6. Managing Schema and Schema Objects

Este capítulo discute o gerenciamento de diferentes tipos de objetos de schema no DBMaker, incluindo tabelas, views, sinônimos, índices, números de série, integridade de dados e domínios. O capítulo inclui tópicos sobre como navegar pelos catálogos do sistema para obter informações sobre objetos de schema e como estimar o espaço de armazenamento em disco necessário para tabelas e índices. O gerenciamento de objetos de schema pode ser realizado usando comandos dmSQL ou através da Ferramenta JDBA. A Ferramenta JDBA contém uma interface gráfica intuitiva, fornece assistentes fáceis de usar para a maioria das tarefas de gerenciamento de banco de dados e exibe a estrutura lógica do banco de dados em um formato inequívoco. O uso da Ferramenta JDBA ajudará os usuários iniciantes do DBMaker a entender o relacionamento entre os objetos de schema. Usuários experientes descobrirão que a exibição lógica auxilia na criação e gerenciamento do schema do banco de dados. As seções a seguir mostram exemplos de como gerenciar objetos de schema de banco de dados por meio do dmSQL. Para obter mais informações sobre como usar a Ferramenta JDBA para gerenciar objetos de schema, consulte o Guia do Usuário da Ferramenta JDBA. Para obter mais informações sobre como usar a linguagem SQL no DBMaker, consulte o Guia do Usuário de Referência de Comando e Função SQL.

6.1 Managing Schema

Schemas são namespaces (agrupamentos lógicos de objetos de banco de dados). Schemas contêm objetos de esquema, como tabelas, visualizações, índices, comandos, procedimentos, um domínio e um sinônimo.

O comando **CREATE SCHEMA** define um novo schema. Após a criação do schema, podemos criar objetos dentro dele. O proprietário do schema é o concedente de quaisquer privilégios concedidos.

O proprietário do schema é determinado da seguinte forma:

- Se uma cláusula **AUTHORIZATION** for especificada, o nome de usuário indicado será o proprietário do schema. E, se o nome do schema não for especificado, o nome de usuário indicado será utilizado como nome do schema.

Exemplo:

```
dmSQL> CREATE SCHEMA AUTHORIZATION JEFFERY;
```

Exemplo 1:

Como usuário com autoridade de RECURSO, JEFFERY cria um schema chamado SCH_JEF. Por padrão, JEFFERY se torna o proprietário do schema.

```
dmSQL> CREATE SCHEMA SCH_JEF;
```

Exemplo 2:

Sendo um usuário com autoridade de DBA, crie um schema com o usuário JEFFERY como proprietário, e o nome de usuário JEFFERY como o nome padrão do schema.

```
dmSQL> CREATE SCHEMA AUTHORIZATION JEFFERY;
```

Exemplo 3:

Como usuário com autoridade de DBA, crie um schema chamado SCH_ForJEF com o proprietário sendo o usuário JEFFERY.

```
dmSQL> CREATE SCHEMA SCH_ForJEF AUTHORIZATION JEFFERY;
```

Exemplo 4:

Um usuário com autoridade de DBA cria um schema chamado "inventory". Em seguida, o usuário cria uma tabela e um índice nessa tabela. Por fim, o usuário concede permissão de acesso à tabela para o usuário JEFFERY.

```
dmSQL> CREATE SCHEMA inventory;
dmSQL> CREATE TABLE inventory.part (partNo smallint not null,
quantity int);
dmSQL> CREATE INDEX partind ON inventory.part (partNo);
dmSQL> GRANT ALL ON inventory.part TO JEFFERY;
```

O comando DROP SCHEMA remove esquemas do banco de dados. Apenas o dono do schema ou um DBA (administrador do banco de dados) pode removê-lo. É importante ressaltar que o dono não pode remover o schema se ele ainda contiver objetos (tabelas, views, etc.). Exemplo 5:

Remover o schema SCH_JEF do banco de dados

```
dmSQL> DROP SCHEMA SCH_JEF;
```

NOTA: *É importante ressaltar que nomes de usuário e nomes de schema não podem ser iguais.*

INFORMATION SCHEMA

Cada banco de dados no DBMaker contém um schema chamado **INFORMATION_SCHEMA**. Este schema contém uma série de visualizações que permitem visualizar, mas não alterar, a descrição dos objetos pertencentes ao banco de dados.

O DBMaker fornece visualizações do **information schema** para a obtenção de metadados. Essas visualizações oferecem uma visão interna e independente das tabelas do sistema em relação aos metadados do DBMaker. As visualizações do **information schema** permitem que as aplicações funcionem corretamente, mesmo que mudanças significativas tenham sido feitas nas tabelas do sistema. As visualizações do **information schema** incluídas no DBMaker estão em conformidade com a definição do padrão SQL-92 para o **INFORMATION_SCHEMA**.

O DBMaker suporta uma convenção de nomenclatura em três partes ao se referir ao servidor atual. O padrão SQL-92 também suporta uma convenção de nomenclatura em três partes. No entanto, os nomes usados em ambas as convenções de nomenclatura são diferentes.

DBMaker Name	EQUIVALENT SQL-92 NAME
Database	Catalog
Owner	Schema
Object	Object
user-defined data type	Domain

Essas visualizações são definidas em um schema especial chamado **INFORMATION_SCHEMA**, que está contido em cada banco de dados. Cada visualização do **INFORMATION_SCHEMA** contém metadados para todos os objetos de dados armazenados naquele banco de dados específico. Esta tabela descreve as relações entre os nomes do DBMaker e os nomes padrão do SQL-92.

Essa convenção de nomenclatura se aplica às visualizações compatíveis com SQL-92 do DBMaker. Essas visualizações são definidas em um schema especial chamado **INFORMATION_SCHEMA**, presente em cada banco de dados. Cada visualização do

INFORMATION_SCHEMA contém metadados para todos os objetos de dados armazenados naquele banco de dados específico.

As visões do INFORMATION_SCHEMA são listadas abaixo:

- COLUMN_DOMAIN_USAGE
- COLUMN_PRIVILEGES
- COLUMNS
- DOMAINS
- SCHEMATA
- TABLE_PRIVILEGES
- TABLES
- VIEW_COLUMN_USAGE
- VIEW_TABLE_USAGE
- VIEWS

Exemplo:

```
dmSQL> SELECT * FROM INFORMATION_SCHEMA.COLUMNS;
```

6.2 Managing Tables

Tabelas são a unidade lógica de armazenamento usada pelo DBMaker para armazenar dados. Uma tabela consiste em várias colunas e linhas. Uma coluna às vezes é chamada de campo ou atributo, e uma linha pode ser chamada de registro ou tupla.

No DBMaker, cada tabela é identificada por um nome de schema e um nome de tabela exclusivos.

Por exemplo, se dois usuários chamados Jeff e Kevin criarem cada um uma tabela chamada "amigo" com o nome de schema padrão, então os nomes de tabela "Jeff.amigo" e "Kevin.amigo" indicam as duas tabelas diferentes.

Na ferramenta JDBA, todas as tabelas de um banco de dados podem ser visualizadas expandindo o nó tabelas na árvore lógica. Selecionar uma tabela exibe o schema dessa tabela.

Creating Tables

Toda tabela é definida com um nome de tabela e um conjunto de colunas. O número de colunas em uma tabela pode variar de 1 a 2000.

Cada coluna possui:

- Um nome de coluna e um tipo de dado ou um domínio, que é descrito na Seção 6.11, Gerenciando Domínios.
- Um tamanho (o tamanho pode ser predeterminado pelo tipo de dado, como INTEIRO), uma precisão e escala (apenas para colunas do tipo de dado DECIMAL) ou um número inicial (apenas para colunas do tipo de dado SERIAL).

O DBMaker suporta um grande número de tipos de dados que podem ser usados para definir colunas. Existem tipos numéricos (SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, DECIMAL, REAL, SERIAL e BIGSERIAL), tipos binários (BINARY e VARBINARY), tipos de caracteres (CHAR, NCHAR, VARCHAR e NVARCHAR), tipos BLOB (LONG VARCHAR, LONG VARBINARY, FILE, tipo de mídia e JSONCOLS) e tipos de tempo (DATE, TIME e TIMESTAMP). Consulte a Referência de Comando e Função SQL para obter mais informações sobre tipos de dados.

Ao criar uma tabela, forneça o nome da tabela, as definições das colunas e o nome do tablespace associado. Uma tabela será colocada no tablespace do sistema por padrão se não estiver associada a outro tablespace. As tabelas podem ser criadas usando o assistente Criar Tabela da ferramenta JDBA ou usando o prompt de comando dmSQL. Para obter informações sobre como criar uma tabela com a ferramenta JDBA, consulte o Guia do Usuário da ferramenta JDBA. O seguinte é um exemplo de como criar uma tabela usando dmSQL. Detalhes sobre a sintaxe e uso do comando SQL CREATE TABLE podem ser encontrados na Referência de Comando e Função SQL.

Exemplo: Para criar a tabela tb_staff no tablespace ts_reg:

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20),
 ID INTEGER,
 name CHAR(30),
 joinDate DATE,
 height FLOAT,
 degree VARCHAR(200),
 picture LONG VARCHAR) IN ts_reg;
```

O DBMaker oferece muitos recursos úteis que podem ser aplicados ao criar tabelas:

- Definir um valor padrão para uma coluna
- Especificar que uma coluna não pode ser nula (NOT NULL)
- Especificar a chave primária ou a chave estrangeira para a tabela

- Definir opções como LOCK MODE, FILLFACTOR ou NOCACHE para melhorar a eficiência do banco de dados
- Especificar a tabela como temporária

DEFAULT VALUES FOR COLUMNS

Uma coluna em uma tabela pode ser atribuída a um valor padrão para que, quando uma nova linha for inserida e um valor para a coluna for omitido, o valor padrão seja automaticamente fornecido.

Os valores padrão para cada coluna em uma tabela podem ser especificados. Se um valor padrão não for definido para uma coluna, o valor padrão será definido como **NULL**. Valores padrão válidos podem ser constantes, **NULL** ou funções internas. Para mais informações sobre funções internas, consulte a Referência de Comandos e Funções SQL.

Agora, o DBMaker suporta a definição de atributos padrão de coluna para operações de inserção e atualização usando três palavras-chave: **USER**, **SYSTEM** e **ON UPDATE**. As palavras-chave **USER/SYSTEM** são opcionais. Essas palavras-chave especificam se os usuários podem ou não modificar o valor da coluna com um valor padrão usando a instrução **INSERT/UPDATE**. **USER** é usado por padrão. A palavra-chave **USER** especifica que os usuários podem modificar seu valor, e a palavra-chave **SYSTEM** especifica que os usuários não podem modificar seu valor. A palavra-chave **ON UPDATE** também é opcional. Essa palavra-chave especifica que o valor da coluna com um valor padrão pode ser atualizado automaticamente quando o valor de outras colunas for alterado. As três palavras-chave são usadas principalmente na definição de uma tabela e, para mais detalhes sobre a definição de uma tabela, consulte **CREATE TABLE**, **ALTER TABLE ADD COLUMN** e **ALTER TABLE MODIFY COLUMN** no capítulo **Comando SQL** da Referência de Comandos e Funções SQL.

Além disso, ao atualizar dados, um usuário pode usar a opção de conexão **SYSTEM DEFAULT** para especificar se o valor de uma coluna com o atributo **SYSTEM DEFAULT** será substituído pelo valor padrão. Se o usuário definir essa opção como **ON**, o valor será atualizado para o valor padrão; se o usuário definir essa opção como **OFF**, o valor original será atualizado para o valor especificado pelo usuário. A configuração padrão para essa opção é **ON**. Além disso, se tiver atribuído um valor à coluna usando a instrução **INSERT/UPDATE**, o usuário pode usar a opção de conexão **LOAD SYSTEM DEFAULT** para especificar se o valor de uma coluna com o atributo **SYSTEM DEFAULT** será substituído durante o processo de carregamento das tabelas do banco de dados. Se o usuário definir essa opção como **ON**, o valor será atualizado para o valor padrão; se o usuário definir essa opção como **OFF**, o valor original será

atualizado para o valor especificado pelo usuário. A configuração padrão para essa opção é **OFF**.

Exemplo: Para especificar o valor padrão da coluna nation, na tabela tb_staff como uma constante 'R.O.C.' e o valor padrão da coluna joinDate como o valor da função embutida curdate():

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20) DEFAULT 'R.O.C',
 ID INTEGER,
 name CHAR(30),
 joinDate DATE DEFAULT CURDATE(),
 height FLOAT,
 degree VARCHAR(200),
 picture LONG VARCHAR) IN ts_reg;
```

Exemplo 2a:

```
dmSQL> CREATE TABLE computer(id INT, buy_time TIMESTAMP DEFAULT
'2012-03-04
12:12:12', price int); //now attributes of buy_time is USER
dmSQL> INSERT INTO computer VALUES(1, '2012-10-10 10:10:20', 3400);
//value of
buy_time will be replaced with '2012-10-10 10:10:20' which is
specified by the
user
1 rows inserted
dmSQL> INSERT INTO computer VALUES(2, '2012-10-11 10:10:20', 5400);
1 rows inserted
dmSQL> SELECT * FROM computer;
 ID BUY_TIME PRICE
=========== ========================== ===========
 1 2012-10-10 10:10:20 3400
 2 2012-10-11 10:10:20 5400
2 rows selected
dmSQL> UPDATE computer SET price=3200 WHERE id=1; //value of
buy_time will not be
updated
1 rows updated
dmSQL> SELECT * FROM computer;
 ID BUY_TIME PRICE
=========== ========================== ===========
 1 2012-10-10 10:10:20 3200
```

```
 2 2012-10-11 10:10:20 5400
2 rows selected
```

Exemplo 2b:

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP
DEFAULT '2012-
03-04 12:12:12' ON UPDATE); //now attributes of buy_time is USER and
ON UPDATE
dmSQL> UPDATE computer SET price=3000 WHERE id=1; //value of
buy_time will be
replaced with the default value'2012-03-04 12:12:12'
1 rows updated
dmSQL> SELECT * FROM computer;
 ID BUY_TIME PRICE
=========== ========================== ===========
 1 2012-03-04 12:12:12 3000
 2 2012-10-11 10:10:20 5400
2 rows selected
dmSQL> UPDATE computer SET price=3000, buy_time='2012-10-10' WHERE
id=1;//value
of buy_time will be replaced with '2012-10-10' which is specified by
the user
1 rows updated
dmSQL> SELECT * FROM computer;
 ID BUY_TIME PRICE
=========== ========================== ===========
 1 2012-10-10 00:00:00 3000
 2 2012-10-11 10:10:20 5400
2 rows selected
```

Exemplo 2c:

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP
SYSTEM DEFAULT
'2012-03-04 12:12:12'); //now attributes of buy_time is SYSTEM
dmSQL> INSERT INTO computer VALUES(3, '2012-11-10 10:10:20', 4700);
//value of
buy_time will not be replaced with '2012-11-10 10:10:20' which is
specified by
```

```
the user
1 rows inserted
dmSQL> INSERT INTO computer VALUES(4, '2012-12-11 10:10:20',
2800);//value of
buy_time will not be replaced with '2012-12-11 10:10:20' which is
specified by
the user
1 rows inserted
dmSQL> SELECT * FROM computer;
 ID BUY_TIME PRICE
========== ========================== ==========
 1 2012-10-10 00:00:00 3000
 2 2012-10-11 10:10:20 5400
 3 2012-03-04 12:12:12 4700
 4 2012-03-04 12:12:12 2800
4 rows selected
dmSQL> UPDATE computer SET price=4500 WHERE id=3; //value of
buy_time will not be
updated
1 rows updated
dmSQL> SELECT * FROM computer;
 ID BUY_TIME PRICE
========== ========================== ==========
 1 2012-10-10 00:00:00 3000
 2 2012-10-11 10:10:20 5400
 3 2012-03-04 12:12:12 4500
 4 2012-03-04 12:12:12 2800
4 rows selected
```

Exemplo 2d:

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP
SYSTEM DEFAULT
'2012-03-04 12:12:12' ON UPDATE); //now attributes of buy_time is
SYSTEM and ON
UPDATE
dmSQL> UPDATE computer SET price=4000, buy_time='2015-01-01' WHERE
id=3; //value
of buy_time will be replaced with the default value'2012-03-04
12:12:12'
1 rows updated
dmSQL> SELECT * FROM computer;
```

```
 ID BUY_TIME PRICE
========== ========================= ==========
1 2012-10-10 00:00:00 3000
 2 2012-10-11 10:10:20 5400
 3 2012-03-04 12:12:12 4000
 4 2012-03-04 12:12:12 2800
4 rows selected
```

NOT NULL

Regras para colunas ou tabelas podem ser especificadas. Essas regras são chamadas de restrições de integridade. Um exemplo é a restrição de integridade NOT NULL definida em uma coluna de uma tabela. Ela impõe a regra de que a coluna não pode conter um valor nulo (null).

Por exemplo, a tabela tb_staff pode sempre precisar de um ID e de um nome para um novo funcionário.

Exemplo: Para criar um ID e nome para novos funcionários na tabela tb_staff.

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20) DEFAULT 'R.O.C',
 ID INTEGER NOT NULL,
 name CHAR(30) NOT NULL,
 joinDate DATE DEFAULT CURDATE(),
 height FLOAT,
 degree VARCHAR(200)) IN ts_reg;
```

PRIMARY KEY AND FOREIGN KEYS

O proprietário da tabela pode especificar a chave primária ou estrangeira com o comando CREATE TABLE. Consulte a Seção 6.9, Gerenciando Integridade de Dados, para obter informações sobre chaves primárias e estrangeiras.

LOCK MODE

O modo de bloqueio de uma tabela identifica o tipo de bloqueio que o DBMaker coloca automaticamente nos objetos ao acessar o banco de dados. O DBMaker suporta três níveis de modo de bloqueio: **TABLE**, **PAGE** e **ROW**. O modo de bloqueio **ROW** é usado por padrão se o modo de bloqueio não for especificado ao criar uma tabela. Se o modo de bloqueio for definido para um nível superior (como **TABLE**), o nível de concorrência nos acessos ao banco de dados será menor, mas os recursos de

bloqueio necessários (memória compartilhada) também serão menores. Se o modo de bloqueio for definido para um nível inferior (como **ROW**), o nível de concorrência nos acessos ao banco de dados será maior, mas os recursos de bloqueio necessários (memória compartilhada) serão maiores. Em outras palavras, se um usuário inserir ou modificar linhas em uma tabela com o modo de bloqueio definido como **TABLE**, ninguém mais poderá acessar a tabela. Isso ocorre porque um bloqueio exclusivo é aplicado em toda a tabela. Para mais informações sobre modos de bloqueio, consulte a Seção 9.4, **Locks**.

Exemplo:

Para especificar o modo de bloqueio da tabela tb_staff:

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20) DEFAULT 'R.O.C',
 ID INTEGER NOT NULL,
 name CHAR(30) NOT NULL,
 joinDate DATE DEFAULT CURDATE(),
 height FLOAT,
 degree VARCHAR(200)) IN ts_reg
 LOCK MODE ROW;
```

FILL FACTOR

O recurso FILL FACTOR otimiza o uso do espaço para páginas de dados reservando espaço para a expansão de registros existentes. Ele especifica a porcentagem de uma página que pode ser preenchida antes de interromper a inserção de novos registros. Com esse método, os registros podem ser acessados com mais eficiência, evitando a necessidade de recuperar informações de um único registro em várias páginas.

Exemplo:

To set the FILLFACTOR of the tb_staff table to be 80%:

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20) DEFAULT 'R.O.C',
 ID INTEGER NOT NULL,
 name CHAR(30) NOT NULL,
 joinDate DATE DEFAULT CURDATE(),
 height FLOAT,
 degree VARCHAR(200)) IN ts_reg
 LOCK MODE ROW
 FILLFACTOR 80;
```

No caso da tabela tb_staff, novas linhas não poderão ser inseridas na página de dados depois que o espaço utilizado for maior que 80%. Os valores válidos para o FILL FACTOR variam de 50% a 100%, sendo o padrão 100%.

NOCAHE

O recurso NOCACHE é útil ao acessar tabelas grandes com uma varredura de tabela (table scan). Embora o DBMaker use buffers de página na memória compartilhada para armazenar em cache os dados recuperados e evitar E/S de disco frequente, varreduras de tabela em tabelas grandes ainda podem causar atividade frequente de E/S de disco. Isso ocorre durante uma varredura de tabela em uma tabela com um número maior de páginas de dados do que o número de buffers de página, o que esgota todos os buffers de página.

Quando a opção NOCACHE é especificada durante a criação de uma tabela, o DBMaker usa apenas um buffer de página para armazenar em cache os dados recuperados da tabela durante uma varredura de tabela (table scan). Isso evita que os buffers de página se esgotem devido a uma única varredura de tabela grande.

