5.3. Restrições

Os tipos de dado são uma forma de limitar os dados que podem ser armazenados na tabela. Entretanto, para muitos aplicativos a restrição obtida não possui o refinamento necessário. Por exemplo, uma coluna contendo preços de produtos provavelmente só pode aceitar valores positivos, mas não existe nenhum tipo de dado que aceite apenas números positivos. Um outro problema é que pode ser necessário restringir os dados de uma coluna com relação a outras colunas ou linhas. Por exemplo, em uma tabela contendo informações sobre produtos deve haver apenas uma linha para cada código de produto.

Para esta finalidade, a linguagem SQL permite definir restrições em colunas e tabelas. As restrições permitem o nível de controle sobre os dados da tabela que for desejado. Se o usuário tentar armazenar dados em uma coluna da tabela violando a restrição, ocorrerá um erro. Isto se aplica até quando o erro é originado pela definição do valor padrão.

5.3.1. Restrições de verificação

Uma restrição de verificação é o tipo mais genérico de restrição. Permite especificar que os valores de uma determinada coluna devem estar de acordo com uma expressão booleana (valor-verdade [1]). Por exemplo, para permitir apenas preços com valores positivos utiliza-se:

```
CREATE TABLE produtos (
    cod_prod   integer,
    nome       text,
    preco      numeric CHECK (preco > 0)
);
```

Como pode ser observado, a definição da restrição vem após o tipo de dado, assim como a definição do valor padrão. O valor padrão e a restrição podem estar em qualquer ordem. A restrição de verificação é formada pela palavra chave CHECK seguida por uma expressão entre parênteses. A expressão da restrição de verificação deve envolver a coluna sendo restringida, senão não fará muito sentido.

Também pode ser atribuído um nome individual para a restrição. Isto torna mais clara a mensagem de erro, e permite fazer referência à restrição quando se desejar alterá-la. A sintaxe é:

```
CREATE TABLE produtos (
    cod_prod  integer,
    nome      text,
    preco     numeric CONSTRAINT chk_preco_positivo CHECK (preco > 0)
);
```

Portanto, para especificar o nome da restrição deve ser utilizada a palavra chave CONSTRAINT, seguida por um identificador, seguido por sua vez pela definição da restrição (Se não for escolhido o nome da restrição desta maneira, o sistema escolherá um nome para a restrição).

Uma restrição de verificação também pode referenciar várias colunas. Supondo que serão armazenados o preço normal e o preço com desconto, e que se deseje garantir que o preço com desconto seja menor que o preço normal:

```
CREATE TABLE produtos (
    cod_prod           integer,
    nome               text,
    preco              numeric CHECK (preco > 0),
    preco_com_desconto numeric CHECK (preco_com_desconto > 0),
    CHECK (preco > preco_com_desconto)
);
```

As duas primeiras formas de restrição já devem ser familiares. A terceira utiliza uma nova sintaxe, e não está anexada a uma coluna em particular. Em vez disso, aparece como um item à parte na lista de colunas separadas por vírgula. As definições das colunas e as definições destas restrições podem estar em qualquer ordem.

Dizemos que as duas primeiras restrições são restrições de coluna, enquanto a terceira é uma restrição de tabela, porque está escrita separado das definições de colunas. As restrições de coluna também podem ser escritas como restrições de tabela, enquanto o contrário nem sempre é possível, porque supostamente a restrição de coluna somente faz referência à coluna em que está anexada (O PostgreSQL não impõe esta regra, mas deve-se segui-la se for desejado que a definição da tabela sirva para outros sistemas de banco de dados). O exemplo acima também pode ser escrito do seguinte modo:

```
CREATE TABLE produtos (
    cod_prod           integer,
    nome               text,
    preco              numeric,
    CHECK (preco > 0),
    preco_com_desconto numeric,
    CHECK (preco_com_desconto > 0),
    CHECK (preco > preco_com_desconto)
);
```

ou ainda

```
CREATE TABLE produtos (
    cod_prod           integer,
    nome               text,
    preco              numeric CHECK (preco > 0),
    preco_com_desconto numeric,
    CHECK (preco_com_desconto > 0 AND preco > preco_com_desconto)
);
```

É uma questão de gosto.

