**实验报告六**

|  |  |
| --- | --- |
| 实验序号：6 | 实验名称：安全机制 |
| 学号：2023015509 | 实验日期：2025年5月29日 |
| 姓名：胡林森 | 实验教室：C4 420机房 |

**一、实验目的**

1．掌握数据库安全管理机制中的常规方法，理解登录、用户、架构、角色、权限的概念、定义及使用；

2．掌握视图、存储过程、触发器的概念、定义及如何发挥特殊的安全控制作用。

**二、实验学时**

2学时

**三、实验准备**

1．SQL Server 2008的安全机制

和SQL Server早先的版本相比，SQL Server 2008在安全问题上作了较多的改进，让数据库管理与程序编写更为安全和富有弹性。其中主要的改变有：用户（User）和架构（Schema）定义的分离、SQL Server定制账号的密码可以遵循Windows系统安全性原则、可创建或装载证书（Certificate）、对称与非对称式加/解密数据表内的数据、签名与验证等。下面详细介绍这些新的安全机制。

**(1) SQL Server 2008主要安全层次**

SQL Server 2008的安全设置，可以根据设置安全的对象层次和主要可分配到这些对象上的安全层次来考虑，其主要的安全层次如表7-1所示。

表7-1 主要安全层次

|  |  |
| --- | --- |
| 安全层次 | 可分配的权限模型 |
| 操作系统 | 组（Group） |
| 域登录（Domain login） |
| 本地登录（Local login） |
| SQL Server实例 | 固定服务器角色（Fixed server role） |
| 登录（Login） |
| 数据库 | 固定数据库角色（Fixed database role） |
| 用户（User） |
| 应用程序角色（Application role） |
| 组（Group） |

对表7-1中的操作系统层，在SQL Server环境中并没有安全对象可进行设置，而对于其他层次，均可进行安全特性的设置，这些安全对象的层次结构如表7-2所示。

表7-2 可设置安全特性的对象层次

|  |  |
| --- | --- |
| 对象层次 | 可设置安全特性的对象 |
| SQL Server实例 | 登录（Login） |
| 端点（Endpoint） |
| 数据库 |
| 数据库 | 应用角色（Application role） |
| 集合（Assembly） |
| 非对称密钥（Asymmetric key） |
| 证书（Certificate） |
| 协议（Contract） |
| 全文目录（Full-Text Catalog） |
| 消息类型（Message Type） |
| 远程服务绑定（Remote service binding） |
| 角色（Role） |
| 路由（Route） |
| 架构（Schema） |
| 服务（Service） |
| 对称密钥（Symmetric key） |
| 用户（User） |
| 架构（Schema） | 函数 |
| 表 |
| 存储过程 |
| 视图 |
| 队列 |
| 类型 |
| 同义字 |
| XML架构集 |

表7-2中的对象权限，可以通过Transact-SQL的GRANT、DENY或REVOKE语句来进行设置。

**(2) 新增安全机制**

SQL Server 2008对安全提供了许多新机制，大体来说有如下几点：

① 登录

登录是SQL Server实例层的安全模型。登录SQL Server的账号一直就有两种，Windows和SQL Server自己建立的账号，而Windows的登录账号的密码可以通过系统安全性原则来管理，SQL Server 2008可以要求SQL Server自身提供的登录账号也遵循Windows系统的密码安全性原则。

② 用户

用户是数据库层的安全模型。

③ 用户和架构的分离

用户和架构定义相分离是指：数据库的对象，例如：数据表、视图表、存储过程等，属于某个架构，而用户、角色（Role）、Application Role等都可以赋予访问架构的权限。即每一个架构属于一个用户，用户是该架构对象的拥有者。SQL Server 2008引入架构层次，使得当需要改变对象的拥有者时，不需要去更改应用程序编码，只需要改变架构的拥有者就可以。