Exemplo:

Para especificar a opção NOCACHE:

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20) DEFAULT 'R.O.C',
 ID INTEGER NOT NULL,
 name CHAR(30) NOT NULL,
 joinDate DATE DEFAULT CURDATE(),
 height FLOAT,
 degree VARCHAR(200)) IN ts_reg
 LOCK MODE ROW
 FILLFACTOR 80
 NOCACHE;
```

TEMPORARY TABLES

Uma tabela temporária pode ser criada para armazenar dados. Tabelas temporárias existem apenas durante uma única sessão e só podem ser usadas por quem as criou. O DBMaker elimina automaticamente as tabelas temporárias quando o usuário que as criou se desconecta do banco de dados. Tabelas temporárias suportam operações de dados rápidas. Usuários clientes também podem criar uma tabela temporária local usando a sintaxe **CREATE LOCAL TEMPORARY TABLE**.

Exemplo 1:

Para criar uma tabela temporária chamada tb_student:

```
dmSQL> CREATE TEMPORARY TABLE tb_student (name CHAR(25) NOT NULL,
 birthday DATE,
 score INTEGER);
```

Exemplo 2:

Para criar uma tabela temporária local chamada tb_student:

```
dmSQL> CREATE LOCAL TEMPORARY TABLE tb_student (name CHAR(25) NOT
NULL,
 birthday DATE,
 score INTEGER);
```

Browsing Table Schema

O esquema de uma tabela pode ser consultado usando o dmSQL ou a Ferramenta JDBA. A Ferramenta JDBA fornece uma representação gráfica do esquema da tabela e permite que ele seja modificado sem a necessidade de digitar nenhum comando SQL. Também é possível usar o comando dmSQL DEF TABLE para consultar diretamente o esquema de uma tabela.

Exemplo:

Para visualizar o esquema da tabela tb_staff:

```
dmSQL> DEF TABLE tb_staff;
CREATE TABLE SYSADM.TB_STAFF (
NATION CHAR(20) default 'R.O.C' ,
ID INTEGER not null ,
NAME CHAR(30) not null ,
JOINDATE DATE default CURDATE() ,
HEIGHT FLOAT DEFAULT NULL ,
DEGREE VARCHAR(200) DEFAULT NULL )
in TS_REG LOCK MODE ROW FILLFACTOR 80 NOCACHE;
```

Altering Tables

Após a criação de uma tabela no DBMaker, um usuário com permissão de modificação pode alterá-la através de:

- **Adição/remoção de colunas:** Incluir novas colunas na tabela ou remover colunas existentes.
- **Modificação de definições de colunas:** Alterar o tipo de dados de uma coluna, seu tamanho, se permite valores nulos, etc.
- **Alteração do valor FILLFACTOR:** Define a porcentagem de preenchimento de páginas de dados, podendo afetar o desempenho de consultas.
- **Ativação/desativação da opção NOCACHE:** Determina se a tabela pode ser armazenada em cache pelo banco de dados.
- **Alteração de tabelas para outros Tablespaces:** Mover a tabela para um local de armazenamento diferente dentro do banco de dados.

O esquema de uma tabela pode ser alterado usando comandos dmSQL ou a Ferramenta JDBA.

ADDING/DROPPING COLUMNS

Um usuário com permissão de modificação pode adicionar ou remover uma ou várias colunas de uma tabela, independentemente de a coluna estar vazia ou não. Adicionar uma nova coluna a uma tabela vazia é o mesmo que expandir o esquema da tabela e colocar a nova coluna na última posição. Um usuário com permissão de modificação também pode adicionar à tabela uma nova coluna antes ou depois de qualquer coluna existente.

Ao adicionar uma nova coluna a uma tabela, o DBMaker não apenas expande o esquema da tabela, mas também preenche todas as linhas da nova coluna com valores NULL por padrão. Se um usuário com permissão de modificação quiser adicionar uma coluna com a restrição de integridade NOT NULL a uma tabela, deve fornecer um valor específico para os registros existentes na coluna (um valor padrão, conforme descrito em "Valores Padrão para Colunas"). Para sintaxe SQL detalhada, consulte a Referência de Comando e Função SQL.

Exemplo 1:

Para adicionar uma coluna chamada foto à tabela tb_staff:

```
dmSQL> ALTER TABLE tb_staff ADD COLUMN photo LONG VARCHAR;
```

Exemplo 2:

Para adicionar uma coluna chamada "cidade" após a coluna existente "nome" na tabela "tb_staff" e definir o valor padrão como 'Taipei':

```
dmSQL> ALTER TABLE tb_staff ADD COLUMN city CHAR(20) default
'Taipei' AFTER name;
```

Exemplo 3:

Se a tabela tb_staff não estiver vazia e um usuário quiser adicionar uma coluna não nula a ela, a palavra-chave GIVE pode ser usada para especificar um valor para os registros existentes na nova coluna. Para adicionar uma coluna não nula chamada HireDate à tabela tb_staff:

```
dmSQL> ALTER TABLE tb_staff ADD (HireDate date NOT NULL give '2000-
02-20');
```

Exemplo 4:

Para remover a coluna foto da tabela tb_staff:

```
dmSQL> ALTER TABLE tb_staff DROP COLUMN photo;
```

MODIFYING COLUMN DEFINITION

É possível alterar a definição de qualquer coluna existente em uma tabela, incluindo nome, tipo de dado, ordem, valor padrão, restrições, etc. Antes de modificar o tipo de dado de uma coluna, certifique-se de que o novo tipo seja compatível com o original. Caso contrário, a operação falhará devido a incompatibilidade de dados. Por exemplo, uma coluna do tipo CHAR não pode ser alterada para o tipo DATE.

Exemplo 1:

Para modificar a coluna chamada cidade na tabela tb_staff:

```
dmSQL> ALTER TABLE tb_staff MODIFY city NAME TO emp_photo;
```

Exemplo 2: Para modificar o tipo de dado da coluna altura na tabela tb_staff:

```
dmSQL> ALTER TABLE tb_staff MODIFY height TYPE TO decimal(10,2);
```

Exemplo 3:

Para alterar a ordem da coluna altura na tabela tb_staff, posicione-a antes da coluna HireDate.

```
dmSQL> ALTER TABLE tb_staff MODIFY height BEFORE HireDate;
```

Exemplo 4:

Para modificar o valor padrão da coluna nação.

```
dmSQL> ALTER TABLE tb_staff MODIFY nation DEFAULT TO 'Taiwan';
```

Exemplo 5:

Para modificar a restrição da coluna altura.

```
dmSQL> ALTER TABLE tb_staff MODIFY height CONSTRAINT TO CHECK value
< 250;
```

CHANGING THE LOCK MODE

Para obter um nível mais alto de simultaneidade em conexões simultâneas a um banco de dados, defina o modo de bloqueio para um nível inferior (como bloqueio de LINHA). No entanto, isso faz com que o DBMaker gaste mais recursos; decidir qual modo de bloqueio usar em uma tabela sempre envolve uma troca. Para obter mais informações sobre modos de bloqueio, consulte a Seção 9.4, Bloqueios.

Exemplo:

Para alterar o valor de FILLFACTOR para uma tabela:

```
dmSQL> ALTER TABLE tb_staff SET LOCK MODE PAGE;
```

CHANGING THE FILLFACTOR VALUE

A opção **FILLFACTOR** pode ser especificada durante a criação da tabela ou modificada posteriormente. Para mais informações sobre a opção **FILLFACTOR**, consulte a subseção **FILLFACTOR** em **Creating Tables**.

Exemplo:

Para alterar o valor do **FILLFACTOR** de uma tabela:

```
dmSQL> ALTER TABLE tb_staff SET FILLFACTOR 90;
```

TURNING NOCACHE ON/OFF

A opção ON/OFF pode ser usada a qualquer momento para NOCACHE. Para obter mais informações sobre a opção NOCACHE, consulte a subseção NOCACHE em "Criando Tabelas".

Exemplo:

Para **desativar** a opção NOCACHE para a tabela tb_staff:

```
dmSQL> ALTER TABLE tb_staff SET NOCACHE OFF;
```

ALTERING TABLES TO ANOTHER TABLESPACES

Você pode mover uma tabela para outro tablespace e, ao mesmo tempo, mover o índice para outro tablespace, desde que o índice e a tabela estejam no mesmo tablespace original. Caso o índice e a tabela estejam em tablespaces diferentes inicialmente, o índice não será movido automaticamente. No entanto, você pode reconstruí-lo no novo tablespace da tabela.

Para melhorar a performance da movimentação, é possível ativar a opção FASTCOPY. Com o FASTCOPY ativado, o sistema copia diretamente uma página de dados para outra, minimizando registros no arquivo de log e dispensando o uso do buffer. Isso reduz significativamente as operações repetidas de log.

Mover uma tabela para outro tablespace permite armazená-la em um disco diferente, evitando que ela fique sem espaço para novos dados caso o disco original fique cheio.

É importante estar ciente de algumas limitações:

- Os usuários não podem alterar uma tabela do sistema, tabela temporária ou visão para outro tablespace.
- Os usuários não podem mover uma tabela permanente para o SYSTABLESPACE ou TMPTABLESPACE.
- Os usuários não podem reconstruir o índice de uma tabela permanente no TMPTABLESPACE.
- Os usuários não podem reconstruir o índice de uma tabela temporária no NON-TMPTABLESPACE.
- Os usuários não podem reconstruir o índice de uma tabela do sistema em outro tablespace.
- Os usuários não podem copiar dados de uma tabela para a mesma tabela.
- Os usuários não podem mover a tabela de um tablespace para o mesmo.

Exemplo:

```
dmSQL> CREATE TABLE tb_staff (c1 int, c2 char(10)) in ts_reg; //
create table
tb_staff in ts_reg
dmSQL> CREATE INDEX idx_desc ON tb_staff (c1); // defaultly store
index in ts_reg
where the table tb_staff is stored
dmSQL> SET FASTCOPY OFF;
dmSQL> ALTER TABLE tb_staff MOVE TABLESPACE ts_app; // slowly move
table tb_staff
and index idx_desc to ts_app
dmSQL> SET FASTCOPY ON;
dmSQL> ALTER TABLE tb_staff MOVE TABLESPACE ts_app; // quickly move
table
tb_staff and index idx_desc to ts_app
dmSQL> REBUILD INDEX idx_desc FOR tb_staff IN ts_shrink; // rebuild
index
idx_desc in ts_shrink
dmSQL> ALTER TABLE tb_staff MOVE TABLESPACE ts_aut; // only move
table tb_staff
to ts_aut, index idx_desc no change
```

Using JSONCOLS Type

JSON é uma forma de intercâmbio de dados leve. É fácil de ler e escrever, além de ser facilmente analisado e gerado por máquinas. JSON é um subconjunto baseado na linguagem de programação JavaScript e ECMAScript; ele usa uma forma de texto

para armazenar e expressar dados, e essa forma de texto é completamente independente de linguagem.

O tipo **JSONCOLS** é uma expressão JSON que transforma todas as colunas dinâmicas em uma saída estruturada em uma tabela. O DBMaker suporta o tipo **JSONCOLS**. O tipo **JSONCOLS** é usado para armazenar todas as colunas dinâmicas na tabela e pode ser usado com colunas dinâmicas. Os usuários devem considerar o uso do tipo **JSONCOLS** quando o número de colunas em uma tabela for grande e os valores da maioria das colunas forem **NULL**, tornando a operação sobre elas individualmente complicada. Para mais detalhes sobre colunas dinâmicas, consulte o capítulo **Using Dynamic Columns**.

O tipo **JSONCOLS** pode ser definido com as instruções **CREATE TABLE** ou **ALTER TABLE**. Para a sintaxe SQL detalhada, consulte o capítulo **SQL Command** na Referência de Comandos e Funções SQL. Após a definição de uma coluna **JSONCOLS**, os usuários podem utilizá-la como uma coluna normal. Uma coluna **JSONCOLS** deriva de **LONG VARBINARY** e, no DBMaker, os usuários não podem criar índices em objetos grandes, portanto, não podem criar um índice em uma coluna **JSONCOLS** que tenha sido definida, mas podem criar um índice de texto nela.

Para definir o tipo **JSONCOLS**, use as palavras-chave **<JSONCOLS_type_name> JSONCOLS** nas instruções **CREATE TABLE** ou **ALTER TABLE**.

```
CREATE TABLE table-name(JSONCOLS-column-name JSONCOLS, common-
column-name datatype,…)
```

ALTER TABLE table_name ADD [COLUMN] JSONCOLS-column-name JSONCOLS;
No DBMaker, o tipo JSONCOLS é representado por uma coluna JSONCOLS. Defina a representação JSON para o tipo JSONCOLS no seguinte formato:"

```
{col1_name:col1_value,col2_name:col2_value,col3_name:col3_value...}
```

Um exemplo é o seguinte:

```
{"ID":1234,"NAME":"linda","PHONE":"1234567"}
```

NOTA: *As colunas dinâmicas que contêm valores nulos são omitidas da representação JSON para o tipo JSONCOLS."*

Se um usuário definir a representação JSON em um formato incorreto ao inserir ou atualizar dados, ocorrerá um erro e será retornada a mensagem "Error (8077): [DBMaker] invalid JSON format".

Se a representação JSON contiver um valor do tipo **DATE**, **TIME** ou **TIMESTAMP**, esse valor será exibido como **Epoch Time** no resultado da consulta quando os usuários consultarem essa coluna **JSONCOLS**. No DBMaker, o **Epoch Time** é definido como o número de milissegundos decorridos desde a meia-noite do Tempo Universal Coordenado (UTC) em 1º de janeiro de 1970.

O DBMaker armazena os dados de uma coluna **JSONCOLS** de maneira interna, portanto, a ordem de exibição das colunas dinâmicas no resultado da consulta pode ser diferente da ordem de inserção.

Se um usuário excluir uma coluna **JSONCOLS** ou a tabela que contém essa coluna **JSONCOLS**, as colunas dinâmicas armazenadas nessa coluna **JSONCOLS** serão automaticamente excluídas pelo sistema.

Ao usar o tipo **JSONCOLS**, os usuários devem considerar as seguintes diretrizes:

- O tipo **JSONCOLS** não pode ser alterado. Para alterar o tipo **JSONCOLS**, os usuários devem excluir e recriar o tipo **JSONCOLS**.
- Apenas um tipo **JSONCOLS** é permitido por tabela.
- Restrições ou valores padrão não podem ser definidos em um tipo **JSONCOLS**.

O modelo de segurança para o tipo **JSONCOLS** é semelhante ao das colunas normais, e a execução das instruções **SELECT**, **INSERT**, **UPDATE** e **DELETE** na coluna **JSONCOLS** exige que o usuário tenha as permissões correspondentes na coluna **JSONCOLS**.

A manipulação de dados de uma coluna **JSONCOLS** pode ser realizada usando o nome das colunas dinâmicas individuais ou referenciando o nome do tipo **JSONCOLS** e especificando os valores do tipo **JSONCOLS** usando a representação JSON do tipo **JSONCOLS**. As colunas dinâmicas podem aparecer em qualquer ordem na coluna **JSONCOLS**.

Exemplo:

Criando uma tabela com coluna JSONCOLS:

```
dmSQL> CREATE TABLE student(name CHAR(30), info JSONCOLS);
```

ou:

```
dmSQL> CREATE TABLE student(name CHAR(30));
dmSQL> ALTER TABLE student ADD COLUMN info JSONCOLS;
```

Inserindo dados na tabela student usando o nome do tipo JSONCOLS:

```
dmSQL> INSERT INTO student (name,info) VALUES
('jessia','{"desk_id":3,"birthday":"1986-09-19","score":90}');
1 rows inserted
dmSQL> INSERT INTO student (name,info) VALUES
('pine','{"desk_id":4,"birthday":"1987-03-03","score":95}');
1 rows inserted
```

Executar a consulta na tabela student usando "SELECT *":

```
dmSQL> SET blobwidth 80;
dmSQL> SELECT * FROM student;
 NAME INFO
==================
==================================================
jessia {"score":90,"birthday":"1986-09-19","desk_id":3}
pine {"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
```

Executando a consulta na tabela student usando o nome do tipo JSONCOLS:

```
dmSQL> SELECT name, info FROM student;
 NAME INFO
==================
==================================================
jessia {"score":90,"birthday":"1986-09-19","desk_id":3}
pine {"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
```

Updating data of table student by using the name of the JSONCOLS type

```
dmSQL> UPDATE student SET info = '{"desk_id":7, "birthday":"1986-09-
19","score":88}' WHERE name='jessia';
1 rows updated
```

Modificando o tipo de dado da coluna chamada "aniversário" para DATA:

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN birthday DATE;
dmSQL> SELECT info FROM student;
 INFO
=====================================================================
=
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
dmSQL> INSERT INTO student (name,desk_id,birthday,score) VALUES
('mike','8','1985-02-15','92');
dmSQL> SELECT info FROM student;
 INFO
=====================================================================
=
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
{"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
3 rows selected
```

Criando um índice de texto na coluna JSONCOLS chamada "info":

```
dmSQL> CREATE TEXT INDEX idx_stu ON student(INFO);
```

Criando uma visão baseada na coluna JSONCOLS chamada "info”

```
dmSQL> CREATE VIEW view1 AS SELECT info FROM student;
dmSQL> SELECT * FROM view1;
 INFO
=====================================================================
=
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
{"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
3 rows selected
```

Using Dynamic Columns

O DBMaker suporta colunas dinâmicas. Uma coluna dinâmica não existe na definição da tabela e é uma chave que pode ser derivada da string JSON, sendo usada apenas quando uma tabela declarou uma coluna como **JSONCOLS**. Colunas dinâmicas são usadas para armazenar dados semi-estruturados, especialmente registros com milhares de atributos ou dados cujo tipo de dados muda frequentemente, e podem ser usadas com o tipo **JSONCOLS**. Os usuários podem considerar o uso de colunas dinâmicas se o número de colunas em uma tabela for grande e os valores da maioria das colunas forem **NULL**. Se os usuários desejarem projetar uma tabela que possa conter colunas instáveis ou precisarem adicionar colunas devido aos requisitos da aplicação, também podem considerar o uso de colunas dinâmicas. Para detalhes do tipo **JSONCOLS**, consulte o capítulo **Using JSONCOLS Type**.

Colunas dinâmicas são armazenadas em uma coluna **JSONCOLS**, e as informações de descrição sobre as colunas dinâmicas são armazenadas em **SYSDESCOL**. Para detalhes sobre **SYSDESCOL**, consulte o capítulo **DBMaker System Catalog Tables**. Para detalhes sobre uma coluna **JSONCOLS**, consulte o capítulo **Using JSONCOLS Type**.

O modelo de segurança para colunas dinâmicas é semelhante ao das colunas normais. Colunas dinâmicas comportam-se como quaisquer outras colunas com as seguintes características:

- Um índice pode ser criado em uma coluna dinâmica.
- Uma coluna dinâmica só suporta a modificação do tipo de dado.
- Uma coluna dinâmica suporta os seguintes tipos de dados: **BIGINT**, **SMALLINT**, **INT**, **FLOAT**, **DOUBLE**, **DATE**, **TIME**, **TIMESTAMP**, **CHAR**, **VARCHAR**, **NCHAR**, **NVARCHAR**.
- O tipo de dado padrão de colunas dinâmicas é **varchar(256)**.
- Uma coluna dinâmica deve ser nula.
- Uma coluna dinâmica não pode ter um valor padrão.
- Uma coluna dinâmica não pode ter uma restrição de coluna.
- Uma coluna dinâmica não pode ser usada em um comando armazenado.
- Uma coluna dinâmica não pode ser usada em um procedimento armazenado.
- Uma coluna dinâmica não pode ser usada em um gatilho.
- Após a criação de uma coluna **JSONCOLS**, as colunas dinâmicas podem ser usadas diretamente sem definição adicional.

Os usuários podem alterar o tipo de dado padrão para outro tipo de dado com o comando **ALTER TABLE ADD DYNAMIC COLUMN**. Além disso, os usuários também podem declarar o tipo de dado de uma coluna dinâmica com esse comando quando

a coluna dinâmica é inserida na tabela. Se os usuários quiserem alterar o tipo de dado declarado com esse comando para outro tipo de dado, o comando **ALTER TABLE MODIFY DYNAMIC COLUMN** pode ser usado. Além disso, se um usuário desejar remover as informações de descrição de uma coluna dinâmica, o comando **ALTER TABLE DROP DYNAMIC COLUMN** pode ser usado. Para a sintaxe SQL detalhada, consulte o capítulo **SQL Command** na Referência de Comandos e Funções SQL. No entanto, se um usuário primeiro inserir dados sem executar **ALTER TABLE ADD DYNAMIC COLUMN**, mas os dados inseridos não puderem ser convertidos para o tipo de dado que é posteriormente declarado com esse comando pelo usuário, os dados serão exibidos como **NULL** quando uma instrução de consulta for executada e nenhum erro ocorrer.

