続SQL

データベースと情報検索　第4回

横浜市立大学　坂巻顕太郎
kentaro.sakamaki@gmail.com

デフォルト値

- 挿入した行のある列の値が指定されていない場合、空欄にはそれぞれの列のデフォルト値が入る

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric DEFAULT 9.99
);
```

  - 明示的に宣言されない場合のデフォルト値はNULL値
  - デフォルト値は式で表すことも可能

検査制約

- 特定の列の値が論理値の式を満たす（真の値）ように指定

```
-- 単独の列を参照
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0)
);

-- 複数の列を参照
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

非NULL制約

- 列がNULL値を取らないことを指定

```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric
);
```

一意性制約

- 列を制約する場合

```
CREATE TABLE products (
    product_no integer UNIQUE,
    name text,
    price numeric
);
```

- 列のグループを参照する場合（列で同じでも組み合わせは一意）

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

一意性制約におけるNULLについて

- NULL値は等価とはみなされない
  - 一意性制約があったとしても、制約対象の列の少なくとも1つにNULL値を持つ行を複数格納することができる
    - この振舞いは標準SQLに準拠している
  - （この規則に従わないSQLデータベースがある）

プライマリキー制約

- 一意性制約と非NULL制約を組み合わせたもの

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

外部キー

- 他のテーブルの行の値と一致しなければならない
  - 関連する2つのテーブルの参照整合性を維持

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

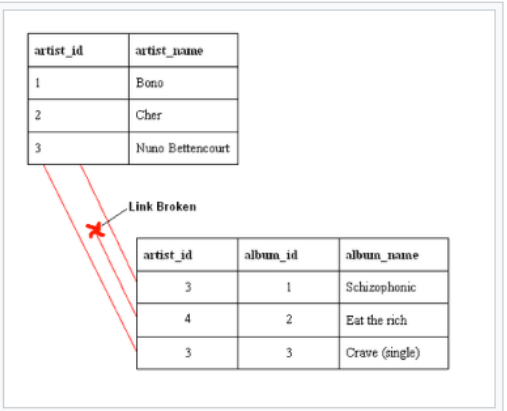
参照整合性

18の言語版

出典: フリー百科事典『ウィキペディア（Wikipedia）』

この記事は検証可能な参考文献や出典が全く示されていないか、不十分です。出典を追加して記事の信頼性向上にご協力ください。
出典検索[?]: "参照整合性" – ニュース・書籍・スカラー・CiNii・J-STAGE・NDL・dlib.jp・ジャパンサーチ・TWL（2023年1月）

参照整合性 (さんしょうせいごうせい、英: referential integrity) は、コンピュータの関係データベースの関係モデルにおいて2つの関連しあった関係変数 (表、テーブル) の間の一貫性 (データ完全性) をいう。 参照整合性は、多くの場合、主キーもしくは主キー以外の候補キーと、外部キーの、組み合わせにより、強制適用される。 参照整合性が強制適用されると、外部キーが宣言された関係変数の外部キーを構成する属性 (列、カラム) の値は、その関係変数の親となる関係変数の主キーの値もしくは主キー以外の候補キーの値として存在しなければならない。 例えば、別の関係変数の外部キーにより参照されている組 (タプル、行) を削除することは、参照整合性を破壊してしまうことになるため、関係データベース管理システム (RDBMS) は参照整合性を保つべく通常は削除の実行を阻止する。 例外として、参照している外部キーを含む組を連鎖して削除することを伴って、削除を実行できる場合があり、この場合は参照整合性が保たれる。 外部キーにより参照されている組を削除することができるかどうかは、データ定義言語 (DDL) による参照整合性制約の定義により定義される。



参照整合性が強制適用されていない関係データベースの例。この例では、アルバム関係変数 (アルバムテーブル) の外部キー (**artist_id**) の値に存在しないアーティストを参照しているものがある。

外部キーの利用

- 外部キーの役割
  - productsテーブルに存在しない非NULLのproduct_no項目を使用して注文を作成（ordersテーブルに入力）することはできない

- テーブル間の関係
  - ordersテーブルを参照テーブル、productテーブルを被参照テーブルとよぶ
  - 参照列と被参照列もある

テーブルの変更

- 列の追加・削除
  - ALTER TABLE products ADD COLUMN description text;
  - ALTER TABLE products DROP COLUMN description;
- 制約の追加・削除
  - ALTER TABLE products ADD CHECK (name <> '');
  - ALTER TABLE products DROP CONSTRAINT some_name;
- 列のデータ型の変更
  - ALTER TABLE products ALTER COLUMN price TYPE numeric(10,2);
- 列名・テーブル名の変更
  - ALTER TABLE products RENAME COLUMN product_no TO product_number;
  - ALTER TABLE products RENAME TO items;

