第5章 数据库管理与维护

数据库管理系统是一种操纵和管理数据库的系统软件，用于建立、使用、管理和维护数据库。一旦数据库创建以后，DBMS 便对数据库中的数据进行统一的管理和控制，以确保用户在共享环境中能合法访问数据，防止数据出错、意外丢失，以及数据库遭到破坏后能迅速恢复正常。因此 DBMS 必须提供数据库管理与维护功能，以保证数据库中数据的正确有效和安全可靠。数据库管理与维护包括数据库的安全性管理、数据库中数据的并发控制和数据库的备份及恢复管理。

5.1 数据库安全性管理

数据库安全性管理是对数据库采取的一种保护措施。安全性管理是指保护数据库，防止非法使用，以避免非法用户对其进行窃取数据、篡改数据、删除数据和破坏数据库结构等操作。

5.1.1 数据库安全性概述


5 数据库安全性管理之安全性概述

安全性问题是计算机系统中普遍存在的一个问题。由于大量数据集中存放在数据库系统中，并为最终用户直接共享，使得数据库系统中的安全问题更加突出。数据库的安全性管理措施是否有效是数据库系统的主要技术指标之一。

威胁数据库安全的因素主要包括以下三方面。

(1) 自然灾害。如火灾、地震、雷击、海啸等。自然灾害轻则造成数据混乱，重则造成数据库系统不可用甚至数据损坏。

(2) 人为破坏。人为破坏包括两种情况：无意的人为破坏和有意的人为破坏。无意的人为破坏是指人为的无意失误和各种误操作造成的安全隐患，如操作人员误删除数据、用户口令设置不当、操作人员把自己的账号口令给他人使用等，这些会给数据库安全带来威胁。有意的人为破坏是指恶意破坏数据库中的数据，如恶意破坏数据的完整性、通过某种手段破坏数据的保密性造成数据泄露等。

(3) 安全环境的脆弱性。数据库的安全环境包括计算机硬件系统、操作系统、网络系统等。计算机硬件系统的故障、操作系统安全的脆弱、网络协议缺陷等都会造成数据库安全性的破坏。如计算机硬件系统中电路短路、接触不良等引起系统不稳定，操作系统中的漏洞，网络监听、伪装信任等均会威胁到数据库的安全。

为了保护数据库，防止恶意破坏，可以从以下两方面提供安全保护措施。

(1) 法律及行政手段。国家颁布有关安全的法律法规来规范和制约人们的思想和行

为，从法律的角度确保数据的安全性，如保密法、计算机犯罪法、计算机安全法等。安全管理部门根据安全管理原则、安全管理需求等建立相应的管理措施，从行政手段的角度确保数据的安全性，如建立多人负责制、中心机房出入管理制度，制定操作规程等。

（2）技术手段。通过数据加密技术、密钥管理技术、访问控制技术、防火墙技术等确保网络和通信安全。通过身份鉴别、安全审计、入侵防范、访问控制等确保计算机硬件系统安全。通过访问控制、隔离控制、存储保护、身份鉴别等确保系统软件安全。通过数据加密技术、访问控制技术、数据备份技术、数字签名技术等确保数据安全。

实现数据库的安全性管理需要考虑多方面的问题，这些问题涉及多项安全措施，本书仅讨论数据库系统本身的安全措施。

5.1.2 数据库安全性控制

在一般的计算机系统中，安全措施是一级一级层层设置的，如图 5-1 所示的计算机系统安全模型。用户在进入计算机系统时，系统先根据用户输入的用户标识进行用户身份鉴别，只有合法的用户才能进入计算机系统；对已进入计算机系统的用户，数据库管理系统还要进行存取控制，只允许用户在自己的权限范围内执行操作；数据库管理系统运行在操作系统之上，操作系统通过自己的安全保护措施，确保数据库中的数据必须由数据库管理系统进行访问，而不允许用户越过数据库管理系统直接操作或访问；数据最后通过加密的方式存储在数据库中。操作系统的安全保护措施可参考操作系统相关书籍，这里仅介绍与数据库相关的安全措施，包括用户身份鉴别、存取控制、视图机制、数据加密、审计等。

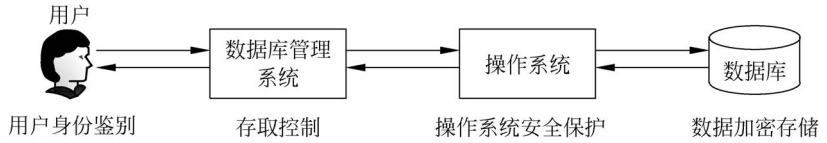


图 5-1 计算机系统的安全模型

1. 用户身份鉴别

用户身份鉴别是数据库管理系统提供的最外层安全保护措施。实现用户身份鉴别包括两项工作：一是用户的标识，即用什么标识用户；二是用户的确认，即怎样识别用户。系统可以采用用户的个人特征，如声音、指纹、虹膜等，用户的特有东西，如磁卡、钥匙等，用户自己设置的内容，如口令或密码等来标识用户，系统内部记录这些用户标识。用户确认是由系统按一定的方式核对用户提供的标识，经过鉴别后才提供使用数据库管理系统的权限。

用户身份鉴别的方法有很多，在使用过程中往往是多种方法的结合，以获得更强的安全性。最常用的用户身份鉴别方法是通过用户账号和口令来鉴别用户身份的合法性，这种方法只需要通过软件来进行用户账号及口令的登记、维护与验证即可，不需要专门的硬件设备，所以简单易行，但容易被别人窃取或破解，安全性较低，因此可以使用更复杂的方法，如口令动态生成，每次登录系统时，使用新口令进行身份鉴别，这种方法增加了口令被窃取或破解的难度，安全性相对高一些。还可以使用个人所具有的生物特征来进行鉴别，如声音、指纹、虹膜等，这种方式需要专门的硬件设备来采集、存储这些特征，再利用图像处理和识别算法等实现鉴别；使用用户持有的磁卡、钥匙等物件进行鉴别，这种方式需要用户随身携带物件，有专门的阅读装置，登录数据库管理系统时，能通过阅读装置读取物件中的身份信息

进行身份验证。

2. 存取控制

存取控制是确保具有授权资格的用户访问数据库,同时使所有未被授权的人员无法访问数据库的机制。存取控制机制主要包括定义并记录用户权限和合法权限检查两部分。数据库管理系统必须提供适当的语言来定义用户权限,这些定义经过编译后记录在数据字典中,作为安全规则或授权规则;当用户发出存取数据库的操作请求后,数据库管理系统查找数据字典,根据安全规则进行合法权限检查,如果用户的操作超出了定义的权限,系统将拒绝执行此操作。

数据库管理系统所采取的存取控制机制主要包括两种:自主存取控制(Discretionary Access Control,DAC)和强制存取控制(Mandatory Access Control,MAC)。在自主存取控制方法中,用户对于不同的数据库对象有不同的存取权限,不同的用户对同一数据库对象也有不同的权限,而且用户还可以将其拥有的权限转授给其他用户,自主存取控制非常灵活。在强制存取控制方法中,每一个数据库对象都被标以一定的密级,每一个用户也都被授予某一个级别的许可证;对于任意一个数据库对象,只有具有合法许可证的用户才能存取;强制存取控制相对比较严格,因此适用于那些对数据有严格而固定密级分类的部分,如军事部门或政府部门,在通用数据库系统中使用较少。下面主要介绍自主存取控制。

大型数据库管理系统都支持自主存取控制,现在的 SQL 标准也对自主存取控制提供了支持,主要是通过 SQL 中的 GRANT 语句和 REVOKE 语句来实现。

用户权限包括数据库对象和操作类型两个要素。定义一个用户的存取权限就是要定义这个用户可以在哪些数据库对象上进行哪些类型的操作。在数据库系统中,定义存取权限就称为授权;取消存取权限就是收回。自主存取控制方式就是通过授权和收回来实现的,包括数据库对象权限的授权和收回、角色的授权和收回。

(1) 数据库对象的授权与收回。

数据库对象权限是指不同的数据库对象,可提供给用户不同的操作,通常由数据库管理员(DBA)或该对象的拥有者(owner)或已经拥有该权限的用户授予。授权语句的语法格式如下:

```
GRANT
ALL PRIVILEGES | PERMISSION [, … ] [(column [, … n])]
ON database_object
TO security_account [, … n]
[WITH GRANT OPTION]
```

说明:ALL [PRIVILEGES]表示所有可授予的权限。PERMISSION 表示在数据库对象上可执行的具体权限,可以是 SELECT、INSERT、UPDATE、DELETE 等。column 表示在表或视图上允许用户将权限局限到某些列上,column 表示列的名字。database_object 表示数据库对象,可以是数据表、视图等。TO 用于指定被授予者,security_account 表示被授予权限的用户。WITH GRANT OPTION 表示被授权者可以把获得的权限转授予其他用户,未使用 WITH GRANT OPTION,则获得权限的用户只能使用该权限,不能转授予其他用户。

收回语句的语法格式如下:

```
REVOKE
ALL PRIVILEGES | PERMISSION [, … ] [(column [, … n])]
ON database_object
FROM security_account [, … n]
[CASCADE|RESTRICT]
```

