第 5 章

MySQL 数据保护

5.1 数 据 保 护

作为实用化最成功的软件系统之一，数据库系统如今已经成为全球化经济基础设施的重要基础部件，对商务管理、事务管理、数据分析、电子商务、慕课等来说，都是必不可少的，人类社会对数据的依赖达到了前所未有的地步，数据安全关系到社会的每个组织单位及个人。数据安全建立在数据保密性、数据完整性、数据可用性之上，数据库系统的特点之一就是由数据库管理系统提供统一的机制保护数据的保密性、完整性、可用性。

数据保密性是指对数据资源的隐藏，通常在数据库中保护保密性就是指仅允许经授权地读数据，需要对数据值进行保密和数据存在性进行保密。数据保密性的需求源于数据的敏感性，例如，军事部门试图实现“需要知道”原则，企业公司对专利设计数据的保护以免竞争对手获取设计成果等。数据存在性有时候比数据值本身更需要保护，比如确认美国政府曾监听某国政要远比监听到的数据本身更为重要。

数据完整性指的是数据的可信度，保护数据完整性通常是指防止非法或者未经授权的数据修改。数据完整性包括数据值的完整性和数据来源的完整性。完整性遭到破坏即产生无效或损坏的数据，将导致用户由于错误的或无效的数据做出错误的决策。

数据可用性是指对数据的期望使用能力，保护数据可用性通常指减少数据库系统停工时间，保持数据持续可访问。可用性遭到破坏将影响用户正常获取数据，意味着用户不能访问数据库或操作困难。

数据保护允许说明数据资源使用的安全策略并提供机制予以支持。策略和机制不同，策略决定做什么，即是对允许什么、禁止什么的规定；机制决定怎么做，即是实施策略的功能、方法、工具、规程。

本书专注于数据库系统层次上的安全保护，本章介绍 MySQL 中提供的与安全保护相关的语句，软件机制将在后续相应章节介绍。描述系统访问策略的最简单模型是访问矩阵，矩阵的行代表用户（角色），列代表数据库对象，矩阵中的元素表示相应用户对相应数据库对象的访问权限。访问控制矩阵通常依据用户在系统应用中担任的角色确定。MySQL 中的授权和收权语句可以赋予或撤销用户相应的访问权限，数据库管理系统确保只有获得授权的用户有资格访问数据库对象，令所有未被授权的人员无法接近数据，保护数据保密性和完整性。视图作为外模式的实现，支持“见之所需”原则，不仅简化了用户操作，而且结合访问控制可以保护视图数据保密性和完整性，更重要的是提供了一种保护数据存在性的手段。MySQL 提供加密函数，以允许对存储数据进行加密处理，支持保护

数据保密性和完整性验证。数据库系统都允许对数据库的并发访问以改进系统响应性能，提高可用性，但是如果对数据库访问的并发执行不加控制将可能破坏数据完整性。另外，数据库系统总会面临这样那样的故障，故障也将可能破坏数据完整性。数据库管理系统通过将数据库操作分组为事务，以事务为单位实施并发控制和故障恢复，MySQL 允许显式或隐式的事务边界定义。数据库管理系统中的并发控制机制维护并发访问情况下的数据完整性；数据库管理系统中的故障恢复机制不仅维护故障情况下的数据完整性，并且由于故障恢复机制对故障的有效处理，它也是保护数据可用性的重要手段。

上述方法都是从数据值以外的因素考虑数据保护。理想情况下，数据值发生变化时，比如进行插入或更新时，系统能够判断数据库中的各个数据项值是否与其对应的现实世界状态一致，即数据是否真实正确。然而，这个目标是无法实现的。退而求其次，可以在系统中定义一些正确数据应该满足的约束，系统自动检查数据库中的数据是否满足这些约束条件，并且只允许满足这些约束条件的数据进入数据库。也就是说，软件系统无法保证数据的真实正确性，但可以保证数据符合可明确定义的约束。这种约束通常称为完整性约束。

