12. Stored Procedures

Uma store procedure é um tipo especial de função definida pelo usuário. O DBMaker suporta store procedure escritos em três linguagens: ESQL/C, Java e SQL. Uma vez criado, um store procedure é armazenado no banco de dados como um objeto executável. Isso permite que o mecanismo de banco de dados evite a compilação e otimização repetidas de SQL, aumentando o desempenho de tarefas frequentemente repetidas. A store procedure é executado tanto como um comando em SQL interativo quanto invocado em programas de aplicação, ações de Triggres ou outros store procedure

Store procedure atendem a uma ampla gama de objetivos, incluindo a melhoria do desempenho do banco de dados, simplificação da escrita de aplicações e limitação ou monitoramento do acesso ao banco de dados.

Como as store procedure são armazenados no banco de dados como objetos executáveis, eles estão disponíveis para todas as aplicações que rodam no banco de dados. Várias aplicações podem usar a mesma store procedure o que reduz o tempo de desenvolvimento de aplicações.

12.1 ESQL Stored Procedures

Uma store procedure ESQL é um programa ESQL/C. Store procedure podem realizar qualquer função que um aplicativo C pode, incluindo a chamada de outras funções C e chamadas de sistema. Portanto, um compilador C é necessário para escrever store procedure.

Um programa ESQL/C para uma store procedure consiste em uma instrução CREATE PROCEDURE, uma seção de declaração, se necessário, e a seção de código. Se o seu programa não utilizar variáveis de host, a seção de declaração pode ser omitida.

Exemplo: Para criar uma store procedure chamado sp_Aphone com um parâmetro de entrada, um parâmetro de saída e um valor de retorno (status):

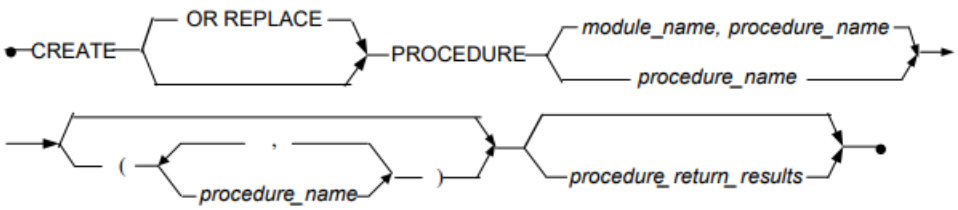
```
EXEC SQL CREATE PROCEDURE sp_Aphone (CHAR(13) name, CHAR(13) phone
OUTPUT)
RETURNS STATUS;
{
 EXEC SQL BEGIN CODE SECTION;
 EXEC SQL SELECT PHONE FROM TBL WHERE NAME = :name INTO :phone;
```

```
 EXEC SQL RETURNS STATUS SQLCODE;
 EXEC SQL END CODE SECTION;
}
```

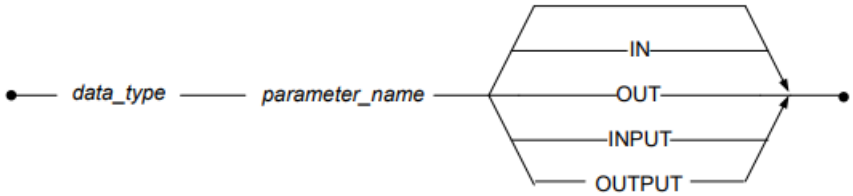
A estrutura deste programa será explicada nas seções seguintes.

Create Procedure Syntax

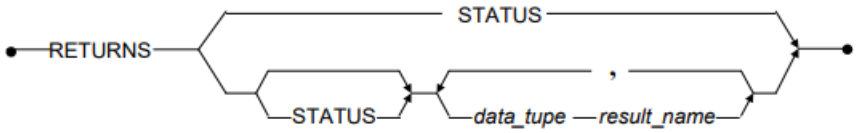
No cabeçalho de uma definição de procedure está a instrução CREATE PROCEDURE. A sintaxe para a instrução CREATE PROCEDURE é:



<*procedure_parameters*> clause



<*procedure_return_result* > clause



OR REPLACE é usado para recriar a store procedure se ela já existir, ou seja, os usuários podem usar essa cláusula para alterar a definição de uma store procedure existente.

NOTA: *Caso a substituição do procedimento falhe, o procedimento armazenado original já terá sido removido.*

Exemplo:

Estes exemplos mostram a sintaxe da instrução CREATE PROCEDURE.

```
dmSQL> CREATE PROCEDURE sp_example1 (INTEGER n IN) RETURNS STATUS;
dmSQL> CREATE PROCEDURE sp_example2 (INTEGER n1 IN, INTEGER n2
OUTPUT) RETURNS
CHAR(12) nm;
dmSQL> CREATE PROCEDURE sp_example3(CHAR(10) par1 OUTPUT, SMALLINT
par2)
 RETURNS STATUS, TIMESTAMP ret1, FLOAT ret2;
dmSQL> CREATE OR REPLACE PROCEDURE sp_example4(CHAR(10) par1 OUTPUT,
SMALLINT
par2) RETURNS STATUS, TIMESTAMP ret1, FLOAT ret2;
```

Em uma instrução CREATE PROCEDURE, o nome da procedure e o nome e tipo de quaisquer parâmetros de entrada/saída devem ser fornecidos.

Using Parameters

Se parâmetros forem necessários, uma lista de pares type-name para os parâmetros deve ser fornecida entre parênteses. Os atributos dos parâmetros IN/OUT (ou INPUT/OUTPUT) devem ser colocados após cada par de type-name. Se não houver um atributo de parâmetro, o padrão será IN. Parâmetros de entrada são usados para passar um valor para uma procedure. No exemplo 1, há um parâmetro de entrada chamado name. Sempre que a procedure for executada, um valor para o parâmetro de entrada deve ser fornecido.

Parâmetros de saída são usados para obter um único resultado, e não um conjunto de resultados, após a execução da procedure. No exemplo 1, a store procedure **sp_Aphone** tem um parâmetro de saída chamado phone. O parâmetro de saída deve ter um buffer atribuído para receber o resultado. Após a execução da procedure, o número de telefone para o nome informado pode ser recuperado do buffer.

A lista de resultados é necessária para uma store procedure recuperar um conjunto de resultados de tuplas do banco de dados. Se a procedure não retornar resultados selecionados, não há necessidade da lista de resultados. A palavra-chave RETURNS é usada para iniciar a lista de resultados, que é uma lista de pares type-name.

A palavra-chave STATUS indica que um valor inteiro deve ser retornado após a execução da store procedure.

Exemplo:

Para executar uma procedure com um parâmetro de entrada e um valor:

```
EXEC SQL CREATE PROCEDURE sp_Select (FLOAT ifl) RETURNS STATUS,
 FLOAT fl,
 DOUBLE db;
{
 EXEC SQL BEGIN CODE SECTION;
 EXEC SQL RETURNS STATUS SQLCODE;
 EXEC SQL RETURNS SELECT fl, db FROM t8 WHERE fl < :ifl into :fl,
:db;
 EXEC SQL END CODE SECTION;
}
```

No DBMaker, os tipos de dados suportados para parâmetros de entrada e saída são: INTEGER, SMALLINT, CHAR(), DATE, TIME, TIMESTAMP, FLOAT, DOUBLE e REAL.

