SQL Server 安全行级安全

不像一些其他 industrial-strength 数据库服务，SQL Server 缺乏一个内置保护个别数据记录的机制，称为行级安全。

行级安全

在发布 SQL Server 2000 和 SQL Server 2005 之间，微软对于安全变得非常严肃。比尔盖茨在 2002 年初所写的 Trustworthy Computing 备忘录，让 SQL Server 2005 接受大量的安全检查。一个新的安全策略是使授权计划远比以前的版本更细粒度化，让你细粒度的(fine-grained)控制谁可以对哪些对象操作。

增加安全粒度的一个明显的遗漏是它没有延伸到行级。你有很多在表级别的访问控制选项，但 SQL Server 在表内没有内置的访问控制机制，通常称为行级安全。在许多情况下这是一个问题，特别是当有保护数据的要求。例如，一个医学数据库中的患者表可能包含所有临床患者的数据，但只有医疗系统管理员有权限访问所有患者信息，而医生和护士只能访问他们直接照顾的患者信息。有许多其他的情况，需要允许访问一个表中记录的子集，甚至下降到单元(cell)级别。但 SQL Server 没有内置的行级别安全手段。

提示：表包含行和列，微软使用单元(cell)代表某一行某一列上的数据。有必要这样划分，因为有些场景需要保护表中的个别单元(cell)。在后面的章节中你会学习更多这方面的知识。

行级安全为数据库表中的个别记录提供细粒度的(fine-grained)访问控制权。所需的结果是，记录被一个查询的执行上下文过滤，这样用户只访问她有权限的记录。这种过滤使用相同的安全功能在 SQL Server 控制访问其他数据库和服务器

对象和操作这些对象。

由于 SQL Server 缺少内置行级安全，开发人员和管理员不得不想出各种巧妙的创意和技术来解决这个问题。虽然这些技术往往需要一点点额外的工作来建立，但是为了发挥细粒度的(fine-grained)控制访问表中记录的子集而不是整张表来说它是值得的。

该技术根据你的需求可以是相对简单的或可以是复杂的，结合访问控制的安全标签和分类级别。安全标签描述了数据项的敏感性，你将在这一篇了解到这项技术。

我们现在的假设是，行级安全最好在数据库级别实现。你可以在应用程序级别上控制访问数据库对象，但你必须在每一个使用该数据库的应用程序中实现安全层。在数据库实现行级安全意味着对应用程序是透明的，甚至应用程序开发人员不需要知道有发生过访问控制。

在云主机数据库需要的驱动下，微软终于开始专注 SQL Server 的行级安全。他们首先添加行级安全到 SQL Azure 数据库，它将会是 SQL Server 2016 的一部分。

视图实现行级安全

正如你所知，SQL Server 2016 会实现内置行级安全。但在它发布以及将你的数据迁移过去之前，我们只能通过自己的方法实现。让我们以一种常见的技术来看看行级安全的简单实现。尽管它很简单，它可以是一个更复杂的方法的基础。比如一个公司的客户记录包含敏感信息，如信用额度。只有系统管理员(或高管)和分配给客户的公司代表，能够查看客户数据，所有这一切都是简单地存储在一个单一的 Customer 表。

代码 10.1 先创建一个示例数据库(RowLevelSecurityDB)，创建一张表

(Customer)，然后往表中插入部分测试数据。一切都是相当简单的，表中包含 UserAccess 字段。这是数据库代码用于限制访问的字段。记住，这是一个简单的例子，有很多其他的方法来做这些。

```
    IF DB_ID('RowLevelSecurityDB') IS NOT NULL DROP DATABAS
E RowLevelSecurityDB;

    CREATE DATABASE RowLevelSecurityDB;

    GO

    USE RowLevelSecurityDB;

    GO


    -- Create the sample table that we want to protect with
row-level security

    CREATE TABLE Customer (

        CustId INT,

        Name NVARCHAR(30),

        City NVARCHAR(20),

        CreditLimit MONEY,

        SocialSecurityNumber NCHAR(11),

        FelonyConvictions INT,

        UserAccess NVARCHAR(50)

    );

    GO


    -- Add some data to the Customer table

    INSERT INTO dbo.Customer

           (CustId, Name, City, CreditLimit, SocialSecurityN
umber, FelonyConvictions, UserAccess)
```

```
    VALUES (1, N'Don Kiely', N'Fairbanks', 5.00, N'123-45-6
789', 17, N'UserOne'),

          (2, N'Kalen Delaney', N'Seattle', 500000.00, N'98
7-65-4321', 0, N'UserOne'),

          (3, N'Tony Davis', N'Cambridge', 5000.00, N'', 0,
 N'UserTwo'),

          (4, N'E.T. Barnette', N'Fairbanks', 0.00, N'555-6
6-7777', 47, N'UserOne'),

          (5, N'Soapy Smith', N'Sitka', 0.00, N'222-33-4444
', 32, N'UserTwo');
```