说明：REVOKE 语法格式与 GRANT 语法格式基本一样，只是将 TO 改成 FROM。各参数的含义相同。CASCADE 选项表示级联操作，即收回某用户的权限时，会把该用户转授予其他用户的权限一并收回。RESTRICT 表示限制操作，即收回某用户的权限时，如果该用户把权限转授予了其他用户，则系统将拒绝执行该收回语句。

注意，有的数据库管理系统默认为 CASCADE，有的则默认为 RESTRICT。

【例 5-1】 授予数据库用户 dbuser2 查询和修改 Goods 表的权限，权限可转授予其他用户。

```
GRANT SELECT, UPDATE
ON Goods
TO dbuser2
WITH GRANT OPTION
```

【例 5-2】 授予数据库用户 dbuser2 修改 Student 表姓名列和专业列的权限。

```
GRANT UPDATE(SName, Major)
ON Student
TO dbuser2
```

【例 5-3】 授予数据库用户 dbuser1、dbuser2 操作 Student 表的所有权限。

```
GRANT ALL PRIVILEGES
ON Student
TO dbuser1, dbuser2
```

【例 5-4】 收回数据库用户 dbuser2 对 Goods 表的查询和修改权限。

```
REVOKE SELECT, UPDATE
ON Goods
FROM dbuser2
```

(2) 角色的授权与收回。

角色是被命名的一组与数据库操作相关的权限，是权限的集合。因此可以为一组具有相同权限的用户创建一个角色，使用角色来管理数据库权限可以简化授权的过程。

在 SQL 中，可以使用 CREATE ROLE 语句创建角色，语句格式为 CREATE ROLE role_name，role_name 为创建的角色名，创建的角色是空的，没有任何内容。然后使用 GRANT 语句给角色授权，使用 REVOKE 语句收回授予角色的权限，语句格式与数据库对象的授权和收回一样，只需要把用户 security_account 改成角色名 role_name。

使用 GRANT 语句把一个或多个角色授予用户，使用 REVOKE 语句把角色从用户那里收回。其语法格式如下：

```
GRANT role_name[, … n]
TO security_account
[WITH ADMIN OPTION]
```

说明：指定 WITH ADMIN OPTION，用户可以把此角色转授予其他用户。

```
REVOKE role_name[, … n]
FROM security_account
```

3. 视图机制

视图可以作为一种安全机制，通过为不同的用户定义不同的视图，把数据对象限制在一定的范围内，即通过创建视图，把要保密的数据对无权存取的用户隐藏起来，从而自动地给数据提供一定程度的安全保护。如果某一用户需要访问视图的结果，则必须授予该用户访问权限，而且用户只能看到该结果集，数据库中其他信息对该用户都是不可见的。

假设校园超市的学生信息由各个专业进行维护。其中，MIS 专业的陈芳老师可以对该专业的所有学生信息进行增、删、查、改，王凤老师只能查看学生信息。先建立 MIS 专业学生视图，再在视图上为两位老师授权相应的权限。

（1）创建视图 mis_student。

```
CREATE VIEW mis_student
AS
SELECT * FROM Student WHERE Major = 'MIS';
```

（2）授权。

```
GRANT ALL PRIVILEGES
ON mis_student
TO 陈芳;
GRANT SELECT
ON mis_student
TO 王凤;
```

通过授权操作，陈芳老师具有增、删、查、改 MIS 专业学生信息的所有权限，而王凤老师只能查看 MIS 专业学生的信息。

4. 数据加密

数据加密是防止数据库中的数据在存储或传输过程中失密的有效手段。加密的基本思想是根据一定算法将原始数据（明文）变换成不可直接识别的格式（密文），从而使得不知道解密算法的人无法获知数据的内容。

现如今数据加密技术已经比较成熟，有关密钥加密、密钥管理的内容这里不再讨论，可参考有关书籍。数据库加密使用已有的加密技术和算法对数据库中存储的数据和传输的数据进行保护，加密后的数据的安全性得到进一步提高。但是数据加密和解密是比较费事的操作，而且数据加密和解密程序会占用大量的系统资源，增加了系统的开销，降低了数据库的性能，另外数据库加密会增加查询处理的复杂性，查询效率也会受到影响。因此，数据加密功能可作为数据库系统的可选功能，允许用户自由选择，只针对那些对保密性要求特别高的数据（如财务数据、军事数据、国家机密等）进行数据加密。

5. 审计

用户身份鉴别、存取控制、视图机制、数据加密等安全措施不可能是完美无缺的，对于蓄意盗窃、破坏数据的人来说，他们总会想方设法打破这些控制。审计功能把用户对数据库的所有操作自动记录下来存放在审计日志中。数据库管理员可以利用审计日志信息，重现导致数据库现有状况的一系列事件，找出非法存取数据的人、时间和内容；也可以通过对审计日志信息分析，对潜在的威胁提前采取措施加以防范。

审计通常是很费时间和空间的,所以数据库管理系统往往将其作为可选功能,允许数据库管理员或数据库拥有者根据应用对安全性的要求进行选择。审计功能一般应用于安全性要求较高的部门。

5.1.3 MySQL 安全性管理

MySQL 数据库管理系统提供了完善的安全管理机制和操作手段,以防止非法用户对数据库进行操作,保证 MySQL 服务器的安全访问。MySQL 的安全访问控制系统可以为不同的用户指定不同的权限,只有拥有相应权限的用户才可以访问数据库中的相应对象,执行相应合法操作。MySQL 的安全访问控制分为身份认证和权限验证两个阶段。当用户试图连接 MySQL 服务器时,MySQL 会将用户提供的信息与存储用户权限信息的 user 表中三个字段(Host、User、authentication_string)进行匹配以实现身份认证,只有用户提供的主机名、用户名和密码与 user 表中对应字段值完全匹配才能成功连接到 MySQL 服务器。成功连接后,MySQL 服务器进入权限验证阶段,针对该连接上的每个操作请求,验证它们要执行的操作以及是否具有足够的权限来执行这些操作;权限验证会用到 user、db、host、tables_priv、columns_priv 等权限表。

MySQL 成功安装后,默认情况下,MySQL 会自动创建 root(超级管理员)用户,管理 MySQL 服务器的全部资源,拥有所有的操作权限。同时会自动创建默认的数据库,这些数据库中存储着与 MySQL 相关的配置信息和一些基本数据,information_schema 数据库中保存着关于 MySQL 服务器所维护的所有其他数据库的信息,这些信息被统称为元数据;performance_schema 数据库主要用于收集数据库服务器性能参数;sys 数据库所有数据来自 performance_schema 数据库,主要是为了降低 performance_schema 数据库的复杂度,数据库管理员能更好地阅读库中内容了解数据库的运行情况;MySQL 数据库是系统的核心数据库,主要负责存储数据库的用户信息、权限设置信息、关键字等 MySQL 需要使用的控制和管理信息。这里主要介绍 MySQL 中的安全访问控制。

5 MySQL 安全性管理之权限表

1. MySQL 权限表

MySQL 服务器通过权限来控制用户对数据库的访问,权限信息存放在名为 mysql 的默认数据库中。当 MySQL 服务启动时,首先会读取 mysql 中的权限表,并将表中的数据装入内存,当用户进行存取操作时,MySQL 会根据这些表中的数据做相应的权限控制。常用的权限表有 user、db、host、tables_priv、columns_priv 和 procs_priv。user 表用于决定连接是否允许,对于允许的连接,user 表授予的权限是全局权限,适用于服务器上的所有数据库;db 和 host 表决定用户能从哪个主机存取哪个数据库进行哪个操作,授予的是数据库级别的权限;tables_priv 表授予表级别的权限,适用于表和它的所有列;columns_priv 表授予列界别的权限,只适用于专用列;procs_priv 表授予程序级别的权限,只适用于单个程序。

1) user 表

user 表是 MySQL 中最重要的一个权限表,记录了允许连接到服务器的账户信息、全局权限及其他非权限列表,是针对所有用户数据库所有表的。user 表中的字段大致分为四类,分别是用户字段、权限字段、安全字段和资源控制字段。通常用得较多的是用户字段和权限字段。当用户进行服务器连接时,先从 user 表中的 host、user 和 authentication_string 三个字段中判断连接的主机、用户名和密码是否存在于表中,如果存在,则通过身份认证,否

则拒绝连接；如果通过身份认证，按照 user、db、tables_priv、columns_priv 权限的顺序得到数据库操作权限。

(1) 用户字段。

Host、User 和 authentication_string 字段属于用户字段，存储了用户连接 MySQL 数据库服务器时需要输入的信息。用户连接时，需要这三个字段同时匹配才允许通过。创建、修改或删除用户信息，也是对 user 表的用户字段进行设置。可以通过 SELECT 语句查询用户字段信息，如图 5-2 所示。

```
1  SELECT Host, User, authentication_string FROM user
```

信息 | 结果 1 | 剖析 | 状态

Host	User	authentication_string
localhost	mysql.infoschema	A005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED
localhost	mysql.session	A005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED
localhost	mysql.sys	A005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED
localhost	root	A005$C□□Z-n□0L,Pm□4QDJm.2sOMYXriH/v.fOuIgxaLOT0oTkhRnDM6uGmMsDB32w6

