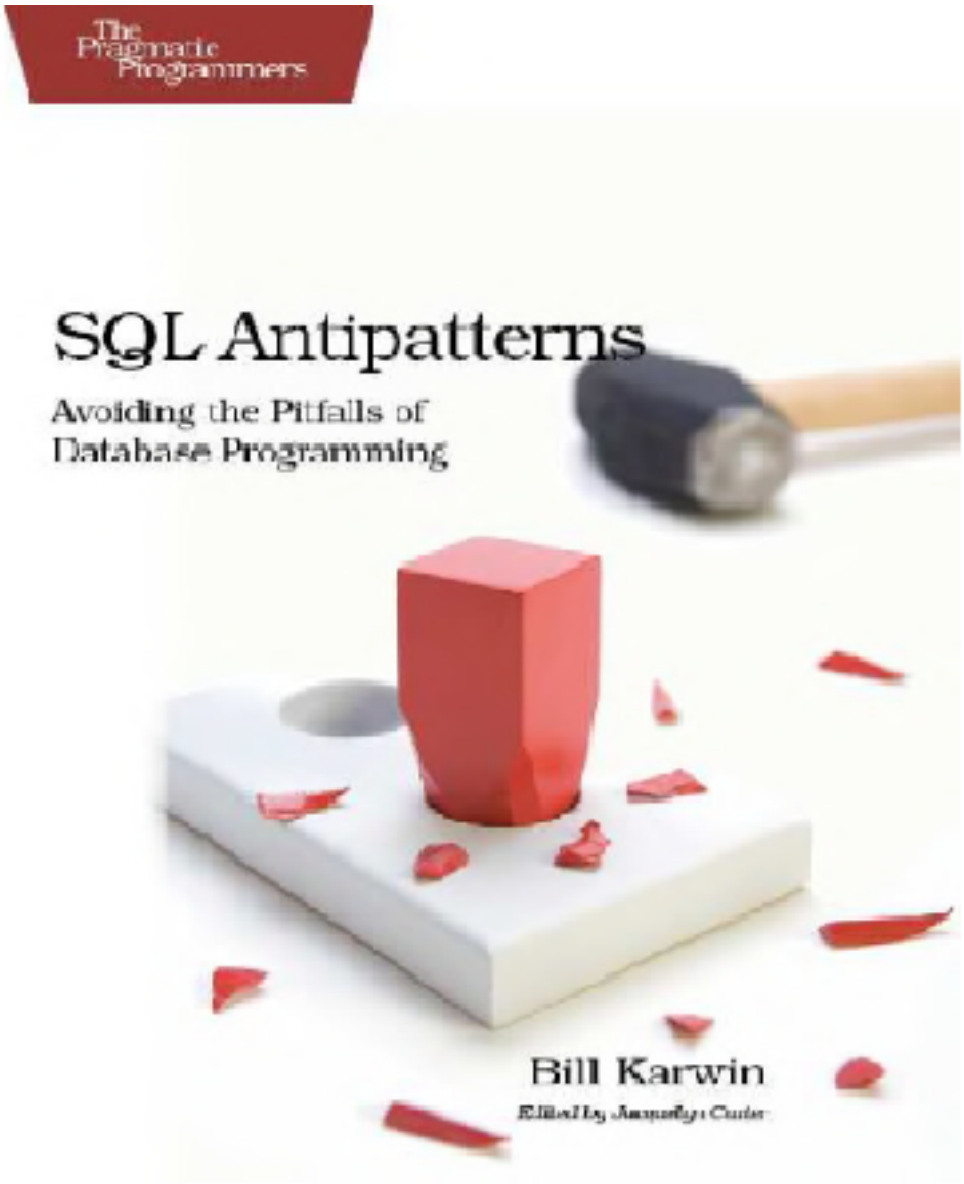
عنوان: **SQL Antipattern #1**
نویسنده: سید مرتضی حسینی
تاریخ: ۱۳۹۳/۰۴/۲۹ ۱۳:۲۰
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, SQL, Antipatterns

در این سلسله نوشتار قصد دارم ترجمه و خلاصه چندین فصل از کتاب ارزشمند ( SQL Antipatterns: Avoiding the Pitfalls of Database Programming (Pragmatic Programmers که حاصل تلاش گروه IT موسسه هدایت فرهیختگان جوان می‌باشد، را ارائه نمایم.