Return Select Statement

Uma procedure pode retornar um conjunto de resultados usando o mecanismo de variáveis de host para passar informações ao usuário que executa a store procedure. No código da store procedure, use a palavra-chave RETURNS para instruir o pré-processador a gerar uma variável de host relacionada ao código C. A palavra-chave RETURNS deve preceder a instrução SELECT que produz o conjunto de resultados. Exemplo:

```
EXEC SQL CREATE PROCEDURE sp_Allphone RETURNS CHAR(12) name,
CHAR(12) phone;
{
 EXEC SQL BEGIN CODE SECTION;
 EXEC SQL RETURNS SELECT NAME, PHONE FROM TBL INTO :name, :phone;
 EXEC SQL END CODE SECTION;
}
```

Há duas ocorrências de RETURNS neste exemplo: uma na instrução CREATE PROCEDURE e outra na instrução SELECT, formando um par. Se um conjunto de resultados for retornado, declare os parâmetros de saída com RETURNS na instrução CREATE PROCEDURE e coloque a palavra-chave RETURNS na instrução SELECT.

Module Names

Quando um usuário cria uma store procedure, o DBMaker usa o owner name e o nome da procedure como o nome padrão da biblioteca de vínculo dinâmico. O usuário pode

chamar ou remover suas proprias procedure usando apenas o nome da procedure. Qualquer usuário pode chamar a procedure de outro usuário usando o nome completo da store procedure: owner.procedure_name.

Um usuário também pode especificar um nome de módulo na sintaxe CREATE PROCEDURE para alterar o nome padrão da biblioteca de vínculo dinâmico. Se um nome de módulo for especificado na sintaxe CREATE PROCEDURE, os usuários precisarão chamar ou remover a procedure usando o nome completo: module_name.owner.procedure_name, mesmo que o usuário o tenha criado.

Variable Declaration

As variáveis de host em store procedure são declaradas da mesma forma que em ESQL/C. A seção de declaração em um store procedure deve ser colocada antes da seção de código, ao contrário dos programas ESQL/C. Variáveis C podem ser colocadas antes ou depois da seção de declaração, mas precisam estar antes da seção de código.

Code Section

Todas as instruções devem estar na seção de código, exceto a declaração de variáveis. Quaisquer instruções que não sejam de declaração colocadas antes da seção de código podem causar problemas, resultando em erros de compilação ou resultados incorretos. Instruções após a seção de CÓDIGO não serão executadas.

Configuration Settings for Stored Procedures

Quando uma store procedure é criada, uma biblioteca de vínculo dinâmico correspondente é construída e armazenada no servidor. Por padrão, o arquivo da biblioteca é colocado no diretório de trabalho do servidor DBMaker. O administrador do banco de dados pode definir um caminho preferencial para armazenar os arquivos de biblioteca das store procedure usando a palavra-chave de configuração DB_SPDir.

A palavra-chave DB_SPLog é usada pelos usuários clientes para definir o diretório em que preferem receber arquivos de mensagens de erro e arquivos de log de rastreamento, transmitidos pelo servidor de banco de dados durante a criação ou execução da store procedure.

Exemplo 1:

Para definir o caminho padrão dos arquivos da biblioteca de vínculo dinâmico para store procedure como /usr1/dbmaker/data/SP, adicione a seguinte linha no arquivo dmconfig.ini:

```
DB_SPDir=/usr1/dbmaker/data/SP
```

Exemplo 2: Para definir o diretório de arquivos de log da store procedure como c:\usr\jerry\data\SP, adicione a seguinte linha no arquivo dmconfig.ini:

```
DB_SPLog=c:\\usr\\jerry\\data\\SP
```

Creating a New Stored Procedure from File

Primeiro, escreva a store procedure e salve-a em um arquivo. Em seguida, use ferramentas do DBMaker, como dmSQL ou JDBATool, para inserir este nova store procedure no banco de dados.

Exemplo:

Para criar uma procedure utilizando vários arquivos:

```
dmSQL> CREATE PROCEDURE FROM 'proc1.ec';
dmSQL> CREATE PROCEDURE FROM '.\\esql\\sp\\proc2.ec';
dmSQL> CREATE PROCEDURE FROM 'c:\\users\\jerry\\sp\\proc3.ec';
```

Os exemplos anteriores mostram como criar store procedure usando dmSQL.

Alternativamente, use o JDBATool:

1. Clique no objeto **Stored Procedure** na Árvore.
2. Clique no botão Criar. A janela de Introdução do assistente Criar Stored Procedure será exibida.
3. Importe uma store procedure selecionando o botão Importar.
4. Selecionar Importar abre a janela Abrir. Arquivos podem ser importados de qualquer fonte, incluindo o diretório SPDIR de outros bancos de dados no servidor ou unidades de rede. Selecione o arquivo desejado digitando o caminho no campo Nome do arquivo ou navegue pela árvore de diretórios até encontrar o caminho correto.

NOTA: *Arquivos importados devem estar no formato ASCII e conter código C++.*

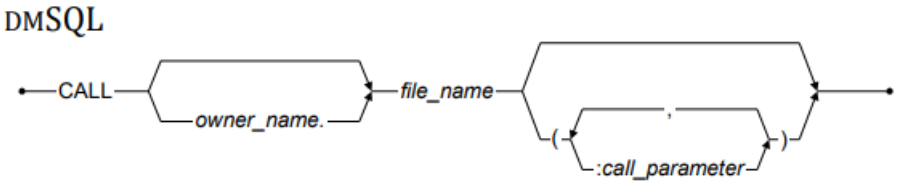
5. Selecione Abrir para abrir o arquivo.

6. A janela Criar Stored Procedure reaparecerá se o arquivo importado contiver texto formatado corretamente (ASCII). Selecione Salvar Como para armazenar a store procedure em outro local ou selecione OK para compilar e armazenar a store procedure no banco de dados.

NOTA: *Se houver erros ao criar uma* store procedure*, eles serão exibidos na parte inferior da janela.*

Executing Stored Procedures

Você pode invocar uma store procedure no dmSQL, em um programa C (ODBC ou ESQL), em outra store procedure ou utilizando uma ação de trigger.



Exemplo 1:

Para executar uma store procedure no dmSQL:

Declaração do procedimento sp_proc1:

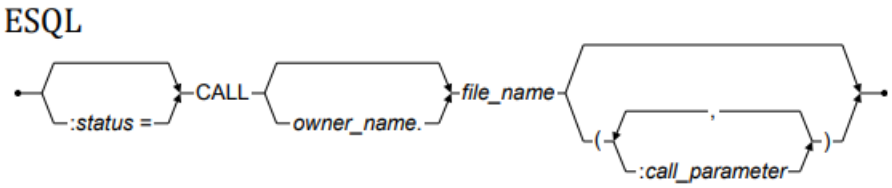
```
dmSQL> CREATE PROCEDURE sp_proc1(CHAR(12) p1, CHAR(12) p2 OUTPUT)
RETURNS INTEGER
r1;
{
EXEC SQL BEGIN CODE SECTION;
EXEC SQL SELECT c2 FROM t1 WHERE c1 = :p1 INTO :p2;
EXEC SQL RETURNS SELECT c1 FROM t2 INTO :r1;
EXEC SQL END CODE SECTION;
}
```

Exemplo 2:

Se a procedure retornar um conjunto de resultados, o dmSQL gerencia automaticamente os parâmetros de saída e exibe o conjunto de resultados na tela. O conjunto de resultados aparece na tela como se você tivesse digitado uma instrução SELECT usando o dmSQL:

```
dmSQL>                    CALL                    sp_Aphone('jeff');
STATUS                                                          PHONE
```

```
========                                              =================
 0                                                         408-255-2689
dmSQL>                          CALL                      sp_Allphone;
 NAME                                                             PHONE
========                                              =================
Jerry                                                      02-775-8615
Jeff                                                      408-255-2689
```



Exemplo:

Para executar uma store procedure em ESQL:

```
EXEC       SQL       :s       =       CALL       sp_example1       (3);
EXEC    SQL    CALL    SYSADM.sp_example2    (5,    :n2)    INTO    :nm;
EXEC SQL :s = CALL jack.sp_example3 (:par1, 7) INTO :ret1, :ret2;
```

A sintaxe utilizada em um programa ESQL é semelhante à do dmSQL. Use variáveis de host para receber o status, os parâmetros de saída e os valores do conjunto de resultados.