④ 目录安全性

不同权限查看不同的元数据（Metadata）。即元数据只对那些对表有权限的用户才可见，这有助于隐藏那些来自用户的未被审核的信息。因此，在SQL Server 2008中用户不可以直接访问系统数据表，而必须要通过系统视图（View）、系统存储过程或系统函数来查看Metadata，而不同权限的用户查看Metadata时，看到的结果不同。SQL Server 2008为此引入了专门针对目录安全性的新的权限View Definition。

⑤ 模块化执行上下文

模块化执行上下文（Module Execution Context）是对SQL Server中拥有权限的补充。在定义存储过程或用户自定义函数时，可使用新的WITH EXECUTE AS语法指定该存储过程，或函数执行时不以调用者的身份执行而是模拟成另外一个账号，以解决Broken Ownership Chains的问题，或是临时转换身份来提升权限，而不必真正且永久赋予某个账号某些权限。

⑥ 粒度化的权限控制

在SQL Server 2008中权限的赋予比其在先前的版本中更加细化。它支持下列粒度化权限控制级别：

* 服务器：在服务器级别，权限能够赋予登录；
* 数据库：在数据库级别，权限可以赋予用户、数据库角色或应用角色；
* 架构：在数据库中的每个架构都有它们自己的相关权限；
* 对象：在架构内的对象都有它们自己相关的权限。

⑦ 口令策略的增强

是指若SQL Server 2008安装在Windows 2003上，则可应用Windows 2003口令安全策略于其上。

2．SQL Server 2008的登录和用户

前已指明，登录是SQL Server实例层对象，若要连接到某个数据库实例，必须要有能够连接到该数据库实例的登录账号。而用户则是数据库层的对象，当通过登录连接到某个数据库实例时，须为数据库指明合适的用户，用于操作数据库对象。所以，一个合法的登录账号仅能表明该账号通过了Windows或SQL Server的认证，并不表示该账号可以对数据库对象进行某种或某些操作。一个登录账号总是和一个或多个数据库用户相对应。

**(1) 创建登录**

创建登录的Transact-SQL语法如下：

CREATE LOGIN login\_name{ WITH<option\_list1>|FROM<sources>}

<sources>::=

WINDOWS[ WITH<windows\_options>[，…]]

|CERTIFICATE cert\_name

|ASYMMETRIC KEY asym\_key\_name

<option\_list1>::=

PASSWORD=’password’ [HASHED][MUST\_CHANGE][,<option\_list2>[,..]]

<option\_list2>::=

SID=sid

|DEFAULT DATABASE = database

|DEFAULT LANGUAGE = language

|CHECK\_EXPIRATION = {ON|OFF}

|CHECK\_POLICY = {ON|OFF}

[CREDENTIAL = credential\_name]

<windows\_options>::=

DEFAULT\_DATABASE = database

|DEFAULT\_LANGUAGE = language

参数说明：

* login\_name：指定创建的登录名称。可有四种类型的登录名称：SQL Server登录名、Windows登录名、证书映射登录名和非对称密钥映射登录名。
* WINDOWS：将指定登录名称映射到Windows登录名。
* CERTIFICATE cert\_name：指定将与cert\_name登录名称关联的证书名称。该证书必须已存在于master数据库中。
* ASYMMETRIC KEY asym\_key\_name：指定将与asym\_key\_name登录名称关联的非对称密钥名称。同样，该密钥必须已存在于master数据库中。
* PASSWORD = 'password'：只适用于SQL Server登录名称。其用于指明正在创建的登录名称的密码，此值提供时可能已经过Hash运算。
* HASHED：只适用于SQL Server登录名，指定在PASSWORD参数后输入的密码已经过Hash运算。
* MUST\_CHANGE：只适用于SQL Server登录名。若有此项，其指定SQL Server在首次使用新登录名时会提示用户输入新密码。
* SID=sid：只适用于SQL Server登录名称，其指定新SQL Server登录名的GUID。
* DEFAULT DATABASE=database：指派给新登录名的默认数据库，若未包含此项，则SQL Server将默认设置为master数据库。
* CHECK\_EXPIRATION={ON | OFF}：只适用于SQL Server登录名，指定是否对此登录名强制实施密码过期策略，默认值为OFF。
* CHECK\_POLICY={ON | OFF}：只适用于SQL Server登录名，指定对此登录名强制实施Windows密码策略，默认值为ON。
* CREDENTIAL= credential\_name：指定映射到新SQL Server登录名的凭据名称，该凭据必须已存在于服务器中。