图 5-2　user 表用户字段信息

(2) 权限字段。

user 表中以 priv 结尾的字段为权限字段，它们决定了用户的权限，用来描述在全局范围内允许对数据和数据库进行的操作。这些权限既包括操作数据库的普通权限，如查询权限、修改权限等，又包括对数据库进行管理的高级权限，如关闭服务器权限、超级权限、加载用户等。权限字段的数据类型为 ENUM，可取的值只有 Y 和 N，其中，Y 表示该用户有对应的权限，N 表示该用户没有对对应的权限。为了安全起见，这些字段的默认值都是 N。如果要修改权限，可以使用 GRANT 语句为用户授予一些权限，也可以通过 UPDATE 语句更新 user 表的方式来设置权限，还可以通过 SELECT 语句查询权限字段信息。图 5-3 展示了 user 表中部分权限字段的信息。

```
1  SELECT Host, User, Select_priv, Insert_priv,Update_priv FROM user
```

信息 | 结果 1 | 剖析 | 状态

Host	User	Select_priv	Insert_priv	Update_priv
localhost	mysql.infoschema	Y	N	N
localhost	mysql.session	N	N	N
localhost	mysql.sys	N	N	N
localhost	root	Y	Y	Y

图 5-3　user 表中部分权限字段的信息

(3) 安全字段。

安全字段主要用来判断用户是否能够连接成功。两个与 ssl 相关：ssl_type 和 ssl_cipher，主要用于加密。两个与 x509 相关：x509_issuer 和 x509_subject，主要用于表示用户。plugin 字段标识可用于认证用户身份的插件，如果该字段为空，则服务器使用内建授权验证机制认证用户身份。三个 password 作为前缀的字段(password_expired、password_last_changed、password_lifetime)主要用于记录密码相关的信息。account_locked 字段用于表示用户是否被锁定。

(4) 资源控制字段。

资源控制字段用来限制用户使用的资源。max_questions 字段规定每小时允许执行查询的操作次数；max_updates 字段规定每小时允许执行更新的操作次数；max_connections 字段规定每小时允许执行的连接操作次数；max_user_connections 字段规定允许同时建立的连接次数。四个字段默认值为 0，表示没有限制。一个小时内用户查询或连接数量超过资源控制限制，用户将被锁定，直到下一个小时才可以再次执行对应的操作。

2) db 表和 host 表

db 表是数据库级别的权限表，存储了用户对某个数据库的操作权限，决定用户能从哪个主机存取哪个数据库；host 表也是数据库级别的权限表，存储了某个主机对数据库的操作权限，和 db 表一起对给定主机上数据库级操作权限做更细致的控制。

db 表和 host 表结构相似，大致分为用户字段和权限字段两类。db 表的用户字段有三个：Host、Db 和 User，分别表示主机名、数据库名和用户名。host 表的用户字段有两个：Host 和 Db，分别表示主机名和数据库名。如果 db 表中找不到 Host 字段的值，就需要到 host 表中去寻找。通常情况下，db 表的设置已经可以满足权限控制的要求，所以 host 表很少用到。db 表和 host 表的权限字段大致相同，这些字段的取值是 Y 或 N。

user 表中的权限是针对所有数据库的。当希望用户只针对某个数据库有操作权限，则需要将 user 表中对应的权限字段值设置为 N，然后在 db 表中设置对应数据库的操作权限。

3) tables_priv 表

tables_priv 是表级别的权限表，可以对单个表进行权限设置。这里指定的权限适用于一个表的所有列。tables_priv 表有八个字段：Host、Db、User、Table_name、Grantor、Timestamp、Table_priv 和 Column_priv。Host、Db、User、Table_name 分别表示主机名、数据库名、用户名和表名；Grantor 表示修改该记录的用户；Timestamp 表示修改该记录的时间；Table_priv 表示对表进行操作的权限，这些权限包括 SELECT、INSERT、UPDATE、DELETE、CREATE、DROP、ALTER 等；Column_priv 表示对表中的列进行操作的权限，这些权限包括 SELECT、INSERT、UPDATE 等。

4) columns_priv 表

columns_priv 是列级权限表，可以对表中的某一列进行权限设置。columns_priv 表有七个字段：Host、Db、User、Table_name、Column_name、Timestamp 和 Column_priv。Column_name 用于指定设置操作权限的列名，其余的字段名与 tables_priv 含义相同。

5) procs_priv 表

procs_priv 是存储过程和函数权限表，可以对存储过程和函数进行权限设置。procs_priv 表有八个字段：Host、Db、User、Routine_name、Routine_type、Grantor、Proc_priv 和 Timestamp。Routine_name 表示存储过程或函数的名称；Routine_type 表示存储过程或函数的类型，有两个取值：procedure 和 function；Proc_priv 表示拥有的权限，包括 execute、alter routine、grant 三种。

5 MySQL 安全性管理之用户管理

在 MySQL 的权限验证中，先判断 user 表中的值是否为 Y，如果是，则不需要验证后面的表；如果 user 表中的值为 N，则依次检查 db 表、tables_priv 表和 columns_priv 表。

2. 用户管理

MySQL 用户分为超级管理员和普通用户。root 是默认的超级管理员，拥有所有权限。

普通用户通常由超级管理员创建，只拥有创建时被赋予的权限。MySQL 的用户信息存放在默认数据库 mysql 的 user 表中。因此，在 MySQL 中对用户管理时，既可以使用 MySQL 特定的用户管理语句，又可以使用操作表的 SQL 语句。需要注意，不管使用哪种方式进行用户管理，都必须有使用这些语句的权限，以及对 mysql 数据库和 user 表操作的权限。MySQL 用户管理主要包括创建用户、修改用户和删除用户。

(1) 创建用户。

root 用户有对整个 MySQL 服务器完全控制的权限。在日常管理和实际操作中，为了避免恶意用户冒名使用 root 账号操作数据库，通常需要创建一系列具备适当权限的用户，尽可能不用或少用 root 登录系统，以此确保数据的安全访问。MySQL 提供了 CREATE USER 语句创建用户，也可以使用图形化管理工具，如 Navicat 创建用户。这里仅介绍语句创建用户。

使用 CREATE USER 语句可以创建一个或多个 MySQL 用户，并设置相应的密码，其语法格式如下：

```
CREATE USER user_name [IDENTIFIED BY 'password']
[, user_name [IDENTIFIED BY 'password']] …;
```

说明：user_name 指定创建的用户账号，格式为'user_name'@'host_name'，其中，user_name 是用户名，host_name 是主机名，即用户连接 MySQL 时所在主机的名字，如果未指定，则主机名会默认为%，表示一组主机。IDENTIFIED BY 用于设置用户账号对应的密码，如果不设置密码，则可省略此子句。password 为用户账号的密码，必须用单引号括起来。

CREATE USER 语句创建用户后，会在 mysql 数据库的 user 表中添加一条新记录。新创建的用户拥有的权限很少，只能执行不需要权限的操作，如登录 MySQL，使用 SHOW 语句查询等。如果两个用户的用户名相同，但主机名不同，MySQL 会将它们看作两个用户，并允许为它们分配不同的权限集合。

【例 5-5】 使用 CREATE USER 语句创建两个新用户，主机名为 localhost，user1 的密码是 pwd1，user2 的密码是 pwd2。

```
CREATE USER
'user1'@'localhost' IDENTIFIED BY 'pwd1', 'user2'@'localhost' IDENTIFIED BY 'pwd2';
```

(2) 修改用户。

系统中存在的用户可以通过 RENAME USER 语句修改用户名，通过 SET PASSWORD 语句修改用户的密码。

修改用户名的语法格式如下：

```
RENAME USER 'old_user_name'@'host_name' TO 'new_user_name'@'host_name'
    [,'old_user_name'@'host_name' TO 'new_user_name'@'host_name'] …;
```

说明：old_user_name 为系统中已经存在的用户名。new_user_name 为新用户名。

修改用户密码的语法格式如下：

```
SET PASSWORD FOR 'user_name'@'host_name' = 'new_password';
```

【例 5-6】 将前面例子中的 user1 用户改名为 new_user1。

```
RENAME USER 'user1'@'localhost' TO 'new_user1'@'localhost';
```

【例 5-7】 将前面例子中的 user2 的密码改为 new_pwd2。

```
SET PASSWORD FOR 'user2'@'localhost' = 'new_pwd2';
```

(3) 删除用户。

对于存在系统中不需要的用户可以通过 DROP USER 语句来删除。DROP USER 语句可以同时删除多个用户，其语法格式如下：

```
DROP USER 'user_name'@'host_name'[,'user_name'@'host_name']…;
```

【例 5-8】 删除前面例子中的 user2 用户。

```
DROP USER 'user2'@'localhost';
```

3. 权限管理

权限用于控制对数据库对象的访问以及指定用户对数据库可以执行的操作。权限管理主要是对登录到 MySQL 服务器的用户进行权限验证，以确保数据库系统的安全。所有用户的权限都存储在 MySQL 的权限表中。MySQL 的权限管理主要包括授予权限和收回权限。

5 MySQL安全性管理之权限管理

1) 授予权限

创建一个新用户后，该用户还没有访问权限，是无法正常操作数据库的，这就需要为该用户授予合适的权限。

MySQL 使用 GRANT 语句进行授权，其语法格式与前面讲的 SQL 标准的 GRANT 语句相近，具体格式如下：

```
GRANT
priv_name[(column [, … n])] [, priv_name [(column [, … n])]]…
ON [object_type] database_object
TO 'user_name'@'host_name'[,'user_name'@'host_name'] …
[WITH GRANT OPTION];
```