データの更新と削除

- 一般にSQLでは行に対して一意のIDを指定しない
  - どの行を更新するかを直接指定できない場合があるため、更新する行が満たすべき条件を指定する
- プライマリキーを設定している場合、確実に個別の行を指定できる
  - ユーザーが宣言したのかどうかには依存しない
- 例：値段が5である全ての商品の値段を10に更新する
  - UPDATE products SET price = 10 WHERE price = 5;
- 例：productsテーブルから価格が10である全ての行を削除する
  - DELETE FROM products WHERE price = 10;

２つの問い合わせの結合（UNION, INTERSECT, EXCEPT)

- UNION
  - クエリ２の結果をクエリ１の結果に付加
    - この順序で実際に行が返される保証はない
    - UNION ALLを指定しないと、DISTINCTと同様、結果から重複している行が削除される
- INTERSECT
  - クエリ１の結果とクエリ２の結果の両方に含まれているすべての行を返す
    - INTERSECT ALLを使用しないと、重複している行は削除される
- EXCEPT
  - クエリ１の結果には含まれるが、クエリ２の結果には含まれないすべての行を返す
    - ２つの問い合わせの差とも言われる
    - EXCEPT ALL を使用しないと、重複している行は削除される

2つの問い合わせの和、積、差が算出できる条件

- 2つの問い合わせは「union互換」でなければいけない
  - 2つの問い合わせが、同じ数の列を返し、対応する列は互換性のあるデータ型でなければならない

括弧を使用した評価の順序の制御

- 括弧がない場合
  - INTERSECTはUNIONとEXCEPTよりも強く結合する
  - UNIONとEXCEPTは左から右に関連付けられる
  - query1 UNION query2 INTERSECT query3
    = query1 UNION (query2 INTERSECT query3)

- 個々のクエリを括弧で囲む場合
  - クエリがLIMITのような句を使用する必要がある場合に重要で、括弧がないと構文エラーが発生する場合がある
  - SELECT a FROM b UNION SELECT x FROM y LIMIT 10
    = (SELECT a FROM b UNION SELECT x FROM y) LIMIT 10
    - SELECT a FROM b UNION (SELECT x FROM y LIMIT 10)ではない

LIMITとOFFSET

- LIMIT
  - 数を指定すると、指定した行数より多くの行が返されることはない
    - 問い合わせの結果が指定した行数より少なければ、それより少なくなる
  - LIMIT ALLは、LIMIT句を省略した場合と同じ
    - LIMITの引数がNULLの場合も同じ

- OFFSET
  - 行を返し始める前に飛ばす行数を指定
    - OFFSET 0は、OFFSET句を省略した場合と同じ
    - OFFSETの引数がNULLの場合も同じ

- OFFSETとLIMITの両者を指定した場合
  - OFFSET分の行を飛ばしてから、返されるLIMIT行を数え始める

比較演算子

- 数値、文字列、日付、時刻などの自然な順序付けを持つすべての組み込みデータ型に適用される
  - 要素となるデータ型が比較可能なら、配列、複合データ型、範囲も比較可能

演算子	説明
datatype < *datatype* → boolean	小なり
datatype > *datatype* → boolean	大なり
datatype <= *datatype* → boolean	等しいかそれ以下
datatype >= *datatype* → boolean	等しいかそれ以上
datatype = *datatype* → boolean	等しい
datatype <> *datatype* → boolean	等しくない
datatype != *datatype* → boolean	等しくない

比較述語

述語: 説明: 例:	*datatype* BETWEEN *datatype* AND *datatype* → boolean 間にある（範囲の端点を含む） 2 BETWEEN 1 AND 3 → t, 2 BETWEEN 3 AND 1 → f
述語: 説明: 例:	*datatype* NOT BETWEEN *datatype* AND *datatype* → boolean 間にない（BETWEENの否定） 2 NOT BETWEEN 1 AND 3 → f
述語: 説明: 例:	*datatype* IS DISTINCT FROM *datatype* → boolean NULLを比較可能な値とした上で、等しくない 1 IS DISTINCT FROM NULL → t , NULL IS DISTINCT FROM NULL → f
述語: 説明: 例:	*datatype* IS NOT DISTINCT FROM *datatype* → boolean NULLを比較可能な値とした上で、等しい 1 IS NOT DISTINCT FROM NULL → f, NULL IS NOT DISTINCT FROM NULL → t
述語: 説明: 例:	*datatype* IS NULL → boolean 値がNULLかどうか検査する 1.5 IS NULL → f
述語: 説明: 例:	*datatype* IS NOT NULL → boolean 値がNULLではないかどうか検査する 'null' IS NOT NULL → t