**示例** 创建登录JOHN，其口令为J12345。

创建登录所使用的Transact-SQL命令为：

CREATE LOGIN JOHN WITH PASSWORD='J12345'

该命令语句执行的结果是在当前SQL Server实例的MASTER数据库中创建了登录名称为JOHN的账号，其登录密码是J12345。

**(2) 创建用户**

创建用户所使用的语法为：

CREATE USER User\_name[{{FOR|FROM}

{ LOGIN login\_name

| CERTIFICATE cert\_name

| ASYMMETRIC KEY asym\_key\_name

}

| WITHOUT LOGIN

}

[WITH DEFAULT\_SCHEMA=schema\_name]

参数说明：

* User\_name：指定此数据库中所创建的该用户名称。
* LOGIN login\_name：指定所要创建数据库用户的SQL Server登录名，该登录名必是服务器中已存在的有效的登录名。通过该语句将数据库级的用户名与实例层的登录名建立关联。
* CERTIFICATE cert\_name：指定要创建数据库用户的证书。
* ASYMMETRIC KEY asym\_key\_name：指定要创建的数据库用户的非对称密钥。
* WITH DEFAULT\_SCHEMA=schema\_name：指定服务器为该数据库用户解析对象名称时将搜索的第一个架构。
* WITHOUT LOGIN：不将用户映射到现有的登录名。

**示例** 为数据库TEACHING\_MIS创建用户JOHN01，所使用的SQL语句为：

USE TEACHING\_MIS

GO

CREATE USER JOHN01

**(3) 用户与架构的分离**

在ANSI SQL99就已经制定了架构（Schema）的规范，用以区分各数据库内的对象，如数据表、视图表、存储过程、自定义函数等，所以这些数据对象的两节名称(two-part name)形式为SchemaName.ObjectName。在以前的版本中，SQL Server将对象的创建者自动变成该对象的拥有者，其两节名称是OwnerName.ObjectName，这会在该对象拥有者离职或不再负责某项系统时引起一些麻烦。例如User01建立的数据表在User02接手管理后，为了避免转换拥有者名称造成前端应用程序访问有困难，或是减少转换的工作，可能User02在登录数据库时干脆都以User01的身份来访问，而不在数据库另外创建与使用User02账号。这样，造成在某数据库内有一大堆数据表，却分不清用途与用户。SQI. Server 2005中用户和架构已经分离，也就是说，可以创建新的架构以管理这些不同的程序以及专用的某些数据表。如此，使得在SQL Server 2008中维护用户更为方便，同时也可避免因为要改变账号的拥有者所引出的一些困难。

所谓SQL Server 2008中提供架构的定义与用户分开，是指当在数据库级别创建用户时，可以指定该账号默认的架构(Default Schema)，如果没有指定，则账号的默认架构为主dbo，这样的设计是为了向前兼容。由于对象归属在架构之下，通过赋予账号是否有权限访问架构将可以避免上述的问题。也就是说用户账号和架构分离具有以下的作用：

① 通过设置架构的拥有者是角色(Role)或windows的组(Group)，可以让多个账号同时有权访问该架构。让用户自建对象时，有较大的弹性。而赋予权限时可以在架构级别，也可以在对象自身。