总的来说，数据安全是数据库技术广泛使用的前提条件之一，也是数据库管理系统的重要目标和优势之一。利用数据保护技术，保护数据安全是以数据库为中心的应用系统必不可少的重要方面。当前的大数据管理技术，主要是利用多副本存储技术维护数据可用性，关于专门针对大数据的保密性和完整性保护越来越受关注并有待进一步探索。

5.2　视　　图

视图主要有如下作用：视图能够简化用户的操作；视图使用户能够从多种角度看待同一数据；视图对重构数据库提供了一定程度的逻辑独立性；视图能够对数据存在性方面的保密性提供安全保护；适当地利用视图可以更清晰地表达查询。

5.2.1　视图的创建和撤销

在 MySQL 中，外模式一级数据结构的基本单位是视图，视图是从若干基本表和(或)其他视图构造出来的“虚表”，采用 SELECT 语句实现。在创建一个视图时，只是把其视图的定义存放在数据字典中，而不存储视图对应的数据，在用户使用视图时才去查询对应的数据。因此，视图被称为“虚表”。基表中的数据发生变化时，从视图中查询出的数据也随之改变。

视图可以把基表结构细节封装起来，表可以随应用进化而变化，但视图以及基于视图的应用程序不受表变化的影响。

用户只能查询或修改通过视图所能见到的数据，看不见数据库中的其他数据；视图可以帮助将授权限制到特定的行或列上。

1. 视图的创建

创建视图可以用“CREATE VIEW”语句实现，其语法如下。

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
```

```
VIEW <视图名>[(<列名> [,<列名>]…)]
AS <子查询>
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

组成视图的属性列名可以全部省略或全部指定，视图属性名省略时，取 SELECT 结果关系属性名。

【例 1】 对于组卷系统数据库中的基表 eeexam，用户经常要用到考生考试号 eeid 和平均成绩 avgachieve 这两列的数据，那么可用下列语句建立视图。

```
CREATE VIEW avgachieve(eeid,average) AS
    SELECT eeid,AVG(achieve)
    FROM eeexam
    GROUP BY eeid;
```

此语句创建了视图，视图名为 avgachieve，有两个属性（eeid 和 average），就好像有一个表 avgachieve 有两个属性（eeid，average）。average 属性的值就是对应 eeid 考生的平均成绩。

也可以使用视图创建视图，例如，可以在 examinee，eeexam，exampaper 三个表的基础上创建视图 eeexamv1，语句是：

```
CREATE VIEW eeexamv1 AS
SELECT examinee.eeid, examinee.eedepa, exampaper.ename, eeexam.achieve
FROM examinee, eeexam,exampaper
WHERE examinee.eeid=eeexam.eeid
AND eeexam.eeid=exampaper.eid;
```

这个语句创建视图 eeexamv1，有 4 个属性：eeid，eedepa，ename，achieve，也就是考生考号、考生院系名、试卷名和成绩。

然后，基于视图 eeexamv1，可以进一步创建视图 eeexamv2。语句是：

```
CREATE VIEW eeexamv2 AS
          SELECT eeid,ename,achieve
          FROM eeexamv1
          WHERE eedepa='历史学院';
```

这个视图 eeexamv2 有三个属性 eeid，ename，achieve，也就是历史学院的考生考号、试卷名和成绩。需要注意的是，虽然可以使用视图定义视图，但是视图定义不能递归。

2. 视图的撤销

在不需要视图时，可以用“DROP VIEW”语句把其从系统中撤销，其语法如下。

```
DROP  VIEW [IF EXISTS] <视图名>[RESTRICT|CASCADE]
```

该语句从数据字典中删除指定的视图定义，如果该视图上还导出了其他视图，使用 CASCADE 级联删除语句，把该视图和由它导出的所有视图一起删除。删除基表时，由该基表导出的所有视图定义都必须显式地使用 DROP VIEW 语句删除。

【例2】 撤销avgachieve视图。

```
DROP  VIEW IF EXISTS avgachieve;
```

5.2.2 对视图的操作

1. 对视图查询

创建视图后,就可以在SQL语句中使用,视图名能够出现在表名可以出现的任何地方。例如,已经定义了视图avgachieve(eeid, average),要查询平均75分以上的人数就可以这样查询:

```
SELECT count(*)
FROM avgachieve
WHERE avgachieve>=75;
```

