Bazy danych

Andrzej Łachwa, UJ, 2015
andrzej.lachwa@uj.edu.pl

6/14

Etapy tworzenia bazy danych

- analiza (zdefiniowanie i poznanie wycinka rzeczywistości, dla którego powstaje baza, wskazanie przyszłych użytkowników i określenie ich wymagań)
- projektowanie (tworzenie modelu logicznego i więzów integralności danych, transformacja do modelu fizycznego oraz jego normalizacja)
- implementacja (wybór systemu zarządzania i wykonanie skryptu definiującego znormalizowany model fizyczny)
- wdrożenie (wypełnienie bazy danymi początkowymi i uruchomienie aplikacji przeznaczonych dla użytkowników)

Semantyka schematu relacyjnej bazy danych

Schemat bazy danych składa się ze schematów relacji i więzów integralności. Każdy schemat relacji ma atrybuty. Atrybuty posiadają znaczenia związane z rzeczywistością, której strukturę informacji odzwierciedlono w schemacie.

Relacją opartą na schemacie nazywamy konkretny zbiór krotek zgodnych ze strukturą i spełniający przyjęte więzy integralności.

Każdą relację opartą na schemacie interpretuje się jako zbiór zdań lub faktów opisujących obszar analizy.

Jeżeli projekt koncepcyjny zostanie wykonany starannie, a po nim dokona się systematycznego odwzorowania na relacje, większość materiału semantycznego zostanie uwzględniona i projekt wynikowy będzie niemal na pewno zrozumiały. Ogólnie rzecz biorąc, im łatwiej jest objaśnić znaczenie relacji, tym lepszy jest projekt jej schematu.

Ramez Elmasri, Shamkant B. Navathe, str. 321

Właściwe pogrupowanie atrybutów w schematy relacji ma ogromny wpływ na przestrzeń pamięciową zajmowaną przez bazę.

Źle zaprojektowany schemat prowadzi do niepotrzebnego zwiększenia koniecznej do przechowywania danych przestrzeni pamięciowej i powoduje m.in. anomalie wstawiania, anomalie usuwania i anomalie modyfikowania.

W razie występowania w schematach anomalii tego rodzaju należy je dokładnie opisać i zapewnić, by procedury aktualizacji danych działały, mimo tych anomalii, poprawnie.

Należy unikać sytuacji, kiedy w wielu krotkach występują wartości NULL. Prowadzi to do zwiększenie przestrzeni pamięciowej oraz do trudności interpretacyjnych.

Wartości puste mogą występować tylko w sytuacjach wyjątkowych i trzeba zadbać o to, by były zawsze właściwie interpretowane.

Oto trzy często stosowane interpretacje wartości pustej:

- atrybut nie ma zastosowania,
- wartość atrybutu nie jest znana,
- wartość jest znana, ale jest nieobecna.

Należy unikać schematów, w których występują pasujące do siebie atrybuty (o tych samych nazwach i dziedzinach), które nie tworzą par: klucz główny, klucz obcy. Złączenia naturalne wykonane po takich atrybutach zwykle prowadzą do powstawania tzw. fałszywych krotek, tj. informacji, które nie odpowiadają opisywanej rzeczywistości.

Aby atrybuty były właściwie pogrupowane w schematy, schematy te były łatwe do interpretacji w obszarze analizy i nie pojawiały się wymienione wyżej problemy, to
– oprócz starannie wykonanego i przemyślanego diagramu, oraz poprawnego mapowania na schemat relacyjnej bazy – trzeba jeszcze przeprowadzić proces NORMALIZACJI.

Mówiąc najbardziej ogólnie proces normalizacji prowadzi do schematu w którym każdy fakt może być zapisany tylko jeden raz! Proces normalizacji korzysta z semantycznych zależności między atrybutami. Najważniejsze są zależności funkcyjne i zależności wielowartościowe.

W dużym uproszczeniu zależność funkcyjna oznacza, że jeżeli znamy wartość jednego atrybutu, to możemy określić wartość drugiego. Na przykład, gdy znamy numer części, to możemy określić jej wagę czy kolor.