Podem ser atribuídos nomes para as restrições de tabela da mesma maneira que para as restrições de coluna:

```
CREATE TABLE produtos (
```

```
    cod_prod           integer,
    nome               text,
    preco              numeric,
    CHECK (preco > 0),
    preco_com_desconto numeric,
    CHECK (preco_com_desconto > 0),
    CONSTRAINT chk_desconto_valido CHECK (preco > preco_com_desconto)
);
```

Deve ser observado que a restrição de verificação está satisfeita se o resultado da expressão de verificação for verdade ou o valor nulo. Como a maioria das expressões retorna o valor nulo quando um dos operandos é nulo, estas expressões não impedem a presença de valores nulos nas colunas com restrição. Para garantir que a coluna não contém o valor nulo, deve ser utilizada a restrição de não nulo descrita a seguir.

5.3.2. Restrições de não-nulo

Uma restrição de não-nulo simplesmente especifica que uma coluna não pode assumir o valor nulo. Um exemplo da sintaxe:

```
CREATE TABLE produtos (
    cod_prod   integer NOT NULL,
    nome       text    NOT NULL,
    preco      numeric
);
```

A restrição de não-nulo é sempre escrita como restrição de coluna. A restrição de não-nulo é funcionalmente equivalente a criar uma restrição de verificação CHECK (*nome_da_coluna* IS NOT NULL), mas no PostgreSQL a criação de uma restrição de não-nulo explícita é mais eficiente. A desvantagem é que não pode ser dado um nome explícito para uma restrição de não nulo criada deste modo.

Obviamente, uma coluna pode possuir mais de uma restrição, bastando apenas escrever uma restrição em seguida da outra:

```
CREATE TABLE produtos (
    cod_prod   integer NOT NULL,
    nome       text    NOT NULL,
    preco      numeric NOT NULL CHECK (preco > 0)
);
```

A ordem das restrições não importa, porque não determina, necessariamente, a ordem de verificação das restrições.

A restrição NOT NULL possui uma inversa: a restrição NULL. Isto não significa que a coluna deva ser nula, o que com certeza não tem utilidade. Em vez disto é simplesmente definido o comportamento padrão dizendo que a coluna pode ser nula. A restrição NULL não é definida no padrão SQL, não devendo ser utilizada em aplicativos portáveis (somente foi adicionada ao PostgreSQL para torná-lo compatível com outros sistemas de banco de dados). Porém, alguns usuários gostam porque torna fácil inverter a restrição no script de comandos. Por exemplo, é possível começar com

```
CREATE TABLE produtos (
    cod_prod   integer NULL,
    nome       text    NULL,
    preco      numeric NULL
```

```
);
```

e depois colocar a palavra chave NOT onde se desejar.

> **Dica:** Na maioria dos projetos de banco de dados, a maioria das colunas deve ser especificada como não-nula.

5.3.3. Restrições de unicidade

A restrição de unicidade garante que os dados contidos na coluna, ou no grupo de colunas, é único em relação a todas as outras linhas da tabela. A sintaxe é

```
CREATE TABLE produtos (
    cod_prod   integer UNIQUE,
    nome       text,
    preco      numeric
);
```

quando escrita como restrição de coluna, e

```
CREATE TABLE produtos (
    cod_prod   integer,
    nome       text,
    preco      numeric,
    UNIQUE (cod_prod)
);
```

quando escrita como restrição de tabela.

Se uma restrição de unicidade fizer referência a um grupo de colunas, as colunas deverão ser listadas separadas por vírgula:

```
CREATE TABLE exemplo (
    a integer,
    b integer,
    c integer,
    UNIQUE (a, c)
);
```

Isto especifica que a combinação dos valores das colunas indicadas deve ser único para toda a tabela, embora não seja necessário que cada uma das colunas seja única (o que geralmente não é).