同样,也可以使用视图定义视图,首先使用examinee表中的eeid、eedepa属性,exampaper表中的ename属性和eeexam表中的achieve属性定义视图eeexamv1,定义eeexamv1的语句如下。

```
CREARE VIEW eeexamv1 AS
    SELECT examinee.eeid, examinee.eedepa, exampaper.ename, eeexam.achieve
    FROM examinee,eeexam,exampaper
    WHERE examinee.eeid=eeexam.eeid  AND  eeexam.eid=exampaper.eid;
```

再使用视图eeexamv1定义由eeid、ename、achieve属性组成的视图eeexamv2,使用如下语句实现。

```
CREARE VIEW eeexamv2 AS
    SELECT eeid,ename,achieve
    FROM eeexamv1
    WHERE eedepa='历史学院';
```

MySQL执行CREARE VIEW语句时只是把视图定义存入数据字典,并不执行其中的SELECT语句;在对视图查询时,按视图的定义从基表中将数据查出。

虽然视图和表都是关系,都可在查询中直接使用,但是视图与表有一点重要不同:数据库中存储表的模式定义和数据,而只存储视图的定义,不存储视图的数据,视图中的数据是在使用视图时按照视图定义中的查询临时计算的。视图定义中引用的表称为基表。当基表中的数据发生变化时,相应的视图数据也随之改变。

视图可以把基表结构细节封装起来,表可以随应用进化而变化,但视图以及基于视图的应用程序可以尽可能少地受表变化的影响,这也就是数据的逻辑独立性。

通过视图,用户只能查询或修改视图中见得到的数据,看不见数据库中的其他数据,这也就提供了数据存在性的保护。

2. 视图上的修改

由于视图是基于对基表的查询来定义的,系统只存储视图的定义,不存视图的数据,

对视图的修改通常是转换为对基表的相应操作来执行，这就要求对视图的修改慎而又慎。

由于视图并不像基表那样实际存在，有些视图的修改不能唯一的有意义地变换成对相应基表的修改，这些视图是不可修改的。

MySQL 只允许对可修改视图进行修改。可修改视图需要满足如下条件：视图是从单个关系只使用投影、选择操作导出；SELECT 子句中只包含属性名，不包含其他表达式、聚集、DISTINCT 声明，查询中不含有 GROUP BY 或 HAVING 子句；WHERE 子句查询不出现基表名（其前面 FROM 子句给出的表名）；并且对视图的修改操作符合一般修改语句的规则，如插入时主键不能为空，需要插入操作的视图的投影列需包含基础关系的键。

【例 3】 如果定义了一个有关男考生的视图：

```
CREATE VIEW eemale     AS
    SELECT  eeid,eename,eeage
    FROM examinee
    WHERE  eesex='男';
```

视图 eemale 是考生表 examinee 中的男生考号、姓名和年龄的投影。由于这个视图是从单个关系只使用选择和投影导出的，满足可修改视图需要满足的条件，是可修改的，允许用户对视图进行插入、删除和更新操作。例如，由于包含主键 eeid，可以执行插入操作：

```
INSERT INTO eemale
VALUES('271610012938','王涛',24);
```

系统会将对视图 eemale 的插入操作传递到基表 examinee 上执行以下语句：

```
INSERT INTO examinee(eeid,eename,eeage)
VALUES('271610012938','王涛', 24);
```

也等价于：

```
INSERT INTO examinee
VALUES('271610012938','王涛', null,24,null);
```

如果在视图定义的末尾包含 WITH CHECK OPTION，数据库管理系统自动检查对视图的修改应满足视图定义中 WHERE 的条件，这种情况下此插入操作便会被拒绝执行。