**بخش اول : Jaywalking**

در این بخش در حال توسعه ویژگی نرم افزاری هستیم که در آن هرکاربر به عنوان یک کاربر اصلی برای یک محصول تخصیص داده می‌شود. در طراحی اصلی ما فقط اجازه می  دهیم یک کاربر متعلق به هر محصول باشد، اما امکان چنین تقاضایی وجود دارد

که چند کاربر نیز به یک محصول اختصاص داده شوند.

در این صورت، اگر پایگاه داده را به نحوی تغییر دهیم که شناسه‌ی حساب کاربران را در لیستی که با کاما از یکدیگر جدا شده‌اند ذخیره نماییم، خیلی ساده‌تر به نظر می‌رسد نسبت به اینکه بصورت جداگانه آنها را ثبت نماییم.

برنامه نویسان معمولا برای جلوگیری از ایجاد جدول واسطی [1] که رابطه‌های چند به چند زیادی دارد از یک لیست که با کاما داده‌هایش از هم جدا شده‌اند، استفاده می‌کنند. بدین جهت اسم این بخش antipatten, jaywalking می‌باشد، زیرا jaywalking نیز عملیاتی است که از تقاطع جلوگیری می‌کند.

### 1.1 هدف: ذخیره کردن چندین صفت

طراحی کردن جدولی که ستون آن فقط یک مقدار دارد، بسیار ساده و آسان می‌باشد. شما می‌توانید نوع داده‌ایی که متعلق به ستون می‌باشد را انتخاب نمایید. مثلا از نوع (int,date...)

چگونه می‌توانیم مجموعه‌ایی از مقادیری که به یکدیگر مرتبط هستند را در یک ستون ذخیره نماییم ؟

در مثال ردیابی خطای پایگاه داده، ما یک محصول را به یک کاربر، با استفاده از ستونی که نوع آن integer است، مرتبط می‌نماییم. هر کاربر ممکن است محصولاتی داشته باشد و هر محصول به یک contact اشاره کند. بنابراین یک ارتباط چند به یک بین محصولات و کاربر برقرار است. برعکس این موضوع نیز صادق است؛ یعنی امکان دارد هر محصول متعلق به چندین کاربر باشد و یک ارتباط یک به چند ایجاد شود. در زیر جدول محصولات را به صورت عادی آورده شده است:

```
CREATE TABLE Products (
product_id SERIAL PRIMARY KEY,
product_name VARCHAR(1000),
account_id BIGINT UNSIGNED,
-- . . .
FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
);

INSERT INTO Products (product_id, product_name, account_id)
VALUES (DEFAULT, 'Visual TurboBuilder' , 12);
```

### 2.1 Antipattern : قالب بندی لیست هایی که با کاما از یکدیگر جدا شده اند

برای اینکه تغییرات در پایگاه داده را به حداقل برسانید، می‌توانید نوع شناسه‌ی کاربر را از نوع varchar قرار دهید. در این صورت می‌توانید چندین شناسه‌ی کاربر که با کاما از یکدیگر جدا شده‌اند را در یک ستون ذخیره نمایید. پس در جدول محصولات، شناسه‌ی کاربران را از نوع varchar قرار می‌دهیم.

```
CREATE TABLE Products (
product_id SERIAL PRIMARY KEY,
product_name VARCHAR(1000),
account_id VARCHAR(100), -- comma-separated list
-- . . .
);

INSERT INTO Products (product_id, product_name, account_id)
VALUES (DEFAULT, 'Visual TurboBuilder' , '12,34' );
```

اکنون مشکلات کارایی و جامعیت داده‌ها را در این راه حل پیشنهادی بررسی می‌نماییم .