② 当删除用户账号时，不必更改该帐号所建对象的名称与拥有者。前端应用程序也不必改写与重新测试。

当数据库对象设置在某个架构之下以后，用户账号也会赋予默认架构，以让它在以单一对象名称访问时，仍能够找到该用户一般常访问的对象，这样做是为了与SQL Server 2000的使用习惯兼容。因此SQL Server 2008提供一个默认的架构dbo。当对象和账号没有明确设置架构时，都属于dbo架构。

在SQL Server 2008中创建架构的语法如下：

CREATE SCHEMA schema\_name\_clause[ <schema\_element>[, ...]]

schema\_name\_clause::

{

schema\_name

| AUTHORIZATION owner\_name

| schema\_name AUTHORIZATION owner\_name

}

schema\_element::

{

table\_definition | view\_definition | grant\_statement

| revoke\_statement | deny\_statement

}

参数说明：

* schema\_name：在数据库内标识架构的名称。
* AUTHORIZATION owner\_name：指定将拥有该架构的数据库级主体(拥有者)名称。该主体还可以拥有其他架构，且可以不使用当前架构作为其默认架构。.
* table\_definition：在指定架构内创建表的语句，执行该语句的主体必对当前数据库具有CREATE TABLE的权限。
* view\_definition：在指定架构内创建视图的语句，执行该语句的主体必对当前数据库具有CREATE VIEW的权限。
* grant\_statement：指定可对除新架构外的任何安全对象授予权限的GRANT语句。
* revoke\_statement：指定可对除新架构外的任何安全对象撤销权限的REVOKE语句。
* deny\_statement：指定可对除新架构外的任何安全对象拒绝授予权限的DENY语句。

另外要说明的是：要创建架构，用户必须对数据库具有CREATE SCHEMA的权限。

**示例** 在数据库TEACHING\_MIS中创建由USER01拥有的包含表TEST01的名为SCOURES的架构。

为体验用户和架构分离所带来的优越性，首先创建2个登录LOGIN1，LOGIN2和2个属于TEACHING\_MIS数据库的用户USER01与USER02，并让它们分别与上面所创建的登录相关联，其所使用的SQL语句为：

CREATE LOGIN LOGIN1 WITH PASSWORD='LOG123'

CREATE LOGIN LOGIN2 WITH PASSWORD='LOG456'

USE TEACHING\_MIS

GO

CREATE USER USER01 FOR LOGIN LOGIN1;

USE TEACHING\_MIS

GO

CREATE USER USER02 FOR LOGIN LOGIN2;

然后为用户USER01指定其架构，同时创建属于该架构的数据表TEST01，所使用的SQL语句如下所示：

USE TEACHING\_MIS

GO

CREATE SCHEMA SCOURES AUTHORIZATION USER01

CREATE TABLE TEST01(SCID CHAR(8)，SCNAME NVARCHAR(30)，SP CHAR(4))

GO

注意此时是以sa登录创建的，因此该架构也属于sa登录用户，当使用以下SQL语句来改变登录用户为USER01，再查询所创建的表TEST01：

SETUSER 'USER01'

SELECT \* FROM SCOURES.TEST01

由于USER01是SCOURES架构的拥有者，所以其具有操作该架构的权限，因此，可以通过SELECT语句正确查询到TEST01表中的数据。

最后，使用以下SQL语句将登录用户改变为USER02，再来查询TEST01表：

SETUSER 'USER02'

SELECT \* FROM SCOURES.TEST01

执行上述代码，出现错误而不能完成指定查询，该输出窗口显示内容如下：

消息15157，级别16，状态1，第1行

由于以下原因之一，setuser失败：数据库主体'USER02'不存在，与数据库主体对应的服务器主体没有服务器访问权限，无法模拟此类型的数据库主体或您没有权限。

其原因是由于该用户并无对SCOURES架构的拥有权限，当然不能访问该架构下的数据表。

本示例说明，在SQL Server 2008中，虽同为一个数据库的用户，但由于USER02没有SCOURES的所有权，所以不能操作该架构下的表TEST01，只有当USER02赋予权限后，才可操作该表，在这里，权限不再和用户相关，用户和架构已经分离。