EXECUTING NESTED STORED PROCEDURES

Invocar uma store procedure aninhado em ESQL/C ocorre exatamente da mesma forma que em qualquer programa ESQL/C. Há uma exceção: programas ESQL regulares não podem usar a palavra-chave RETURNS, mas store procedure podem usá-la ao invocar outra store procedure.

Suponha que a store procedure sp_Allphone retorne um conjunto de resultados com múltiplas tuplas. Um programa ESQL regular deve usar um cursor para buscar as tuplas ao invocar esse procedimento, conforme mostrado na seção anterior. Outro procedimento armazenado, sp_another, pode usar o mesmo método para buscar tuplas, examinar os dados e retornar o conjunto de resultados completo da store procedure chamado diretamente para o chamador, a partir da store procedure atual.

Exemplo: Para chamar uma instrução de dentro da store procedure sp_another:

```
EXEC SQL RETURNS CALL sp_Allphone INTO :oName, :oPhone;
```

Quando uma store procedure retorna o conjunto de resultados de outra store procedure, o chamador deve ter exatamente a mesma lista de resultados ou as primeiras n colunas de resultado da procedure chamado.

EXECUTING STORED PROCEDURES IN ODBC PROGRAMS

Você também pode chamar uma store procedure em um programa ODBC vinculando parâmetros para a procedure e utilizando colunas para retornar o conjunto de resultados. Em um programa ODBC, é possível vincular colunas parciais do conjunto de resultados. Após a execução da procedure, os parâmetros de saída são retornados nas variáveis de host. Use um fetch, assim como em um comando SELECT, para obter o conjunto de resultados.

Exemplo 1:

Declaração do procedimento sp_proc1:

```
dmSQL> CREATE PROCEDURE sp_proc1(CHAR(12) p1, CHAR(12) p2 OUTPUT)
RETURNS INTEGER
r1;
{
EXEC SQL BEGIN CODE SECTION;
EXEC SQL SELECT c2 FROM t1 WHERE c1 = :p1 INTO :p2;
EXEC SQL RETURNS SELECT c1 FROM t2 INTO :r1;
EXEC SQL END CODE SECTION;
}
```

Exemplo 2:

Programa ODBC que chama sp_proc1:

```
SQLPrepare(cmdp,(UCHAR*)"call sp_proc1(?, ?)", SQL_NTS);
strcpy(bpname, "12345");
SQLBindParameter(cmdp, 1, SQL_PARAM_INPUT_OUTPUT, SQL_C_CHAR,
SQL_CHAR,
 20, 0, &p1, 20, NULL);
SQLBindParameter(cmdp, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_CHAR,
SQL_CHAR,
 20, 0, &p2, 20, NULL);
SQLBindCol(cmdp, 1, SQL_C_LONG, &i, sizeof(long), NULL);
```

```
SQLExecute(cmdp); /* get p2 */
while ((rc=SQLFetch(cmdp))!=SQL_NO_DATA_FOUND) /* fetch result set
*/
```

TRACING STORED PROCEDURE EXECUTION

O DBMaker oferece funcionalidade de rastreamento para ajudar os usuários a rastrear a execução de store procedure para depuração.

Exemplo:

Usando o comando TRACE:

```
EXEC SQL TRACE ON; // Start TRACE
EXEC SQL SELECT c1 FROM t1 INTO :var1;
EXEC SQL TRACE ("var1 = %d\\n", var1); // TRACE the value of var1
EXEC SQL TRACE OFF; // END OF TRACE
```

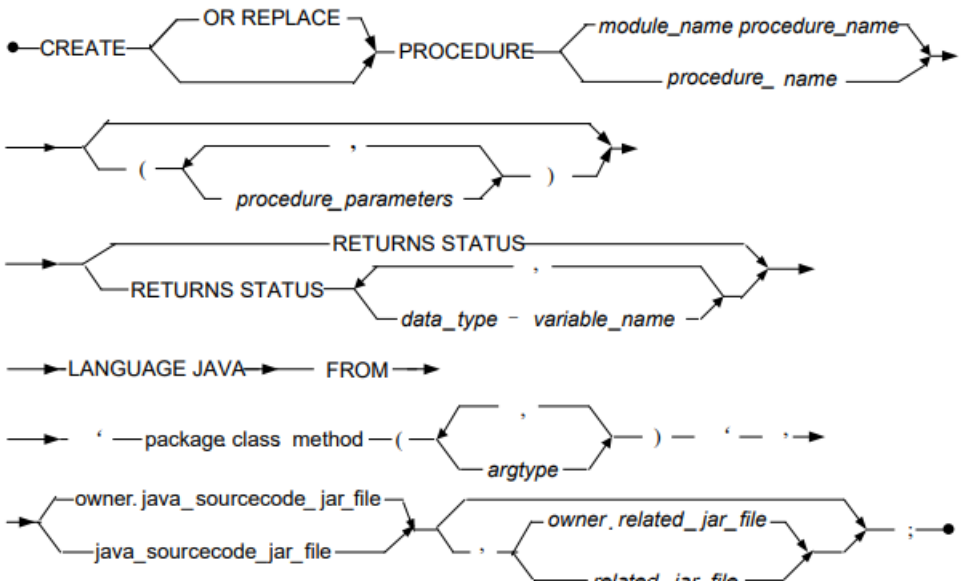
Ative e use a função TRACE para colocar variáveis para rastreamento e imprimir mensagens. Após a execução da store procedure, todas as informações de rastreamento serão escritas em um arquivo chamado **_spusr.log** no diretório especificado pela palavra-chave DB_SPLog encontrada no arquivo dmconfig.ini na máquina cliente.

12.2 JAVA Stored Procedures

Existem vários cenários em que faz sentido usar store procedure em Java. Dada a popularidade do Java hoje em dia, é bastante possível que os membros de uma equipe de desenvolvimento sejam mais proficientes em Java do que em ESQL. O DBMaker suporta store procedure em Java para permitir que programadores Java codifiquem em sua linguagem preferida. Para desenvolvedores experientes em ESQL, usar Java permite aproveitar a linguagem Java para estender a funcionalidade das aplicações de banco de dados. Usar Java também permite reutilizar código existente e aumentar dramaticamente a produtividade.

O DBMaker suporta um método Java como uma store procedure em Java. O DBMaker substitui o argumento URL de DriverManager.getConnection(url, …) por jdbc:default:connection. Você pode usar todas as classes Java para implementar um método Java como uma store procedure em Java, incluindo todas as classes JDBC. O DBMaker tem uma nova sintaxe para registrar arquivos .jar relacionados e criar, executar e excluir uma store procedure em Java.