### بدست آوردن محصولاتی برای یک کاربر خاص

بدلیل اینکه تمامی شناسه‌ی کاربران که بصورت کلید خارجی جدول Products می‌باشند به صورت رشته در یک فیلد ذخیره

شده‌اند و حالت ایندکس بودن آنها از دست رفته است، بدست آوردن محصولاتی برای یک کاربر خاص سخت می‌باشد. به عنوان مثال بدست آوردن محصولاتی که کاربری با شناسه‌ی 12 خریداری نموده به‌صورت زیر می‌باشد:

```
SELECT * FROM Products WHERE account_id REGEXP '[[:<:]]12[[:>:]]' ;
```

**بدست آوردن کاربرانی که یک محصول خاص خریداری نموده اند**

Join کردن account_id با Accounts table یعنی جدول مرجع نامناسب و غیر معمول می‌باشد. مساله مهمی که وجود دارد این است که زمانیکه بدین صورت شناسه‌ی کاربران را ذخیره می‌نماییم، ایندکس بودن آنها از بین می‌رود. کد زیر کاربرانی که محصولی با شناسه‌ی 123 خریداری کرده‌اند را محاسبه می‌کند.

```
SELECT * FROM Products AS p JOIN Accounts AS a
ON p.account_id REGEXP '[[:<:]]' || a.account_id || '[[:>:]]'
WHERE p.product_id = 123;
```

**ایجاد توابع تجمیعی [2]**

مبنای چنین توابعی از قبیل avg ,()count ,()sum() بدین صورت می‌باشد که بر روی گروهی از سطرها اعمال می‌شوند. با این حال با کمک ترفندهایی می‌توان برخی از این توابع را تولید نماییم هر چند برای همه آن‌ها این مساله صادق نمی‌باشد. اگر بخواهیم تعداد کاربران برای هر محصول را بدست بیاوریم ابتدا باید طول لیست شناسه‌ی کاربران را محاسبه کنیم، سپس کاما را از این لیست حذف کرده و طول جدید را از طول قبلی کم کرده و با یک جمع کنیم. نمونه کد آن در زیر آورده شده است:

```
SELECT product_id, LENGTH(account_id) - LENGTH(REPLACE(account_id, ',' , '' )) + 1
AS contacts_per_product
FROM Products;
```

**ویرایش کاربرانی که یک محصول خاص خریداری نموده‌اند**

ما به راحتی می‌توانیم یک شناسه‌ی جدید را به انتهای این رشته اضافه نماییم؛ فقط مرتب بودن آن را بهم میریزیم. در نمونه اسکریپت زیر گفته شده که در جدول محصولات برای محصولی با شناسه‌ی 123 بعد از کاما یک کاربر با شناسه‌ی 56 درج شود:

```
UPDATE Products
SET account_id = account_id || ',' || 56
WHERE product_id = 123;
```

برای حذف یک کاربر از لیست کافیست دو دستور sql را اجرا کرد: اولی برای fetch کردن شناسه‌ی کاربر از لیست و بعدی برای ذخیره کردن لیست ویرایش شده. بطور مثال تمامی افرادی که محصولی با شناسه‌ی 123 را خریداری کرده‌اند، از جدول محصولات انتخاب می‌کنیم. برای حذف کاربری با شناسه‌ی 34 از لیست کاربران، بر حسب کاما مقادیر لیست را در آرایه می‌ریزیم، شناسه‌ی مورد نظر را جستجو می‌کنیم و آن را حذف می‌کنیم. دوباره مقادیر درون آرایه را بصورت رشته درمیاوریم و جدول محصولات را با لیست جدید کاربران برای محصولی با شناسه‌ی 123 بروزرسانی می‌کنیم.

```
<?php
$stmt = $pdo->query(
"SELECT account_id FROM Products WHERE product_id = 123");
$row = $stmt->fetch();
$contact_list = $row['account_id' ];


// change list in PHP code
$value_to_remove = "34";
$contact_list = split(",", $contact_list);
$key_to_remove = array_search($value_to_remove, $contact_list);
unset($contact_list[$key_to_remove]);
$contact_list = join(",", $contact_list);
$stmt = $pdo->prepare(
"UPDATE Products SET account_id = ?
WHERE product_id = 123");
$stmt->execute(array($contact_list));
```