代码 10.1 创建示例数据库、表

提示：自定义的行级安全方案通常需要在某种程度上更改数据库结构，额外的 UserAccess 列只是其中一种实现方法。另一种流行处理更复杂场景的方法是创建一个多对多的表将行链接到用户或角色。

通过视图访问该表的数据。代码 10.2 创建 MyCustomersView 视图——基于用户执行上下文筛选数据。视图中有趣的部分是 where 子句。它使用 USER_NAME 函数来获取执行上下文的用户。然后查询语句返回与 UserAccess 列匹配的记录。另外，如果是 db_owner 数据库角色成员或者是 sysadmin 服务器角色，视图返回表中所有的行。

```
    IF object_id(N'dbo.MyCustomers', 'V') IS NOT NULL

        DROP VIEW dbo.MyCustomers;

    GO

    CREATE VIEW dbo.MyCustomersView AS

        SELECT CustId, Name, City, CreditLimit, SocialSecur
ityNumber, FelonyConvictions, UserAccess FROM dbo.Customer


        WHERE UserAccess = USER_NAME() OR
```

```
            IS_ROLEMEMBER('db_owner') = 1 OR
            IS_SRVROLEMEMBER('sysadmin') = 1;
    GO
```

代码 10.2 创建 MyCustomersView 视图

代码 10.3 创建了两个数据库用户。在本例中，用户不需要映射到服务器级别的登录名，当然可以映射，而且在大多数情况下应该映射。然后，代码拒绝用户对 Customer 表的所有权限，授予对视图的 SELECT 权限。这就阻止了用户能够直接访问表，但允许他们通过视图访问数据。如果你希望允许用户通过视图对数据进行更改，你可以授予这些权限。

```
    CREATE USER UserOne WITHOUT LOGIN;
    CREATE USER UserTwo WITHOUT LOGIN;
    GO
    -- Set permissions
    DENY SELECT, INSERT, UPDATE, DELETE ON dbo.Customer TO
UserOne, UserTwo;
    GRANT SELECT ON dbo.MyCustomersView TO UserOne, UserTw
o;
    GO
```

代码 10.3 创建用户分配权限

使用代码 10.4 测试行级安全。第一个查询以 sysadmin 身份执行，返回表中所有数据如图 10.1 所示：

```
    -- Test as admin
    SELECT * FROM dbo.MyCustomersView;    -- Should succeed
and return all rows
```

```
    GO

    -- Test as regular users

    EXECUTE AS USER = 'UserOne';

    SELECT * FROM dbo.Customer;           -- Should fail

    SELECT * FROM dbo.MyCustomersView;    -- Should succeed
and return 3 rows

    REVERT;

    GO

    EXECUTE AS USER = 'UserTwo';

    SELECT * FROM dbo.Customer;           -- Should fail

    SELECT * FROM dbo.MyCustomersView;    -- Should succeed
and return 2 rows

    REVERT;

    GO
```

代码 10.4 测试行级安全

结果 消息

	CustId	Name	City	CreditLimit	SocialSecurityNumber	FelonyConvictions	UserAccess
1	1	Don Kiely	Fairbanks	5.00	123-45-6789	17	UserOne
2	2	Kalen Delaney	Seattle	500000.00	987-65-4321	0	UserOne
3	3	Tony Davis	Cambridge	5000.00		0	UserTwo
4	4	E.T. Barnette	Fairbanks	0.00	555-66-7777	47	UserOne
5	5	Soapy Smith	Sitka	0.00	222-33-4444	32	UserTwo