A sintaxe para a instrução CREATE JAVA PROCEDURE é:



Exemplo 1:

O DBMaker suporta o método Java xx.yy.AA(String) em /home/usr/mary/sp/aa.jar, mas /home/usr/john/sp/bb.jar ainda é necessário para executar o método AA(String):

Por exemplo: Para registrar os arquivos aa.jar e bb.jar no banco de dados, primeiro, você deve mover o arquivo aa.jar do usuário sysadm, localizado em /home/usr/mary/sp/aa.jar, para /home/sysadm/spdir/jar/SYSADM/.

O DB_SPDir é definido no dmconfig.ini como "/home/sysadm/spdir", portanto, o usuário deve mover o arquivo jar para o diretório /DB_SPDir/jar/uppercase_username/ antes de adicionar o arquivo jar ao banco de dados.

Exemplo 2:

Você tem um método Java xx.yy.AA(String) em aa.jar, mas ainda precisa de bb.jar para executar o método AA(String):

Para registrar os arquivos aa.jar e bb.jar no banco de dados:

NOTA: *Os usuários devem mover os arquivos jar físicos, ou seja, aa.jar e bb.jar, para o diretório DB_SPDir/jar/uppercase_username/ antes de executar a sintaxe "add jarfile". DB_SPDir é a palavra-chave que define o diretório de store procedure do DBMaker no dmconfig.ini.* Para criar um procedimento armazenado em Java JSP_AA(char(10) par1) usando o método Java AA(String), siga os passos abaixo:

```
dmSQL> CREATE PROCEDURE JSP_AA (char(10) par1) RETURNS STATUS
LANGUAGE JAVA FROM
'xx.yy.AA(String)', xaa, xbb;
```

Para executar uma store procedure em Java JSP_AA:

```
dmSQL>       EXECUTE       PROCEDURE       JSP_AA       ('aaaaaa');
dmSQL>              CALL              JSP_AA              ('bbb');
```

Para excluir uma store procedure em Java JSP_AA:

```
dmSQL>              DROP              PROCEDURE              JSP_AA;
```

Para remover os arquivos aa.jar e bb.jar do banco de dados:

```
dmSQL>               REMOVE               JARFILE               xaa;
dmSQL>               REMOVE               JARFILE               xbb;
```

Sintaxe relacionada para CREATE PROCEDURE em Java

Para registrar um arquivo .jar no banco de dados:

```
ADD       JARFILE       logical_file_name       physical_jarfile_name
```

Para remover um arquivo .jar do banco de dados:

```
REMOVE                   JARFILE                   logical_file_name
```

Para criar uma store procedure em Java com o comando CREATE PROCEDURE:

```
CREATE [OR REPLACE] PROCEDURE procedure-name
[(procedure-parameter [, procedure-parameter ...])]
 {
 [RETURNS STATUS]
 | [RETURNS [STATUS,] procedure-result [,procedure-result ...]]
 }
LANGUAGE JAVA FROM 'package.class.method([ ' argtype[,argtype…] '
])',
```

```
[owner.]java-sourcecode-jar-file [, owner.related-jar-file]
```

OR REPLACE é usado para recriar a procedure se ela já existir, ou seja, os usuários podem usar essa cláusula para alterar a definição de uma procedure existente.

Para executar uma stored procedure em Java:

```
EXECUTE                PROCEDURE                [owner.]procedure-name
EXECUTE                   PROC                  [owner.]procedure-name
[?  =]  CALL  [owner.]procedure-name  [(procedure-parameter-value  [,
procedureparameter-value                                        ...])]
```

Para excluir uma ****stored procedure **em Java:**

```
DROP                  PROCEDURE                  [owner.]procedure-name
```

Para realizar o load/unload de uma stored procedure:

```
UNLOAD PROCEDURE/PORC FROM [owner_patt.]proc_patt TO unload_filename
LOAD           PROCEDURE/PORC           FROM           unload_filename
```

Para realizar o load/unload de um arquivo .jar:

```
UNLOAD  JARFILE  FROM  [owner_patt.]jarfile_patt  TO  unload_filename
LOAD             JARFILE             FROM             unload_filename
```

NOTA: *Os usuários devem mover os arquivos .jar físicos para o novo diretório DB_SPDir/jar/uppercase_username/ antes de carregar os arquivos .jar.*

Executing Java Stored Procedures

Explicações usando store procedure em Java são apresentadas com os seguintes exemplos.

Exemplo 1 (parâmetro INPUT)

Inserir uma tupla na tabela **tb_staff** usando uma store procedure em Java.

1. Escreva um método Java addEmployee(int, String) para inserir uma tupla na tabela **tb_staff**. Em seguida, compile o método Java e compacte a classe em um arquivo AA.jar.

```
public static void addEmployee(int id, String name)
{
Connection conn =
DriverManager.getConnection("jdbc:default:connection");
PreparedStatement pstmt = conn.prepareStatement("insert into
tb_staff
values(?,?)");
pstmt.setInt(1, empid);
pstmt.setString(2, name);
pstmt.execute();
}
```

1. Crie uma store procedure em Java para o método Java addEmployee(int, String).

Para executar a instrução SQL para adicionar o novo arquivo JAR:

```
dmSQL>          ADD          JARFILE          logical_AA          AA.jar;
```

Para executar uma das seguintes instruções SQL para criar a store procedure em Java:

```
dmSQL> CREATE PROCEDURE JSP_addEmp (int id, char(10) name) RETURNS
STATUS
LANGUAGE   JAVA   FROM   'xx.yy.addEmployee(int,String)',   logical_AA;
```

ou:

```
dmSQL> CREATE OR REPLACE PROCEDURE JSP_addEmp (int id, char(10) name)
RETURNS
STATUS    LANGUAGE    JAVA    FROM    'xx.yy.addEmployee(int,String)',
logical_AA;
```

1. Execute a store procedure em Java.

Para executar a instrução SQL para rodar o procedimento armazenado em Java:

```
dmSQL>      EXECUTE      PROCEDURE      JSP_addEmp(1234,      'jeff');
```

Exemplo 2 (Parâmetro OUTPUT)

Selecione o nome de um funcionário da tabela **tb_staff** com o predicado (empid) usando uma store procedure em Java.

1. Escreva um método Java oneEmployee(int, byte[]) para obter o nome de um funcionário da tabela **tb_staff** com o predicado (empid). Em seguida, compile o método Java e compacte a classe em um arquivo BB.jar.

```
public static void oneEmployee(int id, byte[] name)
{
Connection conn =
DriverManager.getConnection("jdbc:default:connection");
PreparedStatement pstmt = conn.prepareStatement("select name from
tb_staff where id = ?");
pstmt.setInt(1, id);
ResultSet rs = pstmt.executeQuery();
Rs.next();
String empName = rs.getString(1);
Name = empName.getBytes();
}
```

1. Crie uma store procedure em Java para o método Java oneEmployee(int, byte[]).

Para executar a instrução SQL para adicionar o novo arquivo JAR:

```
dmSQL>         ADD         JARFILE         logical_BB         BB.jar;
```

Para executar a instrução SQL para criar a store procedure em JavaSP:

```
dmSQL> CREATE PROCEDURE JSP_oneEmp (int id, char(10) name OUTPUT)
RETURNS STATUS
LANGUAGE JAVA FROM 'xx.yy.oneEmployee(int,byte[])', logical_BB;
```

1. Execute a store procedure em Java.

Para executar a instrução SQL para rodar o procedimento armazenado em Java:

```
dmSQL> EXECUTE PROCEDURE JSP_oneEmp(1234, ?);
```

Exemplo 3 (Resultset)

Selecione um conjunto de resultados da tabela **tb_staff** usando uma store procedure em Java.