**اعتبارسنجی شناسه‌ی محصولات**

به دلیل آنکه نوع فیلد account_id،varchar می‌باشد احتمال این وجود دارد داده‌ای نامعتبر وارد نماییم چون پایگاه داده‌ها هیچ عکس العملی یا خطایی را نشان نمی‌دهد، فقط از لحاظ معنایی دچار مشکل می‌شویم. در نمونه زیر banana یک داده‌ی غیر معتبر می‌باشد و ارتباطی با شناسه‌ی کاربران ندارد.

```
INSERT INTO Products (product_id, product_name, account_id)
VALUES (DEFAULT, 'Visual TurboBuilder' , '12,34,banana' );
```

**انتخاب کردن کاراکتر جداکننده**

نکته قابل توجه این است که کاراکتری که بعنوان جداکننده در نظر می‌گیریم باید در هیچکدام از داده‌های ورودی ما امکان بودنش وجود نداشته باشد .

**محدودیت طول لیست**

در زمان طراحی جدول، برای هر یک از فیلدها باید حداکثر طولی را قرار دهیم. به عنوان نمونه ما اگر varchar(30) در نظر بگیریم بسته به تعداد کاراکترهایی که داده‌های ورودی ما دارند می‌توانیم جدول را پرنماییم. اسکریپت زیر شناسه‌ی کاربرانی که محصولی با شناسه‌ی 123 را خریده‌اند، ویرایش می‌کند. با این تفاوت که با توجه به محدودیت لیست، در نمونه‌ی اولی شناسه‌ی کاربران دو کاراکتری بوده و 10داده بعنوان ورودی پذیرفته است در حالیکه در نمونه‌ی دوم بخاطر اینکه شناسه‌ی کاربران شش کاراکتری می‌باشد فقط 4 شناسه می‌توانیم وارد نماییم.

```
UPDATE Products SET account_id = '10,14,18,22,26,30,34,38,42,46'
WHERE product_id = 123;
```

```
UPDATE Products SET account_id = '101418,222630,343842,467790'
WHERE product_id = 123;
```

**3.1 موارد تشخیص این Antipattern:**

سؤالات زیر نشان می‌دهند که Jaywalking antipattern مورد استفاده قرار گرفته است:

• حداکثر پذیرش این لیست برای داده ورودی چه میزان است؟

• چه کاراکتری هرگز در داده‌های ورودی این لیست ظاهر نمی‌شود؟ **4.1 مواردی که استفاده از این Antipattern مجاز است:**

دی نرمال کردن کارایی را افزایش می‌دهد. ذخیره کردن شناسه‌ها در یک لیست که با کاما از یکدیگر جدا شده‌اند نمونه بارزی از دی نرمال کردن می‌باشد. ما زمانی می‌توانیم از این مدل استفاده نماییم که به جدا کردن شناسه‌ها از هم نیاز نداشته باشیم. به همین ترتیب، اگر در برنامه، شما یک لیست را از یک منبع دیگر با این فرمت دریافت نمایید، به سادگی آن را در پایگاه داده خود به همان فرمت بصورت کامل می‌توانید ذخیره و بازیابی نمایید و احتیاجی به مقادیر جداگانه ندارید. درهنگام استفاده از پایگاه داده‌های دی‌نرمال دقت کنید. برای شروع از پایگاه داده نرمال استفاده کنید زیرا به کدهای برنامه شما امکان انعطاف بیشتری می‌دهد و کمک می‌کند تا پایگاه داده‌ها تمامیت داده(Data integrity) را حفظ کند. **5.1 راه حل: استفاده کردن از جدول واسط**