Também é possível atribuir nomes às restrições de unicidade:

```
CREATE TABLE produtos (
    cod_prod   integer CONSTRAINT unq_cod_prod UNIQUE,
    nome       text,
    preco      numeric
);
```

De um modo geral, uma restrição de unicidade é violada quando existem duas ou mais linhas na tabela onde os valores de todas as colunas incluídas na restrição são iguais. Entretanto, os valores nulos não são considerados iguais nesta comparação. Isto significa que, mesmo na presença da restrição de unicidade, é possível armazenar um número

ilimitado de linhas que contenham o valor nulo em pelo menos uma das colunas da restrição. Este comportamento está em conformidade com o padrão SQL, mas já ouvimos dizer que outros bancos de dados SQL não seguem esta regra. Portanto, seja cauteloso ao desenvolver aplicativos onde se pretenda haver portabilidade. [2] [3] [4]

5.3.4. Chaves primárias

Tecnicamente a restrição de chave primária é simplesmente a combinação da restrição de unicidade com a restrição de não-nulo. Portanto, as duas definições de tabela abaixo aceitam os mesmos dados:

```
CREATE TABLE produtos (
    cod_prod   integer UNIQUE NOT NULL,
    nome       text,
    preco      numeric
);
```

```
CREATE TABLE produtos (
    cod_prod   integer PRIMARY KEY,
    nome       text,
    preco      numeric
);
```

As chaves primárias também podem restringir mais de uma coluna; a sintaxe é semelhante à da restrição de unicidade:

```
CREATE TABLE exemplo (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

A chave primária indica que a coluna, ou grupo de colunas, pode ser utilizada como identificador único das linhas da tabela (Isto é uma conseqüência direta da definição da chave primária. Deve ser observado que a restrição de unicidade não fornece, por si só, um identificador único, porque não exclui os valores nulos). A chave primária é útil tanto para fins de documentação quanto para os aplicativos cliente. Por exemplo, um aplicativo contendo uma Interface de Usuário Gráfica (GUI), que permite modificar os valores das linhas, provavelmente necessita conhecer a chave primária da tabela para poder identificar as linhas de forma única.

Uma tabela pode ter no máximo uma chave primária (embora possa ter muitas restrições de unicidade e de não-nulo). A teoria de banco de dados relacional dita que toda tabela deve ter uma chave primária. Esta regra não é imposta pelo PostgreSQL, mas normalmente é melhor segui-la.

5.3.5. Chaves Estrangeiras

A restrição de chave estrangeira especifica que o valor da coluna (ou grupo de colunas) deve corresponder a algum valor existente em uma linha de outra tabela. Diz-se que a chave estrangeira mantém a *integridade referencial* entre duas tabelas relacionadas.

Supondo que já temos a tabela de produtos utilizada diversas vezes anteriormente:

```
CREATE TABLE produtos (
    cod_prod   integer PRIMARY KEY,
    nome       text,
    preco      numeric
);
```

Agora vamos assumir a existência de uma tabela armazenando os pedidos destes produtos. Desejamos garantir que a tabela de pedidos contenha somente pedidos de produtos que realmente existem. Para isso é definida uma restrição de chave estrangeira na tabela de pedidos fazendo referência à tabela de produtos:

```
CREATE TABLE pedidos (
    cod_pedido  integer PRIMARY KEY,
    cod_prod    integer REFERENCES produtos (cod_prod),
    quantidade  integer
);
```

Isto torna impossível criar um pedido com cod_prod não existente na tabela de produtos.

Nesta situação é dito que a tabela de pedidos é a tabela *que faz referência*, e a tabela de produtos é a tabela *referenciada*. Da mesma forma existem colunas fazendo referência e sendo referenciadas.

O comando acima pode ser abreviado escrevendo

```
CREATE TABLE pedidos (
    cod_pedido integer PRIMARY KEY,
    cod_prod   integer REFERENCES produtos,
    quantidade integer
);
```

porque, na ausência da lista de colunas, a chave primária da tabela referenciada é usada como a coluna referenciada.

A chave estrangeira também pode restringir e referenciar um grupo de colunas. Como usual, é necessário ser escrito na forma de restrição de tabela. Abaixo está mostrado um exemplo artificial da sintaxe:

```
CREATE TABLE t1 (
  a integer PRIMARY KEY,
  b integer,
  c integer,
  FOREIGN KEY (b, c) REFERENCES outra_tabela (c1, c2)
);
```

Obviamente, o número e tipo das colunas na restrição devem corresponder ao número e tipo das colunas referenciadas.

Pode ser atribuído um nome à restrição de chave estrangeira da forma habitual.

Uma tabela pode conter mais de uma restrição de chave estrangeira, o que é utilizado para implementar relacionamentos muitos-para-muitos entre tabelas. Digamos que existam as tabelas de produtos e de pedidos, e desejamos permitir que um pedido posa conter vários produtos (o que não é permitido na estrutura anterior). Podemos, então, utilizar a seguinte estrutura de tabela:

```
CREATE TABLE produtos (
    cod_prod   integer PRIMARY KEY,
    nome       text,
    preco      numeric
);

CREATE TABLE pedidos (
    cod_pedido       integer PRIMARY KEY,
    endereco_entrega text,
    ...
);

CREATE TABLE itens_pedidos (
    cod_prod    integer REFERENCES produtos,
    cod_pedido  integer REFERENCES pedidos,
    quantidade  integer,
    PRIMARY KEY (cod_prod, cod_pedido)
);
```

Deve ser observado, também, que a chave primária está sobreposta às chaves estrangeiras na última tabela.

Sabemos que a chave estrangeira não permite a criação de pedidos não relacionados com algum produto. Porém, o que acontece se um produto for removido após a criação de um pedido fazendo referência a este produto? A linguagem SQL permite tratar esta situação também. Intuitivamente temos algumas opções:

- Não permitir a exclusão de um produto referenciado
- Excluir o pedido também
- Algo mais?

Para ilustrar esta situação, vamos implementar a seguinte política no exemplo de relacionamento muitos-para-muitos acima: Quando se desejar remover um produto referenciado por um pedido (através de itens_pedidos), isto não será permitido. Se um pedido for removido, os itens do pedido também serão removidos.

```
CREATE TABLE produtos (
    cod_prod   integer PRIMARY KEY,
    nome       text,
    preco      numeric
);

CREATE TABLE pedidos (
    cod_pedido       integer PRIMARY KEY,
    endereco_entrega text,
    ...
);

CREATE TABLE itens_pedidos (
    cod_prod    integer REFERENCES produtos ON DELETE RESTRICT,
    cod_pedido  integer REFERENCES pedidos  ON DELETE CASCADE,
    quantidade  integer,
    PRIMARY KEY (cod_prod, cod_pedido)
);
```

As duas opções mais comuns são restringir, ou excluir em cascata. RESTRICT não permite excluir a linha referenciada. NO ACTION significa que, se as linhas referenciadas ainda existirem quando a restrição for verificada, será gerado um erro; este é o comportamento padrão se nada for especificado (A diferença essencial entre estas duas

opções é que NO ACTION permite postergar a verificação para mais tarde na transação, enquanto RESTRICT não permite). CASCADE especifica que, quando a linha referenciada é excluída, as linhas que fazem referência também devem ser excluídas automaticamente. Existem outras duas opções: SET NULL e SET DEFAULT. Estas opções fazem com que as colunas que fazem referência sejam definidas como nulo ou com o valor padrão, respectivamente, quando a linha referenciada é excluída. Deve ser observado que isto não evita a observância das restrições. Por exemplo, se uma ação especificar SET DEFAULT, mas o valor padrão não satisfizer a chave estrangeira, a operação não será bem-sucedida.

Semelhante a ON DELETE existe também ON UPDATE, chamada quando uma coluna referenciada é alterada (atualizada). As ações possíveis são as mesmas.

Mais informações sobre atualização e exclusão de dados podem ser encontradas no Capítulo 6.

Para terminar, devemos mencionar que a chave estrangeira deve referenciar colunas de uma chave primária ou de uma restrição de unicidade. Se a chave estrangeira fizer referência a uma restrição de unicidade, existem algumas possibilidades adicionais sobre como os valores nulos serão correspondidos. Esta parte está explicada na documentação de referência para *CREATE TABLE*.

5.3.6. Exemplos do tradutor

Exemplo 5-1. Restrição de unicidade com valor nulo em chave única simples

Abaixo são mostrados exemplos de inserção de linhas contendo valor nulo no campo da chave única simples da restrição de unicidade. Deve ser observado que, nestes exemplos, o PostgreSQL e o Oracle consideram os valores nulos diferentes.

PostgreSQL 8.0.0:

```
=> \pset null '(nulo)'
=> CREATE TABLE tbl_unique (c1 int UNIQUE);
=> INSERT INTO tbl_unique VALUES (1);
=> INSERT INTO tbl_unique VALUES (NULL);
=> INSERT INTO tbl_unique VALUES (NULL);
=> INSERT INTO tbl_unique VALUES (2);
=> SELECT * FROM tbl_unique;

   c1
--------
      1
 (nulo)
 (nulo)
      2
(4 linhas)
```

SQL Server 2000:

```
CREATE TABLE tbl_unique (c1 int UNIQUE)
INSERT INTO tbl_unique VALUES (1)
INSERT INTO tbl_unique VALUES (NULL)
INSERT INTO tbl_unique VALUES (NULL)
Violation of UNIQUE KEY constraint 'UQ__tbl_unique__37A5467C'.
Cannot insert duplicate key in object 'tbl_unique'.
The statement has been terminated.
INSERT INTO tbl_unique VALUES (2)
SELECT * FROM tbl_unique
```

```
c1
-----------
NULL
1
2
(3 row(s) affected)
```

Oracle 10g:

```
SQL> SET NULL (nulo)
SQL> CREATE TABLE tbl_unique (c1 int UNIQUE);
SQL> INSERT INTO tbl_unique VALUES (1);
SQL> INSERT INTO tbl_unique VALUES (NULL);
SQL> INSERT INTO tbl_unique VALUES (NULL);
SQL> INSERT INTO tbl_unique VALUES (2);
SQL> SELECT * FROM tbl_unique;

        C1
----------
         1
(nulo)
(nulo)
         2
```

Exemplo 5-2. Restrição de unicidade com valor nulo em chave única composta

Abaixo são mostrados exemplos de inserção de linhas contendo valores nulos em campos da chave única composta da restrição de unicidade. Deve ser observado que, nestes exemplos, somente o PostgreSQL considera os valores nulos diferentes.

PostgreSQL 8.0.0:

```
=> \pset null '(nulo)'
=> CREATE TABLE tbl_unique (c1 int, c2 int, UNIQUE (c1, c2));
=> INSERT INTO tbl_unique VALUES (1,1);
=> INSERT INTO tbl_unique VALUES (1,NULL);
=> INSERT INTO tbl_unique VALUES (NULL,1);
=> INSERT INTO tbl_unique VALUES (NULL,NULL);
=> INSERT INTO tbl_unique VALUES (1,NULL);
=> SELECT * FROM tbl_unique;

   c1    |   c2
---------+--------
       1 |      1
       1 | (nulo)
  (nulo) |      1
  (nulo) | (nulo)
       1 | (nulo)
(5 linhas)
```

SQL Server 2000:

```
CREATE TABLE tbl_unique (c1 int, c2 int, UNIQUE (c1, c2))
INSERT INTO tbl_unique VALUES (1,1)
INSERT INTO tbl_unique VALUES (1,NULL)
INSERT INTO tbl_unique VALUES (NULL,1)
INSERT INTO tbl_unique VALUES (NULL,NULL)
INSERT INTO tbl_unique VALUES (1,NULL)
Violation of UNIQUE KEY constraint 'UQ__tbl_unique__33D4B598'.
Cannot insert duplicate key in object 'tbl_unique'.
```

```
The statement has been terminated.
SELECT * FROM tbl_unique

c1          c2
----------- -----------
NULL        NULL
NULL        1
1           NULL
1           1
(4 row(s) affected)
```

Oracle 10g:

```
SQL> SET NULL (nulo)
SQL> CREATE TABLE tbl_unique (c1 int, c2 int, UNIQUE (c1, c2));
SQL> INSERT INTO tbl_unique VALUES (1,1);
SQL> INSERT INTO tbl_unique VALUES (1,NULL);
SQL> INSERT INTO tbl_unique VALUES (NULL,1);
SQL> INSERT INTO tbl_unique VALUES (NULL,NULL);
SQL> INSERT INTO tbl_unique VALUES (1,NULL);
INSERT INTO tbl_unique VALUES (1,NULL)
*
ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C005273) violated
SQL> SELECT * FROM tbl_unique;