1. Escreva um método Java rsEmployee() para obter o nome de um funcionário da tabela **tb_staff**. Em seguida, compile o método Java e compacte a classe em um arquivo CC.jar.

```
public static ResultSet rsEmployee()
{
Connection conn =
DriverManager.getConnection("jdbc:default:connection");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select id, name from tb_staff");
Return rs;
}
```

1. Crie uma store procedure em Java para o método Java rsEmployee().

Para executar a instrução SQL para adicionar o novo arquivo JAR:

```
dmSQL>         ADD         JARFILE         logical_CC         CC.jar;
```

Para executar a instrução SQL para criar a store procedure em Java:

```
dmSQL> CREATE PROCEDURE JSP_rsEmp RETURNS STATUS, int outId,
char(10) outName
LANGUAGE JAVA FROM 'xx.yy.rsEmployee()', logical_CC;
```

1. Execute a store procedure em Java.

Para executar a instrução SQL para rodar a store procedure em Java:

```
dmSQL> EXECUTE PROCEDURE JSP_rsEmp();
```

1. Busque o conjunto de resultados usando um método de fetch geral (ou fetch estendido).

Input/Output Argument

Os argumentos de entrada/saída para a store procedure em Java no DBMaker suportam os seguintes tipos de dados:

BINARY	VARCHAR
CHAR	FLOAT
REAL	DOUBLE
SMALLINT	INTEGER
TIMESTAMO	DATE
TIME	DECIMAL

Os sete tipos básicos de Java e arrays que o DBMaker suporta são:

JAVA TYPE	ARRAY
byte	bute [ ]
short	shot [ ]
int	int [ ]
long	long [ ]
float	float [ ]
double	double [ ]
char	char [ ]

Além dos tipos básicos e arrays, o DBMaker também suporta as seguintes classes para store procedure em Java:

JAVA TYPE	ARRAY
Byte	Byte [ ]
Short	Short [ ]
Int	Int [ ]
Long	Long [ ]
Float	Float [ ]
Double	Double [ ]
Character	Character [ ]
String	String [ ]

12.3 SQL Stored Procedures

Usar instruções SQL para criar store procedure, em vez de ESQL e Java, pode ser uma abordagem mais eficaz em alguns casos. Uma store procedure SQL é um procedimento com lógica implementada usando apenas instruções SQL. A store procedure SQL contém um conjunto de instruções SQL que podem ser armazenadas em um servidor. Uma vez no servidor, os clientes podem evitar a execução de muitas instruções individuais utilizando a store procedure SQL. Stored Procedures SQL

incluem stored procedures permanentes, stored procedures temporárias e stored procedures anônimas. Para mais informações sobre stored procedures SQL, consulte DBMaker SQL Stored Procedure User's Guide.

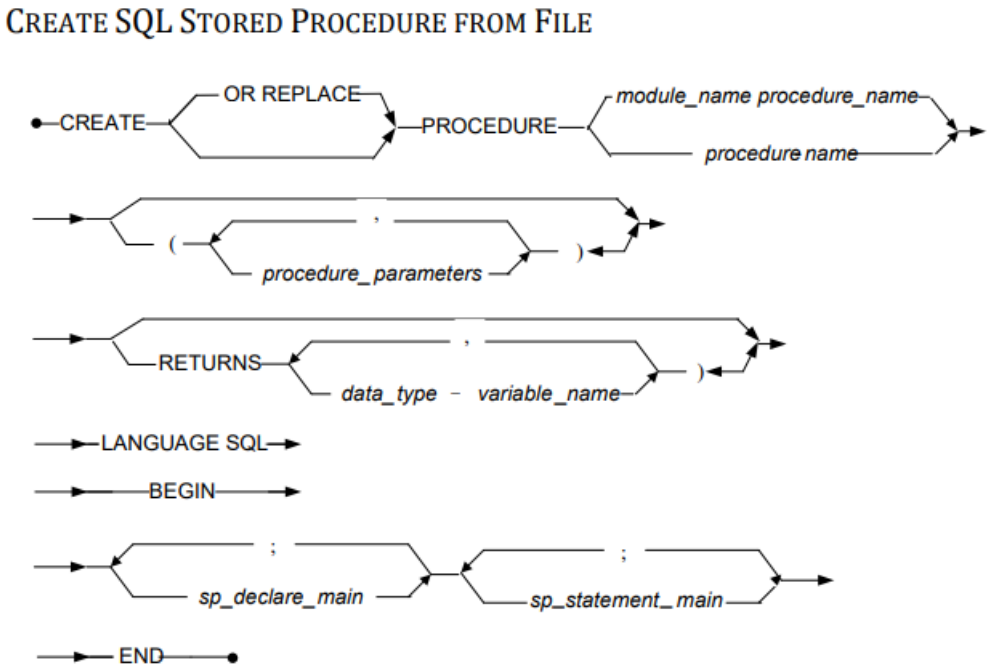
Architecture

As store procedure SQL contêm declarações compostas, que são delimitadas pelas palavras-chave BEGIN e END. O exemplo a seguir ilustra uma instrução da store procedure SQL.

```
BEGIN #block header
Variable declarations
Condition declarations
Cursor declarations
Condition handler declarations
Assignment, flow of control, SQL statements and other compound
statements
END; #block end
```

Esse exemplo mostra como as store procedure SQL consistem em uma ou mais declarações e instruções de componentes que formam um bloco. Blocos suportam alinhamento dentro de uma única store procedure SQL. Algumas declarações de componentes são opcionais: variável, condição e manipulador. No entanto, quando presentes, elas devem preceder as instruções de atribuição, controle de fluxo, SQL e outras instruções compostas. Note que declarações de cursor podem aparecer em qualquer lugar dentro do bloco.

Create SQL Stored Procedure Syntax



OR REPLACE é usado para recriar a procedure se ela já existir, ou seja, você pode usar essa cláusula para alterar a definição de uma procedure existente.

NOTA: *A sintaxe de comando **CREATE OR REPLACE** não é suportada em store procedure.*

NOTA: *A sintaxe **CREATE OR REPLACE PROCEDURE** não é suportada enquanto o **AUTOCOMMIT** estiver desativado.*

Store procedure SQL podem ser criados usando um arquivo externo (*.sp). Utilize a ferramenta de linha de comando **dmSQL** para criar store procedure SQL referenciando arquivos externos, conforme mostrado no exemplo a seguir.

Exemplo 1:

Para criar uma store procedure SQL por referência a um arquivo:

```
dmSQL> CREATE PROCEDURE FROM 'CRETB.SP';
dmSQL> CREATE PROCEDURE FROM '.\\SPDIR\\CRETB.SP';
dmSQL> CREATE PROCEDURE FROM 'D:\\DATABASE\\SPDIR\\CRETB.SP';
```

Exemplo 2:

Para criar ou substituir uma store procedure SQL por referência a um arquivo:

```
dmSQL> CREATE OR REPLACE PROCEDURE FROM 'CRETB.SP';
dmSQL> CREATE OR REPLACE PROCEDURE FROM '.\\SPDIR\\CRETB.SP';
dmSQL> CREATE OR REPLACE PROCEDURE FROM
'D:\\DATABASE\\SPDIR\\CRETB.SP';
```

CREATE SQL STORED PROCEDURE IN SCRIPT

Os usuários podem criar store procedure SQL não apenas a partir de arquivos, mas também diretamente no dmSQL. O usuário pode chamar e excluir suas próprias store procedure SQL e pode executar store procedure SQL para os quais lhe foi concedido privilégio. Para mais informações, consulte o capítulo 12.3, Store Procedure SQL.