5.3 访 问 控 制

MySQL 提供以下访问控制功能。

（1）使用 GRANT 和 REVOKE 语句实现授权/收权存入数据库管理系统的数据字典。

（2）在角色提出操作请求时，按照授权状态进行检查，从而决定是否执行操作请求。

MySQL 权限管理机制允许对整个关系或一个关系的指定属性授予或收回权限。通常不允许对特定元组的授权。

5.3.1 用户与角色管理

MySQL新增了角色(role)的概念,使账号权限的管理更加灵活方便。所谓角色,就是一些权限的集合。可以把该集合授权给某个或者某一批用户,在需要给这些用户增加或者减少权限的时候,只需要修改角色的权限即可,不用对每个用户依次进行修改。

MySQL的访问控制,声明给定用户/角色拥有在给定数据库对象上的给定操作权限。主要包括用户/角色、数据库对象和操作权限三方面。

通常所有者就是执行创建语句的用户/角色。对于大多数类型的对象,只有“所有者”或者“超级用户”用户/角色可以对该对象做任何事情。其他用户初始状态是只具有usage权限(登录数据库),要允许其他用户/角色使用这个对象,必须赋予相应的权限。

MySQL用户管理命令与角色管理命令类似,比如:

创建用户lili:CREATE USER lili;

创建具有口令的用户yanni:CREATE USER yanni IDENTIFIED BY '654321';

给用户lili重命名为ninglili:RENAME USER lini TO ninglili;

删除用户yanni:DROP USER yanni;

创建角色cuichen,lichen,chenchen:CREATE ROLE cuichen,lichen,chenchen;

创建一个具有所有权限的角色:GRANT ALL ON exam.* TO 'chen_developer';

创建可使用SELECT的角色:GRANT SELECT ON exam.* TO 'chen _read';

创建可使用INSERT,UPDATE,DELETE的角色:GRANT INSERT,UPDATE,DELETE ON exam.* TO 'chen _write';

分配给用户所需的角色:GRANT 'chen_read','chen_write' TO ninglili。

启用角色,设置了角色,如果不启用,用户登录的时候,依旧没有该角色的权限:set default role '角色名' to '用户名';

如果一个用户有多个角色,使用前应:set default role all to '用户名';

验证分配给账户的权限:SHOW GRANTS FOR 'lili'@'localhost'。

通过GRANT语句来授予用户不同的权限,ON:表示这些权限对哪些数据库和表生效(格式:数据库名.表名);TO:将权限授予哪个用户(格式:'用户名'@'登录IP或域名',其中%表示没有限制,在任何主机都可以登录。比如:'lili'@'192.168.0.%','表示lili这个用户只能在192.168.0IP段登录)。上述show grant for语句可以查看MySQL中其他用户权限。

一旦角色已经存在了,那么就可以用GRANT和REVOKE命令给用户授予和撤销角色,比如:

```
GRANT <角色名>on exam to <用户名>;
REVOKE <角色名>on exam from <用户名>
```

用户和角色之间的区别在于可用于管理用户和角色的特权:

(1) 在CREATE ROLE语句完创建了一个ROLE,还需要通过GRANT赋予其相应的权限,在DROP ROLE之后该角色的权限就被取消了。

(2) 在CREATE USER语句完成后,默认具有CREATE USER,DROP USER,

RENAME USER 和 REVOKE ALL PRIVILEGES 的权限。

因此，CREATE ROLE 和 DROP ROLE 特权不如 CREATE USER 强大，CREATE USER 可以授予仅应被允许创建和删除角色，而不执行更一般的账户操作的用户。

可以将用户账户视为角色，然后将该账户授予其他用户或角色。结果是将账户的特权和角色授予其他用户或角色，但是不能建立循环的成员关系。比如：

```
CREATE USER 'user1';
CREATE ROLE 'role1';
GRANT SELECT ON exam1.* TO ' user1';
GRANT SELECT ON exam2.* TO ' role1';
CREATE USER ' user2';
CREATE ROLE ' role2';
GRANT ' user1', ' role1' TO ' user2';
GRANT ' user1', ' role1' TO ' role2';
```