Além disso, quando um usuário insere dados em uma coluna dinâmica ou atualiza dados de uma coluna dinâmica com parâmetros, se o usuário não definir o tipo de descrição da coluna dinâmica, o DBMaker suporta por padrão a inserção de dados com o tipo **VARCHAR(256)**. Nesse momento, o usuário pode inserir dados com outros tipos de dados usando a função de conversão implícita de dados. Para detalhes sobre a função de conversão implícita de dados, consulte a seção **Implicit Data Conversion** do manual de Referência de Comandos e Funções SQL.

A manipulação de dados de colunas dinâmicas pode ser realizada usando o nome das colunas dinâmicas individuais ou referenciando o nome do tipo **JSONCOLS** e especificando os valores do tipo **JSONCOLS** usando a representação JSON do tipo **JSONCOLS**. Colunas dinâmicas podem aparecer em qualquer ordem na coluna **JSONCOLS**.

NOTA: Os usuários podem usar JSONPath para inserir, atualizar e selecionar dados em jsoncols e colunas dinâmicas. Consulte o Guia do Usuário JSONPath para mais informações.

Exemplo:

As seguintes operações são baseadas na tabela student. Para detalhes sobre a tabela student, consulte o exemplo em "Usando o Tipo JSONCOLS".

Inserindo dados na tabela student usando os nomes das colunas dinâmicas:

```
/* implicit data conversion is closed by default */
dmSQL> INSERT INTO student (name,score) VALUES(?,?);
dmSQL/Val> 'demi','85'; /* it is ok */
1 rows inserted
dmSQL/Val> 'finly',82; /* INT cannot be converted to CHAR */
ERROR (9629): value list syntax error
```

```
dmSQL/Val> END;
dmSQL> SET itcmd ON;
dmSQL> INSERT INTO student (name,score) VALUES(?,?);
dmSQL/Val> 'finly',82; /* using implicit data conversion */
1 rows inserted
dmSQL/Val> END;
dmSQL> SET itcmd OFF;
dmSQL> INSERT INTO student (name,desk_id,birthday,score)
VALUES('linda','1','1982-01-01','91');
1 rows inserted
dmSQL> INSERT INTO student (name,desk_id,birthday,score)
VALUES('glow','2','1984-
03-25','93');
1 rows inserted
dmSQL> INSERT INTO student (name,desk_id,birthday,score)
VALUES('kitty','abc','1980-02-27','97');
1 rows inserted
```

Consultando a tabela student usando "SELECT *":

```
dmSQL> SELECT * FROM student;
 NAME INFO
============== ================================================
jessia {"score":88,"birthday":"1986-09-19","desk_id":7}
pine {"score":95,"birthday":"1987-03-03","desk_id":4}
mike {"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
demi {"SCORE":"85"}
finly {"SCORE":"82"}
linda {"BIRTHDAY":378662400000,"DESK_ID":"1","SCORE":"91"}
glow {"BIRTHDAY":448992000000,"DESK_ID":"2","SCORE":"93"}
kitty {"BIRTHDAY":320428800000,"DESK_ID":"abc","SCORE":"97"}
8 rows selected
```

Consultando a tabela student usando os nomes das colunas dinâmicas:

```
dmSQL> SELECT name, desk_id, birthday, score FROM student;
 NAME DESK_ID BIRTHDAY SCORE
============== =========== ============ =====================
jessia 7 19* 88
pine 4 19* 95
mike 8 19* 92
```

```
demi NULL NU* 85
finly NULL NU* 82
linda 1 19* 91
glow 2 19* 93
kitty abc 19* 97
8 rows selected
```

Atualizando/excluindo dados da tabela student usando os nomes das colunas dinâmicas:

```
dmSQL> UPDATE student SET score='88' WHERE name='linda';
1 rows updated
dmSQL> DELETE FROM student WHERE desk_id='2';
1 rows deleted
```

Adicionando descrição de colunas dinâmicas a esta tabela:

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN desk_id INT;
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN score DOUBLE;
```

Inserindo dados na tabela student:

```
dmSQL> INSERT INTO student (name, desk_id, age, score)
VALUES('jane','12','1982-
05-07',96);
ERROR (6150): [DBMaker] the insert/update value type is incompatible
with column
data type or compare/operand value is incompatible with column data
type in
expression/predicate
dmSQL> INSERT INTO student (name, desk_id, age, score)
VALUES('jim',8,'1984-09-
26',98);
1 rows inserted
dmSQL> SELECT name, desk_id, birthday, score FROM student;
 NAME DESK_ID BIRTHDAY SCORE
============== ============ ============ =====================
jessia 7 1986-09-19 8.80000000000000e+001
pine 4 1987-03-03 9.50000000000000e+001
mike 8 1985-02-15 9.20000000000000e+001
demi NULL NULL 8.50000000000000e+001
```

```
finly NULL NULL 8.20000000000000e+001
linda 1 1982-01-01 8.80000000000000e+001
kitty NULL 1980-02-27 9.70000000000000e+001
jim 8 NULL 9.80000000000000e+001
8 rows selected
```

Modificando o tipo de dado da coluna dinâmica chamada score:

```
dmSQL> ALTER TABLE student MODIFY DYNAMIC COLUMN score TYPE TO INT;
```

Criando um índice na coluna com nome gerado dinamicamente chamada desk_id

```
dmSQL> CREATE INDEX idx1 ON student(desk_id);
```

Removendo metadados da coluna dinâmica chamada birthday:

```
dmSQL> ALTER TABLE student DROP DYNAMIC COLUMN birthday;
```

Using Dynamic Columns with JSONPATH

Os tipos de dados JSON são usados para armazenar dados em JSON (JavaScript Object Notation), conforme especificado na RFC 7159. Os tipos de dados JSON têm a vantagem de garantir que cada valor armazenado seja válido de acordo com o JSON.

As expressões de caminho SQL/JSON especificam os itens a serem recuperados dos dados JSON, semelhantes às expressões XPath usadas para acesso SQL a XML. No DBMaker, as expressões de caminho são implementadas como o tipo de dado jsonpath.

Também estão disponíveis diversas funções e operadores específicos para JSON para dados armazenados nesses tipos de dados.

As funções e operadores de consulta JSON passam a expressão de caminho fornecida para o mecanismo de caminho para avaliação. Se a expressão corresponder aos dados JSON a serem consultados, o item SQL/JSON correspondente é retornado.

As expressões de caminho são escritas na linguagem de caminho SQL/JSON e também podem incluir expressões e funções aritméticas. As funções de consulta

tratam a expressão fornecida como uma string de texto, portanto, ela deve estar entre aspas duplas.

Uma expressão de caminho consiste em uma sequência de elementos permitidos pelo tipo de dado jsonpath. A expressão de caminho é avaliada da esquerda para a direita, mas você pode usar parênteses para alterar a ordem das operações. Se a avaliação for bem-sucedida, uma sequência de itens SQL/JSON (sequência SQL/JSON) é produzida, e o resultado da avaliação é retornado à função de consulta JSON que completa o cálculo especificado.

- A seguir está uma visão geral completa sobre o elemento de sintaxe JSONPath.

JSONPath	Descration
$	Objeto/elementos raiz
@	Objeto/elementos atuais
. or []	Nó filho
..	Descida recursiva
*	Caractere curinga
[]	Operador de índice (ou subscrito)
[,]	Operador de união para combinar conjuntos de nós. JSONPath permite nomes alternativos ou índices de array como um conjunto. Array slice operator: Operador de fatiamento de array.
[start:end:step]	Operador de fatiamento de array.
?()	Aplica uma expressão de filtro (script)
()	Expressão de script.

Expressões JSONPath podem utilizar a notação ponto.

```
$.store.book[0].title
```

Notação entre colchetes:

```
$['store']['book'][0]['title']
```

Expressões da linguagem de script subjacente () podem ser usadas como alternativa a nomes ou índices explícitos, como em:

```
$.store.book[(@.length-1)].title
```

Usando o símbolo @ para o objeto atual. Expressões de filtro são suportadas pela sintaxe ?() como em:

```
$.store.book[?(@.price<100)].title
```

NOTA: *No JSONPath, os colchetes operam no objeto ou array indicado pelo fragmento anterior do caminho. Os índices sempre começam em 0.*

Exemplo:

Aqui está um exemplo de dados JSON de um rastreador de livros que você pode analisar. Salve o texto JSON em um arquivo chamado "book.json". Usaremos esse arquivo nos próximos exemplos.

```
book.json:
{ "store" :
 {
 "book" : [
 { "category": "reference",
 "author": "Nigel Rees",
 "title": "Sayings of the Century",
 "price": 8.95
 },
 { "category": "fiction",
 "author": "Evelyn Waugh",
 "title": "sword of Honour",
 "price": 12.99
 },
 { "category": "fiction",
 "author": "Herman Melville",
 "title": "Moby Dick",
 "isbn": "0-553-21311-3",
 "price": 8.99
 },
 { "category": "fiction",
 "author": "J. R. R Tolkien",
 "title": "The Load of the Rings",
 "isbn": "0-395-19395-8",
 "price": 22.99
 }
 ],
"bicycle": {
 "color": "red",
 "price": 19.95
}
```

```
 }
}
```

Exemplo:

Aqui está uma visão geral completa do exemplo JSONPath

JSONPath	Result
$.store.book[*].author	Os autores de todos os livros da livraria.
$..author	Todos os autores
$.store.*	Considerando tudo o que há na loja, inclua alguns livros e uma bicicleta
$.store..price	O preço de tudo na loja
$..*	Para obter todos os elementos de um objeto JSON:
$..book[2]	O terceiro livro
$..book[(@.length-1)]	O último livro da série
$..book[-1:]	Os dois primeiros livros
$..book[0,1]	Filtrar todos os livros com número ISBN
$..book[:2]	Filtrar todos os livros por menos de 10
$..book[?(@.isbn)]	Filtrar todos os livros por menos de 10
$..book[?(@.price<10)]	Filtrar todos os livros com preço inferior a 10.
$..book[?(@.price<10)]	Filtrar todos os livros com preço inferior a 10.

- **The child node (. ou [])**

O DBMaker recomenda que o usuário utilize diretamente o nó filho (ponto "." ou colchetes "[]") na consulta JSONPath. Desse modo, a consulta deverá ser mais eficiente.

Por exemplo:

```
$['store']['book'][0]['title']


e:

$.store.book[0].title


e:
```

```
$['store']['book'][0]['title']
```

- **The expr node** (.. ou * ou (v1 expr v2) ou (@.length))

O nó de expressão precisa realizar um cálculo no momento da execução durante a varredura.

Observação: O nó de expressão/filtro precisa usar () para delimitar a expressão a ser avaliada.

Os v1/v2 podem ser um valor constante ou o nome do elemento atual (@).

=, ==, !=, >, >=, <, <=, <>, +, -, *, /, %, AND, &&, OR, ||. **@.length é uma expressão avaliada em tempo de execução no JSONPath.**

NOTA: O nó @.length só permite calcular o tamanho do nó atual. Por isso, a expressão $..book[(@.length-1)].title não é suportada.

- **The slice node([num1 :num2] or ,)**

num1, num2, num3, ... é a lista de marcas definida pelo usuário. num1 e num2 representam as extremidades do array; se forem null, isso significa que correspondem à posição inicial ou final do array.

- T**he fileter node(?(@.element) or ?(expr))** O nó de filtro verifica TRUE/FALSE com um nó de expressão. O nó de filtro precisa usar ? para checar com o resultado da expressão. (@.element) significa se o elemento atual existe na varredura atual.

Exemplo:

As operações a seguir se baseiam em JSONPath de tabela, que inclui os dados JSON anteriores. Primeiramente, é necessário criar o JSONPath de tabela e garantir que os textos JSON sejam inseridos nele.

```
CREATE TABLE the table jsonpath.
dmSQL> CREATE TABLE jsonpath(C0 JSONCOLS);
dmSQL> INSERT INTO jsonpath VALUES(?);
dmSQL/Val> &'book.json’;
1 rows inserted
dmSQL/Val> END;
```

Exemplo:

```
dmSQL> SELECT "$.store.book[0].title" FROM jptab;
 $.store.book[0].title
======================
Sayings of the Century
1 rows selected
dmSQL> SELECT "$.store.book[0].author" FROM jptab;
$.store.book[0].author
======================
Nigel Rees
1 rows selected
dmSQL> SELECT "$.store.bicycle" FROM jptab;
$.store.bicycle
=======================
{"author":"Nigel
Rees","price":8.95,"category":"reference","title":"Sayings of
the Century"}
1 rows selected
dmSQL> SELECT "$.store.book[0]" FROM jptab;
$.store.book[1]
=======================
{"author":"Evelyn
Waugh","price":12.99,"category":"fiction","title":"sword of
Honour"}
1 rows selected
dmSQL> SELECT "$.store.book[0].author" FROM jptab;
$.store.book[0].author
Nigel Rees
1 rows selected
For more expression:
dmSQL> select "$.store.book[(@.length-1)]" from jptab;
$.store.book[(@.length-1)]
============================================================
============
[{"author":"J. R. R Tolkien","title":"The Load of the
Rings","isbn":"0-395-19395-
8","price":22.99,"category":"fiction"}]
1 rows selected
dmSQL> select "$..book[?(@.isbn)].title" from jptab;
$..book[?(@.isbn)].title
============================================================
============
```

```
["Moby Dick","The Load of the Rings"]
1 rows selected
dmSQL> select "$..book[?(@.price<10)].price" from jptab;
$..book[?(@.price<10)].price
================================================================================
============
[8.95,8.99]
1 rows selected
mSQL> select "$..book[?(@.price==8.99)].price" from jptab;
$..book[?(@.price==8.99)].price
================================================================================
============
[8.99]
1 rows selected
dmSQL> select "$..book[?(@.price>=8.99)].price" from jptab;
$..book[?(@.price>=8.99)].price
================================================================================
============
[22.99,12.99,8.99]
1 rows selected
dmSQL> select "$..book[(1+1+1)].price" from jptab;
$..book[(1+1+1)].price
================================================================================
============
[22.99]
1 rows selected
dmSQL> select "$..book[:2].title" from jptab;
$..book[:2].title
================================================================================
============
["Sayings of the Century","sword of Honour"]
1 rows selected
dmSQL> select "$..book[1:4:1].author" from jptab;
$..book[1:4:1].author
================================================================================
============
["Herman Melville","J. R. R Tolkien","Evelyn Waugh"]
1 rows selected
```

Locking Tables

Embora o DBMaker gerencie automaticamente o mecanismo de bloqueio sempre que um banco de dados é acessado, uma tabela pode ser bloqueada manualmente para instruções SELECT ou UPDATE subsequentes. Bloquear uma tabela enquanto um usuário a visualiza ou modifica impede atualizações por outras pessoas.

O DBMaker oferece algumas opções para bloqueio de tabelas, como bloqueios compartilhados para visualização de dados ou bloqueios exclusivos para modificação de dados, e a opção WAIT (esperar) ou NO WAIT (não esperar) usada ao obter um bloqueio. Para obter mais informações sobre esses recursos, consulte a Referência de Comandos e Funções SQL. Para saber mais sobre bloqueios de tabela, controle de simultaneidade e tratamento de transações, consulte o Capítulo 9, Controle de Simultaneidade.

Exemplo:

Para bloquear a tabela tb_staff para seleções posteriores e **não esperar** se não conseguir o bloqueio imediatamente:

```
dmSQL> LOCK TABLE tb_staff IN SHARE MODE NO WAIT;
```

Dropping Tables

Um usuário pode excluir uma tabela quando ela não for mais necessária. Ao excluir uma tabela, todos os dados e índices associados a ela são removidos, e as páginas alocadas pela tabela excluída são liberadas.

Exemplo 1:

Para excluir a tabela tb_staff, use o comando DROP TABLE:

```
dmSQL> DROP TABLE tb_staff;
```

Exemplo 2:

Para excluir a tabela tb_staff somente se ela existir, use o comando DROP TABLE IF EXISTS:

```
dmSQL> DROP TABLE IF EXISTS tb_staff;
```

6.3 Managing Views

O DBMaker permite definir tabelas virtuais chamadas viwes. Essas visões são baseadas em tabelas existentes e armazenadas como uma definição com um nome de visão definido pelo usuário. A definição da visão é armazenada permanentemente no banco de dados, mas os dados reais não são armazenados fisicamente lá. Em vez disso, os dados são armazenados nas tabelas base originais das quais as views foram derivadas. Uma views é definida por uma consulta que referencia uma ou mais tabelas ou outras views.

As views são um mecanismo muito útil em um banco de dados. Por exemplo, consultas complexas podem ser definidas uma vez e usadas repetidamente sem a necessidade de serem reescritas. Além disso, as visões podem ser usadas para melhorar a segurança do banco de dados, restringindo o acesso a um conjunto predeterminado de linhas ou colunas em uma tabela.

Como uma visão é derivada de consultas em tabelas, um usuário não pode determinar a partir dela quais linhas das tabelas devem ser atualizadas. Devido a essa limitação, as visões geralmente só podem ser consultadas, a menos que a visão seja derivada de uma única tabela.

Creating Views

As views podem ser criadas usando dmSQL ou a ferramenta JDBA. Uma visão é definida por um nome e uma consulta que referencia tabelas ou outras views.

Os usuários podem especificar uma lista de nomes de coluna para a views. Se os nomes das colunas não forem especificados, a views herdará os nomes das colunas das tabelas subjacentes.

Utilize a sintaxe CREATE VIEW. Por exemplo, para permitir que outros usuários vejam apenas duas colunas da tabela tb_staff, crie uma views com o comando SQL mostrado abaixo. Através da views vi_staff, os usuários poderão visualizar apenas duas colunas (nome e ID) da tabela tb_staff.

Exemplo 1:

```
dmSQL> CREATE VIEW vi_staff (empName, empId) AS SELECT name, ID FROM
tb_staff;
```

Utilize a sintaxe CREATE OR REPLACE VIEW. Suponha que exista uma visão chamada vi_staff que atualmente permite que outros usuários vejam apenas duas colunas, nome e ID, da tabela tb_staff. No entanto, você precisa modificar a definição da visão para permitir que esses mesmos usuários agora visualizem três colunas da tabela tb_staff, sem alterar as permissões de acesso à visão.

Para isso, você pode utilizar o comando CREATE OR REPLACE VIEW, conforme mostrado no exemplo abaixo. Ele irá substituir a visão vi_staff existente por uma nova definição que inclui a coluna idade:

Exemplo 2:

```
dmSQL> CREATE OR REPLACE VIEW vi_staff (empName, empId, empAge) AS
SELECT name,
ID, age FROM tb_staff;
```

Browsing View Schema

A construção de uma visão pode ser consultada usando dmSQL ou a ferramenta JDBA. Utilize o comando dmSQL DEF VIEW para consultar diretamente o schema de uma tabela.

Exemplo:

Para visualizar a construção da views vi_staff:

```
dmSQL> DEF VIEW vi_staff;
dmSQL> CREATE VIEW_SYSADM.VI_STAFF(empname,empid) AS SELECT name,id
from
SYSADM.TB_STAFF;
```

Dropping Views

Uma visão pode ser removida quando não for mais necessária. Ao remover uma visão, apenas a definição armazenada no catálogo do sistema é eliminada. As tabelas base das quais a visão foi derivada não são afetadas.

Exemplo 1: Para remover a visão vi_staff, utilize o comando DROP VIEW:

```
dmSQL> DROP VIEW vi_staff;
```

Exemplo 2:

Para remover a visão vi_staff, utilize o comando DROP VIEW IF EXISTS:

```
dmSQL> DROP VIEW IF EXISTS vi_staff;
```

6.4 Managing Synonyms

Um sinônimo é um alias para qualquer tabela ou visão. Como um sinônimo é apenas um apelido, ele não requer armazenamento além de uma definição no catálogo do sistema. Sinônimos são úteis para simplificar o nome totalmente qualificado de uma tabela ou visão. O DBMaker normalmente identifica tabelas e visões com nomes totalmente qualificados, que são compostos pelo nome do proprietário e do objeto. Ao usar um sinônimo, qualquer pessoa pode acessar uma tabela ou visão usando o sinônimo correspondente, sem precisar utilizar o nome totalmente qualificado. Como um sinônimo não possui nome de proprietário, cada sinônimo no banco de dados deve ser único para que o DBMaker possa identificá-los. Os sinônimos podem ser criados ou removidos com dmSQL ou a ferramenta JDBA.

Creating Synonyms

Exemplo 1:

Você pode criar sinônimos utilizando o comando CREATE SYNONYM:

```
dmSQL> CREATE SYNONYM staff FOR SYSADM.tb_staff;
```

Vamos supor que o dono da tabela tb_staff seja SYSADM. O comando a seguir cria o alias staff para a tabela SYSADM.tb_staff. Com isso, todos os usuários do banco de dados poderão referenciar diretamente a tabela SYSADM.tb_staff através do sinônimo staff. Exemplo 2:

Utilize o comando CREATE OR REPLACE SYNONYM:

```
dmSQL> CREATE OR REPLACE SYNONYM staff FOR SYSADM.tb_staff;
```

Suponha que um alias "staff" para a tabela SYSAMD.tb_staff já exista. Você pode substituí-lo sem precisar excluí-lo.

Dropping Synonyms

Um sinônimo que não é mais necessário pode ser excluído. Quando um sinônimo é excluído, apenas sua definição é removida do catálogo do sistema.

Exemplo 1:

Para excluir o sinônimo "staff" com o comando DROP SYNONYM:

```
dmSQL> DROP SYNONYM staff;
```

Exemplo 2:

Para excluir o sinônimo "staff" com o comando DROP SYNONYM IF EXISTS:

```
dmSQL> DROP SYNONYM IF EXISTS staff;
```

6.5 Managing Indexes

Um índice fornece suporte para acesso rápido e aleatório a uma linha em uma tabela. A criação de índices para uma tabela acelera as pesquisas. Por exemplo, quando um usuário executa a consulta SELECT NAME FROM tb_staff WHERE id = 306004, é possível recuperar os dados em um tempo muito menor se houver um índice criado para a coluna ID.

Um índice pode ser composto por mais de uma coluna, até um máximo de 32. Todas as colunas da tabela podem ser usadas em um índice.

Um índice pode ser único ou não único. Em um índice único, no máximo uma linha pode ter o mesmo valor de chave, com a exceção de que qualquer número de linhas pode ter valores NULL. Se um usuário criar um índice único em uma tabela, o DBMaker verificará se todas as chaves existentes são distintas ou não. Se houver chaves duplicadas, o DBMaker retornará uma mensagem de erro. Após a criação de um índice único em uma tabela, se um usuário inserir uma linha na tabela, o DBMaker garante que não haja linhas existentes com a mesma chave que a nova linha.

Ao criar um índice, a ordem de classificação de cada coluna do índice pode ser especificada como crescente ou decrescente. Por exemplo, suponha que haja cinco chaves em uma tabela com os valores 1, 3, 9, 2 e 6. Em ordem crescente, a sequência de chaves no índice é 1, 2, 3, 6 e 9, e em ordem decrescente, a sequência de chaves no índice é 9, 6, 3, 2 e 1.

Quando um usuário executa uma consulta, a ordem do índice pode ocasionalmente afetar a ordem de saída dos dados.

Exemplo:

Suponha que a seguinte query seja executada:

```
dmSQL> SELECT name, age FROM friend_table WHERE age > 20
```

Usando um índice com ordem decrescente na coluna 'idade', a saída seria semelhante a:

```
     name               age
---------------- ----------------
     Jeff                49
     Kevin               40
     Jerry               38
     Hughes              30
     Cathy               22
```

Um usuário pode especificar o fill factor para tabelas ao criar um índice. O fill factor indica o quão densas serão as chaves nas páginas do índice. Os valores legais do fill factor estão na faixa de 1% a 100%, e o padrão é 100%. Se um usuário atualiza dados frequentemente após a criação do índice, ele pode definir um fill factor esparso no índice, por exemplo 60%. Se o usuário nunca atualizar os dados nesta tabela, o fill factor pode ser deixado no valor padrão de 100%.

Um usuário também pode especificar a criação de um índice em um tablespace separado. Isso pode resultar em melhor E/S de disco para pesquisas que usam o índice se vários discos forem utilizados.

Antes de criar índices em uma tabela, é recomendável carregar todos os dados primeiro, especialmente se houver uma grande quantidade de dados para essa tabela. Se um usuário criar um índice antes de carregar os dados em uma tabela, os índices serão atualizados sempre que o usuário carregar uma nova linha. É muito mais eficiente criar um índice após carregar uma grande quantidade de dados do que criar um índice antes de carregar os dados.

Creating Indexes

Índices podem ser criados usando o assistente Criar Índice da ferramenta JDBA ou o comando dmSQL CREATE INDEX. Para criar um índice em uma tabela, especifique o nome do índice e as colunas do índice. Especifique a ordem de classificação de cada coluna como ascendente ou descendente. A ordem de classificação padrão é ascendente.

Exemplo:

Para criar um índice, chamado idx_desc, na coluna ID da tabela tb_staff em ordem decrescente, use a opção DESC:

```
dmSQL> CREATE INDEX idx_desc ON tb_staff (ID DESC);
```

Exemplo:

Para criar um índice único, chamado idx_uniq, na coluna ID da tabela tb_staff, use a opção UNIQUE:

```
dmSQL> CREATE UNIQUE INDEX idx_uniq ON tb_staff (ID);
```

Exemplo 3:

Para criar um índice com um fill factor específico, use a opção FILLFACTOR:

```
dmSQL> CREATE INDEX idx_fill ON tb_staff (name, id DESC) FILLFACTOR
60;
```

Exemplo 4:

Para criar um índice em um tablespace específico:

```
dmSQL> CREATE INDEX idx_reg ON tb_staff (name, id DESC) IN ts_reg
FILLFACTOR 60;
```

Creating Expression Indexes

Índices podem ser criados não apenas em colunas simples, mas também em colunas de expressões ou em colunas que utilizam funções definidas pelo usuário (UDFs).

Exemplo 1:

Para criar um índice, chamado idx_expr, na expressão basepay+bonus da tabela tb_salary:

```
dmSQL> CREATE INDEX idx_expr ON tb_salary (basepay+bonus);
```

Exemplo 2:

Para criar um índice, chamado idx_substr, na função substring(nation,1,3) da tabela tb_salary:

```
dmSQL> CREATE INDEX idx_substr ON tb_salary (substring(nation,1,3)
desc);
```

Exemplo 3:

Para criar um índice, chamado idx_udf, na expressão que envolve a função definida pelo usuário (UDF) abs(bonus) para a tabela tb_salary:

```
dmSQL> CREATE INDEX idx_udf ON tb_salary(basepay+abs(bonus)-tax
desc);
```

Creating Indexes on XML column

Para melhorar o desempenho de consultas XML, podemos criar índices XML especiais em colunas XML. O índice XML suporta UDFs XML: extract() e extractvalue(). O exemplo a seguir mostra como criar um índice em uma coluna XML usando dmSQL. Consulte a Referência de Comando e Função SQL para obter detalhes adicionais sobre a sintaxe e uso do comando SQL CREATE INDEX.

Exemplo 1:

Para criar um índice usando a UDF XML extract:

```
dmSQL> CREATE INDEX idx_extr ON tb_extract (extract(id,
'/order/items/item/@product', NULL));
```

Exemplo 2:

Para criar um índice usando a UDF XML extractValue:

```
dmSQL> CREATE INDEX idx_extrV ON tb_extract (extractValue(id,
'/order/items/item/@product', NULL));
```

A principal diferença entre extract() e extractvalue() é:

extract()

- Permite um resultado com vários valores, um único valor ou nenhum valor.
- ordenação ascendente (asc) / descendente (desc) não é permitida
- índice único não é permitido

extractValue()

- Permite um único valor ou nenhum valor do resultado da UDF (quando o resultado da UDF é um valor com vários valores, a criação do índice falha para tuplas existentes e a inserção de dados falha para tuplas recém-inseridas)
- Permite ordenação ascendente (asc) / descendente (desc)
- Permite índice único

Creating Filtered Indexes

Índices Filtrados são índices que possuem uma cláusula WHERE associada. Imagine um índice otimizado, especialmente útil para consultas que selecionam um subconjunto bem definido de dados. Ao contrário de um índice normal que engloba todas as linhas da tabela, o Índice Filtrado é aplicado antes da filtragem propriamente dita. Isso significa que seus dados **não incluem todas as linhas**, apenas uma parte delas, definida pela condição de filtro (cláusula WHERE). Ele utiliza um predicado de filtro para selecionar uma porção específica das linhas da tabela. Um Índice Filtrado bem projetado pode melhorar o desempenho de consultas, além de reduzir custos de manutenção e armazenamento de índices comparado a índices de tabela completa.

Um Índice Filtrado bem projetado **melhora a performance de consultas e a qualidade do plano de execução** por ser menor que um índice de tabela completa e possuir estatísticas filtradas. Essas estatísticas filtradas são mais precisas do que as estatísticas de tabela completa, pois cobrem apenas as linhas presentes no índice filtrado.

A manutenção de um índice só ocorre quando instruções de linguagem de manipulação de dados (DML) afetam os dados contidos nele. Como um Índice Filtrado é menor e só é mantido quando seus dados sofrem alterações, ele resulta em **menor custo de manutenção** comparado a um índice de tabela completa. É possível ter um grande número de Índices Filtrados, especialmente quando contêm dados

modificados com pouca frequência. Da mesma forma, se um índice filtrado contém apenas dados frequentemente modificados, o tamanho menor do índice reduz o custo de atualização das estatísticas.

Criar um Índice Filtrado pode **reduzir o armazenamento em disco**. É possível substituir um índice de tabela completa por múltiplos Índices Filtrados sem aumentar significativamente os requisitos de armazenamento.

A instrução CREATE INDEX com a cláusula WHERE define um índice sobre uma tabela existente. O tipo de índice filtrado permite somente índices não-únicos e índices únicos, mas **não permite chave primária**. O número máximo de colunas que podem ser incluídas na cláuscula WHERE é 32. O DBMaker permite que **um dos seguintes usuários** crie um índice filtrado:

- Usuários com autoridade de DBA (Autoridade de Banco de Dados) ou superior
- O proprietário da tabela
- Usuários com privilégio CREATE INDEX concedido

A cláusula WHERE para Índices Filtrados pode ser qualquer combinação dos seguintes predicados:

- **Qualquer coluna da tabela**
- **Valores constantes**
- **Operadores de comparação:** =, >, >=, <, <=, !=. Ex: c1 >= 3
- **LIKE:** Ex: c3 like 'abc'
- **IS NULL, IS NOT NULL:** Ex: c4 is null
- **Lista IN:** Ex: c5 in (1, 3, 5)
- **Operadores matemáticos:** +, -, *, /. Ex: c1 + c2 > 5
- **Funções Definidas pelo Usuário (UDFs):** Ex: abs(c6) > 5
- **Operadores BLOB:** match, contain
- **Combinação de AND:** Ex: c1 = 3 and c2 = 5 and c3 = 7
- **Combinação de OR:** Ex: c1 = 3 or c2 = 5 or c3 = 7

A cláusula WHERE para Índices Filtrados **não permite** as seguintes construções:

- **Subconsultas**
- **Variáveis de sessão** (host variable)
- **Combinações mistas de AND e OR:** Ex: c1=3 **or** c2=5 **and** c3=7

NOTA: O usuário **não pode especificar uma condição filtrada (WHERE clause)** em um índice de texto.

Exemplo 1: Criando um índice filtrado filidx_basepay usando predicado LIKE na tabela tb_salary:

```
dmSQL> CREATE INDEX filidx_basepay ON tb_salary (id, basepay) where
name like
'abc%';
```

Exemplo 2:

Criando um índice filtrado filidx_income usando cláusula WHERE na tabela tb_salary:

```
dmSQL> CREATE INDEX filidx_income ON
tb_salary(basepay+bonus,tax)where id>30;
```

Dropping Indexes

Índices podem ser removidos usando a ferramenta JDBA ou a instrução dmSQL DROP INDEX. Se o índice for uma chave primária e for referenciado por outras tabelas, ele não poderá ser removido. Para obter informações sobre chaves primárias, consulte a seção Gerenciando Integridade de Dados.

Exemplo:

Para remover o índice `idx_desc` da tabela `tb_staff`:

```
dmSQL> DROP INDEX idx_desc FROM tb_staff;
```

Disabling Indexes

Os índices podem ser desativados usando a ferramenta JDBA ou o comando dmSQL DISABLE INDEX. Desativar um índice não o remove, apenas o desativa. Para desativar um índice em uma tabela, especifique o nome do índice e o nome da tabela. Os usuários podem usar o comando REBUILD ``INDEX para reconstruir o índice.

Use o seguinte comando para desativar um índice:

```
DISABLE INDEX index_name FOR table_name;
```

Exemplo 1: Para desativar o índice idx_fill na tabela tb_staff:

```
dmSQL> DISABLE INDEX idx_fill FOR tb_staff;
```

Nota: Os usuários podem usar "ALL" para representar todos os índices na tabela ao usar a instrução DISABLE INDEX ou REBUILD INDEX.

Exemplo 2: Para desativar todos os índices da tabela tb_staff, use o seguinte comando:

```
dmSQL> DISABLE INDEX ALL FOR tb_staff;
```

Rebuilding Indexes

Índices podem ser reconstruídos usando a ferramenta JDBA ou a instrução dmSQL REBUILD INDEX. Geralmente, a reconstrução de um índice é necessária quando ele se fragmenta, o que reduz sua eficiência. Reconstruir um índice significa remover o antigo e criar um novo.

Um usuário pode mover uma tabela para outro tablespace. Se o índice e a tabela estiverem no mesmo tablespace, o índice também será movido. No entanto, se o índice e a tabela estiverem em tablespaces diferentes, o índice não será movido junto. Nesse caso, para manter a consistência, é recomendável reconstruir o índice no novo tablespace da tabela.

Exemplo 1:

Para reconstruir o índice idx_fill da tabela tb_staff

```
dmSQL> REBUILD INDEX idx_fill FOR tb_staff;
```

Exemplo 2:

Para reconstruir o índice idx na tabela tb_staff no tablespace ts_reg, use o seguinte comando:

```
dmSQL> REBUILD INDEX idx FOR tb_staff IN ts_reg;
```

NOTA: *Os usuários podem usar "ALL" para representar todos os índices na tabela ao usar a instrução DISABLE INDEX ou REBUILD INDEX.*

Exemplo 3: Para reconstruir todos os índices da tabela tb_staff, use o seguinte comando

```
dmSQL> REBUILD INDEX ALL FOR tb_staff;
```

6.6 Managing Auto Indexes

Com o desenvolvimento do banco de dados em nuvem, os índices são gerenciados automaticamente, em vez de manualmente. De acordo com as instruções de consulta executadas pelos usuários, o daemon de auto índice pode analisar os requisitos dos usuários para gerenciar índices de forma mais inteligente.

O DBMaker suporta auto índices, cujo comportamento é semelhante ao de índices não-únicos, mas eles podem ser criados ou removidos automaticamente pelo daemon de auto índice. Se a opção AUTOCOMMIT estiver ativada, o DBMaker requer apenas bloqueio de atualização (Update - U) ao criar um auto índice, o que permite que outros usuários consultem a tabela simultaneamente.

O DBMaker suporta um daemon de auto índice para operar índices automáticos. Ele inclui duas partes principais: Mecanismos de Coleção e Mecanismos de Gerenciamento. Somente se o daemon de auto índice for iniciado, os Mecanismos de Coleção analisarão o plano de execução das instruções SQL executadas pelos usuários e, então, registrarão o resultado da análise (zero ou vários registros por vez) no arquivo DMSCAN.LOG. O Mecanismo de Gerenciamento será acionado de acordo com as configurações de DB_IdxTm e DB_IdxTv, usando o comando dmSQL SYNC AUTO INDEX ou com o assistente de Sincronização de Índice Automático da ferramenta JDBA. As principais tarefas do Mecanismo de Gerenciamento são ler o DMSCAN.LOG, analisar todos os registros para decidir se há necessidade de criar índices automáticos, atualizar informações de uso de índices e remover índices automáticos que não são usados por um período especificado por DB_IdxDp (somente índices automáticos serão removidos; outros tipos de índices não serão afetados). Os índices criados ou removidos neste processo serão registrados no arquivo DMAUTOIDX.LOG, o que é útil para usuários que desejam saber a condição de trabalho do daemon de auto índice.

Na verdade, quando um usuário no cliente executa uma instrução SQL, a informação de log não é registrada imediatamente no arquivo DMSCAN.LOG. Os Mecanismos de Coleção primeiro armazenam a informação de log em um buffer (tamanho fixo de 2560 bytes) do cliente e só a gravam no arquivo DMSCAN.LOG quando o buffer estiver cheio. Além disso, se o usuário desconectar do banco de dados ou executar o comando SYNC AUTO INDEX, os dados no buffer também serão gravados no arquivo DMSCAN.LOG. Desta forma, não apenas a desordem de informações é evitada devido à gravação simultânea de vários usuários no DMSCAN.LOG, mas também ajuda a

concentrar a operação de E/S para melhorar o desempenho dos Mecanismos de Coleção do índice automático.

Para criar e remover automaticamente um índice automático, os usuários precisam definir as palavras-chave DB_IdxSv, DB_IdxLg, DB_IdxTm, DB_IdxTv, DB_IdxDp e DB_IdxLn no arquivo dmconfig.ini para controlar o Daemon de Índice Automático. Com o banco de dados em execução, somente um usuário com autoridade DBA, SYSDBA ou SYSADM pode chamar setSystemOption() para definir essas opções, exceto IDXLG.

A diferença entre índices automáticos e outros tipos de índices se reflete nos quatro aspectos a seguir:

- ndices automáticos podem ser criados automaticamente pelo daemon de índice automático.
- Índices automáticos podem ser removidos automaticamente pelo daemon de índice automático se o índice puder ser mesclado com outros índices ou se não for usado por um número especificado de dias.
- O número máximo de colunas de um índice criado pelo usuário é 32, mas o número máximo de colunas de um índice automático criado pelo daemon de índice automático é 16.
- A criação de um índice automático requer bloqueio U quando definido como COMMIT ON, enquanto a criação de outros índices requer bloqueio X.

O DBMaker também suporta a sintaxe 'SET LOADAUTOINDEX ON|OFF' e a função ODBC SQLSetConnectOption para o usuário decidir se carrega o índice automático ou não. Para obter mais informações sobre funções ODBC, consulte o Guia do Programador ODBC.

Exemplo 1:

Para ligar o servidor de índice automático com o procedimento armazenado do sistema SetSystemOption:

```
dmSQL> CALL SETSYSTEMOPTION('IDXSV','1'); //activate auto index
server
dmSQL> SELECT * FROM tb_staff WHERE joinDate='1986-07-20'; //create
auto index
by daemon
dmSQL> sync auto index; //wake up auto index daemon
dmSQL> SELECT * FROM sysindex;
```

Exemplo 2:

Para redefinir o número de dias para 60 para descartar um índice automático com o procedimento armazenado do sistema SetSystemOption.

```
dmSQL> CALL SETSYSTEMOPTION ('IDXDP','60');
```

Creating Auto Indexes

O índice automático pode ser criado automaticamente pelo daemon de índice automático ou manualmente pelos usuários usando o comando dmSQL CREATE AUTO INDEX. Para saber mais sobre como criar índices automáticos manualmente, consulte a sintaxe CREATE INDEX na Referência de Comando e Função SQL.

Quando o daemon de índice automático cria um índice automático em uma tabela, você especifica o tipo de índice automático, as colunas do índice automático e a ordem de classificação de cada coluna como ascendente ou descendente. A ordem de classificação padrão é ascendente.

O nome do índice automático criado pelo daemon de índice automático é formado por AUTO, sublinhado, identificador da coluna e ordem da coluna (a ordem da coluna é D para descendente e nula para ascendente). Se dois índices com o mesmo nome puderem ser mesclados, não há necessidade de criar um novo índice. Caso não possam ser mesclados, os usuários precisam criar um novo índice e nomeá-lo como nome_do_índice_Rxxx. Nome_do_índice é o nome dos dois índices originais e o número xxx de três dígitos é aleatório.

Exemplo 1:

Para criar o índice automático AUTO_ID_2 nas colunas ID e NAME para a tabela tb_staff em ordem decrescente, use a opção DESC no comando dmsql:

```
dmSQL> CREATE AUTO INDEX AUTO_ID_2 ON tb_staff (ID DESC, NAME);
```

Exemplo 2:

Se o novo índice tiver o mesmo nome de um índice existente e os dois não puderem ser mesclados, você pode estender o nome do novo índice para AUTO_ID_2__R321. O número de três dígitos 321 é aleatório.

