Pequeno. Rápido. Confiável.
Escolha os três.

Datatypes Em SQLite Versão 3

► Índice

1. Datatypes em SQLite

A maioria dos mecanismos de banco de dados SQL (todo o mecanismo de banco de dados SQL que não seja o SQLite, na medida em que sabemos) usa tipagem estática e rígida. Com a digitação estática, o tipo de dados de um valor é determinado pelo seu contêiner - a coluna particular na qual o valor é armazenado.

O SQLite usa um sistema de tipo dinâmico mais geral. Em SQLite, o tipo de dados de um valor está associado ao próprio valor, não ao seu contêiner. O sistema de tipo dinâmico do SQLite é compatível com os sistemas de tipos estáticos mais comuns de outros mecanismos de banco de dados, no sentido de que as instruções SQL que funcionam em bancos de dados digitados estaticamente devem funcionar da mesma maneira no SQLite. No entanto, a digitação dinâmica no SQLite permite que ele faça coisas que não são possíveis em bancos de dados tradicionais com digitação rígida.

2. Classes de armazenamento e tipos de dados

Cada valor armazenado em um banco de dados SQLite (ou manipulado pelo mecanismo de banco de dados) possui uma das seguintes classes de armazenamento:

- **NULL** . O valor é um valor NULL.
- **INTEGER** . O valor é um inteiro assinado, armazenado em 1, 2, 3, 4, 6 ou 8 bytes dependendo da magnitude do valor.
- **REAL** . O valor é um valor de ponto flutuante, armazenado como um número de ponto flutuante IEEE de 8 bytes.
- **TEXTO** . O valor é uma string de texto, armazenada usando a codificação do banco de dados (UTF-8, UTF-16BE ou UTF-16LE).
- **BLOB** . O valor é um blob de dados, armazenado exatamente como foi inserido.

Uma classe de armazenamento é mais geral do que um tipo de dados. A classe de armazenamento INTEGER, por exemplo, inclui 6 tipos de dados inteiros diferentes de diferentes comprimentos. Isso faz diferença no disco. Mas assim que os valores INTEGER são lidos do disco e na memória para processamento, eles são convertidos para o tipo de dados mais geral (inteiro assinado de 8 bytes). E, na maioria das vezes, "classe de armazenamento" é indistinguível do "tipo de dados" e os dois termos podem ser usados de forma intercambiável.

Qualquer coluna em um banco de dados SQLite versão 3, exceto uma coluna INTEGER PRIMARY KEY , pode ser usada para armazenar um valor de qualquer classe de armazenamento.

Todos os valores nas instruções SQL, sejam eles literais incorporados no texto da instrução SQL ou parâmetros vinculados a instruções SQL pré - compiladas, possuem uma classe de armazenamento implícita. Sob as circunstâncias descritas abaixo, o mecanismo de banco de dados pode converter valores entre classes de armazenamento numérico (INTEGER e REAL) e TEXT durante a execução da consulta.

2.1. Tipo de dados booleano

O SQLite não possui uma classe de armazenamento booleana separada. Em vez disso, os valores booleanos são armazenados como números inteiros 0 (falso) e 1 (verdadeiro).

2.2. Datatype de data e hora

O SQLite não possui uma classe de armazenamento reservada para armazenar datas e / ou horários. Em vez disso, as funções de data e hora incorporadas do SQLite são capazes de armazenar datas e horas como valores TEXTO, REAL ou INTEGER:

- **TEXTO** como cadeias ISO8601 ("AAAA-MM-DD HH: MM: SS.SSS").
- **REAL** como números do dia juliano, o número de dias desde meio-dia em Greenwich em 24 de novembro de 4714 aC de acordo com o calendário gregoriano proletípico.
- **INTEGER** como Unix Time, o número de segundos desde 1970-01-01 00:00:00 UTC.

Os aplicativos podem optar por armazenar datas e horas em qualquer um desses formatos e converter livremente entre formatos usando as funções de data e hora incorporadas .

3. Tipo de afinidade

Os mecanismos de banco de dados SQL que usam a digitação rígida geralmente tentarão converter valores automaticamente para o tipo de dados apropriado. Considere isto:

```
CREATE TABLE t1 (a INT, b VARCHAR (10));
INSERT INTO t1 (a, b) VALUES ('123', 456);
```

O banco de dados com digitação rígida converterá a seqüência '123' em um inteiro 123 e o inteiro 456 em uma seqüência '456' antes de fazer a inserção.