执行这些语句后，user2 和 role2 被授予与用户（user1）和角色（role1）相同的权限。

5.3.2 授予权限

MySQL 提供 GRANT 语句来给角色授予数据库操作权限，GRANT 语句的一般格式为：

```
GRANT<权限>[,<权限>]…[ON<对象类型><对象名>]TO<角色>[,<角色>][WITH GRANT OPTION];
```

这个语句可以赋予给定角色对给定对象的给定操作权限。当此 GRANT 语句包含 WITH GRANT OPTION 选项时，在此语句获得权限的角色可以将所获得权限授予其他角色。

对于不同的操作对象，有不同的操作权限。常见的操作权限如表 5-1 所示。

表 5-1 常见的操作权限

对　　象	对 象 类 型	可以授予的权限
属性列、视图	TABLE	SELECT、INSERT、UPDATE、DELETE、ALL PRIVILEGES
基表	TABLE	SELECT、INSERT、UPDATE、DELETE，ALTER、INDEX、ALL PRIVILEGES
数据库	DATABASE	CREATE TABLE

接受权限的角色可以是一个或者多个角色。

如果指定了 WITH GRANT OPTION 子句，则获得某种权限的角色可以把这种权限再授予其他角色，否则该角色只能使用所获得的权限，而不能将该权限传播给其他角色。

在以下的例子中，假设在上述考试系统数据库中已创建了 uZhang、uChang、uWang、uYing、uLong 和 uLiang 等角色。

【例 4】 把查询组卷表 erexam 权限授给角色 uZhang。

```
GRANT   SELECT
```

```
ON TABLE   erexam
TO uZhang;
```

【例5】 把对试卷表 exampaper 和考官表 erexam 的全部权限授予角色 uChang 和 uWang。

```
GRANT ALL PRIVILEGES
ON TABLE erexam
TO uChang, uWang;
GRANT ALL PRIVILEGES
    ON TABLE exampaper
TO uChang, uWang;
```

【例6】 把对院系表 department 的查询权限授予所有用户。

```
CREATE ROLE if not exists 'SELECT_department';
GRANT SELECT ON TABLE department to 'SELECT_department';
```

【例7】 把查询 department 表和插入院系联系电话的权限授给角色 uYing。

```
GRANT INSERT(dtele), SELECT
ON TABLE department
TO uYing;
```

授予关于属性列权限时,应当指出相应的属性列名。

【例8】 把更新 department 表的权限授予 uLong 角色,并允许他再将此权限授予其他角色。

```
GRANT UPDATE
ON TABLE department
TO uLong
WITH GRANT OPTION;
```

uLong 不仅拥有了更新 department 表的权限,还可以传播此权限,即 uLong 角色使用上述 GRANT 命令给其他角色授权。

【例9】 uLong 可以将此权限授予 uLiang。

```
GRANT UPDATE
ON TABLE department
TO uLiang
WITH GRANT OPTION;
```

【例10】 uLiang 还可以将此权限授予 uKuang。

```
GRANT UPDATE
ON TABLE department
TO uKuang;
```

但是由于 uLiang 未给 uKuang 传播的权限,因此 uKuang 不能再传播此权限。

5.3.3 收回权限

MySQL 提供 REVOKE 语句来收回角色的数据库操作权限。收回权限的 REVOKE 语句的一般格式为：

```
REVOKE<权限>[,<权限>] [ON<对象类型><对象名>] FROM<角色>[,<角色>];
```

【例 11】 把角色 uYing 插入院系联系电话的权限收回。

```
REVOKE INSERT(dtele)
ON TABLE department
FROM uYing;
```

【例 12】 把角色 uLong 更新 department 表的权限收回。

```
REVOKE UPDATE
ON TABLE department
FROM uLong CASCADE;
```