در این روش شناسه‌ی کاربران را در جدول جداگانه‌ایی که هر کدام از آنها یک سطر را به خود اختصاص داده اند، ذخیره می‌نماییم.

```
CREATE TABLE Contacts ( product_id BIGINT UNSIGNED NOT NULL,
account_id BIGINT UNSIGNED NOT NULL,
PRIMARY KEY (product_id, account_id),
FOREIGN KEY (product_id) REFERENCES Products(product_id),
FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
);
```

جدول Contacts یک جدول رابطه ایی بین جداول Products,Accounts

**بدست آوردن محصولات برای کاربران و موارد مربوط به آن**

ما براحتی می‌توانیم تمامی محصولاتی که مختص به یک کاربر هستند را بدست آوریم. در این شیوه خاصیت ایندکس بودن شناسه‌ی کاربران حفظ می‌شود به همین دلیل queryهای آن برای خواندن و بهینه کردن راحت‌تر می‌باشند. در این روش به کاراکتری برای جدا کردن ورودی‌ها از یکدیگر نیاز نداریم چون هر کدام از آنها در یک سطر جداگانه ثبت می‌شوند. برای ویرایش کردن کاربرانی که یک محصول را خریداری نموده اند، کافیست یک سطر از جدول واسط را اضافه یا حذف نماییم. درنمونه کد زیر، ابتدا در جدول Contacts کاربری با شناسه‌ی 34 که محصولی با شناسه‌ی 456 را خریداری کرده، درج شده است و در خط بعد عملیات حذف با شرط آنکه شناسه‌ی کاربر و محصول به ترتیب 34،456 باشد روی جدول Contacts اعمال شده است.

```
INSERT INTO Contacts (product_id, account_id) VALUES (456, 34);

DELETE FROM Contacts WHERE product_id = 456 AND account_id = 34;
```

**ایجاد توابع تجمیعی**

به عنوان نمونه در مثال زیر براحتی ما می‌توانیم تعداد محصولات در هر حساب کاربری را بدست آوریم:

```
SELECT account_id, COUNT(*) AS products_per_account
FROM Contacts
GROUP BY account_id;
```

**اعتبارسنجی شناسه محصولات**

از آنجاییکه مقادیری که در جدول قرار دارند کلید خارجی می‌باشند میتوان صحت اعتبار آنها را بررسی نمود. بعنوان مثال Contacts.account_id به Account.account_id اشاره می‌کند. در ضمن برای هر فیلد نوع آن را می‌توان مشخص کرد تا فقط همان نوع داده را بپذیرد.

**محدودیت طول لیست**

نسبت به روش قبلی تقریبا در این حالت محدودیتی برای تعداد کاراکترهای ورودی نداریم.

**مزیت‌های دیگر استفاده از جدول واسط**

کارایی روش دوم بهتر از حالت قبلی می‌باشد چون ایندکس بودن شناسه‌ها حفظ شده است. همچنین براحتی میتوانیم فیلدی را به این جدول اضافه نماییم مثلا (time, date... )

Intersection Table [1]

Aggregate [2]

## نظرات خوانندگان

نویسنده: م منفرد
تاریخ: ۱۳۹۳/۰۴/۳۰ ۰:۴۸

سلام
منظورتان از گروه it موسسه هدایت فرهیختگان چه بود؟ ایشان کتاب را ترجمه کردند؟ چگونه می‌توان آن را تهیه کرد؟
با تشکر

نویسنده: فرید
تاریخ: ۱۳۹۳/۰۶/۲۶ ۲۰:۱۸

سلام؛ به نظر میرسه این خصیصه مربوط به mysql است. من نتوانستم آن‌را در sqlserver اجرا کنم.