Para maximizar a compatibilidade entre o SQLite e outros mecanismos de banco de dados, e para que o exemplo acima funcione no SQLite, como acontece com outros mecanismos de banco de dados SQL, o SQLite suporta o conceito de "tipo de afinidade" nas colunas. O tipo de afinidade de uma coluna é o tipo recomendado para os dados armazenados nessa coluna. A idéia importante aqui é que o tipo é recomendado, não é necessário. Qualquer coluna ainda pode armazenar qualquer tipo de dados. É apenas que algumas colunas, dada a escolha, preferem usar uma classe de armazenamento sobre outra. A classe de armazenamento preferencial para uma coluna é chamada de "afinidade".

Cada coluna em um banco de dados SQLite 3 é atribuída a uma das seguintes afinidades de tipo:

- TEXTO
- NUMÉRICO
- INTEGRA
- REAL
- BLOB

(Nota histórica: A afinidade do tipo "BLOB" costumava ser chamada "NENHUMA". Mas esse termo foi fácil de confundir com "sem afinidade" e, portanto, foi renomeado.)

Uma coluna com afinidade TEXT armazena todos os dados usando classes de armazenamento NULL, TEXT ou BLOB. Se os dados numéricos forem inseridos em uma coluna com afinidade TEXT, ele será convertido no formulário de texto antes de ser armazenado.

Uma coluna com afinidade NUMÉRICA pode conter valores usando as cinco classes de armazenamento. Quando os dados de texto são inseridos em uma coluna NUMERIC, a classe de armazenamento do texto é convertida em INTEGER ou REAL (por ordem de preferência) se essa conversão for sem perdas e reversíveis. Para conversões entre classes de armazenamento TEXT e REAL, a SQLite considera a conversão como sem perda e reversível se os primeiros 15 dígitos decimais significativos do número forem preservados. Se a conversão sem perdas de TEXT para INTEGER ou REAL não for possível, o valor é armazenado usando a classe de armazenamento TEXT. Nenhuma tentativa é feita para converter valores NULL ou BLOB.

Uma seqüência de caracteres pode parecer um literal de ponto flutuante com um ponto decimal e / ou uma notação de expoente, enquanto o valor pode ser expresso como um inteiro, a afinidade NUMÉRICA o converterá em um número inteiro. Portanto, a string '3.0e + 5' é armazenada em uma coluna com afinidade NUMERIC como o número inteiro 300000, não como o valor do ponto flutuante 300000.0.

Uma coluna que usa a afinidade INTEGER comporta-se igual a uma coluna com afinidade NUMÉRICA. A diferença entre a afinidade INTEGER e NUMERIC é apenas evidente em uma expressão CAST .

Uma coluna com afinidade REAL se comporta como uma coluna com afinidade NUMÉRICA, exceto que força valores inteiros em representação de ponto flutuante. (Como uma otimização interna, pequenos valores de ponto flutuante sem componente fracionário e armazenados em colunas com afinidade REAL são escritos no disco como números inteiros, a fim de ocupar menos espaço e são automaticamente convertidos novamente em ponto flutuante à medida que o valor é lido. a otimização é completamente invisível no nível SQL e só pode ser detectada examinando os bits brutos do arquivo de banco de dados.)

Uma coluna com afinidade BLOB não prefere uma classe de armazenamento sobre outra e nenhuma tentativa é feita para coagir dados de uma classe de armazenamento para outra.

3.1. Determinação da afinidade da coluna

A afinidade de uma coluna é determinada pelo tipo declarado da coluna, de acordo com as seguintes regras na ordem mostrada:

1. Se o tipo declarado contiver a string "INT", é atribuída a afinidade INTEGER.
2. Se o tipo declarado da coluna contiver qualquer uma das cadeias "CHAR", "CLOB" ou "TEXT", essa coluna possui afinidade TEXT. Observe que o tipo VARCHAR contém a string "CHAR" e, portanto, é atribuído a afinidade TEXT.
3. Se o tipo declarado para uma coluna contiver a string "BLOB" ou se nenhum tipo for especificado, a coluna terá afinidade BLOB.
4. Se o tipo declarado para uma coluna contiver qualquer uma das cadeias "REAL", "FLOA" ou "DOUB", então a coluna tem afinidade REAL.
5. Caso contrário, a afinidade é NUMÉRICA.