将角色 uLong 更新 department 表的权限收回的时候必须级联(CASCADE)收回，即系统将收回直接或间接从 uLong 处获得的权限。

MySQL 中的授权和收权语句可以赋予或撤销角色相应的访问权限，数据库管理系统确保只有获得授权的角色有资格访问数据库对象，从而保护数据保密性和完整性。

5.4 完整性约束

5.4.1 约束含义

假如 DBMS 足够智能，理想情况下，当数据库中任意数据项值发生变化时，比如执行插入、删除或更新操作时，DBMS 能够判断出数据项的新值是否与其对应的现实世界状态一致，即数据是否真实正确。当然，这个目标是无法实现的。能做的是，在系统中定义一些正确数据应该满足的约束条件，系统自动检查数据是否满足这些约束条件，并且只允许满足这些约束条件的数据进入数据库。换句话说，DBMS 无法保证数据始终与其对应的现实世界状态一致，但可以保证数据始终与系统中明确定义的约束一致。这种约束通常称为完整性约束。典型的完整性约束包括主键约束(也称为实体完整性)、外键约束(也称为引用完整性)、值非空、值唯一等。典型的完整性约束包括主键约束、外键约束、值非空、值唯一等，约束的一般形式就是一个任意谓词。

1. 主键

主键约束意味着各元组主键值不能重复，且不能为空。将一个表的一个或几个属性定义为主键后，插入或对主键列进行修改操作时，系统自动检查主键的各个属性是否为空，只要有一个为空就拒绝插入或修改；并且检查主键值是否唯一，如果不唯一则拒绝插入或修改。由于实际上删除操作不会导致违背主键约束，只有插入或对主键列进行修改时才可能发生违背主键约束，因此，只有对关系进行插入或修改时系统才检验主键约束。

2. 外键

外键约束意味着各元组外键值必须来自于引用表或为空。违背外键约束的那些元组通常称为悬浮元组。将一个表的一个或几个属性声明为外键后,引用表插入或对外键列进行修改操作造成违背外键约束时系统拒绝相应操作;被引用表删除或修改时,系统也会自动检查是否违背外键约束,如果违背外键约束,有以下5种处理策略。

(1) 拒绝(NO ACTION)执行,当且仅当在引用关系中构造出一个或多个悬浮元组时,对于被引用关系的删除和修改操作将予以禁止。生成一个错误,表明删除或更新将产生一个违反外键约束的动作。如果约束被声明为DEFERABLE,则约束检查将推迟到当前事务完成时执行。这是默认动作。

(2) 级联(CASCADE)执行,删除任何引用了被删除行的行,或者分别把引用行的属性值更新为被参考属性的新数值。

(3) 设置为空值(SET NULL),把每个悬浮元组中的外键值都设置为NULL,需要悬浮元组的外键列没有声明限定词NOT NULL。

(4) 限制(RESTRICT)操作,生成一个表明删除或更新将导致违反外键约束的错误。和NO ACTION一样,只是约束被声明为DEFERABLE。

(5) 设置为默认值,把引用属性设置为它们的默认值。

3. 非空

非空约束意味着每个元组对应列值不能为空。将一个或几个属性声明为非空后,系统会阻止相应属性值为空的数据输入或更新。

4. 唯一值

唯一值约束意味着相应的属性(或属性组)为候选键。将一个表的一个或几个属性声明为唯一值约束(候选键)后,系统自动检查是否违背唯一值约束。与主键约束不同,除非显式定义为非空,候选键的属性可为空值。

5. CHECK约束

CHECK(P)子句指定一个谓词P,关系中的每一个元组都必须满足谓词P。CHECK约束可以通过在CREATE TABLE或者ALTER TABLE语句中,根据用户的实际完整性要求来定义。它分别对列或表实施CHECK约束,使用语法为:CHECK(expr)。

其中,expr是一个SQL表达式,用于指定需要检查的限定条件。在更新表数据时,MySQL会检查更新后的数据行是否满足CHECK约束中的限定条件。该限定条件可以是简单的表达式,也可以是复杂的表达式(如子查询)。

5.4.2 声明及检验

在MySQL中,可以对完整性约束进行添加、修改和删除等操作。其中,为了删除和修改完整性约束,需要在定义约束的同时对其进行命名。命名完整性约束的方式是在各种完整性约束的定义说明之前加上CONSTRAINT子句。

1. 主键声明

主键用CONSTRAINT ＜约束名＞ PRIMARY KEY来声明。