图 10.1 sysadmin 身份查询视图

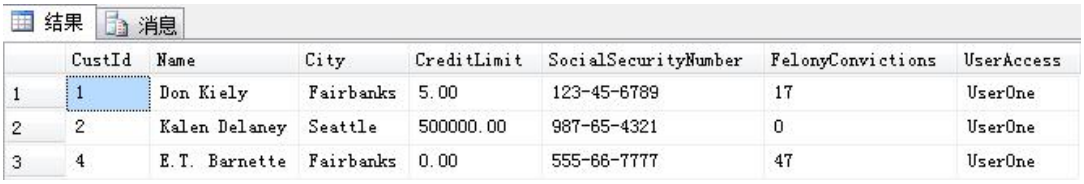
代码 10.4 中的第二段代码在 UserOne 用户上下文执行。例子中有两个查询语句。第一个试图从表中直接读取数据，测试 DENY 权限。第二个测试使用视图。正如预料，第一句失败(图 10.2)然而第二句返回三行数据(图 10.3)，UserAccess 列是 UserOne 的行。

结果 消息

消息 229，级别 14，状态 5，第 2 行
拒绝了对对象 'Customer' (数据库 'RowLevelSecurityDB'，架构 'dbo')的 SELECT 权限。

(3 行受影响)

图 10.2 以 UserOne 用户身份测试表和视图消息

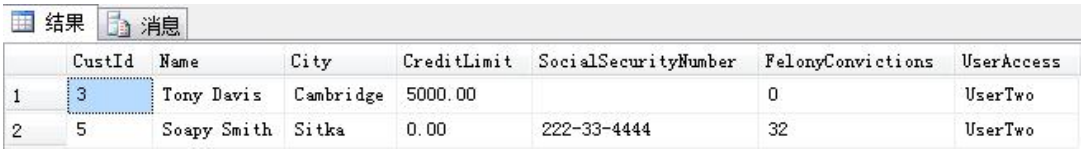


结果 消息

	CustId	Name	City	CreditLimit	SocialSecurityNumber	FelonyConvictions	UserAccess
1	1	Don Kiely	Fairbanks	5.00	123-45-6789	17	UserOne
2	2	Kalen Delaney	Seattle	500000.00	987-65-4321	0	UserOne
3	4	E.T. Barnette	Fairbanks	0.00	555-66-7777	47	UserOne

图 10.3 以 UserOne 用户身份执行视图返回记录

代码 10.4 中的最后一段代码，直接从表获取数据失败，从视图返回两行数据，UserTwo 有权限访问的行，如图 10.4 所示



结果 消息

	CustId	Name	City	CreditLimit	SocialSecurityNumber	FelonyConvictions	UserAccess
1	3	Tony Davis	Cambridge	5000.00		0	UserTwo
2	5	Soapy Smith	Sitka	0.00	222-33-4444	32	UserTwo

图 10.4 以 UserTwo 用户身份执行视图返回记录

这是一个非常简单的实施行级安全的方案。它是简单的，但有助于理解基本概念。但是这样做有几个问题：

->如果你需要保护多个表，你可能需要实施多个不同程度的复杂逻辑视图，以允许不同的用户和角色访问受保护的数据。当表结构发生改变时，这些视图需要与表保持同步。

->这个例子是基于用户的名字，并且假定每一行只能被一个用户访问。你很可能需要使用角色过滤的方式，你还必须处理用户名不唯一的情况。

->当一个用户名改变时，你必需更新一个或多个表中的数据。

最佳行级安全实践

一个更全面的、real-world-enabled 行级安全例子超出 SQL Server 安全系列的范围。微软在白皮书中已经发布了这样一个样本。

白皮书中描述的方法是为公共部门的安全需求，最高级别安全需求的分类数据库所设计。它使用安全标签和视图来提供细粒度的访问控制，甚至超过行级保护访

问个别单元(cell)。白皮书描述了一个这样的安全标签：

安全标签是描述数据项(对象)灵敏度的一串信息。它是一个包含一个或多个类别标记的字符串。每个主题的权限可以被认为是自己的标签。对比主题的标签与对象的标签，以确定对象的访问。

一个保护层次的例子，美国青睐使用 SECRET、TOP SECRET、UNCLASSIFIED 类别。这种设计允许多个访问标准 ,如安全类别的情况下 ,进一步限制于各项目或部门。例如 ,在美国国家安全局的拥有顶级机密类别的人不一定能够访问美国联邦调查局的机密文件。