其中，object_type：

```
TABLE | PROCEDURE | FUNCTION
database_object:
 * | *.* | db_name. * | db_name.table_name | table_name | db_name.routine_name
```

说明如下。

priv_name：指定权限的名称，如 SELECT、INSERT、UPDATE 等操作。

column：为可选项，如果定义列级权限，则需要指定列名。

ON [object_type] database_object：用于指定权限授予的对象。object_type 为可选项，用于指定授权对象的类型，如表、存储过程和函数。database_object 为具体的对象名称，可以是数据库名、表名、视图名等。具体的名称和书写形式跟 GRANT 授权的级别有关。

TO 'user_name'@'host_name'：用于指定被授权的用户。

WITH GRANT OPTION：为可选项，用于实现权限的转移。

(1) 列级授权。

授予列级权限时，priv_name 的值只能是 SELECT、UPDATE、INSERT，权限后面需要加上列名。ON 子句为 ON [TABLE] db_name. table_name，db_name. table_name 表示指

定数据库中的表名或视图名。

【例 5-9】 授予系统中的用户 user1 在超市数据库 SuperMarket 的 Student 表上拥有 SELECT 和 UPDATE 学号列和姓名列的权限。

```
GRANT SELECT(Sno,Sname), UPDATE(Sno,Sname)
ON SuperMarket.Student
TO 'user1'@'localhost'
```

(2) 表级授权。

授予表级权限时,ON 子句为 ON [TABLE] table_name | db_name.table_name,table_name 表示当前数据库的表名或视图名,db_name.table_name 表示指定数据库中的表名或视图名。priv_name 后面不加列名,权限名称为表级名称,见表 5-1。

表 5-1 表级权限名称

权限名称	权限描述
SELECT	查询特定表的权限
INSERT	向特定表插入数据的权限
UPDATE	修改特定表的权限
DELETE	删除特定表记录的权限
CREATE	创建表的权限
DROP	删除表或视图的权限
ALTER	修改表的权限
REFERENCES	创建外键来参照特定表的权限
INDEX	在特定表上创建或删除索引的权限
CREATE VIEW	在特定表上创建视图的权限
ALL PRIVILEGES	所有权限

【例 5-10】 授予系统中的用户 user2 在超市数据库 SuperMarket 的 Student 表上拥有 UPDATE 和 DELETE 的权限,并允许其将这些权限授予其他用户。

```
GRANT UPDATE, DELETE
ON SuperMarket.Student
TO 'user2'@'localhost'
WITH GRANT OPTION;
```

(3) 存储过程级授权。

授予存储过程级权限时,ON 子句为 ON [PROCEDURE|FUNCTION] db_name.routine_name,db_name. routine_name 表示指定数据库中的存储过程名或函数名。priv_name 的取值可以是执行权限 EXECUTE、修改权限 ALTER ROUTINE。

【例 5-11】 授予系统中的用户 user1 在超市数据库 SuperMarket 中拥有执行存储过程 pro_inventory 的权限。

```
GRANT EXECUTE
ON SuperMarket. pro_inventory
TO 'user1'@'localhost';
```

【例 5-12】 授予系统中的用户 user2 在超市数据库 SuperMarket 中拥有执行和修改函数 fun_judgeprice 的权限。

```
GRANT EXECUTE, ALTER ROUTINE
```

```
ON SuperMarket. fun_judgeprice
TO 'user2'@'localhost';
```

(4) 数据库级授权。

授予数据库级权限时,ON 子句为 ON * | db_name. *,* 表示当前数据库的所有对象,db_name. * 表示指定数据库中的所有对象。priv_name 表示的权限名称为数据库级权限名称,见表 5-2。

表 5-2 数据库级权限名称

权限名称	权限描述
SELECT	查询特定数据库中所有表和视图的权限
INSERT	向特定数据库中所有表插入数据的权限
UPDATE	修改特定数据库中所有表的权限
DELETE	删除特定数据库中所有表记录的权限
CREATE	在特定数据库中创建表的权限
DROP	删除特定数据库中表或视图的权限
ALTER	修改特定数据库中所有表的权限
REFERENCES	在特定数据库中创建外键来参照表的权限
INDEX	在特定数据库中所有表上创建或删除索引的权限
CREATE VIEW	在特定数据库中创建视图的权限
CREATE ROUTINE	在特定数据库中创建存储过程和函数的权限
ALTER ROUTINE	在特定数据库中更新和删除存储过程与函数的权限
EXECUTE ROUTINE	执行特定数据库中存储过程和函数的权限
TRIGGER	在特定数据库中创建、删除触发器的权限
LOCK TABLES	锁定特定数据库中已有表的权限
ALL PRIVILEGES	所有权限

【例 5-13】 授予系统中的用户 user3 在超市数据库 SuperMarket 中拥有创建表、查询表和删除表的权限。

```
GRANT CREATE, SELECT, DROP
ON SuperMarket. *
TO 'user3'@'localhost';
```

(5) 服务器级授权。

授予服务器级权限时,ON 子句为 ON *. *,*. * 表示所有数据库中的所有对象。priv_name 的取值除了表 5-2 的数据库级权限以外,还包括对整个 MySQL 服务器的管理权限,见表 5-3。

表 5-3 服务器级权限名称

权限名称	权限描述
RELOAD	执行 FLUSH HOSTS、FLUSH LOGS、FLUSH PRIVILEGES 等刷新命令的权限
SHUTDOWN	停止服务器运行的权限
PROCESS	查看服务器上正在执行的线程和关闭线程的权限
FILE	执行 LOAD DATA INFILE、执行 file 的权限
SHOW DATABASES	执行 SHOW DATABASES 查看所有已有的数据库定义的权限

续表

权限名称	权限描述
SUPER	执行 CHANGE MASTER TO、KILL、PURGE BINARY LOGS、SET GLOBAL 以及 mysqladmin 的 DEBUG 等命令的权限
REPLICATIOIN SLAVE	拥有从服务器连接主服务器的权限
REPLICATION CLIENT	执行 SHOW MASTER STATUS、SHOW SLAVE STATUS 命令的权限
CREATE USER	创建用户、删除用户的权限
CREATE TABLESPACE	创建、修改以及删除表空间或日志文件组的权限

【例 5-14】 创建一个用户 server_user，授予服务器级的所有权限。

```
CREATE USER 'server_user'@'localhost' IDENTIFIED BY 'pwd';
GRANT ALL PRIVILEGES ON *.*
TO 'server_user'@'localhost';
```

2）收回权限

收回权限是取消已经授予用户的某些权限。收回用户不必要的权限在一定程度上可以保证数据的安全性。收回权限以后，用户权限信息将从系统表 db、tables_priv、columns_priv、procs_priv 中删除，但用户账号信息仍然保存在 user 表中。

REVOKE 语句收回所有权限的语法格式如下：

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM 'user_name'@'host_name'[,'user_name'@'host_name'] …;
```

REVOKE 语句收回指定权限的语法格式如下：

```
REVOKE
    priv_name[(column [, … n])] [, priv_name [(column [, … n])]]…
    ON [object_type] database_object
    FROM 'user_name'@'host_name'[,'user_name'@'host_name'] …;
```

说明：REVOKE 语句和 GRANT 语句的语法格式相似，各个部分的含义相同。

【例 5-15】 收回用户 user1 的所有权限。

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM 'user1'@'localhost';
```

【例 5-16】 收回用户 user2 在超市数据库 SuperMarket 的 Student 表上拥有的 UPDATE 权限。

```
REVOKE UPDATE
ON SuperMarket.Student
FROM 'user2'@'localhost'
```

3）查看用户权限

可以使用 SHOW GRANTS 语句查询用户的权限，也可以通过 SELECT 语句查询 mysql.user 表中的数据记录来查看用户的权限。SHOW GRANTS 语句查看权限的格式如下：

```
SHOW GRANTS FOR 'user_name'@'host_name';
```

说明：user_name 表示用户名，host_name 表示主机名或主机 IP。

【例 5-17】 查看用户 user2 的权限。

```
SHOW GRANTS FOR 'user2'@'localhost';
```

使用 SELECT 语句查询用户权限的语句如下：

```
SELECT * FROM mysql.user;
```

4. 角色管理

角色是一组相关权限的集合。MySQL 为用户授予角色时，用户就具备该角色的所有权限。MySQL 数据库管理员只对角色进行权限设置便可以实现对所有用户权限的设置，大大减少了管理员的工作量。

（1）创建角色。

MySQL 创建角色的语法格式如下：

```
CREATE ROLE 'role_name'@'host_name'[,'role_name'@'host_name'] …;
```

说明：role_name 为角色名，host_name 为主机名。创建的角色信息存放在系统数据库 mysql 的 user 表中。

5 MySQL 安全性管理之角色管理

【例 5-18】 在本地主机上创建只读角色 read_role、更新角色 write_role、管理员角色 admin_role。

```
CREATE ROLE 'read_role'@'localhost',
    'write_role'@'localhost', 'admin_role'@'localhost';
```