O **SQLSP** contém mais de uma instrução SQL, e cada instrução é terminada por ';'. Portanto, o dmSQL deve suportar delimitadores de bloco. O delimitador de bloco pode ser uma sequência de caracteres de a-z, A-Z, @, %, ^, e deve conter pelo menos dois e no máximo sete caracteres. No delimitador de bloco, ';' não indica o fim da entrada. Os usuários devem definir o delimitador de bloco antes de escrever a store procedure SQL no dmSQL; caso contrário, será retornado um erro.

Exemplo:

Para criar a sintaxe de store procedure em um script:

```
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> @@
CREATE PROCEDURE sp_in_script2
LANGUAGE SQL
BEGIN
INSERT INTO t1 VALUES(1);
END;
@@
dmSQL> SET BLOCK DELIMITER;
```

Using Parameters

Valores SQL são passados para e a partir de store procedure SQL por meio de parâmetros. Parâmetros podem ser úteis em uma store procedure SQL ao implementar lógica que depende de um determinado valor de entrada ou conjunto de valores escalares, ou quando você precisa retornar um ou mais valores escalares de saída sem retornar um conjunto de resultados.

A Store Procedure SQL suporta expressões nos parâmetros de entrada, incluindo adição, subtração, multiplicação, divisão e funções definidas pelo usuário (UDF), entre outras. No entanto, atualmente, a expressão não inclui variáveis de host nos parâmetros de entrada.

```
dmSQL> @@
CREATE OR REPLACE PROCEDURE SP_PARAM(IN C1 INT,OUT C2 INT)
LANGUAGE SQL
BEGIN
SET C2 = C1*100;
END;
@@
dmSQL> CALL SP_PARAM(1000,?);
C2 : 100000
dmSQL> CALL SP_PARAM(1000+1,?);
C2 : 100100
dmSQL> CALL SP_PARAM(999+1-100,?);
C2 : 90000
dmSQL> CALL SP_PARAM(999+10*100,?);
C2 : 199900
```

Variable Declaration

O suporte a variáveis locais em store procedure SQL permite atribuir e recuperar valores SQL para dar suporte à lógica da store procedure. As variáveis em uma store procedure SQL são definidas com a instrução DECLARE. Use DECLARE para definir itens locais de uma rotina, ou seja, variáveis locais, condições, manipuladores e cursores. O DECLARE deve seguir diretamente um BEGIN como parte de uma instrução composta BEGIN ... END. Nenhuma outra instrução pode preceder uma declaração DECLARE. As declarações devem seguir esta ordem: primeiro devem ser declaradas as variáveis e condições, em seguida, os cursores e, por fim, os manipuladores.

O DBMaker suporta dois tipos de variáveis SQL: Variável de Conexão (CV) e Variável de Instrução (SV). Ambas as variáveis são utilizadas para aprimorar a extensibilidade e portabilidade do comando dmsql. As seções a seguir fornecem ilustrações e exemplos das variáveis CV e SV.

CONNECTION VARIABLE (CV)

CV (Variável de Conexão) é uma variável que só pode ser definida em conexões locais. As variáveis de conexão em uma conexão são independentes daquelas em outras conexões. Ou seja, as variáveis de conexão só podem ser usadas pela conexão que as possui e não podem ser acessadas ou utilizadas por outras conexões.

Para os usuários, uma variável de conexão é uma variável global de comando SQL na conexão local, e as variáveis de conexão podem ser usadas na ferramenta de linha de comando **dmSQL** e em **SQLSP**. O usuário pode atribuir um valor a uma variável de conexão em uma instrução e referenciá-la em outra instrução. Isso permite passar valores de uma instrução para outra. Uma vez que a conexão é desconectada do banco de dados, todas as variáveis de conexão são automaticamente liberadas. **CV** é um tipo de dado pré-definido. Você pode definir o tipo da variável antes de usá-la. Se você referenciar uma variável que não foi inicializada, será retornado o erro **"Error (6344): [DBMaker] invalid variable name".**

NOTA: ***CV*** *não suporta os tipos de dados SERIAL, BIGSERIAL, FILE, OID, LONG VARCHAR, LONG VARBINARY e Media.*

Exemplo 1:

CV pode ser declarado e utilizado com a ferramenta de linha de comando **dmSQL**. No exemplo a seguir, @a, @aa e @b são variáveis de conexão. Elas podem ser usadas não apenas em operações de inserção, atualização e exclusão, mas também em cláusulas WHERE ou chamadas de funções UDF, entre outras.

```
dmSQL> DECLARE SET INT @a = 1;
dmSQL> SELECT @b;
ERROR (6344): [DBMaker] invalid variable name : B name
dmSQL> DECLARE SET INT @aa = NULL;
dmSQL> SELECT @aa;
 @AA
=======================
NULL
dmSQL> DECLARE SET INT @b = 2;
dmSQL> SELECT @a, @b;
 @A @B
========== ==========
1 2
dmSQL> SELECT * FROM t1 WHERE c1=@a;
dmSQL> INSERT INTO t1 VALUES(@b+100);
dmSQL> UPDATE t2 SET c2 = @b+2 WHERE c1 = @a+1;
```

```
dmSQL> DELETE FROM t1 WHERE c1=@b;
dmSQL> SELECT SUM(c1) INTO @b FROM t1;
```

Exemplo 2: No **SQLSP** (SQL Stored Procedure), **CV** pode ser declarado e usado como uma variável normal. No exemplo abaixo, @val2 e @val3 são variáveis de conexão.

```
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> @@
CREATE PROCEDURE tsp2
LANGUAGE SQL
BEGIN
 DECLARE SET INT @val2 =100+100;
DECLARE SET INT @val3 =LENGTH('test');
 SET @val3 =LENGTH('test');
END;
@@
```

Exemplo 3:

```
In SQLSP, CV can be used from select count(*) command. In this
example, @count is
connection variable. dmSQL> SET BLOCK DELIMITER @@;
dmSQL> declare set int @count = 0;
dmSQL>
@@
CREATE OR REPLACE PROCEDURE TSP
LANGUAGE SQL
BEGIN
 SET @count = select count(*) from SYSPROCINFO;
END;
@@
dmSQL> call tsp;
dmSQL> select @count;
 @COUNT
===========
 36
```

STATEMENT VARIABLE (SV)

SV (Variável de Declaração) é uma variável declarada em uma instrução e pode ser usada somente nessa instrução específica. Variáveis de declaração em diferentes comandos são independentes umas das outras e só podem ser usadas em **SQLSP** (Store Procedure SQL).

A definição das variáveis de declaração é a mesma das variáveis locais em **SQLSP**. Uma vez que a instrução SQL é terminada, as variáveis de declaração são automaticamente liberadas.