当用户访问对象并没有以两节名称明确指定架构时，SQL Server 2008会先找与用户默认架构相同的架构下的对象，若找不到该对象，再去dbo架构下查找是否有相同的对象。

**四、实验内容**

1．理解登录、用户、架构、角色、权限的概念，在Enterprise Manager里浏览SQL Server 2008中存在的登录名、用户、架构、角色和权限，形成一个总结。

2．利用Query Analyzer完成以下操作：

⑴ 建立采油一矿的作业项目的视图，把该视图的查询权限授予给采油一矿的用户user11，以user11的身份查询该视图，观察执行情况；再以其他用户的身份查询该视图，观察执行情况。

⑵ 创建一个用户user12，以user12的身份执行实验五中所定义的存储过程，观察记录是否成功执行；然后把该存储过程的执行权限授予给user12，再次以user12的身份执行该存储过程，观察记录是否成功执行。

⑶ 定义触发器，实现只能在工作时间内更新“作业项目表”的数据，然后通过选择不同的时间进行适当的更新操作来验证。

**五、实验报告**

## 一、实验目的

本次实验旨在通过实际操作，加深对数据库视图、用户权限管理、存储过程和触发器的理解和应用能力。通过建立视图、管理用户权限、创建存储过程以及定义触发器，掌握数据库的基本操作和安全管理技术。