Observe que a ordem das regras para determinar a afinidade da coluna é importante. Uma coluna cujo tipo declarado é "CHARINT" irá coincidir com as regras 1 e 2, mas a primeira regra tem precedência e, portanto, a afinidade da coluna será INTEGER.

3.1.1. Exemplos de nomes de afinidade

A tabela a seguir mostra quantos nomes de tipos de dados comuns das implementações SQL mais tradicionais são convertidos em afinidades pelas cinco regras da seção anterior. Esta tabela mostra apenas um pequeno subconjunto dos nomes de tipos de dados que a SQLite aceitará. Observe que os argumentos numéricos entre parênteses que seguem o nome do tipo (ex: "VARCHAR (255)") são ignorados pelo SQLite. O SQLite não impõe restrições de comprimento (além do limite global global SQLITE_MAX_LENGTH) no comprimento de strings, BLOBs ou valores numéricos.

Tipos de exemplo do exemplo Declaração CREATE TABLE ou expressão CAST	**Afinidade resultante**	**Regra usada para determinar a afinidade**
INT INTEGRA TINYINT SMALLINT MEDIUMINT	INTEGRA	1

BIGINT UNNIGADO GRANDE INT INT2 INT8		
CARÁTER (20) VARCHAR (255) CARÁTER VARIÁVEL (255) NCHAR (55) CARÁTER NATIVO (70) NVARCHAR (100) TEXTO CLOB	TEXTO	2
BLOB *nenhum tipo de dados especificado*	BLOB	3
REAL DUPLO DUPLA PRECISÃO FLUTUADOR	REAL	4
NUMÉRICO DECIMAL (10,5) BOLEANO ENCONTRO DATA HORA	NUMÉRICO	5

Note-se que um tipo declarado de "PONTO FLUTUANTE" daria afinidade INTEGER, não afinidade REAL, devido ao "INT" no final de "PONTO". E o tipo declarado de "STRING" tem uma afinidade de NUMERIC, não TEXT.

3.2. Afinidade de expressões

Toda coluna de tabela tem uma afinidade de tipo (uma de BLOB, TEXT, INTEGER, REAL ou NUMERIC), mas as expressões não têm necessariamente uma afinidade.

A afinidade de expressão é determinada pelas seguintes regras:

- O operando direito de um operador IN ou NOT IN não tem afinidade se o operando for uma lista e tiver a mesma afinidade que a afinidade da expressão do conjunto de resultados se o operando for SELECT.
- Quando uma expressão é uma referência simples a uma coluna de uma tabela real (não uma VISTA ou uma subconsulta), a expressão tem a mesma afinidade que a coluna da tabela.
    - Parênteses ao redor do nome da coluna são ignorados. Portanto, se X e YZ são nomes de colunas, então (X) e (YZ) também são considerados nomes de colunas e têm a afinidade das colunas correspondentes.
    - Qualquer operador aplicado aos nomes das colunas, incluindo o operador "+" unário não-op, converte o nome da coluna em uma expressão que sempre não possui afinidade. Portanto, mesmo que X e YZ sejam nomes de colunas, as expressões + X e + YZ não são nomes de colunas e não têm afinidade.
- Uma expressão do formulário "CAST (*expr tipo* AS)" tem uma afinidade que é a mesma que uma coluna com um tipo declarado de " *tipo* ".
- Caso contrário, uma expressão não tem afinidade.

3.3. Afinidade de coluna para visualizações e subconsultas

As "colunas" de uma sub -procuração de cláusula VIEW ou FROM são realmente as expressões no conjunto de resultados da instrução SELECT que implementa o VIEW ou a subconsulta. Assim, a afinidade para colunas de uma VISTA ou subconsulta é determinada pelas regras de afinidade de expressão acima. Considere um exemplo:

```
CREATE TABLE t1 (a INT, b TEXT, c REAL);
CREATE VIEW v1 (x, y, z) AS SELECT b, a + c, 42 FROM t1 WHERE b! = 11;
```

A afinidade da coluna v1.x será a mesma que a afinidade de t1.b (INTEGER), uma vez que v1.x mapeia diretamente em t1.b. Mas as colunas v1.y e v1.z ambas não têm afinidade, uma vez que essas colunas mapeiam a expressão a + c e 42, e as expressões sempre não têm afinidade.