        C1         C2
---------- ----------
         1          1
         1 (nulo)
(nulo)              1
(nulo)     (nulo)
```

Exemplo 5-3. Cadeia de caracteres vazia e valor nulo

Abaixo são mostrados exemplos de consulta a uma tabela contendo tanto o valor nulo quanto uma cadeia de caracteres vazia em uma coluna. Deve ser observado que apenas o Oracle 10g não faz distinção entre a cadeia de caracteres vazia e o valor nulo. Foram utilizados os seguintes comandos para criar e inserir dados na tabela em todos os gerenciadores de banco de dados:

```
CREATE TABLE c (c1 varchar(6), c2 varchar(6));
INSERT INTO c VALUES ('x', 'x');
INSERT INTO c VALUES ('VAZIA', '');
INSERT INTO c VALUES ('NULA', null);
```

PostgreSQL 8.0.0:

```
=> \pset null '(nulo)'
=> SELECT * FROM c WHERE c2 IS NULL;

  c1  |   c2
------+--------
 NULA | (nulo)
(1 linha)
```

SQL Server 2000:

```
SELECT * FROM c WHERE c2 IS NULL
```

```
c1     c2
------ ------
NULA   NULL
(1 row(s) affected)
```

Oracle 10g:

```
SQL> SET NULL (nulo)
SQL> SELECT * FROM c WHERE c2 IS NULL;

C1     C2
------ ------
VAZIA  (nulo)
NULA   (nulo)
```

DB2 9.1:

```
db2 > SELECT * FROM c WHERE c2 IS NULL;

C1     C2
------ ------
NULA   -
```

Exemplo 5-4. Coluna sem restrição de não nulo em chave primária

Abaixo são mostrados exemplos de criação de uma tabela definindo uma chave primária em uma coluna que não é definida como não aceitando o valor nulo. O padrão SQL diz que, neste caso, a restrição de não nulo é implícita, mas o DB2 não implementa desta forma, enquanto o PostgreSQL, o SQL Server e o Oracle seguem o padrão. Também são mostrados comandos para exibir a estrutura da tabela nestes gerenciadores de banco de dados.

PostgreSQL 8.0.0:

```
=> CREATE TABLE c (c1 int, PRIMARY KEY(c1));
=> \d c

        Tabela "public.c"
 Coluna |  Tipo   | Modificadores
--------+---------+---------------
 c1     | integer | not null
Índices:
    "c_pkey"chave primária, btree (c1)
```

SQL Server 2000:

```
CREATE TABLE c (c1 int, PRIMARY KEY(c1));
sp_help c

Name Owner Type       Created_datetime
---- ----- ---------- -----------------------
c    dbo   user table 2005-03-28 11:00:24.027

Column_name Type Computed Length Prec  Scale Nullable
----------- ---- -------- ------ ----- ----- -------- ...
c1          int  no       4      10    0     no

index_name      index_description                                  index_keys
--------------- -------------------------------------------------- ----------
```

```
PK__c__403A8C7D clustered, unique, primary key located on PRIMARY c1
```

Oracle 10g:

```
SQL> CREATE TABLE c (c1 int, PRIMARY KEY(c1));
SQL> DESCRIBE c

 Name Null?    Type
 ---- -------- ----------
 C1   NOT NULL NUMBER(38)

SQL> SELECT index_name, index_type, tablespace_name, uniqueness
 2  FROM user_indexes
 3  WHERE table_name='C';

INDEX_NAME  INDEX_TYPE TABLESPACE_NAME UNIQUENES
----------- ---------- --------------- ---------
SYS_C005276 NORMAL     USERS           UNIQUE
```

DB2 9.1:

```
db2 => CREATE TABLE c (c1 int, PRIMARY KEY(c1));

SQL0542N  "C1" não pode ser uma coluna de uma chave primária ou exclusiva,
porque pode conter valores nulos.  SQLSTATE=42831

db2 => CREATE TABLE c (c1 int NOT NULL, PRIMARY KEY(c1));