算術関数

関数	説明	例	結果
abs(x)	絶対値	abs(-17.4)	17.4
ceil(dp or numeric)	引数より小さくない最小の整数	ceil(-42.8)	-42
exp(dp or numeric)	指数	exp(1.0)	2.71828182845905
floor(dp or numeric)	引数より大きくない最大の整数	floor(-42.8)	-43
ln(dp or numeric)	自然対数	ln(2.0)	0.693147180559945
log(dp or numeric)	10を底とした対数（常用対数）	log(100.0)	2
pi()	″円周率（π）″定数	pi()	3.14159265358979
power(a dp, b dp)	aのb乗	power(9.0, 3.0)	729
random()	0.0～1.0の乱数値	random()	
round(dp or numeric)	四捨五入	round(42.4)	42
round(v numeric, s int)	sの桁で四捨五入	round(42.4382, 2)	42.44
sqrt(dp or numeric)	平方根	sqrt(2.0)	1.4142135623731
trunc(dp or numeric)	切り捨て	trunc(42.8)	42
trunc(v numeric, s int)	sの桁で切り捨て	trunc(42.4382, 2)	42.43

文字列関数1

述語: 説明: 例:	text || text → text 2つの文字列を結合する 'Post' || 'greSQL' → PostgreSQL
述語: 説明: 例:	text || anynonarray → text（anynonarray || text → text） 非文字列の入力をテキストに変換したのちに2つの文字列を結合する 'Value: ' || 42 → Value: 42
述語: 説明: 例:	position (*substring* text IN *string* text) → integer *string*中の*substring*で指定する文字列の最初の開始位置を返す 0ならその文字列は存在しない position('om' in 'Thomas') → 3
述語: 説明: 例:	substring (*string* text [FROM *start* integer] [FOR *count* integer]) → text *start*が指定されていれば*start*番目の文字で始まる*string*の部分文字列を返す *count*が指定されていれば*count*数の文字を取り出す substring('Thomas' from 2 for 3) → hom substring('Thomas' from 3) → omas substring('Thomas' for 2) → Th

文字列関数2

述語: 説明: 例:	lower (text) → text データベースの照合順のルールに従い、文字列をすべて小文字に変換する lower('TOM') → tom
述語: 説明: 例:	upper (text) → text データベースの照合順のルールに従い、文字列をすべて大文字に変換する upper('tom') → TOM

パターンマッチ（LIKE式）

- パターンがパーセント記号もしくはアンダースコアを含んでいない場合
  - パターンは文字列そのもので、LIKE式は等号演算子のように振舞う
  - 'abc' LIKE 'abc'　true
  - 'abc' LIKE 'c'　false

- パターンがパーセント記号もしくはアンダースコアを含んでいる場合
  - アンダースコア（_）は任意の一文字との一致、パーセント記号（%）は0文字以上の並びとの一致を意味する
  - 'abc' LIKE 'a%'　true
  - 'abc' LIKE '_b_'　true

- LIKEによるパターン一致は常に文字列全体に対して行われる
  - 文字列内の任意位置における並びと一致させたい場合には、パーセント記号を先頭と末尾に付ける必要がある

SIMILAR TO正規表現

- SIMILAR TO演算子
  - パターンが与えられた文字列に一致するかどうかにより、真もしくは偽を返す
  - 標準SQLの正規表現定義を使用してパターンを解釈するという点以外は、LIKEに類似している

- 動作特性
  - LIKEと同様、SIMILAR TO演算子は、そのパターンが文字列全体に一致した場合のみ真を返す
    - パターンが文字列の一部分であっても一致する一般的な正規表現の動作とは異なる
  - LIKEと同様、SIMILAR TOでは、%および_を、それぞれ任意の文字列および任意の単一文字を意味するワイルドカード文字として使用する

CASE句

- 他のプログラミング言語のif/else構文に類似した汎用条件式

```
SELECT * FROM test;

 a
---
 1
 2
 3
```

```
SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 'other'
       END
    FROM test;

 a | case
---+-------
 1 | one
 2 | two
 3 | other
```

型の変換（cast）

- cast(A as B)　-- Aを型Bに変換する
  - A: 型変換をする文字・数字・日付などの列
  - B: 変換する型を指定
  - select cast('0123.45' as numeric);

- 「::型」で表現する方法もある
  - 「‘123’::integer」、「123::text」など
  - select '0123'::integer;