Quando a instrução SELECT que implementa uma subconsulta de cláusula VIEW ou FROM é um composto SELECT , a afinidade de cada suposta coluna do VIEW ou subconsulta será a afinidade da coluna de resultados correspondente para uma das instruções SELECT individuais que compõem o composto . No entanto, é indeterminado qual das instruções SELECT será usado para determinar a afinidade. Diferentes declarações SELECT constituintes podem ser usadas para determinar a afinidade em diferentes momentos durante a avaliação da consulta. A melhor prática é evitar misturar afinidades em um composto SELECT.

3.4. Exemplo de comportamento de afinidade de coluna

O SQL a seguir demonstra como o SQLite usa afinidade de coluna para digitar conversões quando os valores são inseridos em uma tabela.

```
CREATE TABLE t1 (
    t TEXT, - afinidade de texto pela regra 2
    nu NUMÉRIC, - afinidade numérica pela regra 5
    INTEGER, - afinidade inteira pela regra 1
    r REAL, - afinidade real pela regra 4
    não BLOB - sem afinidade pela regra 3
);
```

```
- Valores armazenados como TEXT, INTEGER, INTEGER, REAL, TEXT.
INSERIR PARA VALORES T1 ('500.0', '500.0', '500.0', '500.0', '500.0');
SELECTE typeof (t), typeof (nu), typeof (i), typeof (r), typeof (não) FROM t1;
texto | inteiro | inteiro | real | texto

- Valores armazenados como TEXT, INTEGER, INTEGER, REAL, REAL.
DELETE FROM t1;
INSERIR EM VALORES T1 (500,0, 500,0, 500,0, 500,0, 500,0);
SELECTE typeof (t), typeof (nu), typeof (i), typeof (r), typeof (não) FROM t1;
texto | inteiro | inteiro | real | real

- Valores armazenados como TEXT, INTEGER, INTEGER, REAL, INTEGER.
DELETE FROM t1;
INSERTE PARA VALORES T1 (500, 500, 500, 500, 500);
SELECTE typeof (t), typeof (nu), typeof (i), typeof (r), typeof (não) FROM t1;
texto | inteiro | inteiro | real | inteiro

- BLOBs são sempre armazenados como BLOBs, independentemente da afinidade da coluna.
DELETE FROM t1;
INSERT IN T1 VALUES (x'0500 ', x'0500', x'0500 ', x'0500', x'0500 ');
SELECTE typeof (t), typeof (nu), typeof (i), typeof (r), typeof (não) FROM t1;
blob | blob | blob | blob | blob

- NULLs também não são afetados pela afinidade
DELETE FROM t1;
INSERT IN T1 VALUES (NULL, NULL, NULL, NULL, NULL);
SELECTE typeof (t), typeof (nu), typeof (i), typeof (r), typeof (não) FROM t1;
nulo | nulo | nulo | nulo | nulo
```

4. Expressões de comparação

SQLite versão 3 tem o conjunto usual de operadores de comparação de SQL, incluindo "=", "==", "<", "<=", ">", "> =", "! =", "", "IN" , "NOT IN", "ENTRE", "IS" e "IS NOT",.

4.1. Ordem de classificação

Os resultados de uma comparação dependem das classes de armazenamento dos operandos, de acordo com as seguintes regras:

- Um valor com classe de armazenamento NULL é considerado menor do que qualquer outro valor (incluindo outro valor com a classe de armazenamento NULL).
- Um valor INTEIRO ou REAL é inferior a qualquer valor TEXTO ou BLOB. Quando um INTEGER ou REAL é comparado com outro INTEGER ou REAL, uma comparação numérica é realizada.
- Um valor TEXT é inferior a um valor BLOB. Quando dois valores de TEXT são comparados, uma seqüência de agrupamento apropriada é usada para determinar o resultado.
- Quando dois valores BLOB são comparados, o resultado é determinado usando memcmp ().

4.2. Conversões de tipos antes da comparação

O SQLite pode tentar converter valores entre as classes de armazenamento INTEGER, REAL e / ou TEXT antes de realizar uma comparação. Seja ou não qualquer tentativa de conversão antes da comparação, dependerá do tipo de afinidade dos operandos.

Para "aplicar afinidade" significa converter um operando em uma determinada classe de armazenamento se e somente se a conversão for sem perdas e reversível. A afinidade é aplicada aos operandos de um operador de comparação antes da comparação de acordo com as seguintes regras na ordem mostrada:

- Se um operando tem afinidade INTEGER, REAL ou NUMERIC e o outro operando tem TEXT ou BLOB ou nenhuma afinidade, então a afinidade NUMERIC é aplicada a outro operando.
- Se um operando tiver afinidade TEXT e o outro não tiver afinidade, então a afinidade TEXT é aplicada ao outro operando.
- Caso contrário, nenhuma afinidade é aplicada e ambos os operandos são comparados como estão.