## 二、实验内容与步骤

### 1. 视图创建与权限管理

**创建采油一矿的作业项目视图**：

编写 SQL 语句，创建一个名为 采油一矿作业项目视图 的视图，该视图仅包含采油一矿相关的作业项目。

**创建用户 user11**：

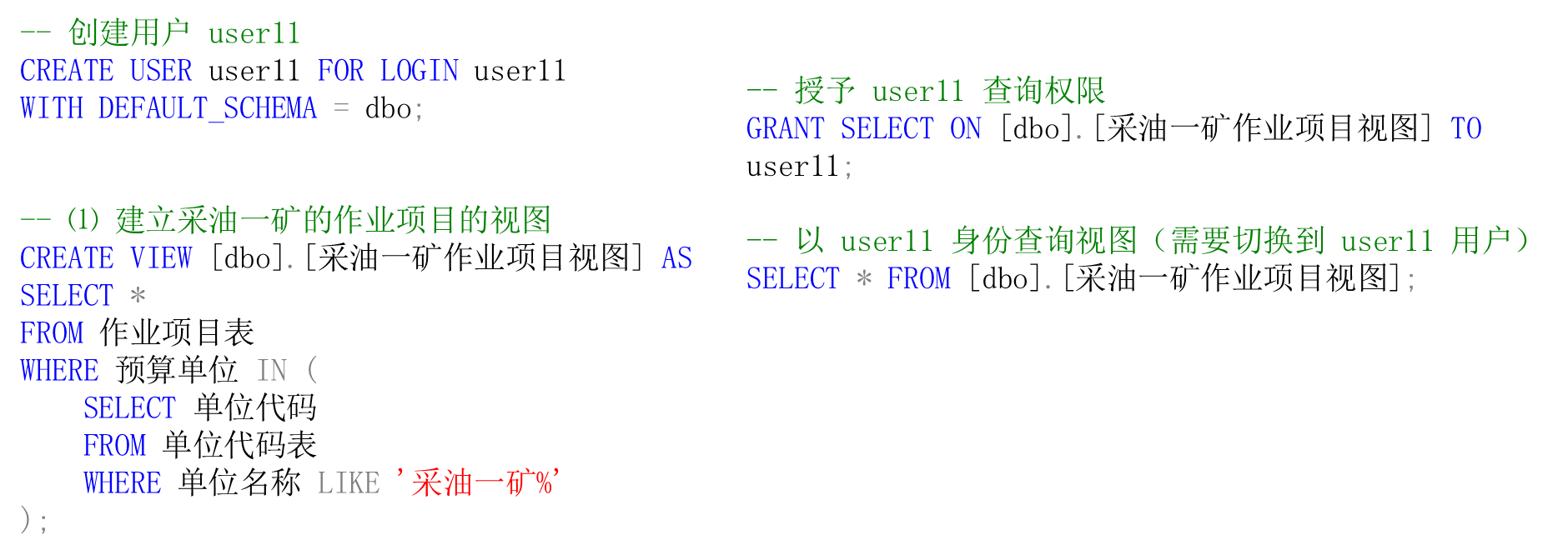
在 SQL Server 中创建一个名为 user11 的新用户。

**授予查询权限**：

授予 user11 对 采油一矿作业项目视图 的查询权限。

**以 user11 身份查询视图**：

切换到 user11 用户身份，执行查询语句，验证是否能够成功查询视图。

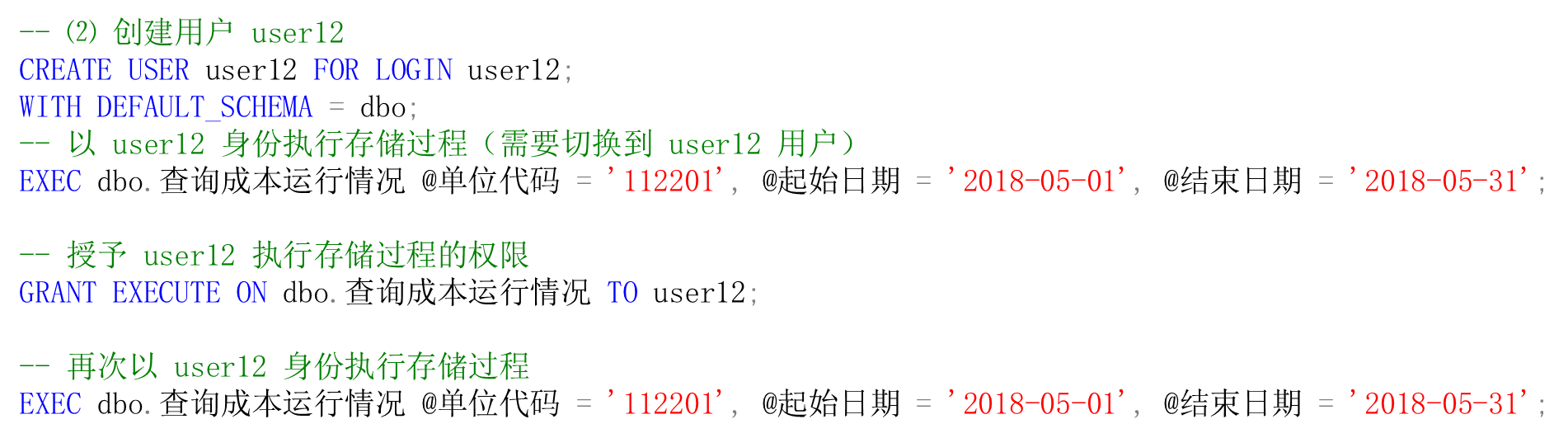




### 2. 用户创建与存储过程权限测试

**创建用户 user12**：

在 SQL Server 中创建一个名为 user12 的新用户。



### 3. 触发器定义与验证

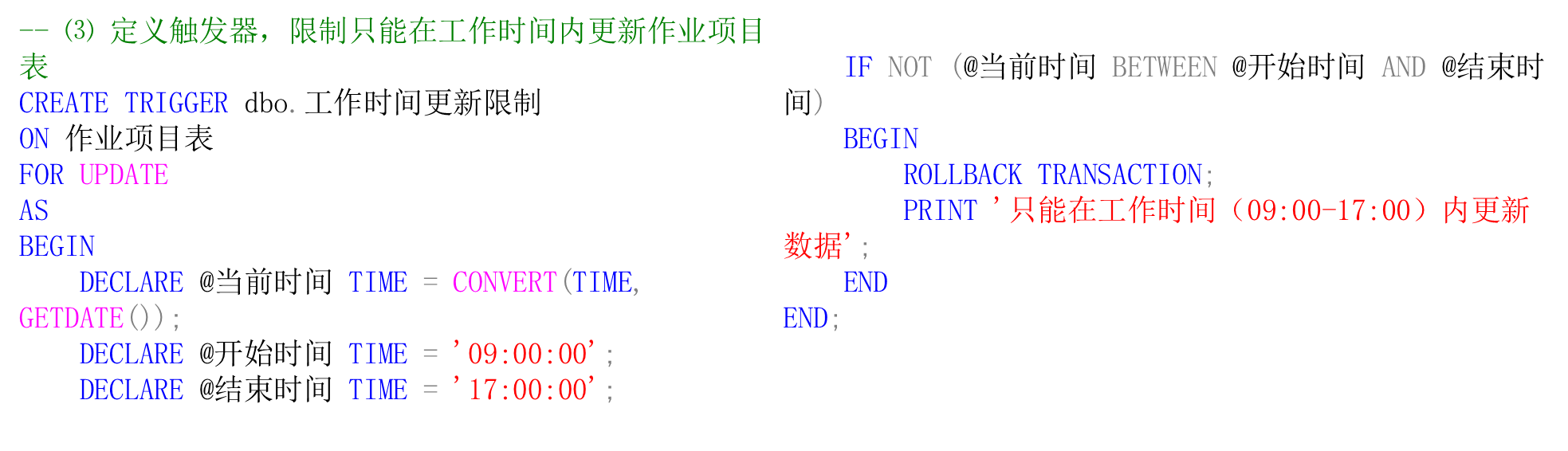
**定义触发器**：

编写 SQL 语句，创建一个触发器，限制只能在工作时间（09:00-17:00）内更新 作业项目表 的数据。

**验证触发器**：

在工作时间外尝试更新 作业项目表 的数据，观察更新是否成功。

在工作时间内更新 作业项目表 的数据，观察更新是否成功。



## 四、实验总结

通过本次实验，我们成功实现了以下目标：