（2）授予角色权限。

授予角色权限的语法格式与授予用户权限相同，只需将 GRANT 语句中 TO 子句后面的用户名改为角色名即可。

【例 5-19】 分别授予 read_role 角色读取数据库 SuperMarket 所有表的权限、write_role 角色更新数据库 SuperMarket 所有表的权限、admin_role 角色访问数据库 SuperMarket 的所有权限。

```
#为 read_role 角色授权
GRANT SELECT
ON SuperMarket.*
TO 'read_role'@'localhost';
#为 write_role 角色授权
GRANT INSERT, UPDATE, DELETE
ON SuperMarket.*
TO 'write_role'@'localhost';
#为 admin_role 角色授权
GRANT ALL PRIVILEGES
ON SuperMarket.*
TO 'admin_role'@'localhost';
```

（3）授予用户角色。

使用 GRANT 语句授予用户角色，其语法格式如下：

```
GRANT 'role_name'@'host_name'[,'role_name'@'host_name',…]
TO 'user_name'@'host_name'[,'user_name'@'host_name',…];
```

【例 5-20】 分别将角色 read_role 授予新用户 reader，write_role 角色授予新用户 writer，admin_role 角色授予新用户 admin1、admin2。

```
#创建新用户
CREATE USER
    'reader'@'localhost' IDENTIFIED BY 'reader', 'writer'@'localhost' IDENTIFIED BY 'writer',
    'admin1'@'localhost' IDENTIFIED BY 'admin1', 'admin2'@'localhost' IDENTIFIED BY 'admin2';
#为用户分配角色
GRANT 'read_role'@'localhost' TO 'reader'@'localhost';
GRANT 'write_role'@'localhost' TO 'writer'@'localhost';
GRANT 'admin_role'@'localhost' TO 'admin1'@'localhost', 'admin2'@'localhost';
```

注意：用户在使用角色权限之前必须先激活角色，激活的语句如下。

```
SET GLOBAL activate_all_roles_on_login = ON;
```

(4) 收回用户角色。

使用 REVOKE 语句收回用户角色，其语法格式如下：

```
REVOKE 'role_name'@'host_name'[,'role_name'@'host_name',…]
FROM 'user_name'@'host_name'[,'user_name'@'host_name',…];
```

【例 5-21】 收回用户 admin1 的角色 admin_role。

```
REVOKE 'admin_role'@'localhost' FROM 'admin1'@'localhost';
```

(5) 删除角色。

删除角色的语法格式如下：

```
DROP ROLE 'role_name'@'host_name'[,'role_name'@'host_name'] …;
```

【例 5-22】 删除角色 read_role 和 admin_role。

```
DROP ROLE 'read_role'@'localhost', 'admin_role'@'localhost';
```

5.2 并发控制

数据库是一个多用户的共享数据集合，在多个用户同时执行某些操作时，由于操作间的互相干扰，有可能产生错误的结果。即使这些操作在单独执行时都是正确的，但是在并发执行时有可能存取不正确的数据，破坏数据的一致性。因此，数据库管理系统必须提供并发控制机制以保证数据的正确性。

5.2.1 事务概述

5 并发控制之事务概述

1. 事务的概念

事务由用户定义的一系列数据操作语句构成，这些操作语句要么全部执行要么全部不执行，是数据库运行的最小的、不可分割的工作单位。所有对数据库的操作都要以事务为一个整体单位来执行或撤销，同时事务也是保证数据一致性的基本手段。无论什么情况下，DBMS 都应该保证事务能正确、完整地执行。在关系数据库中，一个事务可以是一条 SQL 语句、一组 SQL 语句或整个程序。

2. 事务的特性

事务由有限的数据库操作序列组成，但不是任意的操作序列都能成为事务，它必须同时满足以下四个特性：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)和持续性

(Durability)。这四个特性也简称 ACID 特性。

(1) 原子性。

一个事务对于数据的所有操作都是不可分割的整体,这些操作要么全部执行,要么全部不执行。原子性是事务概念本质的体现和基本要求。

(2) 一致性。

事务执行完成后,数据库中的内容必须全部更新,确保事务执行后使数据库从一个一致性状态变成另一个一致性状态,此时数据库中的数据具备正确性和完整性。如果数据库只包含事务成功提交的结果,则说明数据库处于一致性状态;如果数据库系统运行过程中发生了故障,有些事务尚未完成就被迫中断,这些未完成事务对数据库所做的更新操作有一部分已写入物理数据库,这时数据库就处于一种不正确的状态,或者说不一致的状态,为了保证一致性,系统会对事务中对数据库的所有已完成的操作全部撤销,回滚到事务开始时的一致性状态。

(3) 隔离性。

隔离性也称独立性,表明一个事务的执行不能被其他事务干扰,即一个事务内部的操作及使用的数据对其他并发事务是隔离的,并发执行的各个事务之间不能互相干扰。

(4) 持续性。

持续性也称永久性,表明一个事务一旦提交,它对数据库中的数据的改变就应该是永久的,接下来的其他操作或故障不应该对其执行结果有任何影响。

事务是并发控制的基本单位,保证事务的 ACID 特性是事务处理的重要任务。事务的 ACID 特性可能遭到破坏的因素一般有两种。

(1) 多个事务并行运行时,不同事务的操作交叉执行。此时 DBMS 必须保证多个事务的交叉运行不影响这些事务的原子性。

(2) 事务在运行过程中被强行停止。此时 DBMS 必须保证被强行停止的事务对数据库和其他事务没有任何影响。

5.2.2 并发控制概述

数据库系统是多用户共享数据库资源,尤其是多个用户可以同时存取相同数据,如银行系统数据库、超市管理数据库等都是多个用户共享的数据库系统。在这些系统中,同一时间可同时运行数百个事务。若对多用户的并发操作不加以控制,就会造成数据存取错误,破坏数据库的一致性和完整性。

在 DBMS 运行多个事务时,如果一个事务完成以后,再开始另一个事务,这种执行方式为事务的串行执行。如果 DBMS 可以同时接受多个事务,并且这些事务在时间上可以重叠执行,这种执行方式为事务的并发执行。并发执行能提高系统资源的利用率,改善短事务的响应时间等。但并发执行可能会破坏事务的 ACID 特性。下面举例说明并发执行带来的数据不一致的问题。

设有两个校园超市收银台 A 和 B,其中 A 和 B 同时收取同一商品(洗衣粉)的费用,修改洗衣粉的库存数量。其操作过程及顺序如下。

收银台 A(事务 A)读出目前洗衣粉的库存数量,假设为 20 袋。

收银台 B(事务 B)读出目前洗衣粉的库存数量也为 20 袋。

收银台 A 此时要卖出 3 袋洗衣粉，则修改库存数量 20－3＝17，并将 17 写回数据库中。

收银台 B 此时也要卖出 4 袋洗衣粉，则修改库存数量 20－4＝16，并将 16 写回数据库中。

从上述操作可以看出，事务 B 覆盖了事务 A 对数据库的修改，使数据库中的数据不可信，这种情况称为数据的不一致性。这种不一致性就是由并发执行引起的。由于在并发执行下 DBMS 对事务 A 和 B 操作序列的调度是随机的，这会产生数据不一致，而这种不一致性是致命的，且在现实生活中是绝对不允许发生的。因此数据库管理员必须想办法避免这种情况，这就是数据库管理系统在并发控制中要解决的问题。

1. 并发操作导致的问题

数据库的并发操作会导致三种问题：丢失更新、读“脏”数据和不可重复读。下面分别介绍这三种问题。

5 并发控制概述之并发操作导致的问题

（1）丢失更新。

丢失更新是指当两个或两个以上的事务选择同一数据值，在更新最初的读取值时，会发生丢失更新的问题。两个事务 T1 和 T2 从数据库读取同一数据并进行更新，T1 执行更新后提交，T2 在 T1 更新后也对该数据进行了更新，此时 T2 提交的结果就破坏了 T1 提交的结果，导致 T1 的修改被 T2 覆盖掉，这样 T1 的更新就被丢失了。这是由于每个事务都不知道其他事务的存在，最后的更新将重写由其他事务所做的更新，这将导致数据丢失。

丢失更新是由于多个事务对同一数据并发进行写入操作引起的。前面例子中收银台 A 和收银台 B 同时对洗衣粉数量进行更新时，最后进行的更新数量必将替代第一个更新的数量，得到错误的结果 16 袋。如果收银台 A 完成收费以后，收银台 B 再收费就可避免这样的问题发生。

（2）读“脏”数据。

读“脏”数据是指一个事务读取了另一个事务失败运行过程中的数据。也就是说，事务 T1 更新了某一数据，并将更新结果写入磁盘，然后事务 T2 读取了这一数据（T1 更新后的数据）。过了一段时间，由于某种原因 T1 撤销了更新操作，T1 修改过的数据又恢复为原值，此时 T2 读取的数值与数据库中实际数据值不一致。这种数据就是“脏”数据。

前面例子中，收银台 A、B 同时修改洗衣粉数量，收银台 A 修改洗衣粉数量为 17，未做提交操作，这时收银台 B 将修改后的数量 17 读取出来，之后收银台 A 执行回滚操作，数量恢复为原值 20，而收银台 B 仍然在使用已回滚的数量 17。这种修改了但未提交随后又被回滚的数据就是“脏”数据。