A expressão "A BETWEEN b AND c" é tratada como duas comparações binárias separadas "a> = b AND a <= c", mesmo que isso signifique afimidades diferentes aplicadas a 'a' em cada uma das comparações. As conversões de tipo de dados em comparações da forma "x IN (SELECT y ...)" são tratadas é se a comparação fosse realmente "x = y". A expressão "a IN (x, y, z, ...)" é equivalente a "a = + x OR a = + y OR a = + z OU ...". Em outras palavras, os valores à direita do operador IN (os valores "x", "y" e "z" neste exemplo) são considerados sem afinidade, mesmo que sejam valores de coluna ou expressões CAST .

4.3. Exemplo Comparativo

```
CREATE TABLE t1 (
    um TEXTO, - afinidade de texto
    b NÚMERO, - afinidade numérica
    c BLOB, - sem afinidade
    d - sem afinidade
);

- Os valores serão armazenados como TEXT, INTEGER, TEXT e INTEGER respectivamente
INSERIR PARA VALORES T1 ('500', '500', '500', 500);
SELECIONE o typeof (a), typeof (b), typeof (c), typeof (d) FROM t1;
texto | número inteiro | texto | número inteiro

- Como a coluna "a" tem afinidade de texto, valores numéricos no
- lado direito das comparações são convertidos em texto antes
- a comparação ocorre.
SELECIONE um <40, a <60, a <600 FROM t1;
0 | 1 | 1

- A afinidade do texto é aplicada aos operandos à direita, mas desde
- eles já são TEXTO este é um não-op;  não ocorrem conversões.
```

```
SELECIONE um <'40', a <'60', a <'600' FROM t1;
0 | 1 | 1

- A coluna "b" tem afinidade numérica e, portanto, a afinidade numérica é aplicada
- para os operandos à direita.  Como os operandos já são numéricos,
- a aplicação da afinidade é um não-op;  não ocorrem conversões.  Todos
- os valores são comparados numericamente.
SELECCIONE B <40, b <60, b <600 FROM t1;
0 | 0 | 1

- A afinidade numérica é aplicada aos operandos à direita, convertendo-os
- de texto para números inteiros.  Em seguida, ocorre uma comparação numérica.
SELECCIONAR b <'40', b <'60', b <'600' FROM t1;
0 | 0 | 1

- Não ocorrem conversões de afinidade.  Os valores do lado direito de todos
- classe de armazenamento INTEGER que são sempre menores que os valores de TEXTO
-- à esquerda.
SELECCIONE c <40, c <60, c <600 FROM t1;
0 | 0 | 0

- Não ocorrem conversões de afinidade.  Os valores são comparados como TEXT.
SELECTE c <'40', c <'60', c <'600' FROM t1;
0 | 1 | 1

- Não ocorrem conversões de afinidade.  Os valores do lado direito de todos
- classe de armazenamento INTEGER que compara numericamente com o INTEGER
- valores à esquerda.
SELECIONE d <40, d <60, d <600 FROM t1;
0 | 0 | 1

- Não ocorrem conversões de afinidade.  Os valores INTEGER à esquerda são
- sempre inferior aos valores de TEXTO à direita.
SELECIONE d <'40', d <'60', d <'600' FROM t1;
1 | 1 | 1
```

Todo o resultado no exemplo é o mesmo se as comparações forem comutadas - se as expressões da forma "a <40" forem reescritas como "40> a".

5. Operadores

Todos os operadores matemáticos (+, -, *, /,%, <<, >>, &, e |) lançam ambos os operandos para a classe de armazenamento NUMERIC antes de serem executados. O elenco é realizado, mesmo que seja perplexo e irreversível. Um operando NULL em um operador matemático produz um resultado NULL. Um operando em um operador matemático que não parece de forma alguma numérica e não é NULL é convertido em 0 ou 0.0.

6. Seleção, Agrupamento e SELECTs compostos

Quando os resultados da consulta são ordenados por uma cláusula ORDER BY, os valores com a classe de armazenamento NULL vêm primeiro, seguidos pelos valores INTEGER e REAL intercalados em ordem numérica, seguidos de valores TEXT na ordem de sequência de agrupamento e, finalmente, valores BLOB na ordem memcmp (). Nenhuma conversão de classe de armazenamento ocorre antes do ordenamento.