```
dmSQL> CREATE AUTO INDEX AUTO_ID_2__R321 ON tb_staff (ID DESC,
NAME);
```

Creating Expression Auto Indexes

Índices automáticos podem ser criados automaticamente não apenas em colunas simples, mas também em colunas de expressão ou colunas de função definida pelo usuário (UDF).

Exemplo 1:

Para criar um índice automático, auto_idx_expr, na expressão basepay+bonus para a tabela tb_salary:

```
dmSQL> CREATE AUTO INDEX auto_idx_expr ON tb_salary (basepay+bonus);
```

Exemplo 2:

Para criar um índice automático, auto_idx_substr, na UDF substring(nation,1,3) para a tabela tb_salary:

```
dmSQL> CREATE AUTO INDEX auto_idx_substr ON tb_salary
(substring(nation,1,3)
DESC);
```

Dropping Auto Indexes

O daemon de índice automático pode remover automaticamente índices automáticos não usados por um período especificado por DB_IdxDp. Além disso, os usuários também podem remover índices automáticos com o comando dmSQL DROP AUTO INDEX. Se o índice for criado em uma coluna como chave primária ou referenciado por outras tabelas, ele não poderá ser removido. Para obter informações sobre chaves primárias, consulte a seção Gerenciando Integridade de Dados.

Exemplo:

Para remover o índice AUTO_1D_2 da tabela tb_staff.

```
dmSQL> DROP INDEX AUTO_1D_2 FROM tb_staff;
```

Disabling Auto Indexes

Índices automáticos podem ser desativados usando a ferramenta JDBA ou o comando dmSQL DISABLE INDEX.

Para desativar o índice automático aidx_fill da tabela tb_staff:

```
dmSQL> DISABLE INDEX aidx_fill FOR tb_staff;
```

Rebuilding Auto Indexes

Um índice automático pode ser reconstruído usando a ferramenta JDBA ou o comando dmSQL REBUILD INDEX. O comando REBUILD INDEX eliminará o índice antigo e criará um novo.

Exemplo: Para reconstruir o índice automático aidx_fill da tabela tb_staff:

```
dmSQL> REBUILD INDEX aidx_fill FOR tb_staff;
```

NOTA: Os usuários podem usar "ALL" para representar todos os índices na tabela ao usar o comando DISABLE/REBUILD INDEX.

6.7 Managing Text Indexes

Um índice de texto é um mecanismo que fornece acesso rápido às linhas de uma tabela que contenham uma ou mais palavras ou fras es em determinadas colunas. Índices de texto armazenam uma representação de todo o texto encontrado nas colunas às quais estão associados, mas esses dados são codificados e estruturados para permitir uma recuperação bem mais veloz do que a busca direta na tabela. Uma vez que um usuário cria um índice de texto para uma tabela, sua operação se torna transparente. O Sistema de Gerenciamento de Banco de Dados (SGBD) utiliza o índice para melhorar o desempenho de consultas de texto completo sempre que possível.

O DBMaker oferece dois métodos de indexação de texto: assinatura e arquivo invertido (IVF).

Índices de texto podem ser criados em todas as colunas do tipo caractere, incluindo CHAR, VARCHAR, LONG VARCHAR, LONG VARBINARY e tipos de dados FILE. Uma tabela pode ter vários índices de texto, e um único índice pode ser construído usando várias colunas. O usuário pode criar índices de texto usando a ferramenta JDBA ou o comando dmSQL CREATE [SIGNATURE | IVF] TEXT INDEX.

Exemplo:

Para usar automaticamente um índice de texto na coluna de dados (sem especificá-lo):

```
dmSQL> SELECT id FROM tb_book WHERE data MATCH 'compute';
```

Os operadores de string do DBMaker incluem MATCH, CONTAIN, CONTAINS e LIKE. Apenas os operadores MATCH e CONTAINS podem ser aplicados a uma pesquisa por índice de texto.

O DBMaker oferece dois tipos diferentes de índice de texto: assinatura e arquivo invertido. O índice de texto por assinatura é mais eficiente para pequenas quantidades de dados. O índice de texto por arquivo invertido geralmente consome mais espaço de armazenamento, mas sua resposta a consultas é mais rápida para grandes volumes de dados.

Creating Signature Text Indexes

O DBMaker cria índices de texto por assinatura se o método de indexação de texto não for especificado pelo comando. O usuário pode criar índices de texto usando a ferramenta JDBA, o comando dmSQL CREATE TEXT INDEX ou o comando CREATE SIGNATURE TEXT INDEX. Exemplo: Para criar um índice de texto por assinatura, tidx_name, na coluna name da tabela tb_staff:

```
dmSQL> CREATE SIGNATURE TEXT INDEX tidx_name ON tb_staff(name);
```

SIGNATURE TEXT INDEX PARAMETERS

O DBMaker oferece dois parâmetros para configurar convenientemente o desempenho e o tamanho de armazenamento de índices de texto por assinatura.

- **Tamanho total do texto (MB):** o tamanho total estimado de todos os documentos de origem, em megabytes (MB). O intervalo é de 1 a 200 e o padrão é 32. É importante observar que o tamanho total real do texto não está limitado a 200 MB; se o tamanho for maior que 200, defina como 200. No entanto, recomendamos fortemente o uso do índice de texto IVF para indexar grandes quantidades de dados para um desempenho de consulta significativamente melhor. • Ajuste a proporção esperada entre o tamanho do índice e o tamanho total do texto. Se um usuário definir o tamanho total do texto como 20 (MB) e esperar que o índice de texto use 10 MB de armazenamento,

ele deve definir a escala como 50 (por cento). Quanto maior a escala, melhor será o desempenho da pesquisa. O intervalo é de 10 a 200 e o valor padrão é 40 (por cento).

Exemplo:

Para criar um índice de texto, defina tidx_scale na coluna da tabela tb_staff que contém cerca de 40 megabytes de dados, considerando que desejamos que o índice de texto use cerca de 20 megabytes de espaço de armazenamento:

```
dmSQL> CREATE SIGNATURE TEXT INDEX tidx_scale ON tb_staff(name)
 TOTAL TEXT SIZE 40 MB
 SCALE 50;
```

Os usuários podem utilizar a configuração padrão como parâmetros do índice de texto. Para obter um melhor desempenho do índice de texto ou reduzir seu tamanho, é possível alterar esses parâmetros. Defina os parâmetros, monitore o desempenho do índice e então reajuste-os conforme necessário.

Creating Inverted-File (IVF) Text Indexes

Um usuário pode criar índices de texto invertidos usando o comando CREATE IVF TEXT INDEX.

Exemplo:

Para criar um índice de texto invertido, ivfidx_name, na coluna name da tabela tb_staff:

```
dmSQL> CREATE IVF TEXT INDEX ivfidx_name ON tb_staff(name);
```

INVERTED-FILE TEXT INDEX PARAMETERS

Storage path - Diretório lógico de trabalho onde os arquivos invertidos serão armazenados. Os usuários devem definir esse diretório lógico no arquivo dmconfig.ini. O padrão é o valor de DB_DbDir, que é o diretório inicial do banco de dados. O gerenciamento detalhado do armazenamento e a convenção de nomeação do índice de arquivo invertido serão descritos na próxima seção.

T**otal text size (MB)** - é o tamanho aproximado do total de documentos que serão indexados no futuro. A unidade de tamanho é megabyte (MB). Com base no tamanho, o DBMaker decide quantas partições serão criadas. O valor pode variar entre 1 MB e 10.000 MB, sendo o valor padrão 500 MB. Exemplo:

Para criar um índice de texto invertido, chamado ivfidx_name, no caminho \IVFDIR para a coluna na tabela tb_staff que contém aproximadamente 400 MB de dados:

Primeiramente, adicione um caminho lógico na seção dmconfig.ini do banco de dados.

```
MYPATH1 = \\IVFDIR
```

Utilize o seguinte comando.

```
dmSQL> CREATE IVF TEXT INDEX ivfidx_name ON tb_staff(name)
 2> STORAGE PATH MYPATH1
 3> TOTAL TEXT SIZE 400 MB;
```

Durante a criação de índices de texto invertido, uma grande quantidade de memória é necessária. O DBMaker usa uma regra simples para definir o uso máximo de memória para esse processo. Se o DBMaker não conseguir detectar memória livre ou se a memória livre for inferior a 128 MB, o uso máximo de memória será de 64 MB. Caso contrário, será usada metade da memória livre disponível. É possível especificar manualmente um limite superior aproximado para o uso de memória por meio do arquivo dmconfig.ini. Para isso, adicione a entrada de palavra-chave DB_IFMem com o valor em megabytes (MB).

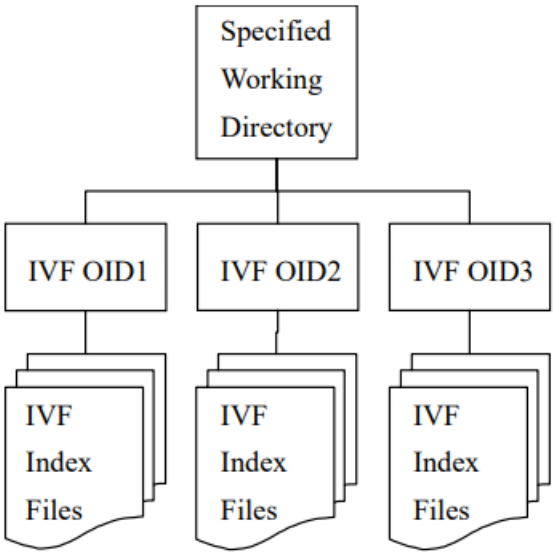
Exemplo:

Para especificar um uso de memória de 100 MB para a criação de um índice de texto invertido no arquivo dmconfig.ini:

```
DB_IFMem = 100
```

STORAGE OVERVIEW

Além do diretório de trabalho especificado pelo parâmetro do caminho de Armazenamento, o DBMaker irá gerar subdiretórios nesse diretório para gerenciar diferentes índices invertidos. Cada índice invertido possui uma versão de tempo exclusiva, possibilitando ao DBMaker utilizar essa propriedade para gerar um subdiretório exclusivo para armazenar arquivos de índice. A nomeação do subdiretório será descrita posteriormente. Subdiretórios e arquivos invertidos não podem ser revertidos quando um índice invertido é descartado.



Por exemplo, suponhamos que \DBMaker5.4 seja o diretório de trabalho especificado e que criemos um IVF nesse diretório com o nome de índice IVF1 e a versão de tempo 1024476670. Nesse caso, um subdiretório chamado IVF1.1024476670 será criado. Todos os arquivos invertidos residem nesse subdiretório. Existem três tipos de arquivo invertido, cada qual com um tipo de termo diferente: termo Single-Byte, termo Uni-Gram e termo Bi-Gram. Além disso, cada arquivo invertido possui várias partições cujo tamanho é determinado pelo tamanho do texto.

A seguir, a estrutura conceitual dos arquivos de índice IVF com quatro partições.

Escolher entre assinatura e arquivo invertido dependerá dos seguintes fatores:

1. Tamanho do índice: o tamanho do índice de assinatura não excederá a proporção definida pelo parâmetro Scale, que é 40% do tamanho total dos dados por padrão. O tamanho médio dos índices de arquivo invertido é cerca de 1,5 vezes o tamanho dos dados, mas pode crescer para duas ou até três vezes, dependendo da propriedade dos dados.
2. Tempo de resposta para consultas: em um computador pessoal moderno com memória e poder de processamento suficientes, os usuários podem esperar um tempo de resposta inferior a um segundo de índices de arquivo invertido, mesmo que o tamanho dos dados seja em gigabytes. Os índices de assinatura levarão mais tempo para responder, especialmente quando o tamanho dos dados fica grande.
3. Integração com banco de dados: ao contrário dos índices de assinatura que são armazenados como objetos BLOB, os índices de arquivo invertido são

armazenados como arquivos externos. Portanto, por exemplo, os usuários não podem reverter uma operação de eliminação de um índice de arquivo invertido.

Tente ambos os tipos de índice de texto para descobrir qual se adapta melhor às características dos seus dados. Como regra geral, para dados com tamanho inferior a 100 MB, os índices de assinatura respondem a consultas de forma razoavelmente rápida e normalmente ocupam menos espaço de armazenamento.

Creating Text Indexes on Multiple Columns

Um índice de texto pode ser construído usando múltiplas colunas. Utilize CONTAINS e o operador de concatenação (||) para realizar consultas de texto em várias colunas. Usuários podem pesquisar em todas as colunas do índice ou apenas em uma parte delas. Ou seja, a lista de colunas em uma consulta de correspondência deve estar contida na lista de colunas de um índice de texto para que ele seja utilizado. Os usuários também podem usar a sintaxe de consulta multicoluna mesmo se nenhum índice de texto for criado na lista de colunas, mas nenhum índice de texto será usado nesse caso.

Buscar em várias colunas é logicamente equivalente a mesclar todos os dados das colunas e, em seguida, realizar a pesquisa.

Exemplo 1:

Para criar um índice de texto invertido em várias colunas chamado ivfidx_multiple nas colunas author, subject e content da tabela tb_document:

```
dmSQL> CREATE IVF TEXT INDEX ivfidx_multiple ON
tb_document(author,subject,content);
dmSQL> SELECT author FROM tb_document WHERE
 2> CONTAINS(author || subject || content, 'reagan');
```

Exemplo 2:

Para pesquisar em listas de colunas parciais:

```
dmSQL> SELECT author FROM tb_document WHERE
2> CONTAINS(author || content, 'reagan');
dmSQL> SELECT author FROM tb_document WHERE CONTAINS(subject,
'reagan');
dmSQL> SELECT author FROM tb_document WHERE subject MATCH 'reagan';
```

Exemplo 3:

Neste exemplo, a coluna subject está incluída no índice de texto ivfidx_multiple, mas abstract não está. Portanto, esta consulta não usará nenhum índice de texto.

```
dmSQL> SELECT author FROM tb_document WHERE
2> CONTAINS(subject || abstract, 'reagan'); // no text index used
```

Exemplo 4:

Este exemplo ilustra o comportamento da consulta em várias colunas.

```
dmSQL> CREATE TABLE tb_example (c1 char(20), c2 char (20), c3
serial);
dmSQL> INSERT INTO tb_example VALUES('apple orange', 'banana
grape');
dmSQL> INSERT INTO tb_example VALUES('grape orange', null);
dmSQL> CREATE TEXT INDEX idx_example ON tb_example (c1, c2);
dmSQL> SELECT c3 FROM tb_example WHERE CONTAINS (c1 || c2, 'apple');
 C3
===========
 1
1 rows selected
dmSQL> SELECT c3 FROM tb_example WHERE CONTAINS (c1 || c2, 'orange &
grape');
 C3
===========
 1
 2
2 rows selected
```

Creating Text Indexes on Media Types

As colunas de objetos grandes do DBMaker podem registrar tipos de mídia. Por exemplo, uma coluna LONG VARBINARY pode reconhecer que seu conteúdo é um arquivo do Microsoft Word. Isso permite ao DBMaker invocar as funções apropriadas para realizar pesquisas de texto completo em documentos do Microsoft Word. O DBMaker também fornece UDF de mídia para converter alguns formatos de mídia em texto puro e uma UDF (CHECKMEDIAFORMAT) para consultar o formato de mídia. A Tabela 6-1 resume os tipos de mídia disponíveis e seus comandos SQL associados.

MEDIA TYPE	DATA TYPE	FILE TYPE
Microsoft Word	MsWordType	MsWordFileType
HTML	HtmlType	HtmlFileType
XML	XmlType	XmlFileType
Microsoft PowerPoint	MsPPTType	MsPPTFileType
Microsoft Excel	MsExcelType	MsExcelFileType
PDF	PDFType	PDFFileType

Internamente, MsWordType, MsPPTType, MsExcelType e PDFType são tratados como objetos LONG VARBINARY; HtmlType e XmlType são objetos LONG VARCHAR e MsWordFileType, HtmlFileType, XmlFileType, MsPPTFileType, MsExcelFileType e PDFFileType são objetos FILE.

FULL-TEXT SEARCH ON MEDIA-TYPE COLUMNS

Os usuários podem criar um índice de texto e realizar uma pesquisa de texto completo no tipo de mídia, mas primeiro o formato de mídia precisa ser convertido em texto puro. O DBMaker não reconhecerá novos formatos de mídia, portanto, a conversão do formato de mídia em texto puro não será possível, nem a pesquisa de texto completo no formato de mídia.

O DBMaker fornece as seguintes UDFs de mídia para converter alguns formatos de mídia em texto puro:

DOC, XLS, PPT, HTM, PDF.

- DOCTOTXT(BLOB) RETURNS NCLOB;
- XLSTOTXT(BLOB) RETURNS NCLOB;
- PPTTOTXT(BLOB) RETURNS NCLOB;
- HTMTOTXT(CLOB) RETURNS CLOB;
- PDFTOTXT(BLOB) RETURNS NCLOB;

MATCH e CONTAINS também podem ser usados para realizar pesquisas de texto completo em colunas de tipo de mídia (mediatype), assim como em colunas de texto comuns.

Exemplo 1:

Converte um documento do PowerPoint em um BLOB temporário contendo o texto puro do blob como Unicode.

```
dmSQL> CREATE TABLE tb_ppt(pptfile LONG VARBINARY);
dmSQL> INSERT INTO tb_ppt VALUES(?);
```

```
dmSQL/Val> &e:\\udf\\pptfile\\pfile.ppt;
dmSQL/Val> END;
dmSQL> SELECT PPTTOTXT(pptfile) FROM tb_ppt;
```

Exemplo 2:

Criar uma tabela com uma coluna do tipo MS Word, inserir alguns dados e pesquisar.

```
dmSQL> CREATE TABLE tb_minutes(id INT, doc MsWordFileType);
dmSQL> INSERT INTO tb_minutes VALUES(1, 'c:\\meeting\\20020403-
1.doc');
dmSQL> SELECT id FROM tb_minutes WHERE doc MATCH 'Jeff';
 id
===========
 1
1 rows selected
```

Exemplo 3:

Criar um índice de texto de assinatura tidx_doc na coluna doc da tabela tb_minutes e pesquisar.

```
dmSQL> CREATE TEXT INDEX tidx_doc ON tb_minutes(doc);
dmSQL> SELECT id FROM tb_minutes WHERE doc MATCH 'Jeff';
 id
===========
 1
1 rows selected
```

CHECK COLUMN DATA'S MEDIA TYPE

É possível que uma coluna de tipo de mídia contenha dados de diferentes formatos. O DBMaker pode verificar o conteúdo durante a inserção ou atualização de dados nessas colunas. Ele fornece a função CHECKMEDIAFORMAT para verificar se os dados da coluna correspondem ao formato de mídia especificado. Se os tipos corresponderem, a função retorna 1; caso contrário, retorna 0. É importante ressaltar que esta verificação de formato de mídia é compatível somente com as versões do Office de 2007 a 2010. O suporte do DBMaker para formatos além desse intervalo não é mencionado neste trecho.

O DBMaker suporta os seguintes formatos de tipos de mídia: DOC, XLS, PPT, HTM e PDF.

- DOC: Microsoft Words Document
- XLS: Microsoft Excel Document
- PPT: Microsoft Power Point Document
- HTM: Hypertext Markup Language
- PDF: Portable Document Format

NOTA: ***Os formatos de PDF suportados pelo DBMaker são de 1.2 a 1.7.***

Para verificar se o formato do tipo de mídia está correto:

```
dmSQL> CREATE TABLE tb_checkmedia(note LONG VARBINARY);
dmSQL> INSERT INTO tb_checkmedia VALUES(?);
dmSQL/Val> &E:\\DOCS\\Media.doc;
dmSQL/Val> &E:\\DOCS\\Media2007.docx;
dmSQL/Val> END;
dmSQL> SELECT checkmediaformat(note,'doc') FROM tb_checkmedia;
CHECKMEDIAFORMAT(NOTE,'DOC')
============================
 0
0
2 rows selected
```

O CHECKMEDIAFORMAT retornará 0, 1 ou NULL. • Retorna 1 quando o conteúdo do BLOB corresponde ao formato de mídia especificado. • Retorna 0 quando o conteúdo do BLOB não corresponde ao formato de mídia especificado. • Retorna NULL quando o BLOB é NULL. • Quando a coluna da tabela é definida com o tipo de mídia original (domínio do sistema) e o formato de mídia não é o correto, a migração pode falhar. Para resolver esse problema, remova os dados de mídia inválidos ou altere o tipo de dados para CLOB ou BLOB para evitar a etapa de verificação do formato de mídia correto.

Dropping Text Indexes

Indiciadores de texto ou índices de texto IVF podem ser descartados usando a ferramenta JDBA ou a instrução dmSQL DROP TEXT INDEX.

Para descartar o índice de texto `tidx_name` da tabela `tb_staff`:

```
dmSQL> DROP TEXT INDEX tidx_name FROM tb_staff;
```

Rebuilding Text Indexes

Ao contrário dos índices tradicionais, os índices de texto não refletem automaticamente o conteúdo da tabela quando novos registros são inseridos ou registros antigos são atualizados. Por isso, é necessário reconstruí-los manualmente. Dados atualizados após a reconstrução mais recente não serão encontrados durante uma pesquisa no índice de texto.

Exemplo 1:

```
dmSQL> CREATE TABLE tb_song (id INT, name VARCHAR(20));
dmSQL> INSERT INTO tb_song VALUES(1,'Endless Love');
1 rows inserted
dmSQL> CREATE TEXT INDEX tidx_name ON tb_song(name);
dmSQL> INSERT INTO tb_song VALUES(2,'Love Story');
1 rows inserted
dmSQL> SELECT * FROM tb_song WHERE name MATCH 'love';
 id name
=========== ====================
 1 Endless Love
1 rows selected
```

Deveria haver dois registros correspondentes ao padrão de pesquisa, mas apenas um é recuperado. O índice de texto IVF é um tipo especial de índice de texto, portanto, as instruções de uso de índice de texto nesta seção são as mesmas para o índice de texto IVF.

Se um grande número de documentos for excluído ou atualizado, você precisa executar os seguintes comandos para descartar e recriar o índice de texto:

Exemplo 2:

Para reconstruir o índice de texto de assinatura tidx_name da tabela tb_song como um índice invertido:

```
dmSQL> DROP TEXT INDEX tidx_name FROM tb_song;
dmSQL> CREATE IVF TEXT INDEX tidx_name ON tb_song(name);
```

Exemplo 3:

Para reconstruir completamente o índice de texto de assinatura tidx_name da tabela tb_song com um novo tamanho total de texto:

```
dmSQL> DROP TEXT INDEX tidx_name FROM tb_song;
dmSQL> CREATE TEXT INDEX tidx_name FROM tb_song(name) TOTAL TEXT
SIZE 60 MB;
```

Exemplo 4:

Para reconstruir um índice de texto invertido ivfidx_name para a tabela tb_staff e exibir os resultados:

```
dmSQL> REBUILD TEXT INDEX ivfidx_name FOR tb_staff;
dmSQL> SELECT * FROM tb_staff WHERE name MATCH 'CITY';
 ID NAME
========== ====================
 1 XIAN CITY
 2 SHANGHAI CITY
2 rows selected
```

INCREMENTALLY REBUILD TEXT INDEXES

O comando REBUILD TEXT INDEX reconstrói os dados atualizados de forma incremental, coletando todos os registros novos e atualizados, criando novos vetores de assinatura e anexando esses novos vetores ao final do índice de texto. Quando apenas alguns registros foram alterados, o comando REBUILD TEXT INDEX <index_name> INCREMENTAL é o método mais rápido para a reconstrução. O termo "INCREMENTAL" pode ser omitido. Exemplo:

Para reconstruir um índice de texto incrementalmente e exibir os resultados:

```
dmSQL> REBUILD TEXT INDEX tidx_name FOR tb_song;
dmSQL> SELECT * FROM tb_song WHERE name MATCH 'love';
 id name
========== ====================
 1 Endless Love
 2 Love Story
2 rows selected
```

FULLY REBUILD TEXT INDEXES

Se um grande número de documentos for excluído ou atualizado, use o comando REBUILD TEXT INDEX FULL para reconstruir completamente o índice de texto com seu tipo original (assinatura ou arquivo invertido) e parâmetros. É importante ressaltar que a palavra FULL é obrigatória nesse comando.

Exemplo:

Para reconstruir completamente o índice de texto tidx_name da tabela tb_song:

```
dmSQL> REBUILD TEXT INDEX tidx_name FOR tb_song;
```

Boolean Text Search

O operador MATCH permite buscar não apenas um padrão de texto simples, mas também operações Booleanas complexas.

Um usuário pode especificar os seguintes caracteres Booleanos em um padrão de pesquisa:

'&' AND

'|' OR '-' EXCLUDE '(' Left bracket ')' Right bracket

A precedência dos caracteres booleanos é: parênteses > EXCLUDE = AND > OR. Quando uma string MATCH contém caracteres booleanos, todos os outros caracteres entre eles são processados como padrões de pesquisa simples. Por exemplo, se a string MATCH for "café | chá | suco de maçã", então o padrão de pesquisa inclui "café", "chá" e "suco de maçã".

Exemplo 1:

Para pesquisar documentos que contenham 'amor' e 'amigo':

```
dmSQL> SELECT * FROM tb_song WHERE name MATCH 'love & friend';
```

Exemplo 2:

A próxima consulta pesquisa documentos que contenham 'amor' ou 'amigo', mas que não incluam 'amor sem fim'.

```
dmSQL> SELECT * FROM tb_song WHERE name MATCH '(love | friend) -
endless love';
```

Exemplo 3:

Os operadores Booleanos da sintaxe SQL, como AND e OR, podem ser usados para obter os mesmos resultados que os operadores Booleanos do padrão MATCH. No entanto, possui um desempenho inferior, pois somente a última parte do padrão de pesquisa utiliza o índice de texto. Por exemplo, o seguinte comando SQL aplicará a varredura do índice de texto somente ao pesquisar a string 'amigo' e usará uma pesquisa padrão não indexada para a string 'amor':

```
dmSQL> SELECT * FROM tb_song WHERE name MATCH 'love' AND name MATCH
'friend';
```

Fuzzy Search

Às vezes, os usuários gostariam de pesquisar por padrões imprecisos. Se apenas frases exatas forem permitidas, a consulta "William Clinton" não encontrará "William Jefferson Clinton" e vice-versa. Uma expressão booleana como "William & Clinton" pode retornar muitos resultados irrelevantes. O DBMaker oferece um recurso de pesquisa fuzzy, permitindo que os usuários realizem consultas imprecisas sem receber muitos resultados irrelevantes.

Uma frase iniciada por um ponto de interrogação (?) será avaliada como uma expressão fuzzy, por exemplo, "?intel pentium". Palavras usadas para pesquisa em uma expressão fuzzy podem ser separadas por até quatro palavras no texto alvo. Por exemplo, "?intel pentium" encontrará "A Intel lançará seu processador Pentium III de 1 GHz" e "?amd k7 athlon" encontrará "A AMD renomeou seu processador K7 para Athlon".

É possível que alguns termos da consulta estejam ausentes do resultado. Por exemplo, "?William Jefferson Clinton" pode encontrar "William Clinton" e "William J Clinton", mas a primeira palavra da consulta precisa aparecer; a consulta "?William Clinton" não encontrará "Bill Clinton".

Expressões fuzzy podem ser combinadas com outras operações Booleanas de texto.

Exemplo:

```
dmSQL> SELECT content FROM tb_document WHERE content MATCH '?intel
pentium & ?amd
```

```
k6';
dmSQL> SELECT title FROM tb_document WHERE title MATCH 'al gore |
?george bush';
```

A frase em uma expressão fuzzy não pode conter nenhum outro operador. Portanto, a expressão "?intel pentium & ?amd k6" é avaliada como "((?intel pentium) & (?amd k6))", e "?(intel & pentium)" gerará um erro.

Near logic full-text search

Uma correspondência imprecisa (fuzzy match) permite que os usuários realizem consultas inexatas sem receber muitos resultados irrelevantes. A pesquisa por correspondência próxima é semelhante à pesquisa imprecisa, porém mais exata. Ela garante que todas as palavras na consulta apareçam no texto. Uma frase iniciada por um til (~) será avaliada como uma expressão de correspondência próxima. Por exemplo, "~amd vendas 1ghz athlon" encontrará "A AMD anunciou vendas trimestrais de seu chip Athlon de 1 GHz", mas não "A AMD anunciou vendas trimestrais de seu chip Athlon".

Expressões de pesquisa por correspondência próxima podem ser combinadas com outras operações Booleanas de texto.

Exemplo:

```
dmSQL> SELECT content FROM tb_document WHERE content MATCH '~intel
pentium & ~amd
k6';
dmSQL> SELECT title FROM tb_document WHERE title MATCH 'al gore |
~george bush';
```

Fuzzy/Near Logic Matching Rules

1. A primeira palavra da consulta deve aparecer. Por exemplo, a consulta "?William Clinton" não encontrará "Bill Clinton".
2. Palavras podem ser separadas por um número predefinido de outras palavras ("proximidade"). Por exemplo, a consulta "?intel pentium" encontrará resultados como "A Intel lançará o processador Pentium III de 1 GHz" e "?amd k7 athlon" encontrará resultados como "A AMD renomeou seu processador K7 para Athlon". Atualmente, o número de palavras adicionais permitidas entre as palavras da consulta nos resultados é de no máximo 4.

3. A number of words in the search pattern may be missing from the result set, e.g.,'?William Jefferson Clinton' can find 'William Clinton' and 'William J Clinton'. The maximum allowed number of missing words is determined by the formula:

max_miss = num_words - round(num_words ×threshold).

4. Todas as palavras na consulta devem aparecer na ordem original. Por exemplo, "?amd 1ghz k7 athlon" encontrará "A AMD anunciará o Athlon de 1 GHz", mas não "AMD Athlon, anteriormente conhecido como K7”

Uma frase iniciada por um '~' (til) será avaliada como uma expressão próxima, por exemplo, '~intel pentium'. A pesquisa próxima é um caso especial de pesquisa imprecisa que atende às regras 1, 2 e 4 acima, mas não permite palavras ausentes.

User-Defined Stopword

Ao contrário de um sistema baseado em palavras-chave, o software de recuperação de texto completo indexa todas as palavras em um documento, com exceção das palavras de parada. Palavras de parada são termos que o software de recuperação de texto completo é programado para ignorar durante os processos de indexação e recuperação, para evitar a recuperação de registros irrelevantes. Geralmente, uma lista de palavras de parada inclui artigos, pronomes, adjetivos, advérbios e preposições (por exemplo, o, a, os, as, eles, elas, muito, não, de) que são comuns no idioma inglês. Essa mesma regra também pode ser aplicada ao idioma chinês ou a qualquer texto codificado em byte duplo, por exemplo: 的、呢、啊 e 哈.

SEARCH PATH FOR STOPWORD LIST

DB_StpWd = <string>

Essa palavra-chave indica o nome do arquivo de definição da lista de stopwords, localizado no subdiretório shared/stopword do diretório de instalação do DBMaker. O arquivo de definição da lista de stopwords é um arquivo de texto puro, que pode afetar o resultado da indexação de texto no DBMaker. Essa palavra-chave é usada durante a criação e recuperação do índice de texto do banco de dados. Sem essa palavra-chave, a pesquisa no banco de dados usa a lista de stopwords predefinida baseada no LCode.

default value:

DB_LCode	Stopword List Definition
0 English (ASCII)	en.tab
1 Traditional Chinese (BIG5)	tw.tab

2 Japanese (Shift JIS + Half Corner)	jp.tab
3 Simplified Chinese (GB)	cn.tab
4 Latin1 code (ISO-8859-1)	en.tab
5 Latin2 code (ISO-8859-2)	en.tab
6 Cyrillic code (ISO-8859-5)	en.tab
7 Greek code (ISO-8859-7)	en.tab
8 Japanese code（EUC-JP）	jp.tab
9 Simplified Chinese (GB18030)	cn.tab
10 Unicode (UTF-8)	en.tab

valid range: O nome do arquivo de definição da lista de stopwords definida pelo usuário.

see also: DB_LCode

where to use: servidor (somente para criação e pesquisa de índice de texto)

Default Stopword List

- Se você não especificar nenhuma configuração, o DBMaker carregará stopwords padrão que estão em um arquivo pré-definido e baseado no LCODE. Esse recurso pode ser usado por usuários de versões anteriores do DBMaker.
- O DBMaker pesquisa o arquivo pré-definido no diretório local e no diretório de instalação.

User Defined Stopword List

- Você pode especificar uma lista de stopwords por meio do arquivo de configuração DB_StpWd. O DBMaker carrega o arquivo ao criar um índice de texto ou recuperar objetos do índice de texto.
- O DBMaker pesquisa o arquivo no diretório local, no diretório especificado pelo usuário e no diretório de instalação.

Disable Stopword List

Existem duas maneiras de desabilitar a lista de stopwords:

- **Renomear ou remover o arquivo predefinido:** Isso impedirá o DBMaker de carregar a lista padrão.
- **Definir uma lista de stopwords inexistente no arquivo de configuração:** Ao especificar um arquivo que não existe no DB_StpWd, o DBMaker efetivamente desabilita o uso de stopwords.

6.8 Managing Memory Tables

As tabelas de memória, para quase todos os fins, funcionam da mesma maneira que uma tabela permanente no DBMaker. A diferença principal é que as tabelas de memória são temporárias e baseadas em conexão. Isso significa que uma tabela de memória só existe enquanto houver uma conexão com o banco de dados. Quando um usuário encerra sua conexão com o banco de dados, ao sair do sistema para encerrar o expediente do dia, por exemplo, a tabela de memória e seu conteúdo serão perdidos para o usuário.

As tabelas de memória do DBMaker só ficam visíveis durante a conexão com o banco de dados. Uma vez perdida a conexão, a tabela não pode mais ser acessada. Ao contrário de uma tabela permanente, as tabelas de memória residem apenas na memória da máquina que as criou. Elas não podem ser acessadas por um grupo e só permitem a seleção ou inserção de dados. Funcionalidades de atualização ou exclusão de dados não são suportadas. No entanto, as tabelas de memória oferecem suporte a controles de transação, incluindo commit, rollback, definição de savepoint, rollback para savepoint e savepoint interno.

Para criar uma tabela de memória no DBMaker, use a sintaxe dmSQL CREATE MEMORY TABLE. Detalhes sobre a sintaxe e uso do comando SQL CREATE MEMORY TABLE podem ser encontrados na referência de comando e função SQL do DBMaker.

Exemplo:

Para criar a tabela de memória tb_memory:

```
dmSQL> CREATE MEMORY TABLE tb_memory (id INT, name CHAR(10),
brithday DATE);
```

Hash Index Management

As tabelas de memória são armazenadas na memória da máquina que a criou, por esse motivo, elas não suportam a estrutura B-tree de outros tipos de tabela. Para auxiliar os usuários ao utilizar tabelas de memória, é possível criar índices hash nessas tabelas. Índices hash só podem ser criados em tabelas de memória. A vantagem de um índice hash é que os usuários têm acesso muito rápido aos dados armazenados nele. Os índices hash também melhoram o desempenho de expressões de igualdade e junções de igualdade. Para criar um índice hash em uma tabela, os usuários podem utilizar o comando CREATE HASH INDEX nome_do_indice ON nome_da_tabela

nome_da_coluna,) [bucket n]; onde nome_do_indice é o nome do índice hash sendo criado, nome_da_tabela é o nome da tabela de memória, nome_da_coluna é o nome da coluna na tabela de memória sendo afetada (este valor não pode especificar colunas asc/desc) e bucket n define o tamanho do array para a tabela hash sendo criada. Por exemplo, com a tabela de memória criada, um índice hash hidx pode ser criado na tabela de memória tb_memory, usando as colunas id e nome com um tamanho de array de 31.

Exemplo:

Para criar o índice hash hidx na tabela de memória tb_memory

```
dmSQL> CREATE HASH INDEX hidx ON tb_memory (id, name) bucket 31;
```

6.9 Managing Data Integrity

A aplicação de restrições, ou regras, para garantir que os dados atendam a certos critérios, pode assegurar a integridade dos dados. Por exemplo, verificar se um valor de entrada para um item de dados específico está dentro da faixa correta de valores, como a idade de um novo funcionário que deve estar entre 16 e 90 anos, é um exemplo de integridade de dados.

De maneira geral, os diferentes tipos de integridade de dados aplicáveis a tabelas incluem aqueles descritos nas subseções a seguir.

Not Null

Por padrão, todas as colunas em uma tabela permitem valores NULL. NOT NULL indica que valores NULL não são permitidos em uma coluna definida com a palavra-chave NOT NULL.

Unique Indexes

Indices exclusivos, mencionados na seção 6.5, Gerenciando Índices, podem ser usados para garantir que nenhuma linha da tabela tenha valores duplicados, exceto valores NULL, em uma coluna ou conjunto de colunas especificado.

Unique Constraints

Uma restrição UNIQUE pode ser definida em uma coluna, em um conjunto de colunas ou em uma tabela inteira. A restrição UNIQUE garante que cada linha em uma coluna

tenha um valor diferente. Nenhuma linha pode ter o mesmo valor na coluna ou nas colunas onde uma restrição UNIQUE é aplicada.

Exemplo:

Para criar uma tabela com restrição UNIQUE na coluna Nome:

```
dmSQL> CREATE TABLE tb_student (Name CHAR(50) CONSTRAINT u UNIQUE,
mathematics
SMALLINT);
```

Check Constraints

Uma restrição CHECK em uma coluna ou conjunto de colunas exige que uma condição específica seja verdadeira para todas as linhas da tabela. Se uma instrução INSERT ou UPDATE for emitida e a condição da restrição CHECK for avaliada como falsa, a instrução falhará.

De modo geral, uma restrição CHECK pode ser definida em uma coluna (restrição de coluna) ou em um conjunto de colunas (restrição de tabela).

COLUMN CONSTRAINTS

Uma restrição de coluna é definida em uma coluna específica e não afeta as outras colunas da mesma tabela. Ao inserir uma nova linha ou atualizar uma linha existente, cada restrição de coluna é avaliada.

TABLE CONSTRAINTS

Uma restrição de tabela é definida em um conjunto de colunas. Ao inserir uma nova linha ou atualizar uma linha existente, a restrição de tabela é avaliada **depois** que todas as restrições de coluna forem avaliadas como verdadeiras. Somente após a restrição de tabela também ser avaliada como verdadeira, a instrução será processada.

Exemplo 1:

Para criar uma tabela com restrições de coluna e tabela:

```
dmSQL> CREATE TABLE tb_student (mathematics SMALLINT
 CHECK VALUE >= 0 AND VALUE <= 100,
 chemistry SMALLINT
 CHECK VALUE >= 0 AND VALUE <= 100,
```

```
CHECK mathematics + chemistry <= 200);
```

Exemplo 2:

Para criar uma tabela com restrições de coluna e tabela usando a sintaxe padrão SQL99:

```
dmSQL> CREATE TABLE tb_student (mathematics SMALLINT
 CONSTRAINT con_math CHECK VALUE >= 0 AND VALUE <= 100,
 chemistry SMALLINT
 CONSTRAINT con_chem CHECK VALUE >= 0 AND VALUE <= 100,
 CONSTRAINT con_sum CHECK mathematics + chemistry <= 200);
```

A palavra-chave VALUE é usada para representar o valor da coluna em restrições de coluna, enquanto os nomes das colunas são usados para representar os valores das colunas em uma restrição de tabela.

Primary Keys

Uma tabela pode ter uma chave primária, que inclui uma coluna ou um grupo de colunas com valores únicos para identificar cada linha. A chave primária é similar a um índice exclusivo, exceto pelo fato de suas colunas não poderem conter valores NULL. Quando um usuário cria uma chave primária, o DBMaker cria um índice exclusivo chamado PrimaryKey na tabela. Após a criação de uma tabela, as chaves primárias podem ser modificadas ou adicionadas desde que todas as colunas que a componham contenham valores únicos e não nulos. Uma chave primária pode ser adicionada a uma tabela ou modificada usando a instrução ALTER TABLE. Além disso, uma chave primária adicionada a uma tabela ou modificada dessa forma pode ser armazenada em um tablespace diferente da tabela.

CREATING PRIMARY KEYS

Exemplo 1:

Para criar uma tabela com sua chave primária na coluna ID:

```
dmSQL> CREATE TABLE tb_student (
 ID INTEGER PRIMARY KEY,
 name CHAR(30),
 nation CHAR(20)
```

```
 );
```

Exemplo 2:

Para criar uma tabela com uma chave primária composta pelas colunas ID e nome no tablespace ts_reg:

```
dmSQL> CREATE TABLE tb_student (
 ID INTEGER,
 name CHAR(30),
 nation CHAR(20),
 PRIMARY KEY (ID, name)
 ) in ts_reg;
```

Exemplo 3:

Para adicionar uma chave primária à tabela tb_student:

```
dmSQL> ALTER TABLE tb_student PRIMARY KEY (ID , name);
```

Exemplo 4:

Para adicionar uma chave primária PK1 à tabela tb_student no tablespace ts_reg usando a sintaxe padrão SQL99:

```
dmSQL> ALTER TABLE tb_student ADD CONSTRAINT PK1 PRIMARY KEY (ID ,
name) IN ts_reg;
```

Exemplo 5:

Para criar uma tabela com sua chave primária na coluna ID usando a sintaxe padrão SQL99:

```
dmSQL>          CREATE          TABLE          tb_student          (
 ID       INTEGER       CONSTRAINT       pk1       PRIMARY       KEY,
 name                                                        CHAR(30),
 nation                                                       CHAR(20)
 );
```

DROPPING PRIMARY KEYS

Sim, é possível remover uma chave primária de uma tabela quando ela não é mais necessária. No entanto, antes de fazer isso, é crucial remover todas as chaves estrangeiras que se referem a essa chave primária.

Exemplo:

Para remover a chave primária da tabela tb_student:

```
dmSQL> ALTER TABLE tb_student DROP PRIMARY KEY;
```

DISABLING PRIMARY KEYS

Exemplo: Para desativar a chave primária da tabela tb_student:

```
dmSQL> DISABLE INDEX primarykey for tb_student;
```

REBUILDING PRIMARY KEYS

Os usuários podem reconstruir a chave primária quando ela for necessária novamente.

Exemplo:

```
dmSQL> REBUILD INDEX primarykey for tb_student;
```

Para reconstruir a chave primária da tabela tb_student:

Nota: *Os usuários podem usar "ALL" para representar todos os índices da tabela ao utilizar o comando DISABLE/REBUILD INDEX.*

Foreign Keys (Referential Integrity)

Uma coluna em uma tabela que armazena valores iguais aos da chave primária de outra tabela é chamada de **chave estrangeira**. Essa chave estrangeira **estabelece uma relação** entre as duas tabelas. Um usuário pode criar uma chave estrangeira em uma única coluna ou em um conjunto de colunas de uma tabela. Esta chave estrangeira **referencia** uma coluna ou conjunto de colunas (também conhecidas como colunas referenciadas) em outra tabela. É importante ressaltar que as colunas

referenciadas devem ser definidas como **chave primária** ou possuir um **índice único** e **não podem conter valores nulos (NULL)**.

As colunas referenciadas na tabela estrangeira **precisam conter previamente os valores de chave que estão sendo inseridos** em uma nova linha. Se esses valores não estiverem presentes na tabela referenciada, o usuário não poderá inserir a linha na tabela estrangeira. Isso garante a integridade referencial do banco de dados. Além disso, **todos os valores de chave na tabela estrangeira devem ser excluídos** antes de excluir os valores de chave na tabela referenciada. Isso evita que a tabela estrangeira contenha referências inválidas (órfãs) a registros inexistentes na tabela referenciada.

Um usuário pode criar ou remover chaves primárias e estrangeiras sempre que necessário. O DBMaker verifica a unicidade de uma chave primária quando ela é criada. Da mesma forma, o DBMaker verifica se todos os valores de chave já existem na tabela referenciada quando uma chave estrangeira é criada. Isso garante a consistência dos dados e evita erros de integridade referencial.

CREATING FOREIGN KEYS

Uma chave estrangeira é usada para referenciar outra tabela especificando as colunas de referência e referenciadas. Ambas as colunas, de referência e referenciada, devem ser mapeadas uma para a outra; seus esquemas devem ser iguais. As colunas de mapeamento devem ser do mesmo tipo e tamanho. As colunas referenciadas (especificadas pela chave primária ou índice único) devem ser NOT NULL (não nulas), mas as colunas de referência (especificadas pela chave estrangeira) podem ser NOT NULL ou NULL. Se as colunas referenciadas não forem especificadas, a chave primária na tabela referenciada é considerada a(s) coluna(s) referenciada(s). Chaves estrangeiras podem ser criadas usando o assistente de criação de chave estrangeira da ferramenta JDBA Tool ou a opção FOREIGN KEY do dmSQL.

Exemplo 1:

Para criar uma chave estrangeira f1 na tabela tb_salary, que referencia a tabela tb_staff com uma chave primária composta pelas colunas ID e nome:

```
dmSQL> ALTER TABLE tb_salary FOREIGN KEY f1(ID, name) REFERENCES
tb_staff;
```

Exemplo 2:

Alternativamente, é possível especificar a chave estrangeira durante a criação da tabela tb_salary:

```
dmSQL> CREATE TABLE tb_salary (
 ID INT,
 name CHAR(30),
 basepay INT,
 bonus INT,
 tax INT,
 FOREIGN KEY f1 (ID, name) REFERENCES tb_staff);
```

Exemplo 3:

Sintaxe padrão SQL99, para especificar a chave estrangeira f1 durante a criação da tabela tb_example:

```
dmSQL> CREATE TABLE tb_example (
 c1 int,
 c2 int CONSTRAINT f1 REFERENCES tb_other (c1) ON DELETE SET NULL);
```

Se a tabela tb_staff possui uma chave primária, uma chave estrangeira pode ser criada em outra tabela para referenciá-la, sem a necessidade de especificar as colunas referenciadas.

DROPPING FOREIGN KEYS

Se o relacionamento definido por uma chave estrangeira não for mais necessário, ela pode ser removida utilizando a ferramenta JDBA Tool ou o comando DROP FOREIGN KEY do dmSQL.

Exemplo:

Para remover uma chave estrangeira da tabela tb_salary:

```
dmSQL> ALTER TABLE tb_salary DROP FOREIGN KEY f1;
```

6.10 Managing Serial Numbers

O DBMaker oferece um recurso para gerar números de série automaticamente. Esse recurso é particularmente útil em ambientes com vários usuários, pois permite gerar

e retornar números sequenciais únicos sem a sobrecarga de E/S de disco ou bloqueio de transação.

Os números de série no DBMaker são inteiros de 32 bits assinados. Uma tabela só pode ter uma coluna contendo o tipo de dado SERIAL para gerar números de série.

O DBMaker permite especificar um número inicial para a coluna do tipo SERIAL ao criar uma tabela. Se nenhum valor inicial for especificado, será definido como 1.

Para que o DBMaker gere um número de série, insira uma nova linha e forneça um valor NULL para a coluna serial. Se você inserir uma nova linha e fornecer um valor inteiro em vez de NULL, o DBMaker não gerará um número de série. Se o valor inteiro fornecido for maior que o último número de série gerado, o DBMaker reiniciará a sequência de números de série gerados para começar com o valor inteiro fornecido. É importante observar que colunas do tipo SERIAL não podem ser definidas com valores padrão ou restrições.

Creating Serial Columns

Uma coluna do tipo SERIAL pode ser criada usando a ferramenta JDBA Tool ou o dmSQL. Para definir uma coluna serial, utilize a palavra-chave SERIAL ou BIGSERIAL e, opcionalmente, um número inicial.

Exemplo:

Para criar uma coluna do tipo SERIAL chamada ID na tabela tb_staff e especificar seu número inicial como 1001:

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20) DEFAULT 'R.O.C',
 ID SERIAL(1001),
 name CHAR(30) NOT NULL,
 joinDate DATE DEFAULT CURDATE(),
 height FLOAT,
 degree VARCHAR(200)) IN ts_reg;
```

Generating Serial Numbers

O DBMaker gera números de série automaticamente **quando um valor NULL é inserido** na coluna do tipo SERIAL.

Exemplo: Para inserir uma nova linha na tabela tb_staff e fazer com que o DBMaker gere um número de série para a coluna ID:

```
dmSQL> INSERT INTO tb_staff VALUES
 ('U.S.A', NULL, 'Jeff', 6.6, 'Director', NULL);
```

Retrieving Serial Numbers

O DBMaker armazena o último número de série gerado na coluna LAST_SERIAL da tabela de sistema SYSCONINFO para cada conexão. Após inserir um registro contendo um número de série, esse número pode ser recuperado da coluna LAST_SERIAL.

Exemplo: Para obter o número de série que acabou de ser gerado para o registro inserido:

```
dmSQL> SELECT LAST_SERIAL FROM SYSCONINFO;
LAST_SERIAL
===========
 200
1 rows selected
```

Resetting Serial Numbers

O DBMaker permite **reiniciar o contador** de uma coluna do tipo SERIAL. Isso possibilita iniciar uma nova sequência de numeração sem a necessidade de modificar a tabela.

Exemplo:

Para alterar o valor do contador da coluna serial na tabela tb_staff do seu valor atual para 3000:

```
dmSQL> ALTER TABLE tb_staff SET SERIAL 3000;
```

6.11 Managing Domains

Um domínio é um tipo de restrição de integridade usada na definição de uma coluna. Domínios definem o tipo de dado para a coluna e podem especificar um valor padrão ou uma restrição de valor. Quando uma coluna é definida usando um domínio, ela herda as propriedades do domínio (tipo de dado, valor padrão e restrição de valor), sem exigir que o usuário as especifique.

Usar domínios para especificar o valor padrão e a restrição de valor atinge o mesmo resultado que especificá-los em uma definição de coluna padrão. Se um usuário especificar um valor padrão para uma coluna, ele substituirá o valor padrão especificado em um domínio.

Restrições de valor especificadas na definição da coluna serão usadas junto com aquelas definidas no domínio. Se um usuário definir uma coluna usando um domínio e especificar restrições de valor adicionais, essas restrições adicionais não deverão entrar em conflito com as definidas no domínio.

É importante observar que o DBMaker não verifica conflitos entre restrições de valor. Por isso, é possível definir restrições que impeçam a inserção de qualquer valor. Todos os tipos de dados suportados pelo DBMaker, exceto o tipo SERIAL, podem ser usados em domínios.

Creating Domains

Um domínio é definido por um nome, um valor padrão opcional e uma restrição opcional. Por exemplo, um usuário pode querer garantir que todas as colunas que lidem com algum tipo de título (como filme, CD ou videocassete) tenham um tipo de dado VARCHAR, não possuam mais de 35 caracteres e não permitam a inserção de valores NULL. Domínios podem ser criados usando a ferramenta JDBA Tool ou a instrução CREATE DOMAIN do dmSQL.

Exemplo 1:

A palavra-chave VALUE é usada para representar o valor da coluna definida no domínio. Vamos criar um domínio específico que será utilizado nas instruções CREATE TABLE posteriores:

```
dmSQL> CREATE DOMAIN title_type VARCHAR(35) CHECK VALUE IS NOT NULL;
```

Exemplo 2:

Para definir colunas como no comando CREATE TABLE:

```
dmSQL> CREATE TABLE movie_titles (title title_type, ..., ...);
```

CREATING DOMAIN WITH TEXT CONVERTER

Um tipo de mídia é um domínio com características específicas relacionadas ao formato da mídia. O usuário pode criar domínios usando a sintaxe TEXT CONVERTER

na cláusula CREATE DOMAIN. Quando o usuário especifica a sintaxe TEXT CONVERTER no domínio, o DBMaker usa a expressão TEXT CONVERTER para converter dados CLOB, NCLOB, BLOB ou FILE em texto puro para criar um índice de texto e UDF PURETEXT(). O nome da função TEXT CONVERTER só deve conter tipos de argumento relacionados a BLOB. O tipo de retorno deve ser CLOB ou NCLOB, caso contrário o DBMaker retornará um erro. Não é possível criar mais de 32.767 domínios com a sintaxe TEXT CONVERTER.

NOTA: O usuário não pode especificar expressões, funções sem argumentos ou funções com mais de um argumento na cláusula TEXT CONVERTER.

Exemplo:

Para definir um domínio MSWORDTYPE:

```
dmSQL> CREATE DOMAIN MSWORDTYPE AS BLOB
TEXT CONVERTER DOCTOTXT
CHECK VALUE IS NULL OR CHECKMEDIAFORMAT(VALUE,'DOC') = 1;
```

Para criar uma tabela tb_MT com o domínio MSWORDTYPE:

```
dmSQL> CREATE TABLE tb_MT (C1 MSWORDTYPE);
```

Dropping Domains

Um domínio só pode ser removido (dropped) se não houver colunas que o referenciam. Você pode remover domínios usando a ferramenta JDBA ou a instrução DROP DOMAIN do dmSQL.

Exemplo:

Para utilizar a instrução DROP DOMAIN do dmSQL.

```
dmSQL> DROP DOMAIN title_type;
```

6.12 Unloading and Loading Objects

Às vezes, o usuário pode precisar salvar dados do banco de dados em um arquivo de texto externo. O DBMaker fornece os comandos UNLOAD e LOAD exatamente para esse propósito. Objetos descarregados do banco de dados não são removidos do banco de dados; eles são simplesmente salvos como um ou mais arquivos de texto

externos. Quando um objeto é carregado em um banco de dados, o esquema desse objeto também é recriado.

Unloading Objects

O Unload é uma ferramenta fornecida pelo dmSQL usada para transferir o conteúdo de um banco de dados para um arquivo de texto externo. Após o sucesso do procedimento de unload, o dmSQL irá gerar dois arquivos de texto. Um armazena o script, com a extensão .s0, para estabelecer o objeto do banco de dados e o outro armazena os dados BLOB, com a extensão .bn.

Existem oito opções para o comando unload: banco de dados, tabela, esquema, dados, projeto, módulo, procedimento e definição de procedimento. Para descarregar um objeto, o usuário precisa ter o privilégio SELECT sobre ele. Por exemplo, se um usuário possui privilégio SELECT em uma tabela, somente ele poderá descarregar o conteúdo dessa tabela. Apenas usuários com autoridade DBA, SYSDBA ou SYSADM podem descarregar o banco de dados inteiro.

UNLOAD [DB | DATABASE]

Um usuário com autoridade DBA, SYSDBA ou SYSADM pode descarregar o conteúdo de um banco de dados para um arquivo de texto externo. Este arquivo incluirá informações sobre segurança, tablespaces, definições, índices, sinônimos e dados. Para cada banco de dados, o dmSQL irá gerar pelo menos dois arquivos externos: um script e um arquivo de dados BLOB.

Exemplo 1:

```
dmSQL> UNLOAD DB TO empdb;
```

O nome do arquivo de texto externo é empdb. Por padrão, o dmSQL irá criar esses arquivos no diretório de trabalho atual. No exemplo acima, pelo menos dois arquivos de texto são criados, **empdb.s0** e **empdb.b0**. Se o arquivo BLOB descarregado empdb.b0 exceder o tamanho máximo permitido pelo sistema operacional, o dmSQL irá gerar sequencialmente arquivos **empdb.b1, empdb.b2,** empdb.bn, até um número máximo de 99. O dmSQL sempre gerará um arquivo de script empdb.s0, com tamanho máximo limitado pelo sistema operacional. No entanto, se um usuário usar o comando SET UNLOAD EXTERNAL 'connection_string' (onde o formato de 'connection_string' é "DSN=;UID=;PWD=;") antes de usar o comando UNLOAD DB TO file_name, o dmSQL não descarregará os dados no arquivo de script empdb.s0. Em vez disso, o dmSQL imprimirá "set external db 'connection_string'" em empdb.s0 e a

descarga dos dados das tabelas será impressa como "load external db from 'select * from external_table_name' into local_table_name'". Consulte o seguinte exemplo:

Exemplo 2:

```
dmSQL> SET UNLOAD EXTERNAL 'DSN=DBSAMPLE5;UID=SYSADM;PWD=;';
dmSQL> UNLOAD DB TO empdb;
```

Aqui, o arquivo de script empdb.s0 fica assim:

```
…
set external db 'DSN= DBSAMPLE5;UID=SYSADM;PWD=;';
create table Lauser1.Latb3 (
c1 SMALLINT default null ,
c2 FLOAT default null ,
c3 DOUBLE default null ,
c4 DECIMAL(10, 3) default null ,
c5 CHAR(10) default null ,
c6 BINARY(12) default null )
in DEFTABLESPACE lock mode page fillfactor 100 ;
load external database from 'select * from Lauser1.Latb3' into
Lauser1.Latb3;
create index idx31 on Lauser1.Latb3 ( c1 asc ) in DEFTABLESPACE;
create index idx32 on Lauser1.Latb3 ( c3 desc ) in DEFTABLESPACE;
create index idx33 on Lauser1.Latb3 ( c5 asc ) in DEFTABLESPACE;
```

UNLOAD TABLE

O comando UNLOAD TABLE descarrega tabelas para um arquivo externo e irá registrar a definição, sinônimos, índices, chave primária, chaves estrangeiras e dados da tabela. Você pode usar os caracteres curinga "*" *e "%" nesse comando, que correspondem a "?" e* "" do DOS no nome do dono e da tabela. O curinga "*" representa um único caractere, e "%" representa um conjunto de caracteres.

UNLOAD SCHEMA

O uso desta opção é bem similar ao comando UNLOAD TABLE. Ele só pode descarregar a definição de uma tabela; não é possível descarregar os dados da tabela. Ele utiliza os mesmos caracteres curinga da opção UNLOAD TABLE.

UNLOAD DATA

Esta opção descarregará todos os dados de uma tabela. Ela não irá descarregar a definição da tabela. O comando UNLOAD DATA utiliza os mesmos caracteres curinga das duas opções anteriores. Somente usuários com o privilégio SELECT na tabela a ser descarregada podem executar o comando UNLOAD DATA. O DBMaker 3.6 e versões posteriores oferecem uma sintaxe adicional para descarregar dados:

```
UNLOAD DATA FROM (select statement) TO file_name
```

Exemplo 1:

Sintaxe Válida:

```
dmSQL> UNLOAD DATA FROM (SELECT t1.c1, t1.c2 FROM t1, t2 WHERE
t1.c1= t2.c1) TO
f1;
```

Exemplo 2:

Sintaxe Inválida:

```
dmSQL> UNLOAD DATA FROM (SELECT t1.c1, t2.c1 FROM t1, t2 WHERE t1.c1
= t2.c1) TO
f1;
```

Exemplo 3:

Sintaxe Inválida:

```
dmSQL> UNLOAD DATA FROM (SELECT avg(c1) FROM t1) TO f1;
dmSQL> UNLOAD DATA FROM (SELECT now()FROM t1) TO f1;
```

Exemplo 4:

Sintaxe Válida:

```
dmSQL> UNLOAD DATA FROM (SELECT * FROM s1 WHERE c1 > 10) TO f1;
dmSQL> UNLOAD DATA FROM (SELECT * FROM v1 WHERE c1 < 10) TO f1;
```

UNLOAD PROJECT

Esta opção permite ao usuário descarregar um projeto ESQL/C para um arquivo de texto externo.

UNLOAD MODULE

Esta opção permite ao usuário descarregar um módulo para um arquivo de texto externo.

UNLOAD [PROC | PROCEDURE]

Essa opção permite ao usuário descarregar as stored procedures para um arquivo de texto externo.

UNLOAD [PROC DEFINITION | PROCEDURE

DEFINITION]

Essa opção permite ao usuário descarregar a definição da stored procedure para um arquivo de texto externo.

Exemplo 1:

O código a seguir descarregará as tabelas "e" e "tab" para o usuário atual. Se houver espaços em branco no nome da tabela, ele será colocado entre aspas duplas.

```
dmSQL> UNLOAD TABLE FROM "e tab" TO empfile;
```

Exemplo 2:

O seguinte descarregará todas as tabelas com nomes iniciando com "emp" para o proprietário SYSADM, por exemplo, emptab, empname, … :

```
dmSQL> UNLOAD TABLE FROM SYSADM.emp% TO empfile;
```

Exemplo 3:

O seguinte descarregará o esquema de todas as tabelas com o nome "ktab".

```
dmSQL> UNLOAD SCHEMA FROM %.ktab TO kfile;
```

O seguinte descarregará a tabela cujo nome contenha caracteres curinga. Utilize o caractere de escape "" ou aspas duplas para delimitar o nome da tabela.

Exemplo 4:

O comando a seguir descarregará dados da tabela nomeada "abc%". É importante observar que o uso do caractere curinga "%" nesse contexto pode causar resultados inesperados, dependendo do seu ambiente de banco de dados.

```
dmSQL> UNLOAD DATA FROM abc\\% TO abcfile;
dmSQL> UNLOAD DATA FROM "abc%" TO abcfile;
```

Loading Objects

O comando LOAD é uma ferramenta fornecida pelo dmSQL e é usado para transferir um objeto do banco de dados que já foi descarregado para um arquivo de texto, de volta para o banco de dados. Existem sete opções: carregar banco de dados, carregar tabela, carregar esquema, carregar dados, carregar projeto, carregar módulo e carregar procedimento. Um arquivo deve ser descarregado e carregado com a mesma opção. Por exemplo, para carregar um banco de dados a partir de um arquivo de texto que foi descarregado com a opção "database". Ao carregar um arquivo de texto, defina o número de comandos para realizar a confirmação automática da transação. O padrão é 1000. O tamanho do afetará o sucesso da transação e a velocidade de carregamento. Um valor alto de pode facilmente encher o log de transações e causar falha na transação. Por outro lado, um valor baixo de aumentará o número de transações confirmadas, tornando o carregamento mais lento. Se ocorrerem erros durante o procedimento de carregamento, as mensagens de erro serão registradas em um arquivo de log. O sistema usará esse arquivo para desfazer os comandos executados. O arquivo de log é armazenado no mesmo diretório do arquivo de texto externo que está sendo carregado e não interrompe o procedimento de carregamento.

LOAD [DB | DATABASE]

Use o comando LOAD [DB | DATABASE] para transferir o conteúdo de um banco de dados para um novo banco. Primeiramente, descarregue o banco de dados que deseja transferir para um arquivo de texto externo. Em seguida, use o comando LOAD DB para carregar o conteúdo do banco de dados a partir do arquivo de texto. Antes de carregar um banco de dados, **crie um novo**. O nome do novo banco de dados pode ser diferente do antigo. Somente um DBA, SYSDBA ou um SYSADM pode executar este comando.