Zależność wielowartościowa oznacza, że jeżeli znamy wartość jednego atrybutu, to możemy określić zbiory wartości innych atrybutów. Na przykład, gdy znamy numer części, to możemy określić numery wszystkich części, z których jest zbudowana.

Wybrane elementy MySQL

W różnych produktach SQL spotkamy różne rozwiązania szczegółowe i pewne odchylenia od standardu języka. Skrypt języka SQL (zgodny z jakimś standardem) musi być niemal zawsze dopasowany do serwera na którym ma zostać uruchomiony!!!

Poniżej wybrałem kilkanaście elementów specyficznych dla serwera MySQL.

1.

```
SELECT Name FROM City WHERE Name LIKE 'kr%';

SELECT Name AS "Nazwy 6 literowe na literę K"
   FROM City
   WHERE Name LIKE 'k_____';

SELECT Name FROM City WHERE Name LIKE BINARY 'Kr%';
```

– domyślenie wielkość liter nie ma znaczenia (ale można zdefiniować kolumnę wymuszając rozróżnianie wielkości liter, np.

```
Name VARCHAR(20) BINARY
```

2.

```
SELECT 'David_' LIKE 'David\_';
-> 1

SELECT 'David_' LIKE 'David|_' ESCAPE '|';
-> 1

SELECT 10 LIKE '1%';
-> 1
```

```
mysql> SELECT filename, filename LIKE '%\\' FROM t1;
+--------------+----------------------+
| filename     | filename LIKE '%\\' |
+--------------+----------------------+
| C:           | 0                    |
| C:\          | 1                    |
| C:\Programs  | 0                    |
| C:\Programs\ | 1                    |
+--------------+----------------------+
```

3.

```
CREATE TABLE foo (bar VARCHAR(10));
INSERT INTO foo VALUES (NULL), (NULL);


SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%';
> 0

SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%';
> 0

SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR
bar IS NULL;
> 2
```

4.

IF(expr1,expr2,expr3)

Jeżeli *expr1* jest TRUE (*expr1* <> 0 oraz *expr1* <> NULL) to zwraca *expr2*; w przeciwnym wypadku zwraca *expr3*.

IF() zwraca wartość liczbową lub łańcuch znaków, zależnie od kontekstu.

```
SELECT IF(1>2,2,3);
-> 3
SELECT IF(1<2,'yes','no');
-> 'yes'
```

CASE ma dwie wersje.

Pierwsza:

```
CASE WHEN [condition] THEN result [WHEN [condition]
THEN result ...] [ELSE result] END

SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
-> 'true'
```

Druga wersja CASE:

```
CASE value WHEN [compare_value] THEN result [WHEN
[compare_value] THEN result ...] [ELSE result] END

SELECT CASE BINARY 'B' WHEN 'a' THEN 1 WHEN 'b' THEN
2 END;
-> NULL
```

5.

```
SELECT count(*) FROM City
    WHERE Population IS NULL;

SELECT IFNULL(101,'yes');
-> 101

SELECT IFNULL(NULL,'yes');
-> 'yes'

SELECT Name, IFNULL(Region,'---?---') FROM Country …

SELECT NULLIF(101,101);
-> NULL

SELECT NULLIF(101,102);
-> 101
```

6.

```
SELECT * FROM City
    WHERE (Population BETWEEN 1000 AND 2000) AND
          (Population NOT BETWEEN 1200 AND 1800);


SELECT 2 BETWEEN 2 AND '3';
-> 1
```

7.

```
SELECT Name FROM City ORDER BY Name LIMIT 10,5;
```

- opuszcza pierwszych 10 wierszy, wyświetla 5 kolejnych i opuszcza wszystkie następne
-

8.

FIND_IN_SET(*str*,*strlist*)

- zwraca wartość od 1 do *n*, gdy *strlist* zawiera *n* elementów.

```
SELECT FIND_IN_SET('b','a,b,c,d');
-> 2
```

9.