Exemplo: Para distinguir a diferença entre a definição de **CV** e **SV** :

```
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> DECLARE SET INT @aa = 100;
dmSQL> CREATE TABLE tab(c1 INT);
dmSQL>@@
CREATE OR REPLACE PROCEDURE tsp(OUT c1 INT)
LANGUAGE SQL
BEGIN
 DECLARE vals INT DEFAULT 100; --vals is sv
 INSERT INTO tab VALUES(@aa); --aa is cv
 INSERT INTO tab VALUES(vals); --vals is sv
 SET c1 = @aa; --c1 is sv
 SET @aa = 200; --error not declare cv in sqlsp
END;
@@
```

Cursors

Usados em Store Procedure SQL, os cursores permitem a definição de conjuntos de resultados e a realização de lógica complexa em cada linha dentro do conjunto. Vale ressaltar que um conjunto de resultados é simplesmente um conjunto de linhas de dados. Usando o mesmo método, as store procedure SQL também podem definir conjuntos de resultados e retorná-los diretamente ao chamador ou a uma aplicação cliente.

Pense em um cursor como um ponteiro para uma linha dentro de um conjunto de linhas. O cursor pode apontar para qualquer linha no conjunto de resultados, no entanto, ele só pode referenciar uma única linha de cada vez.

A instrução DECLARE CURSOR primeiro define um cursor, e as seguintes instruções SQL são usadas para manipular o cursor: OPEN, FETCH e CLOSE.

Assignment Statements

As instruções de atribuição são usadas para atribuir valores a variáveis e parâmetros SQL. Valores podem ser atribuídos às variáveis usando uma instrução SET ou uma instrução CURSOR FOR SELECT FROM. Além disso, uma variável pode ter um valor padrão que foi definido quando a variável foi declarada. Literais, expressões, resultados de consultas e valores de registradores especiais podem ser atribuídos às variáveis. Valores de variáveis podem ser atribuídos a parâmetros de store prodeure SQL, outras variáveis em store procedure SQL e podem ser referenciados como parâmetros dentro de instruções de store procedure SQL que são executadas dentro da rotina.

A instrução SET Variable atribui valores a variáveis locais, parâmetros de saída e novas variáveis de transição. A instrução SET Variable está sob controle de transação. As instruções de atribuição SET aceitam expressões simples e complexas.

NOTA: *A atribuição de variáveis do tipo de dado string deve ser menor que 1024 bytes.*

SIMPLE EXPRESSIONS

Expressões simples são classificadas pelos tipos de dados numéricos, caractere, timestamp e binário. Os tipos de dados numéricos são: INTEGER, BIGINT, SMALLINT, DOUBLE, FLOAT, REAL e DECIMAL. Os tipos de dados de caractere são: CHAR, NCHAR, VARCHAR e NVARCHAR. Os tipos de dados de timestamp são: DATE, TIME e TIMESTAMP.

Uma expressão simples inclui operadores (+, -, *, /), variáveis, constantes, valores e strings. Expressões simples têm uma eficiência de implementação muito maior do que expressões complexas. Em particular, declarações com múltiplos loops melhoram significativamente a velocidade de execução.

COMPLEX EXPRESSIONS

Expressões complexas incluem todos os mesmos valores de atribuição encontrados em expressões simples, além de funções SQL, como funções SQL integradas e funções SQL definidas pelo usuário.

Control Flow Statements

As instruções de controle SQL se dividem nas seguintes categorias: instruções relacionadas a variáveis, instruções condicionais (CASE e IF), instruções de loop (FOR, LOOP, WHILE e REPEAT), instruções GOTO, instruções de retorno, instruções de transferência de controle (ITERATE e LEAVE), rótulos e instruções compostas de store procedure SQL.

Returning Result Sets

Os cursores podem ser usados para mais do que simplesmente iterar através das linhas de um conjunto de resultados. Em store procedures SQL, os cursores também podem retornar conjuntos de resultados para o programa que os chamou.

Return Status of SQL Stored Procedures

Normalmente, os usuários podem obter o status de execução das store procedure SQL através do JDBATool ou da ferramenta dmSQL.

O código de status reflete se uma store procedure foi executada com sucesso. Os usuários não podem definir o código de status em uma store procedure.

Códigos de status: -1: erro na execução da store procedure 0: execução bem-sucedida da store procedure 1: execução bem-sucedida da store procedure, mas com aviso

Se você deseja retornar o status da store procedure, deve adicionar RETURN STATUS antes de LANGUAGE SQL.

Exemplo:

Para chamar outra store procedure SQL:

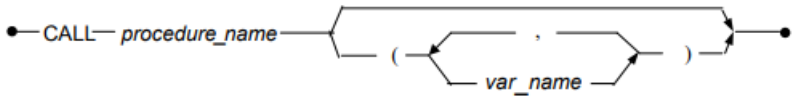
```
CREATE PROCEDURE call_test
RETURNS STATUS
LANGUAGE SQL
BEGIN
DECLARE cur CURSOR WITH RETURN FOR select * from call_tb;
OPEN cur;
END;
CREATE PROCEDURE CASE_TEST_1(IN inval INT, OUT outval1 INT, OUT
outval2 INT)
LANGUAGE SQL
```

```
BEGIN
 SET outval1 = 1;
 SET outval2 = 2;
END;
CREATE PROCEDURE call1(IN inval INT, OUT outval1 INT, OUT outval2
INT)
LANGUAGE SQL
BEGIN
 CALL CASE_TEST_1(inval, outval1, outval2);
END;
```

Executing SQL Stored Procedures

Store Procedure SQL são executadas usando a instrução CALL. A instrução CALL pode ser executada utilizando ferramentas de interface gráfica, como o JDBATool, ou diretamente a partir da Ferramenta de Linha de Comando do DBMaker, dmSQL.

A instrução CALL executável chama uma prodecure. Essa instrução pode ser incorporada em um programa de aplicação, emitida usando instruções SQL dinâmicas ou preparada dinamicamente.



RECURSIVE SQL STORED PROCEDURES

Store Procedure SQL podem ser chamados recursivamente, da mesma forma que funções recursivas. A profundidade recursiva de uma store procedure SQL é limitada pelo número máximo de execuções recursivas permitidas para store procedure SQL. O valor padrão é 8, e o intervalo válido de entrada vai de 8 a 256.

Como o número de execuções recursivas de SQLSP pode aumentar indefinidamente, o espaço de pilha no disco local também aumentará. Se um aplicativo de banco de dados gerar a mensagem de erro 6377 “profundidade recursiva excede o máximo”, isso significa que o número de recursões de procedimentos armazenados SQL no DBMaker excedeu o valor de limite de profundidade recursiva especificado. O valor da profundidade recursiva da store procedure SQL pode ser alterado pela instrução SQL “set sp_recursion=v”.

NOTA: *Você deve aumentar o tamanho da pilha local do banco de dados ao aumentar o valor de sp_recursion.* Exemplo:

Para chamar outra store procedure SQL recursiva:

```
dmSQL> SET sp_recursion = 18;
dmSQL> DECLARE set int @a1 = 0;
dmSQL> @@
CREATE OR REPLACE PROCEDURE SP_CURSION(IN N INT,OUT RES INT)
LANGUAGE SQL
BEGIN
 SET @A1 = @A1 +1;
 IF @a1 < 20 THEN
 CALL SP_CURSION(N, RES);
 END IF;
END;
@@dmSQL> CALL SP_CURSION(20,?);
ERROR (6377): [DBMaker] recursive depth exceeds maxinum : 18
[exproc.c
5952],0,0,0
dmSQL> SET sp_recursion = 28;
dmSQL> DECLARE set int @a1 = 0;
dmSQL> CALL SP_CURSION(20,?);
```

Anonymous SQL Stored Procedures

As Store Procedure SQL anônimos contêm um conjunto de instruções SQL que podem ser criadas e executadas temporariamente por um banco de dados. Não há necessidade de armazenar permanentemente as instruções SQL no DBMaker como um objeto de banco de dados. A store procedure SQL anônimo existe apenas temporariamente em um único bloco SQL, e é usado apenas uma vez pelo criador.