（3）不可重复读。

不可重复读是指一个事务读取数据后，另一个事务对该数据进行更新，当前一个事务再次读取这个数据时，所得到的数据与之前读取的数据不一致。这里的更新操作包括以下三种情况。

事务 T1 按一定条件从数据库中读取某些记录后，事务 T2 在其中修改了部分数据，当 T1 再次按相同条件读取数据时，发现数据与前一次读取的数据不一样。

事务 T1 按一定条件从数据库中读取某些记录后，事务 T2 在其中插入数据，当 T1 再次按相同条件读取数据时，发现数据库中多出了一些数据。

事务 T1 按一定条件从数据库中读取某些记录后，事务 T2 在其中删除数据，当 T1 再次按相同条件读取数据时，发现数据库中之前的数据消失了。

前面例子中,收银台 A、B 同时修改洗衣粉数量,收银台 A 在某一时刻读取的数量是 20 袋,过了一段时间,收银台 B 卖出 4 袋将数量修改为 16,此时收银台 A 读取的值不再是最初的 20 了。

并发操作破坏了事务的隔离性从而导致出现以上三种问题。并发控制是用某种方法来执行并发操作,使一个事务的执行不受其他事务的干扰,避免造成数据的不一致。

2. 并发控制的方法

实现并发控制的主要方法是使用封锁机制。锁可以防止事务的并发问题,在多个事务并发执行时能够保证数据库的完整性和一致性。封锁是指一个事务 T 在对某个数据对象操作之前,先向系统发出请求,对其加锁。加锁后事务 T 对该数据对象有一定的控制,在事务结束之后释放锁。而在事务 T 释放锁之前,其他事务不能更新此数据对象,以保证数据操作的正确性和一致性。封锁是一种并发控制技术,用来调整对数据库中共享数据进行并行存取的技术。前面超市的例子中,当收银台 A 要修改洗衣粉数量,在读取出数量前先封锁数量,再对数量进行读取和修改操作,这是收银台 B 就不能读取和修改数量,直到收银台 A 完成操作,将修改后的数量重新写回数据库,并释放对数量的封锁后,收银台 B 才可以读取和修改数量,这样就不会导致数据不一致的问题。

5 并发控制概述之并发控制方法

1) 基本锁

具体的控制由封锁的类型决定。基本的封锁类型有两种:排他锁(Exclusive Locks,简称 X 锁)和共享锁(Share Locks,简称 S 锁)。

(1) 排他锁(X 锁)。

排他锁又称写锁,可以防止并发事务对数据进行访问,其他事务不能读取或更新锁定的数据。如果事务 T 对数据对象 R 加上 X 锁,则只允许事务 T 读取和更新 R,其他任何事务不能再对 R 加任何类型的锁,直到事务 T 释放 R 上的锁。这就保证了其他事务在 T 释放 R 上的锁之前不能再读取和更新 R。由此可见,X 锁采用的方法是禁止并发操作。

(2) 共享锁(S 锁)。

共享锁又称读锁,允许并发事务读取数据。若事务 T 对数据对象 R 加上 S 锁,则事务 T 读取 R 但不能修改 R,其他任何事务只能再对 R 加 S 锁,而不能加 X 锁,直到事务 T 释放 R 上的 S 锁。这保证了其他事务可以读取 R,而不能再释放 R 上的 S 锁之前对 R 进行修改操作。

对数据库中数据进行读取操作不会破坏数据的完整性,而更新操作才会破坏数据的完整性。加锁的真正目的在于防止更新操作对数据一致性的破坏。S 锁只允许多个事务同时读取同一数据,不能对数据进行更新操作;X 锁只允许一个事务对同一数据进行读取和更新操作,其他事务只能等待 X 锁的释放,才能对该数据进行相应的操作。

2) 基本锁的兼容性

排他锁和共享锁的控制可以用如表 5-4 所示的锁的兼容性来表示。

表 5-4　锁的兼容性

T2	T1		
	排他锁(X 锁)	共享锁(S 锁)	--(没有锁)
排他锁(X 锁)	否	否	是
共享锁(S 锁)	否	是	是
--(没有锁)	是	是	是

在表 5-4 锁的兼容性内容中，最上面一行是事务 T1 已经获取的数据对象上的锁类型，其中，--表示没有加锁。最左侧一列是事务 T2 针对同一数据对象发出的封锁请求，该请求是否被满足，用“是”和“否”在表格中表示出来。“是”表示事务 T2 的封锁请求与 T1 所获取的锁兼容，可以满足请求；“否”表示事务 T2 的封锁请求与 T1 的锁不兼容，请求被拒绝。

3）锁的粒度

锁的粒度是指封锁对象的大小。根据对数据的不同处理，封锁的对象可以是字段、记录、表、数据库等逻辑单元，也可以是页、块等物理单元。

封锁粒度与系统的并发度和并发控制的开销密切相关。封锁粒度越小，系统能够被封锁的对象就越多，并发度越大，封锁机制越复杂，系统开销越大。相反，封锁粒度越大，系统能被封锁的对象就越少，并发度越小，封锁机制越简单，系统开销就越小。

在实际的应用中选择封锁的粒度，需要同时考虑封锁机制和并发度两个因素，对系统开销与并发度进行权衡，以获得最优的效果。一般来说，需要处理大量元组的事务可以以表作为封锁单元，而对于处理少量元素的事务，则可以以元组作为封锁单元，以提高系统的并发度。


5 并发控制概述之封锁协议

3. 封锁协议

在使用排他锁和共享锁对数据对象进行加锁时，还需要约定一些规则：何时申请锁、持锁时间、何时释放锁等。这些规则称为封锁协议。对封锁方式规定不同的规则，就形成了不同级别的封锁协议，不同级别的协议能达到的数据一致性级别也不同。下面介绍三种封锁协议。

（1）一级封锁协议。

一级封锁协议是指事务 T 在修改数据对象之前必须先对其加 X 锁，直到事务结束（包括正常结束和非正常结束）时才释放锁。一级封锁协议可以防止丢失更新问题的发生。

在一级封锁协议中，如果事务仅仅是读数据而不是更新数据，则不需要加锁。所以一级封锁协议不能保证可重复读和读“脏”数据。

（2）二级封锁协议。

二级封锁协议是指在一级封锁协议基础上，加上事务 T 对要读取的数据之前必须先对其加 S 锁，读取完后立即释放 S 锁。二级封锁协议可以防止数据丢失更新问题，还可以防止读“脏”数据。

在二级封锁协议中，由于事务 T 读取完数据后立即释放了 S 锁，因此不能保证可重复读数据。

（3）三级封锁协议。

三级封锁协议是指在一级封锁协议基础上，加上事务 T 在读取数据之前必须先对其加 S 锁，读取完后并不释放 S 锁，直到事务 T 结束才释放。三级封锁协议除可以防止丢失更新和不读“脏”数据外，还可以防止不可重复读。

三个封锁协议均规定对数据对象的更新必须加 X 锁，而它们的主要区别在于读取操作是否需要申请封锁，何时释放锁。三个级别的封锁协议的主要规则及能解决的问题如表 5-5 所示。

表 5-5　不同级别的封锁协议

封锁协议	排他锁(X 锁)	共享锁(S 锁)	不丢失更新	不读脏数据	可重复读
一级封锁协议	必须加锁，直到事务结束才释放	不加锁	是		
二级封锁协议	必须加锁，直到事务结束才释放	加锁，读取完后立即释放锁	是	是	
三级封锁协议	必须加锁，直到事务结束才释放	必须加锁，直到事务结束才释放	是	是	是

4. 死锁和活锁

封锁技术可以有效地解决并发操作的一致性问题，但也会带来一些新的问题：活锁和死锁等问题。

1）活锁

当两个或多个事务请求对同一数据进行封锁时，可能会存在某个事务处于永远等待锁的情况，这种现象称为活锁。例如事务 T1 封锁了数据对象 R 后，事务 T2 也申请封锁 R，于是 T2 等待；接着事务 T3 也申请封锁 R。当 T1 释放了 R 上的封锁后，系统首先批准了 T3 的请求，T2 仍然等待。这时事务 T4 又申请封锁 R，当 T3 释放了 R 上的封锁后，系统又批准了 T4 的请求，这样依次继续，T2 有可能永远等待，这就是活锁。


5 并发控制概述之死锁和活锁

避免活锁最简单的方法就是采用先来先服务的策略。当多个事务请求封锁同一数据对象时，封锁子系统按申请封锁的先后顺序对事务进行排队，数据对象上的锁一旦释放就批准申请队列中的第一个事务获得锁。

2）死锁

在同时处于等待状态的两个或多个事务中，其中每一个事务又在等待其他事务释放封锁后才能继续执行，这样出现多个事务彼此相互等待的状态就称为死锁。例如事务 T1 封锁了数据对象 R1，事务 T2 封锁了数据对象 R2。之后 T1 又申请封锁数据对象 R2，由于 T2 已经封锁了 R2，于是 T1 的申请被拒绝只能等待，直到 T2 释放 R2 上的锁。接着 T2 又申请封锁 R1，由于 R1 已经被 T1 封锁，于是 T2 的申请被拒绝只能等待，直到 T1 释放 R1。这样就出现了 T1 在等待 T2 而 T2 又在等待 T1 的局面，T1 和 T2 两个事务永远不能结束，形成死锁。