**视图创建与权限管理**：

创建了采油一矿的作业项目视图，并实现了对特定用户的权限控制。通过授予 user11 查询权限，确保了只有授权用户能够查看视图中的数据。

**存储过程权限管理**：

验证了存储过程的执行权限管理。通过创建用户 user12 并测试存储过程的执行权限，确保了只有授权用户能够执行特定操作。

**触发器应用**：

通过触发器实现了对 作业项目表 更新时间的限制，确保数据操作符合业务规则。验证了触发器在工作时间外限制更新和在工作时间内允许更新的功能。

**实验心得：**

**视图的作用**：视图可以用于封装复杂的查询逻辑，简化用户的查询操作。同时，视图可以用于实现数据的逻辑抽象和安全控制，通过授予用户对视图的权限，限制其对底层表的直接访问，从而保护数据的安全性和完整性。

**用户权限管理的重要性**：合理的用户权限管理是数据库安全的重要保障。通过为不同用户授予不同的权限，可以确保用户只能执行其职责范围内的操作，防止数据泄露和非法篡改。

**存储过程的优势**：存储过程可以封装复杂的业务逻辑，提高代码的复用性和执行效率。通过授予权限，可以控制哪些用户能够执行特定的存储过程，进一步加强了数据库的安全性。

**触发器的应用场景**：触发器可以用于实现自动的数据验证和业务规则强制。在本实验中，触发器成功限制了对 作业项目表 的更新时间，确保数据操作符合业务要求。然而，触发器的过度使用可能会增加系统复杂性，应在设计时谨慎考虑。

综上所述，本次实验加深了对数据库视图、用户权限管理、存储过程和触发器的理解和应用能力，为后续的数据库开发和管理工作奠定了坚实的基础。