *javax.sql.ConnectionPoolDataSource* e *javax.sql.PooledConnection*

O estabelecimento de uma conexão com o banco de dados pode ser uma operação relativamente dispendiosa em ambientes cliente/servidor. O estabelecimento da conexão uma vez, seguida pela utilização da mesma conexão por várias requisições, pode melhorar muito o desempenho do banco de dados.

A implementação do Derby de *ConnectionPoolDataSource* e *PooledConnection* permite a um servidor de *pool* de conexões manter um conjunto de conexões com o gerenciador de recursos (Derby). Em um ambiente incorporado as conexões são muito menos dispendiosas, tornando o *pool* de conexões desnecessário.

- *javax.sql.XAConnection*

Uma *XAConnection* produz um *XAResource*, e durante o seu tempo de vida várias *Connection*. Permite transações distribuídas.

- *javax.sql.XADataSource*

Um *XADataSource* é simplesmente um *ConnectionPoolDataSource* que produz *XAConnection*.

Além disso, o Derby fornece três métodos para *XADataSource*, *DataSource* e *ConnectionPoolDataSource*. O Derby suporta algumas propriedades de fonte de dados adicionais:

- *setCreateDatabase(String create)*

Define a propriedade para criar o banco de dados na próxima conexão. O argumento cadeia de caracteres deve ser "create".

- *setShutdownDatabase(String shutdown)*

Define a propriedade para parar o banco de dados. Para o banco de dados na próxima conexão. O argumento cadeia de caracteres deve ser "shutdown".

Note: Estas propriedades devem ser definidas antes de obter a conexão.

API do Derby

O Derby fornece arquivos HTML *Javadoc* das classes e interfaces da API no subdiretório *javadoc*.

Este apêndice fornece uma breve visão geral da API. O Derby não fornece Javadoc para os pacotes *java.sql*, a API principal para trabalhar com o Derby, porque está incluída na API do JDBC. Para obter informações sobre a implementação do JDBC no Derby deve ser consultada a [Referência do JDBC](#).

Este documento divide as classes e interfaces da API em várias categorias. As ferramentas e utilitários autônomos (*stand-alone*) são classes Java que se sustentam por si próprias, e são chamadas em uma janela de comandos. As classes de implementação do JDBC são APIs do JDBC padrão, não sendo chamadas na linha de comando. Em vez disso, são chamadas apenas em um contexto específico a partir de outro aplicativo.

Ferramentas e utilitários autônomos

Estas classes estão nos pacotes *org.apache.derby.tools*.

- *org.apache.derby.tools.ij*

Uma ferramenta de script SQL que pode ser executada como um aplicativo incorporado ou cliente/servidor. Consulte o *Guia das Ferramentas e Utilitários do Derby*.

- *org.apache.derby.tools.sysinfo*

Um utilitário de linha de comando, do lado servidor, que mostra informações sobre a JVM e o produto Derby. Consulte o *Guia das Ferramentas e Utilitários do Derby*.

- *org.apache.derby.tools.dblook*

Um utilitário para ver todas as partes da Linguagem de Definição de Dados (DDL) para um determinado banco de dados. Consulte o *Guia das Ferramentas e Utilitários do Derby*.

Classes de implementação do JDBC

Driver de JDBC

Este é o *driver* de JDBC para o Derby:

- *org.apache.derby.jdbc.EmbeddedDriver*

Utilizado para inicializar um *driver* de JDBC nativo incorporado e o sistema Derby.

- *org.apache.derby.jdbc.ClientDriver*

Utilizado para conectar ao Network Server do Derby no modo cliente-servidor.

Consulte o *Guia do Desenvolvedor do Derby*.

Classes de fonte de dados

Estas classes são todas relacionadas à implementação do Derby de *javax.sql.DataSource* e APIs relacionadas. Para obter mais informações deve ser consultado o *Guia do Desenvolvedor do Derby*.

Ambiente incorporado:

- *org.apache.derby.jdbc.EmbeddedDataSource*

- *org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource*
- *org.apache.derby.jdbc.EmbeddedXADataSource*

Ambiente cliente-servidor:

- *org.apache.derby.jdbc.ClientDataSource*
- *org.apache.derby.jdbc.ClientConnectionPoolDataSource*
- *org.apache.derby.jdbc.ClientXADataSource*

Utilitários e interfaces diversas

- *org.apache.derby.authentication.UserAuthenticator*
- Uma interface fornecida pelo Derby. As classes que fornecem esquema de autenticação de usuário alternativa devem implementar esta interface. Para obter informações sobre usuários deve ser consultado "Trabalhando com autenticação de usuário", no capítulo 7 do *Guia do Desenvolvedor do Derby*.

Territórios suportados

A seguir está a relação dos territórios suportados:

Território	Definição de território no Derby (derby.territory)
Japonês	ja_JP
Coreano	ko_KR
Chinês (Tradicional)	zh_TW
Chinês (Simplificado)	zh_CN
Francês	fr
Alemão	de_DE
Italiano	it
Espanhol	es
Português (Brasil)	pt_BR

Limitações do Derby

Esta seção lista as limitações associadas ao Derby.