No entanto, se um usuário utilizar o comando CONFIGURAR DESCARREGAMENTO EXTERNO 'connection_string'(the format of connection_string is "DSN=<db_name>;UID=<user_name>;PWD=<password>;") antes de usar o comando DESCARREGAR BANCO DE DADOS PARA nome_do_arquivo, dmSQL não gravará os dados no arquivo de script. Portanto, quando um usuário carrega o banco de dados com este arquivo de script, o dmSQL se conectará à fonte de dados do gerenciador de drivers ODBC, lerá os dados dela e, em seguida, salvará os dados diretamente no banco de dados local. O dmSQL usa "set external [database|db] 'string_de_conexao'" no arquivo de script para se conectar a um banco de dados externo. Se a conexão falhar, um erro será retornado. O dmSQL mantém apenas a conexão mais recente com um banco de dados externo. Isso significa que, ao definir uma nova conexão externa, ele fecha automaticamente qualquer conexão anterior existente. Além disso, como não há um comando específico para desconectar, o banco de dados externo só será desconectado quando a ferramenta dmSQL for fechada.

O banco de dados opera em modo normal se LOAD DB for definido como SAFE. Nesse modo, a ferramenta de carregamento reverte para o último comando confirmado caso ocorra um erro durante o carregamento. Mensagens de erro são exibidas na tela e gravadas no arquivo de log da ferramenta. Ao utilizar o set LOAD DB no modo rápido, a regra para carregar a ferramenta em versões anteriores à 3.6 do DBMaker fará com que todo o procedimento de carregamento funcione no modo "no journal". Definir LOAD DB no modo rápido irá acelerar a ferramenta de carregamento, porém, caso ocorra qualquer erro, o banco de dados será encerrado no modo "no journal". Imagine, por exemplo, que o arquivo de carregamento contenha a criação de um tablespace, mas ele não esteja especificado no arquivo dmconfig.ini. Se LOAD DB estiver definido para usar a opção segura, a seguinte mensagem de erro será exibida: "ERROR(8002): [DBMaker] entrada de palavra-chave necessária para o arquivo de configuração" e, em seguida, o comando load será revertido. Se LOAD DB estiver definido para usar a opção rápida, a seguinte mensagem de erro ocorrerá: "ERROR(30017)".

Ocorreram erros do [DBMaker] no modo sem registro, encerrando o banco de dados". A opção padrão é SET LOADDB SAFE.

Exemplo:

O seguinte comando "set" para LOAD DB foi adicionado nas versões acima do DBMaker 3.6:

```
SET LOADDB [safe | fast]
```

LOAD TABLE

Esta opção permite carregar o conteúdo de uma tabela, incluindo esquema e dados, a partir de um arquivo de texto. Ao carregar uma tabela de um arquivo de texto, certifique-se de que o nome da tabela seja único.

LOAD SCHEMA

Esta opção permite aos usuários carregar o esquema, sem incluir os dados, de uma tabela contida em um arquivo de texto. Ao carregar o esquema de uma tabela a partir de um arquivo de texto, certifique-se de que o nome da tabela seja único.

LOAD DATA

Ao carregar dados de um arquivo de texto externo, uma tabela correspondente já deve existir no banco de dados. Nas versões anteriores à 3.6, caso ocorra algum erro durante o procedimento LOAD DATA, o processo será revertido para o último comando confirmado.

Exemplo:

O DBMaker 3.6 e versões posteriores suportam as seguintes opções:

```
SET LOADDATA SKIP [error] | STOP [on error]
```

Se LOAD DATA SKIP ERROR for definido, as seguintes mensagens de erro serão ignoradas durante o carregamento dos dados:

- **ERROR(401)**: Violação de chave única
- **ERROR(410)**: Violação de restrição referencial: valor não existe na chave pai
- **ERROR(6521)**: Tabela ou visão não existe
- **ERROR(6002)**: Erro de sintaxe
- **ERROR(6015)**: Entrada incompleta de instrução SQL

O valor padrão para esta opção é CARREGAR DADOS IGNORAR ERRO. Todas as mensagens de erro que ocorrerem durante o carregamento dos dados serão gravadas no arquivo de log.

LOAD MODULE

Essa opção permite ao usuário carregar um módulo de um arquivo de texto externo.

LOAD PROJECT

Essa opção permite ao usuário carregar um projeto ESQL/C de um arquivo de texto externo.

LOAD [PROC | PROCEDURE]

Essa opção permite ao usuário carregar uma stored procedure (procedimento armazenado) de um arquivo de texto externo.

Exemplo 1:

O comando a seguir carrega o banco de dados de um arquivo chamado empdb e realiza a confirmação automaticamente a cada 100 comandos durante o carregamento. O sistema irá gerar um arquivo de log chamado empdb.log no mesmo diretório.

```
dmSQL> LOAD DB FROM empdb 100;
```

Exemplo 2:

O comando a seguir carregará uma tabela do arquivo chamado empfile. Ele realizará a confirmação automaticamente a cada 50 comandos durante o carregamento.

```
dmSQL> LOAD TABLE FROM empfile 50;
```

Exemplo 3:

O comando a seguir permitirá o carregamento de dados de um arquivo de dados externo chamado datafile e realizará a confirmação automaticamente a cada 1.000 comandos utilizando a configuração padrão.

```
dmSQL> LOAD DATA FROM datafile;
```

6.13 Browsing System Catalogs

O DBMaker mantém informações detalhadas sobre todos os objetos do esquema nas tabelas de catálogo do sistema. Para obter mais informações sobre tabelas de catálogo do sistema, consulte a Referência de Catálogo do Sistema.

SCHEMA OBJECT INFORMATION	SYSTEM CATALOG TABLE NAME

Tables	SYSTABLE
Columns	SYSCOLUMN
Views	SYSVIEWDATA
Synonyms	SYSSYNONYM
Indexes	SYSINDEX
Domains	SYSDOMAIN
Serial numbers	SYSCONINFO
Table constraints	SYSTABLE
Column constraints	SYSCOLUMN
Domain constraints	SYSDOMAIN

6.14 Calculating the Space Required

Como afirmado nas seções anteriores, somente tabelas e índices ocupam espaço físico no disco. Para gerenciar o espaço em disco, é necessário estimar o tamanho de cada objeto e decidir a qual tablespace cada objeto pertencerá antes de criá-los. Na fase de estimativa, o administrador do banco de dados deve ter uma visão clara de como construir tablespaces usando tabelas e quanta integridade física será necessária para suportar o banco de dados no futuro. (Note: I replaced "hardware" with "integridade física" as a more suitable translation in this context.)

Ao contrário do que se pensava anteriormente, dividir tabelas em múltiplas tablespaces não necessariamente melhora a performance. Na verdade, tabelas armazenadas em um único tablespace grande costumam ter um desempenho melhor. Por outro lado, é importante ressaltar que o gerenciamento de muitos tablespaces pequenos pode ser mais complexo.

How to Estimate the Size of a Table

As fórmulas a seguir demonstram, é possível estimar o tamanho de uma tabela e de um índice:

table size = row size × number of rows × 1.05 index size = key size × number of rows × 1.20

Estas duas fórmulas servem para estimar o tamanho necessário para um tablespace ao somar o tamanho de todas as tabelas e índices contidos nele. Nas fórmulas acima, 1,05 e 1,20 são estimativas da sobrecarga de recursos usada para calcular os recursos do sistema necessários. O tamanho da linha e o tamanho da chave contêm o tamanho do cabeçalho interno do registro. As subseções a seguir mostram como calcular o tamanho de uma linha e de uma chave.

ROW SIZE

O tamanho de armazenamento de uma linha, excluindo dados BLOB, não pode exceder 3.996 bytes e consiste no espaço necessário para armazenamento de dados e um cabeçalho interno de registro.

O tamanho de um cabeçalho interno de registro é igual a:

internal record header size = (number of columns + 1) × 4

O tamanho do cabeçalho interno de registro é igual a: **(número de colunas + 1) × 4**

Cada tipo de dado possui um espaço necessário para armazenamento:

TYPE	COLUMN LENGTH
BIGINT	8
BINARY(n)	N
BIGSERIAL	8
CHAR(n)	N
SMALLINT	2
INTEGER	4
FLOAT	4
SERIAL	4
DOUBLE	8
DECIMAL(p,s)	[(p + 1) / 2] + 2
TIME	4
DATE	4
TIMESTAMP	11
OID	16
VARCHAR(n)	1-3992n
FILE	20
LONG VARBINARY	48 + X
LONG VARCHAR	48 + X

NOTA: *VARCHAR é um tipo de dado de comprimento variável que aceita qualquer caractere digitável no teclado. Uma coluna do tipo BLOB (LONG VARCHAR ou LONG VARBINARY) ocupa pelo menos 48 bytes no arquivo de dados, e o dado real é armazenado em um arquivo BLOB separado ou no próprio arquivo de dados. Para obter informações mais detalhadas, consulte o Capítulo 7, Gerenciamento de Objetos Grandes. Se o valor em uma coluna for NULL, ele não ocupa nenhum espaço.*

Exemplo:

Para criar uma tabela com cinco colunas definidas:

```
dmSQL> CREATE TABLE tb_staff(ID INTEGER NOT NULL,
 name CHAR(30) NOT NULL,
 height FLOAT,
 degree VARCHAR(200),
 picture LONG VARCHAR);
```

Após executar este comando, você pode inserir linhas na tabela e calcular o tamanho do registro:

(3001, "Jeff Yang", 175.5, "Stanford PhD.", [pic1]) where pic1 is an image

DATA ITEM	**TYPE**	**SIZE**
ID	integer	4 bytes
name	char	30 bytes
height	float	4 bytes
degree	varchar	13 bytes
picture	long varchar	48 bytes
row header	-	24 bytes
	Total	123 bytes

= (4 + 30 + 4 + 13 + 48) + (5 + 1) × 4 = 123 bytes

(3002, "George Wang", 180.0, "NCTU Ms.", NULL)

DATA ITEM	**TYPE**	**SIZE**
ID	integer	4 bytes
name	char	30 bytes
height	float	4 bytes
degree	varchar	8 bytes
picture	long varchar	0 bytes
row header	-	24 bytes
	Total	**70 bytes**

= (4 + 30 + 4 + 8 + 0) + (5 + 1) × 4 = 70 bytes

O DBMaker verificará se o tamanho da linha não ultrapassa MAXTUPLEN1 bytes ao inserir ou atualizar linhas. Ao criar uma tabela, o DBMaker também verifica se o menor tamanho de linha possível não excede MAXTUPLEN bytes.

O menor tamanho de linha no exemplo acima pode ser calculado da seguinte maneira:

minimum row size = (4 + 30 + 0 + 0 + 0) + (5 + 1) × 4 = 58 bytes

O tamanho mínimo da linha não excede MAXTUPLEN bytes, portanto o DBMaker permitirá a criação desta tabela.

KEY SIZE

O tamanho de armazenamento da chave é composto pelo espaço necessário para armazenar os dados nas colunas do índice e um cabeçalho interno do registro. Ele também requer espaço adicional de 16 bytes para um identificador interno da linha. Vale ressaltar que esse identificador interno da linha também consome 4 bytes no cabeçalho do registro.

O tamanho do cabeçalho interno do registro é igual a: **(número de colunas do índice + 1) × 4**

Internal record header size = (no. of columns + 1 + 1) × 4

Por exemplo, se um índice for criado em uma única coluna do tipo SMALLINT, o tamanho de cada chave será:

key size = 2 + 16 + (1 + 1 + 1) × 4 = 30 bytes

1 MAXTUPLEN : The value is 3968 in 4KB page size, 8064 in 8KB page size, 16256 in 16KB page size and 32640 in 32KB page size, respectively.

No caso, dois bytes são usados para os dados na coluna da chave, 16 bytes são usados para o identificador interno da linha para cada chave e 12 bytes são usados para o cabeçalho do registro.

ESTIMATING THE SIZE OF TABLESPACES AND TABLES

O exemplo a seguir demonstra como estimar o tamanho de um tablespace e suas tabelas. Suponha que exista um tablespace contendo três tabelas, A, B e C, e um índice D criado para a tabela A. As colunas da tabela A são definidas como INTEGER e CHAR(10). As colunas da tabela B são definidas como SMALLINT, CHAR(10), FLOAT e VARCHAR(200). As colunas da tabela C são definidas como SMALLINT, INTEGER e LONG VARCHAR. O índice D é criado na primeira coluna da tabela A. As tabelas A, B e C possuem 1500, 3000 e 250 linhas, respectivamente.

O tamanho das linhas e chaves para este banco de dados pode ser calculado conforme demonstrado abaixo. Suponha que o comprimento médio da coluna VARCHAR na tabela B seja de 80 bytes e o tamanho de cada coluna BLOB no arquivo de dados da tabela C seja de 48 bytes:

In table A:	row size = (4 + 10) + 3 ×4 = 26 bytes
In table B:	row size = (2 + 10 + 4 + 80) + 5 ×4 = 116 bytes

In table C:	row size = (2 + 4 + 48) + 4 ×4 = 70 bytes

Se o tamanho médio de cada item BLOB na tabela C for 9000 bytes, então especifique o tamanho do quadro BLOB como 11KB. Consulte o Capítulo 7, Gerenciamento de Objetos Grandes, para obter mais informações sobre dados BLOB.

Index D:	keysize = 4 + 16 + 3 ×4 = 32 bytes

O tamanho das tabelas para este banco de dados pode ser calculado conforme demonstrado abaixo. Observe que o tamanho da tabela A também inclui o tamanho do índice D.

Table A:	table size = (26 □ 1500 □ 1.05) + (32 □ 1500 □ 1.2) = 98550 bytes
Table B:	table size = 116 □ 3000 □ 1.05 = 365400 bytes
Table C:	table size = 70 □ 250 □ 1.05 = 18375 bytes

No arquivo BLOB, o tamanho da tabela C é de 250 quadros (cada linha necessita de um quadro).

Após analisar os valores acima, é possível estimar o tamanho necessário para um tablespace que armazene as tabelas e índices apresentados. O tablespace deve conter pelo menos um arquivo de dados (482.325 bytes) e um arquivo BLOB (250 quadros com tamanho de quadro de 11 KB). Estimar o tamanho do tablespace durante a criação é importante para evitar a necessidade de adicionar ou aumentar arquivos posteriormente.

6.15 Checking Database Consistency

O DBMaker inclui diversos comandos que usuários com privilégios de DBA podem utilizar para verificar a consistência de um banco de dados. Exemplos de inconsistência do banco de dados incluem um índice que possui uma chave que não existe na tabela ou uma chave que existe em uma tabela estrangeira, mas não existe na tabela pai. O DBMaker suporta seis comandos para verificar diferentes níveis de consistência. Como esses comandos podem consumir tempo quando o banco de dados é grande e geram bloqueios, os administradores devem utilizá-los somente quando necessário.

Checking Indexes

O DBMaker permite que um usuário com privilégios no índice em questão verifique o índice e sua relação com a tabela. Ele verifica se a estrutura do índice (por exemplo, B-tree) está correta, se os dados estão ordenados e se as chaves do índice

correspondem exatamente aos registros de dados. Se um índice parece ter algum problema, use este comando para verificar se o problema realmente existe. Caso o DBMaker encontre inconsistências no índice, exclua-o e reconstrua-o para corrigir o problema.

Exemplo:

Para verificar a consistência do índice idx_desc na tabela tb_staff:

```
dmSQL> CHECK INDEX tb_staff.idx_desc;
```

Checking Tables

O DBMaker permite que um usuário com privilégios sobre os objetos verifique todos os registros, índices, dados BLOB associados a uma tabela, e também o relacionamento entre tabelas estrangeiras e suas tabelas pai. Se for encontrada qualquer inconsistência na tabela, o ideal é descarregar todos os registros da tabela, excluí-la, recriá-la e, por fim, reinserir todos os registros.

Exemplo:

Para verificar a consistência da tabela tb_staff:

```
dmSQL> CHECK TABLE tb_staff;
```

Checking Catalogs

O DBMaker permite que um usuário com privilégios de DBA verifique a consistência das tabelas do sistema. Se os catálogos do sistema contiverem erros, o banco de dados pode estar seriamente corrompido.

Exemplo:

Para verificar a consistência dos catálogos do sistema:

```
dmSQL> CHECK CATALOG;
```

Checking Databases

O DBMaker também permite que um usuário com privilégios de DBA verifique todo o banco de dados, incluindo os catálogos do sistema e todos os tablespaces.

Exemplo:

Para verificar a consistência de todo o banco de dados:

```
dmSQL> CHECK DB;
```

Se for identificada corrupção no banco de dados e houver um backup recente, utilize-o para restaurar o sistema. Para obter mais informações, consulte o Capítulo 15, Recuperação de Banco de Dados, Backup e Restauração.

Quando o banco de dados não possui backup e um índice está corrompido, exclua e reconstrua o índice afetado. Se ocorrer qualquer outro tipo de corrupção, faça imediatamente um backup do banco de dados, incluindo todos os arquivos de dados e journals. Em seguida, tente desligar e reiniciar o banco de dados e execute os comandos CHECK novamente. É possível que alguns tipos de corrupção sejam corrigidos automaticamente pelo DBMaker após a recuperação de uma falha. Se alguma inconsistência persistir, entre em contato com o suporte técnico da CASEMaker para ajudar a solucionar os problemas restantes do banco de dados.

Checking Users' Files

O DBMaker permite aos usuários verificar seus arquivos quando o banco de dados é iniciado em um warm start. Se os usuários ativarem esta função, o DmServer verificará todos os arquivos dos usuários para garantir que eles ainda estejam no disco quando o banco de dados for iniciado dessa forma. Caso contrário, o DBMaker emitirá uma mensagem de aviso para alertar um usuário com autoridade de DBA ou superior para evitar a operação de arquivos que já tenham sido movidos. Essa mensagem de aviso é registrada no arquivo de log DMEVENT.LOG. Os usuários podem ativar esta função configurando o DB_ChkFl.

NOTA: *Esta função não suporta modo de usuário único*

6.16 Updating Statistics for Schema Objects

Valores de estatística desatualizados para objetos de esquema (tabelas, índices, colunas) podem fazer com que o otimizador do DBMaker use um plano ineficiente para uma instrução SQL. Se os usuários inseriram grandes quantidades de dados no banco de dados desde a última atualização dos valores de estatística pelo administrador do banco de dados, atualize os valores novamente.

As estatísticas do banco de dados somente são atualizadas após a inicialização. Além disso, essa atualização consome recursos do processador e pode afetar o

desempenho do sistema. Para evitar queda de performance e ainda ter estatísticas atualizadas, é recomendado escolher um intervalo e horário que não coincidam com o pico de uso das tabelas

Com o banco de dados em execução, um usuário pode alterar o valor de estatística especificado usando a stored procedure (procedimento armazenado) do sistema SetSystemOption.

Exemplo 1:

Para atualizar os valores das estatísticas para todos os objetos do schema:

```
dmSQL>                          UPDATE                          STATISTICS;
```

Exemplo 2:

Para atualizar forçadamente os valores das estatísticas para todos os objetos do schema:

```
dmSQL> UPDATE STATISTICS ALL;
```

Se um banco de dados for extremamente grande, levará muito tempo para atualizar os valores estatísticos de todos os objetos do schema. Um método alternativo é atualizar as estatísticas apenas em objetos específicos do schema que foram modificados desde a última atualização, e definir a taxa de amostragem.

Exemplo 3:

Para atualizar estatísticas para tabelas:

```
dmSQL> UPDATE STATISTICS table1, table2, user1.table3;
```

Exemplo 4:

Para atualizar estatísticas para o índice idx_desc na tabela tb_staff:

```
dmSQL> UPDATE STATISTICS tb_staff (index idx_desc);
```

Exemplo 5:

Para atualizar estatísticas para o índice idx_desc na tabela tb_staff:

```
dmSQL> UPDATE TABLESPACE STATISTICS ts_reg;
```

Exemplo 6:

Para atualizar estatísticas para os índices idx_desc e idx_fill na tabela tb_staff:

```
dmSQL> UPDATE STATISTICS tb_staff (index idx_desc, idx_fill);
```

Quando o banco de dados está em execução, um usuário pode alterar o valor da estatística especificada com o procedimento armazenado do sistema SetSystemOption.

Exemplo 7:

A sintaxe a seguir é usada para definir a amostragem de atualização de estatísticas para 60% quando o banco de dados está em execução:

```
dmSQL> CALL SETSYSTEMOPTION('STSSP', '60');
```

Consulte o Capítulo 18, Otimização de Desempenho, para obter mais informações sobre atualização de estatísticas e o otimizador SQL.