Uma Store Procedure SQL anônimo é um tipo especial de store procedure SQL, conhecido como Bloco SQL Anônimo. Ele permite que os usuários executem mais de uma instrução SQL (um lote de SQL) de uma só vez no lado do cliente, e suporta todos os blocos de sintaxe SQL, incluindo variáveis, lógica gramatical e cursores, entre outros. A store procedure SQL anônimo não pode usar parâmetros de store procedure e comandos específicos da ferramenta dmSQL (como SET, por exemplo). Os usuários devem usar delimitadores de bloco antes de escrever o bloco SQL anônimo no dmSQL; caso contrário, um erro será retornado. Para mais informações sobre variáveis e lógica de sintaxe de procedimentos armazenados SQL anônimos, consulte o Capítulo 12.3, Store Procedure SQL.

NOTA: *As instruções compostas de uma Store Procedure SQL anônimo são delimitadas pelas palavras-chave "BEGIN" e "END", com delimitadores de bloco.*

Diferentemente das store procedure SQL, não há nome para os store procedure SQL anônimos, e eles não podem ser referenciados por outros objetos de banco de dados. Isso significa que a execução de uma store procedure SQL anônimo é imediata. Quando uma store procedure SQL anônimo é criado com sucesso, ele entra no estado de execução.

As store procedure SQL anônimos são temporários. Uma vez que a procedure termina sua execução, o DBMaker não armazenará nenhuma informação sobre a procedure. As informações sobre a store procedure SQL anônimo não serão salvas na tabela de sistema SYSPROCINFO e não podem ser armazenadas permanentemente no DBMaker para reutilização. As store procedure SQL anônimos reduzem o intervalo de tempo entre as atualizações de código e a execução do programa, melhorando assim a eficiência do diagnóstico de problemas, prototipagem e execução de código de teste, e oferecendo conveniência para atualizações e execuções de múltiplas tarefas.

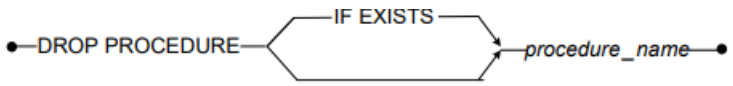
Exemplo:

Para criar uma store procedure SQL anônimo em um bloco SQL:

```
dmSQL> SET BLOCK DELIMITER @@ @@;
dmSQL> @@
BEGIN
 CREATE TABLE TAB(C1 INT,C2 INT);
 INSERT INTO TAB VALUES(123,456);
END;
@@dmSQL> SELECT * FROM tab;
 C1 C2
=========== ===========
 123 456
1 rows selected
dmSQL>
@@
BEGIN
 DECLARE c1 INT;
 DECLARE SET INT @a2 = 100;
 SET c1 = 200;
 INSERT INTO tab VALUES(@a2,c1);
END;
@@dmSQL> SELECT * FROM tab;
 C1 C2
=========== ===========
 123 456
100 200
```

```
2 rows selected
```

12.4 Dropping a Stored Procedure



Exemplo 1: A primeira instrução exclui a store procedure sp_proc1, e a segunda instrução exclui a store procedure user1.sp_proc2.

```
dmSQL> DROP PROCEDURE sp_proc1;
dmSQL> DROP PROCEDURE user1.sp_proc2;
```

Exemplo 2: A primeira instrução exclui a store procedure sp_proc1 se ele existir, e a segunda instrução exclui a store procedure user1.sp_proc2 se ele existir.

```
dmSQL> DROP PROCEDURE IF EXISTS sp_proc1;
dmSQL> DROP PROCEDURE IF EXISTS user1.sp_proc2;
```

12.5 Getting Procedure Information

Exemplo 1: Para usar o dmSQL e obter informações sobre a procedure a partir da tabela do sistema SYSPROCINFO:

```
dmSQL> SELECT * FROM SYSPROCINFO;
```

Exemplo 2: Para usar o dmSQL e obter informações sobre a procedure a partir da tabela do sistema SYSPROCPARAM:

```
dmSQL> SELECT * FROM SYSPROCPARAM;
```

NOTA: *As funções ODBC SQLProcedure() e SQLProcedureColumns() são usadas para obter informações sobre procedure e parâmetros para programas.*

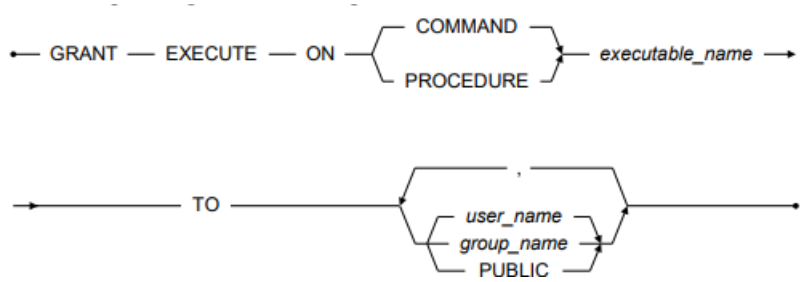
Exemplo 3:

Para usar a seguinte instrução para verificar a definição da store procedure que foi criado anteriormente:

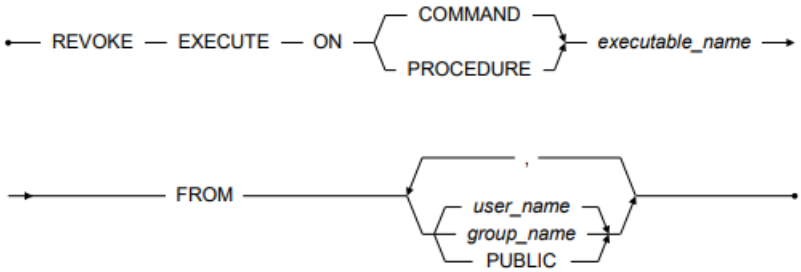
```
def proc[procedure] [modulename.][ownername.]spname;
```

12.6 Security

Somente o proprietário ou um usuário com autoridade DBA ou superior pode executar inicialmente uma store procedure. Outros usuários podem executar a procedure quando o privilégio de execução foi concedido a eles ou a um grupo do qual o usuário seja membro. Apenas o proprietário ou um usuário com autoridade DBA ou superior pode conceder o privilégio EXECUTE PROCEDURE em uma store procedure para outros usuários.



O proprietário ou um usuário com autoridade DBA ou superior também pode revogar o privilégio de execução em uma store procedure para outros usuários.



Exemplo 1:

user1 cria uma store procedure chamado sp_proc1 e concede o privilégio de execução ao user2 usando dmSQL:

```
dmSQL> GRANT EXECUTE ON PROCEDURE sp_proc1 TO user2;
```

Exemplo 2:

user1 cria uma store procedure chamado sp_proc1 e concede o privilégio de execução ao PUBLIC usando dmSQL:

```
dmSQL> GRANT EXECUTE ON PROCEDURE sp_proc1 TO PUBLIC;
```

Exemplo 3:

user1 revoga o privilégio de execução do user2 usando dmSQL:

```
dmSQL> REVOKE EXECUTE ON PROCEDURE sp_proc1 FROM user2;
```

Exemplo 4:

user1 revoga o privilégio de execução do PUBLIC usando dmSQL:

```
dmSQL> REVOKE EXECUTE ON PROCEDURE sp_proc1 FROM PUBLIC;
```