Limitações do comprimento de identificador

Tabela 103. Limitações do comprimento de identificador

A tabela a seguir lista as limitações dos comprimentos dos identificadores no Derby.

Identificador	Número máximo de caracteres permitido
nome de restrição	128
nome de correlação	128
nome de cursor	128
nome de coluna de fonte de dados	128
nome de índice de fonte de dados	128
nome de fonte de dados	128
nome de ponto de salvamento	128
nome de esquema	128
nome de coluna não qualificado	128
nome de função não qualificado	128
nome de índice não qualificado	128
nome de procedimento não qualificado	128
nome de parâmetro	128
nome de gatilho não qualificado	128
nome de tabela, nome de visão, nome de procedimento armazenado não qualificados	128

Limitações numéricas

Tabela 104. Limitações numéricas

A tabela a seguir contém as limitações dos valores numéricos no Derby.

Valor	Límite
Menor INTEGER	-2.147.483.648
Maior INTEGER	2.147.483.647
Menor BIGINT	-9.223.372.036.854.775.808
Maior BIGINT	9.223.372.036.854.775.807
Menor SMALLINT	-32.768
Maior SMALLINT	32.767
Maior precisão decimal	31.255
Menor DOUBLE	-1.79769E+308
Maior DOUBLE	1.79769E+308
Menor DOUBLE positivo	2.225E-307
Maior DOUBLE negativo	-2.225E-307

Valor	Limite
Menor REAL	-3.402E+38
Maior REAL	3.402E+38
Menor REAL positivo	1.175E-37
Maior REAL negativo	-1.175E-37

Limitações das cadeias

Tabela 105. Limitações das cadeias

A tabela a seguir contém as limitações dos valores cadeia no Derby.

Valor	Limite máximo
Comprimento de CHAR	254 caracteres
Comprimento de VARCHAR	32.672 caracteres
Comprimento de LONG VARCHAR	32.700 caracteres
Comprimento de CLOB	2.147.483.647 caracteres
Comprimento de BLOB	2.147.483.647 caracteres
Comprimento de constante caractere	32.672
Comprimento de cadeia de caracteres concatenada	2.147.483.647
Comprimento de cadeia binária concatenada	2.147.483.647
Número de dígitos da constante hexadecimal	16.336
Comprimento da constante com valor DOUBLE	30 caracteres

Limitações de DATE, TIME e TIMESTAMP

Tabela 106. Limitações de DATE, TIME e TIMESTAMP

A tabela a seguir lista as limitações dos valores data, hora e carimbo do tempo no Derby.

Valor	Limite
Menor valor DATE	0001-01-01
Maior valor DATE	9999-12-31
Menor valor TIME	00:00:00
Maior valor TIME	24:00:00
Menor valor TIMESTAMP	0001-01-01-00.00.00.000000
Maior valor TIMESTAMP	9999-12-31-24.00.00.000000

Limitações dos valores do gerenciador de banco de dados

Tabela 107. Limitações do gerenciador de banco de dados

A tabela a seguir lista as limitações de vários valores do gerenciador de banco de dados no Derby.

Valor	Limite
Número máximo de colunas em uma tabela	1.012
Número máximo de colunas em uma visão	5.000

Valor	Limite
Número máximo de parâmetros em um procedimento armazenado	90
Número máximo de índices em uma tabela	32.767 ou a capacidade de armazenamento
Número máximo de tabelas referenciadas em uma instrução SQL ou em uma visão	capacidade de armazenamento
Número máximo de elementos na lista de seleção	1.012
Número máximo de predicados na cláusula WHERE e HAVING	capacidade de armazenamento
Número máximo de colunas na cláusula GROUP BY	32.677
Número máximo de colunas na cláusula ORDER BY	1.012
Número máximo de instruções preparadas	capacidade de armazenamento
Número máximo de cursores declarados em um programa	capacidade de armazenamento
Número máximo de cursores abertos ao mesmo tempo	capacidade de armazenamento
Número máximo de restrições na tabela	capacidade de armazenamento
Nível máximo de aninhamento da subconsulta	capacidade de armazenamento
Número máximo de subconsultas em uma única instrução	capacidade de armazenamento
Número máximo de linhas alteradas em uma unidade de trabalho	capacidade de armazenamento
Número máximo de constantes em uma instrução	capacidade de armazenamento
Profundidade máxima de gatilhos em cascata	16

Marcas registradas

Os seguintes termos são marcas registradas de outras empresas e foram utilizados em pelo menos um dos documentos da biblioteca de documentação do Apache Derby:

Cloudscape, DB2, DB2 Universal Database, DRDA e IBM são marcas registradas da International Business Machines Corporation nos EUA, outros países, ou ambos.

Microsoft, Windows, Windows NT e o logotipo do Windows são marcas registradas da Microsoft Corporation nos EUA, outros países, ou ambos.

Java e todas as marcas registradas baseadas no Java são marcas registradas da Sun Microsystems, Inc. nos EUA, outros países, ou ambos.

UNIX é uma marca registrada do *The Open Group* nos EUA e outros países.

Outros nomes de empresas, produtos ou serviços podem ser marcas registradas ou marcas de serviços de terceiros.