实现这种横切、分层安全的方案 ,用于行和单元(cell)级安全需要 a fair bit of code，既要建立数据库对象(主要是表和视图)来存储和提供访问数据，还要维护系统。白皮书包括大量代码！

除了在白皮书中的代码，微软创建了一个 SQL Server 标签安全工具包，托管在 CodePlex 上，伴随着上述白皮书。该工具箱包括一个标签策略设计器应用程序、文档和各种使用方法的实例。一个 SQL Server 标签策略设计器的例子如图 10.5 所示。这个应用程序提供了一个很好的图形界面，让你可以定义你的类别。然后工具会产生创建所有必要对象的 SQL 代码 ,你可以在所选数据库下直接运行代码或者将代码保存到文件用于后续修改和执行。

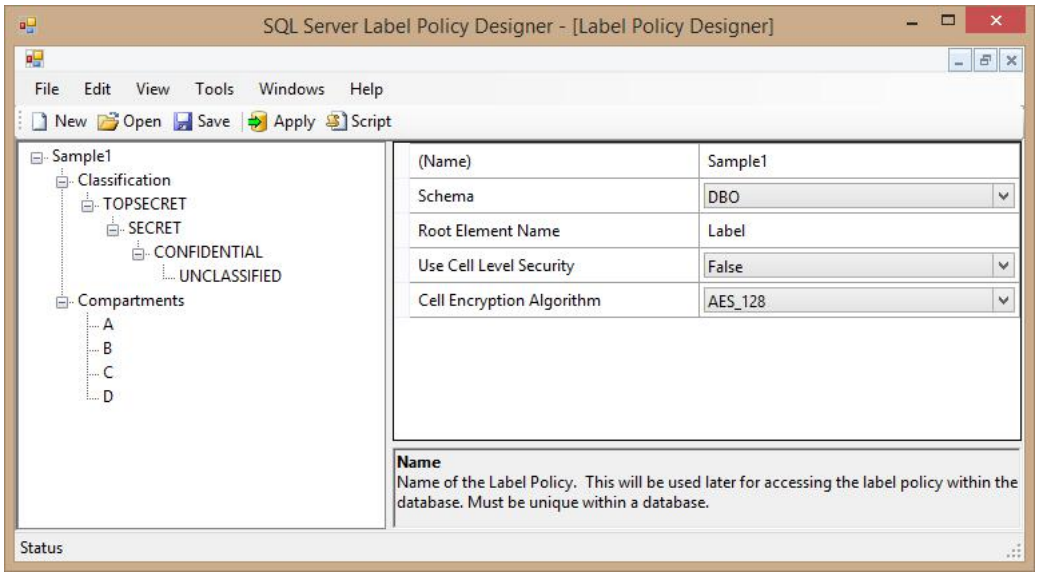


图 10.5 SQL Server 标签策略设计器

即使你不需要像白皮书中描述的那种灵活和精细的安全方案，你可以通过阅读白皮书和探索开发工具包学到很多。在你自己的数据库和应用程序实施行级安全的选择只会受限于你对 SQL Server 所提供众多工具的创造性。

SQL Server 2016 中的行级安全

创造性的需要，自定义行级安全方案在 SQL Server 2016 中将过时。微软已经宣布，该版本将包括内置的行级安全——其他数据库引擎一个使用很久的功能。该功能仍在开发中，所以这一篇不会深入覆盖。但它可能的工作方式是，你将定义一个安全谓词过滤作为内联、表值函数，当用户或应用程序访问受保护的数据时，它成为安全策略调用的一部分。代码 10.5 显示了 SQL Server 2016 CTP2 中的部分行级安全代码，这和前面使用 UserAccess 列和视图筛选/过滤数据的代码很相似。

```
    CREATE SCHEMA Security;

    GO

    CREATE FUNCTION Security.fn_securitypredicate(@SalesRe
p AS sysname)
```

```
        RETURNS TABLE
    WITH SCHEMABINDING
    AS
        RETURN SELECT 1 AS fn_securitypredicate_result
            WHERE @SalesRep = USER_NAME() OR USER_NAME() = '
Manager';
    GO
    CREATE SECURITY POLICY SalesFilter
        ADD FILTER PREDICATE Security.fn_securitypredicate
(SalesRep) ON dbo.Sales
        WITH (STATE = ON);
    GO
```

代码 10.5 SQL Server 2016 创建安全策略