Quando o agrupamento de valores com os valores da cláusula GROUP BY com diferentes classes de armazenamento é considerado distinto, exceto para valores INTEGER e REAL que são considerados iguais se forem numericamente iguais. Nenhuma afinidade é aplicada a qualquer valor como resultado de uma cláusula GROUP by.

Os operadores compostos SELECT UNION, INTERSECT e EXCEPT realizam comparações implícitas entre valores. Nenhuma afinidade é aplicada aos operandos de comparação para as comparações implícitas associadas a UNION, INTERSECT ou EXCEPT - os valores são comparados como estão.

7. Seqüências de agrupamento

Quando o SQLite compara duas seqüências de caracteres, ele usa uma sequência de agrupamento ou uma função de agrupamento (duas palavras para a mesma coisa) para determinar qual string é maior ou se as duas cadeias de caracteres são iguais. O SQLite possui três funções de intercalação integradas: BINARY, NOCASE e RTRIM.

- **BINARY** - Compara dados de string usando memcmp (), independentemente da codificação de texto.
- **NOCASE** - O mesmo que binário, exceto os 26 caracteres em maiúscula de ASCII são dobrados para os equivalentes de minúsculas antes da comparação ser realizada. Observe que apenas os caracteres ASCII são dobrados. O SQLite não tenta fazer o dobramento completo da caixa UTF devido ao tamanho das tabelas necessárias.
- **RTRIM** - O mesmo que o binário, exceto que os caracteres do espaço final são ignorados.

Um aplicativo pode registrar funções de agrupamento adicionais usando a interface sqlite3_create_collation () .

7.1. Atribuindo Seqüências de intercalação do SQL

Cada coluna de cada tabela possui uma função de agrupamento associada. Se nenhuma função de classificação for explicitamente definida, a função de agrupamento será padrão para BINARY. A cláusula COLLATE da definição da coluna é usada para definir funções alternativas de agrupamento para uma coluna.

As regras para determinar qual função de classificação para usar para um operador de comparação binário (=, <,>, <=,> =,! =, IS e IS NOT) são as seguintes:

1. Se um ou outro operando tiver uma atribuição de função de agrupamento explícita usando o operador COLLATE posto , a função de classificação explícita é usada para comparação, com precedência para a função de agrupamento do operando esquerdo.

2. Se qualquer um dos operandos for uma coluna, a função de agrupamento dessa coluna é usada com precedência para o operando esquerdo. Para os propósitos da frase anterior, um nome de coluna precedido por um ou mais operadores "+" unários ainda é considerado um nome de coluna.

3. Caso contrário, a função de classificação BINARY é usada para comparação.

Um operando de uma comparação é considerado como tendo uma atribuição de função de intercalação explícita (regra 1 acima) se qualquer subexpressão do operando usar o operador COLLATE postfix. Assim, se um operador COLLATE é usado em qualquer lugar em uma expressão de comparação, a função de agrupamento definida por esse operador é usada para comparação de seqüência, independentemente das colunas de tabela que possam ser parte dessa expressão. Se duas ou mais subexpressões do operador COLLATE aparecem em qualquer lugar em uma comparação, a função de agrupamento mais explícita da esquerda é usada independentemente de quão profundamente os operadores COLLATE estão aninhados na expressão e independentemente de como a expressão é entre parênteses.

A expressão "x BETWEEN y e z" é logicamente equivalente a duas comparações "x> = y AND x <= z" e funciona em relação às funções de agrupamento como se fossem duas comparações separadas. A expressão "x IN (SELECT y ...)" é tratada da mesma forma que a expressão "x = y" para fins de determinação da sequência de agrupamento. A seqüência de agrupamento usada para expressões da forma "x IN (y, z, ...)" é a seqüência de agrupamento de x.

Os termos da cláusula ORDER BY que faz parte de uma instrução SELECT podem ser atribuídos a uma seqüência de classificação usando o operador COLLATE , caso em que a função de classificação especificada é usada para classificação. Caso contrário, se a expressão ordenada por uma cláusula ORDER BY for uma coluna, a seqüência de agrupamento da coluna é usada para determinar a ordem de classificação. Se a expressão não for uma coluna e não tiver uma cláusula COLLATE, então a seqüência de agrupamento BINARY é usada.