DB20000I  O comando SQL terminou com sucesso.

db2 => DESCRIBE TABLE c;

Nome da Esquema   Nome
coluna  do tipo   do tipo Tamanho Escala Nulos
------- --------- ------- ------- ------ ------
C1      SYSIBM    INTEGER       4      0 Não

  1 registro(s) selecionado(s).

db2 => DESCRIBE INDEXES FOR TABLE c SHOW DETAIL;

Esquema do Nome do            Regra     Número de
índice     índice             unicidade colunas   Nomes das colunas
---------- ------------------ --------- --------- ------------------
SYSIBM     SQL050328184542990 P                 1 +C1

  1 registro(s) selecionado(s).
```

Notas

[1]

truth-value — Na lógica, o valor verdade, ou valor-verdade, é um valor indicando até que ponto uma declaração é verdadeira; Na lógica clássica, os únicos valores verdade possíveis são verdade e falso. Entretanto, são possíveis outros valores em outras lógicas. A lógica intuicionista simples possui os valores verdade: verdade, falso e desconhecido. A lógica *fuzzy*, e outras formas de lógica multi-valoradas, também possuem mais valores verdade do que simplesmente verdade e falso; Algebricamente, o conjunto {verdade, falso} forma a lógica booleana simples. Dictionary.LaborLawTalk.com (N. do T.)

[2]

Oracle — Para satisfazer a restrição de unicidade, não podem haver haver duas linhas na tabela com o mesmo valor para a chave única. Entretanto, a chave única formada por uma única coluna pode conter nulos. Para satisfazer uma chave única composta, não podem haver duas

linhas na tabela ou na visão com a mesma combinação de valores nas colunas chave. Qualquer linha contendo nulo em todas as colunas chave satisfaz, automaticamente, a restrição. Entretanto, duas linhas contendo nulo em uma ou mais colunas chave, e a mesma combinação de valores para as outras colunas chave, violam a restrição. Oracle® Database SQL Reference 10g Release 1 (10.1) Part Number B10759-01 (N. do T.)

[3]

SQL Server — As restrições de unicidade podem ser utilizadas para garantir que não serão entrados valores duplicados em colunas específicas que não participam da chave primária. Embora tanto a restrição UNIQUE quanto a restrição PRIMARY KEY imponham a unicidade, deve ser utilizado UNIQUE em vez de PRIMARY KEY quando se deseja impor a unicidade de uma coluna, ou combinação de colunas, que não seja chave primária. Podem ser definidas várias restrições UNIQUE em uma tabela, mas somente uma restrição PRIMARY KEY. Também, ao contrário de PRIMARY KEY, a restrição UNIQUE permite o valor nulo. Entretanto, da mesma maneira que qualquer valor participante da restrição UNIQUE, é permitido somente um valor nulo por coluna. — A restrição UNIQUE também pode ser referenciada pela restrição FOREIGN KEY. — Quando é adicionada uma restrição UNIQUE a uma coluna, ou colunas, existentes em uma tabela, os dados existentes nas colunas são verificados para garantir que todos os valores, exceto os nulos, são únicos. SQL Server 2005 Books Online — UNIQUE Constraints (N. do T.)

[4]

DB2 — A restrição de unicidade é a regra que especifica que os valores de uma chave são válidos apenas se forem únicos na tabela. As colunas especificadas em uma restrição de unicidade devem ser definidas como NOT NULL. O gerenciador de banco de dados usa um índice único para para impor a unicidade da chave durante as alterações nas colunas da restrição de unicidade. A restrição de unicidade que é referenciada por uma chave estrangeira de uma restrição referencial é chamada de chave ancestral. Deve ser observado que existe uma distinção entre definir uma restrição de unicidade e criar um índice único: embora ambos imponham a unicidade, o índice único permite colunas com valor nulo e, geralmente, não pode ser utilizado como uma chave ancestral. DB2 Version 9 for Linux, UNIX, and Windows (N. do T.)

Anterior
Valor padrão

Principal
Acima

Próxima
Colunas do sistema