FORMAT(*X*,*D*[,*locale*])

- zwraca *X* jako łańcuch znaków w formacie '#,###,###.##', zaokrąglonym do *D* miejsc dziesiętnych.

```
SELECT FORMAT(12332.123456, 4);
-> '12,332.1235'
SELECT FORMAT(12332.1,4);
-> '12,332.1000'
SELECT FORMAT(12332.2,0);
-> '12,332'
SELECT FORMAT(12332.2,2,'de_DE');
-> '12.332,20'
```

10.

```
SELECT DISTINCT Continent FROM Country;

SELECT max(Population), min(Population),
avg(Population), sum(Population), count(Population)
   FROM Country
   WHERE Continent LIKE 'Europe' AND
         Population IS NOT NULL;

SELECT Ci.Name, Co.Name
   FROM City Ci, Country Co
   WHERE Code=CountryCode;

  [WHERE Co.Code=Ci.CountryCode]
```

11.

```
CREATE TEMPORARY TABLE Names AS
    (SELECT Name FROM City
     UNION
     SELECT Name FROM Country);
```

Sprawdź:

```
SELECT count(*) FROM City;
SELECT count(*) FROM Country;
SELECT count(*) FROM Names;
```

i wyjaśnij!

12.

```
SELECT O.id "Nr zam.",LEFT(C.name,10) AS "Klient",
LEFT(P.name,10) AS "Produkt", O.payment_type AS
"Platnosc", DATE_FORMAT(O.date_ordered, '%d-%m-%Y')
AS "Data", I.price AS "Cena",  I.quantity AS "Ilosc"
FROM customer C, ord O,item I, product P WHERE
C.id=O.customer_id AND  O.id=I.ord_id AND P.id
=I.product_id AND O.payment_type = 'CREDIT' AND
O.date_ordered BETWEEN '1992-08-01' AND '1992-08-31'
ORDER BY O.id, P.name;
```

```
SELECT O.id "Nr zam.",
       LEFT(C.name, 20) "Klient",
       LEFT(C.city, 20) "Miasto",
       SUM(I.price * I.quantity) "Suma"
  FROM customer C, ord O, item I, product P
  WHERE C.id = O.customer_id AND
        O.id = I.ord_id AND
        P.id = I.product_id AND
        O.payment_type = 'CREDIT' AND
        O.date_ordered BETWEEN '1992-08-01' AND
            '1992-08-31'
  GROUP BY O.id, C.name, C.city
  ORDER BY O.id;
```

13.

```
SELECT first_name, last_name, salary, title
  FROM emp E1
  WHERE E1.salary <
       (SELECT AVG(salary) FROM emp E2
        WHERE E2.title = E1.title)
  ORDER BY title, salary;
```

14.

Funkcje RAND() i RAND(N) zwracają wartość losową z przedziału [0, 1). W drugim przypadku produkuje powtarzalną sekwencję liczb pseudolosowych.

```
mysql> CREATE TABLE t (i INT);
mysql> INSERT INTO t VALUES(1),(2),(3);
```

```
mysql> SELECT i, RAND() FROM  t;
+------+------------------+
| i | RAND()               |
+------+------------------+
| 1 | 0.61914388706828    |
| 2 | 0.93845168309142    |
| 3 | 0.83482678498591    |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND() FROM  t;
+------+------------------+
| i | RAND()               |
+------+------------------+
| 1 | 0.35877890638893    |
| 2 | 0.28941420772058    |
| 3 | 0.37073435016976    |
+------+------------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i | RAND(3)              |
+------+------------------+
| 1 | 0.90576975597606    |
| 2 | 0.37307905813035    |
| 3 | 0.14808605345719    |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i | RAND(3) |
+------+------------------+
| 1 | 0.90576975597606 |
| 2 | 0.37307905813035 |
| 3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.01 sec)
```

```
mysql> SELECT FLOOR(7+(RAND()*5));
```
zwraca całkowitą wartość losową z przedziału [7, 12)

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```
zwraca wiersze w kolejności losowej

```
mysql> SELECT * FROM tbl_name ORDER BY RAND()LIMIT 10;
```
zwraca 10 losowo wybranych wierszy

15.

```
SELECT SUBSTRING('Quadratically',5);
-> 'ratically'

SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'

SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'

SELECT SUBSTRING('Sakila', -3);
-> 'ila'

SELECT SUBSTRING('Sakila', -5, 3);
-> 'aki'

SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
-> 'ki'
```