目前在数据库中解决死锁问题的方法主要有两类：一类是采取一定的措施来预防死锁的发生；另一类是允许死锁的发生，但需采取一定的手段定期诊断系统中有无死锁，若有则解除它。

（1）死锁的预防。

预防死锁就是要破坏产生死锁的条件，通常有如下两种方法。

① 一次性封锁法。一次性封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行。例如针对前面死锁中的例子，事务 T1 将需要的数据对象 R1 和 R2 一次加锁，T1 就可以执行，而事务 T2 等待。当 T1 执行完后释放 R1、R2 上的锁，T2 就获得 R1 和 R2 上的锁，继续执行。这样就不会发生死锁。一次性封锁法虽然可以有效地防止死锁的发生，但也存在不足：将事务以后要用的全部数据对象加锁，扩大了封锁的范围，降低了系统的并发度，从而影响了系统的效率；另外，需要事先精确地确定每个事务所要封

锁的所有数据对象，这对于不断变化的数据库来讲是很困难的，因此只能扩大封锁范围，将事务可能需要用到的数据进行加锁，这会进一步降低并发度。

② 顺序封锁法。顺序封锁法是要求所有事务必须按照一个预先约定的封锁顺序对所要用到的数据对象进行封锁。例如规定事务封锁数据对象 R1、R2 的顺序依次是 R1、R2，则事务 T1 和 T2 必须先封锁 R1 再封锁 R2，当 T2 请求 R1 的封锁时，由于 T1 已经封锁了 R1，则 T2 就只能等待，T1 释放 R1 和 R2 的锁之后，T2 就继续执行，这样就不会发生死锁。顺序封锁在一定程度上可以有效地防止死锁，但仍然存在不足：很难预先确定所有数据对象的加锁顺序；当封锁的数据对象很多时，随着数据的不断更新，维护数据对象的顺序也很困难。

因此，预防死锁策略难以实施，在解决数据库死锁的问题上 DBMS 普遍采用诊断并解除死锁的方法。

(2) 死锁的诊断与解除。

死锁的解除是指允许产生死锁，在死锁发生后通过一定手段予以解除。一般使用超时法或事务等待图法。

① 超时法。超时法是指对每个锁设定一个时限，如果某个事务的等待时间超过了该时限，就认为发生了死锁，此时调用解锁程序，以解除死锁。超时法实现简单，但存在明显不足：时限难以设置，若设置太长，则会导致死锁发生后不能及时发现；有可能误判死锁，事务可能因为其他原因使等待超时，系统会误认为发生了死锁。

② 事务等待图法。事务等待图是一个特殊的有向图 $G=(T,U)$。T 为结点的集合，每个结点表示正在运行的事务；U 为边的集合，每条边表示事务等待的情况。若 T1 等待 T2，则 T1、T2 之间划一条有向边，从 T1 指向 T2。建立事务等待图之后，诊断死锁的问题就变成了判断有向图 G 中是否存在回路的问题。事务等待图动态地反映了所有事务的等待情况，并发控制子系统周期性地生成事务等待图，并进行检测，如果图中没有回路，则没有发生死锁，反之则说明发生了死锁。

一旦检测到系统存在死锁，DBMS 就要设法解除。通常采用的方法是选择一个处理死锁代价最小的事务，将其撤销，释放该事务持有的所有锁，使其他事务得以继续运行下去。当然，为了保证数据的一致性，对撤销事务所执行的数据更新操作必须加以恢复。

5. 并发调度的可串行性

数据库管理系统对并发事务中的操作调度是随机的，不同的调度会产生不同的结果。什么样的调度是正确的呢？显然，串行调度是正确的。一般来讲，如果多个事务在某个调度下的执行结果与这些事务在某个串行调度下的执行结果相同，那么这个调度也是正确的。虽然以不同顺序串行执行事务可能会产生不同的结果，但不会将数据库置于不一致的状态，因此这个调度是正确的。

5 并发控制概述之并发调度的可串行性

多个事务的并发执行是正确的，当且仅当结果与按某一顺序串行地执行这些事务时的结果相同，则称这种调度策略为可串行化的调度。

可串行性是并发事务正确调度的准则。按这个准则规定，一个给定的并发调度，当且仅当它可串行化时，才认为它是正确的调度。为保证并发操作的正确性，数据库管理系统的并发控制机制必须提供一定的手段来保证调度是可串行化的。

【例 5-23】 假设有两个事务 T1 和 T2，分别包含下列操作。

事务 T1：读取 B；A＝B－3；写回 A。

事务 T2：读取 A；B＝A－3；写回 B。

假设 A、B 的初值均为 20，若按 T1→T2 的顺序执行后，其结果 A＝17，B＝14；若按 T2→T1 的顺序执行后，其结果 A＝14，B＝17。当并发调度时，如果执行的结果是这两者之一，则认为都是正确的并发调度策略。图 5-4 给出了这两个事务的四种调度策略。

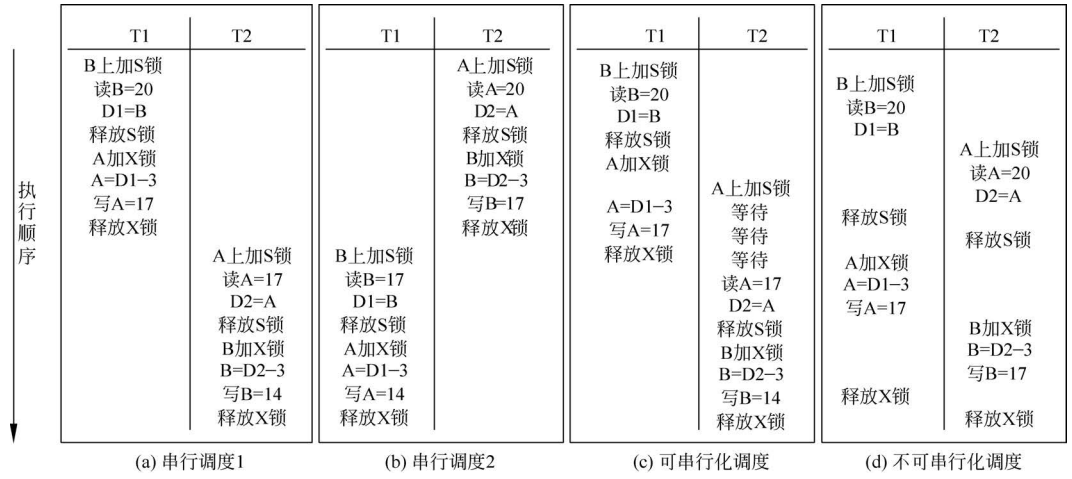


图 5-4　并发事务的不同调度策略


5 并发控制概述之两段锁协议

图 5-4(a)和图 5-4(b)是不同的串行调度策略，虽然执行结果不同，但它们都是正确的调度。图 5-4(c)虽不是串行调度，但其执行的结果与串行调度的结果相同，所以该调度是正确的。图 5-4(d)的执行结果与前两个串行调度的结果都不同，所以是错误的调度。

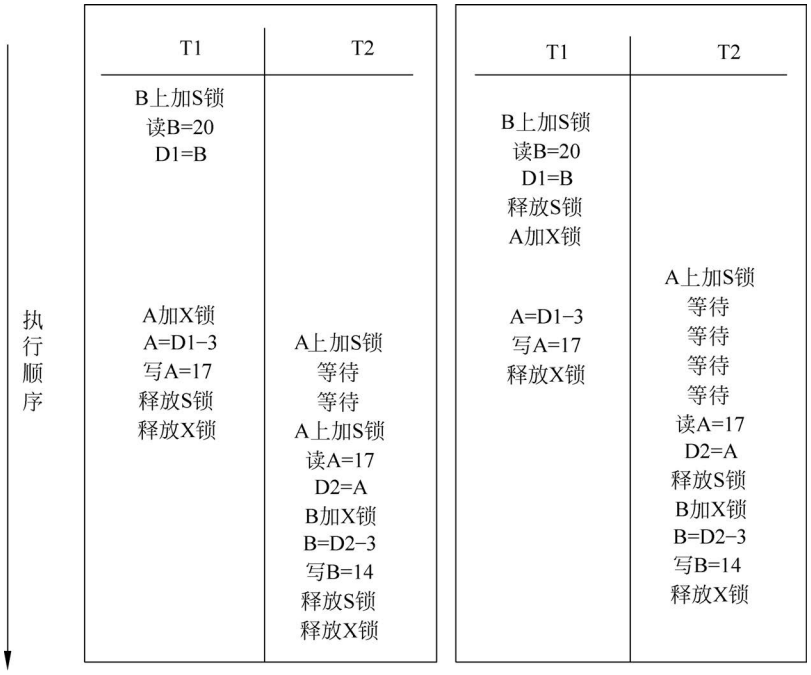
6. 两段锁协议

为保证并发调度的正确性，数据库管理系统的并发控制机制必须提供一定的手段来保证调度的可串行化。目前，数据库管理系统普遍采用两段锁协议来实现并发调度的可串行化，从而保证调度的正确性。

两段锁协议是最常用的一种封锁协议。它是指所有的事务必须分为两个阶段对数据对象进行加锁和解锁。具体包括两方面的内容：在对任何数据进行读写操作之前，要先申请并获得对该数据的封锁；在释放一个封锁之后，事务不再申请和获得对该数据的封锁。

所谓两段锁就是事务分为两个阶段：第一阶段是申请封锁，在这个阶段，事务可以申请获得任何数据对象上的任何类型的锁，但是不允许释放任何锁；第二阶段是释放封锁，在这个阶段，事务可以释放任何数据对象上的任何类型的锁，但不允许申请任何锁。如果并发执行的所有事务都遵守两段锁协议，则这些事务的任何并发调度策略都是可串行化的。

事务遵守两段封锁协议是可串行化调度的充分条件，而不是必要条件。也就是说，如果并发事务都遵守两段锁协议，则对这些事务的任何并发调度策略都是可串行化的。反之，若对并发事务的调度是可串行化的，并不意味着这些事务都符合两段锁协议。如图 5-5 所示，图 5-5(a)遵守两段锁协议，图 5-5(b)不遵守两段锁协议，但它们都是可串行化的调度。



(a) 遵守两段协议　　(b) 不遵守两段协议

图 5-5　可串行化调度

5.2.3　MySQL 的事务与并发控制


5 MySQL的事务与并发控制之事务处理模型

1. MySQL 的事务处理模型

MySQL 的事务处理模型有三种：自动提交事务模型，MySQL 默认的事务处理模型，每条单独的语句就是一个事务；显式事务模型，允许用户定义事务的启动和结束，通常指定显示的开始标记 BEGIN WORK(或 START TRANSACTION)和结束标记 COMMIT(或 ROLLBACK)；隐式事务模型，在当前事务完成提交或回滚后，新事务自动启动，隐式事务模型不需要使用开始标记标识事务的开始，但需要用结束标记语句来提交或回滚事务。

系统变量@@autocommit 的值为 1 时，表示 MySQL 采用自动提交事务模型，当用户执行一条 SQL 语句后，该语句对数据库的修改就立即被提交成为永久性修改保存到磁盘上，事务执行结束。当事务由多条 SQL 语句构成时，需要关闭自动提交事务模型，通过语句 SET @@autocommit=0 来实现，此时需要明确地指示每个事务的结束标记。

(1) 开始事务。

MySQL 默认事务都是自动提交的，要显示启动事务必须使用 BEGIN WORK 或 START TRANSACTION 语句标识事务的开始。其语法格式如下：

```
START TRANSACTION | BEGIN WORK;
```

说明：在存储过程中只能使用 START TRANSACTION 语句来开启一个事务，因为 MySQL 会自动将 BEGIN 识别为 BEGIN…END 语句。

(2) 提交事务。

COMMIT 语句用于结束一个用户定义的事务，保证对数据的修改已经成功写入数据

库,此时事务正常结束。其语法格式如下:

```
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE];
```

说明:提交事务的最简单形式,只需 COMMIT 语句,详细写法是 COMMIT WORK。AND CHAIN 子句会在当前事务结束时立刻启动一个新事务,并且新事务与刚结束的事务有相同的隔离等级。RELEASE 子句在终止当前事务后,会让数据库服务器断开与当前客户端的连接。NO 关键字用以控制 CHAIN 或 RELEASE 完成。

(3) 回滚事务。

回滚事务使用 ROLLBACK 语句,回滚会结束用户的事务,并撤销正在进行的事务开始标记后的所有修改。其语法格式如下:

```
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE];
```

(4) 设置保存点。

ROLLBACK 语句除了撤销整个事务以外,还可以用来使事务回滚到某个点,在这之前需要使用 SAVEPOINT 语句来设置一个保存点。其语法格式如下:

```
SAVEPOINT point_name;
```

说明:point_name 为设置的保存点名称,一个事务中可以有多个保存点。设置保存点后,可以使用 ROLLBACK[WORK] TO SAVEPOINT point_name 语句回滚到 point_name 处。注意,当事务回滚到某个保存点后,在该保存点之后设置的保存点将被删除。

【例 5-24】 向 Goods 表插入数据。

```
START TRANSACTION;
INSERT INTO Goods(GoodsNO, GoodsName) values('GN011','松子');
INSERT INTO Goods(GoodsNO, GoodsName) values('GN012','瓜子');
INSERT INTO Goods(GoodsNO, GoodsName) values('GN013','花生');
INSERT INTO Goods(GoodsNO, GoodsName) values('GN014','开心果');
COMMIT;
```

【例 5-25】 学生在校园超市购买商品时,商品的库存数量不能小于 0(Goods 表中 Number 字段类型为非负整数)。创建存储过程 pro_transale 实现学生购买商品的业务。

```
CREATE PROCEDURE pro_transale(IN sno varchar(20), IN gno varchar(20), IN num int)
    BEGIN
        DECLARE CONTINUE HANDLER FOR 1690
            BEGIN
                SELECT '商品库存数量不足';
                ROLLBACK;
            END;
        START TRANSACTION;
        UPDATE Goods SET Number = Number - num WHERE GoodsNO = gno;
        INSERT INTO SaleBill VALUES(gno,sno,CURRENT_TIMESTAMP(),num);
        COMMIT;
    END;
```

注意,如果 Goods 表中 number 设置了 CHECK 约束,确定范围为 Number >=0,则在存储过程中使用"DECLARE CONTINUE HANDLER FOR 3819"。

【例 5-26】 设置保存点示例。向 Goods 表插入数据。

```
START TRANSACTION;
```

```
INSERT INTO Goods(GoodsNO, GoodsName) values('GN016','松子');
INSERT INTO Goods(GoodsNO, GoodsName) values('GN017','瓜子');
SAVEPOINT p;
INSERT INTO Goods(GoodsNO, GoodsName) values('GN018','花生');
INSERT INTO Goods(GoodsNO, GoodsName) values('GN019','开心果');
ROLLBACK TO p;
```

2. MySQL 的并发控制

数据库管理系统的并发控制用于实现事务的并发操作，避免出现数据的不一致问题，确保事务的一致性。MySQL 使用封锁的方法进行并发控制，防止事务修改另一个未完成事务的数据。

MySQL 的锁分为表级锁和行级锁。表级锁是以数据表为单位进行封锁，行级锁是以元组为单位进行封锁。表级锁的粒度大，行级锁的粒度小。封锁粒度越小，并发访问的性能越高，越适合并发更新操作；封锁粒度越大，并发访问性能越低，更适合做并发查询操作。封锁粒度越小，完成某个功能时需要加锁、解锁的次数会增加，需要消耗的系统资源较多，可能出现资源的恶性竞争，或发生死锁。

（1）表级锁。

表级锁用于锁定整个数据表，包括读锁定和写锁定两种。对于任何针对数据表的查询或更新操作，MySQL 会隐式地加表级锁。隐式锁的生命周期非常短，且不受数据库开发人员控制。使用 LOCK TABLES 语句显示加表级锁，其语法格式如下：

```
LOCK TABLES table_name READ | [table_name WRITE]…;
```

说明：READ 表示加表级读锁，WRITE 表示加表级写锁。在对数据表加了读锁后，事务可以对数据进行查询操作，如果对该表进行更新操作时出错，其他事务对该表可以进行数据查询操作，但更新操作会被阻塞。在对数据表加了写锁后，事务可以对数据进行查询和更新操作，但其他事务对该表进行查询和更新操作时会被阻塞。

MySQL 解锁的语法格式如下：

```
UNLOCK TABLES;
```

（2）行级锁。

行级锁用于锁定数据表的行，相比表级锁对数据进行更精细的控制。行级锁包括共享锁、排他锁。事务获得数据行的共享锁，则可以对该数据行进行查询操作而不能进行更新操作，此时其他事务可以成功申请对该数据行加共享锁，但不能成功申请加排他锁；事务获得数据行的排他锁，则可以对数据行进行查询和更新操作，此时其他事务不能成功申请对数据行加任何锁。

在查询语句中，MySQL 使用 LOCK IN SHARE MODE 和 FOR UPDATE 两个语句为满足条件的数据行加共享锁和排他锁，其语法格式如下：

```
SELECT * FROM table_name WHERE boolean_expression
LOCK IN SHARE MODE | FOR UPDATE;
```

说明：LOCK IN SHARE MODE 表示加共享锁；FOR UPDATE 表示加排他锁。但为更新语句（INSERT、UPDATE、DELETE）时，MySQL 会对符合条件的数据行自动加隐式排他锁。

(3) 意向锁。

意向锁(Intention Locks,简称I锁)是一种表级锁,锁定的粒度是整个数据表。引入意向锁的目的是方便检测表级锁和行级锁是否兼容,从而实现多粒度封锁机制。意向锁分为意向共享锁(IS)和意向排他锁(IX)两类。意向共享锁是指事务向数据行加行级共享锁时,事务必须先取得该表的IS锁;意向排他锁是指事务向数据行加行级排他锁时,事务必须先取得该表的IX锁。

在MySQL中,意向锁是自动加的,不需要用户干预。当事务在向数据表中某些行加共享锁时,MySQL会自动向该数据表加意向共享锁;当事务向数据表中某些行加排他锁时,MySQL会自动地向该数据表加意向排他锁。如果一个事务请求的锁类型与当前锁兼容,MySQL就将该请求的锁授予该事务。

5.3 备份及恢复管理

尽管数据库管理系统采用了许多措施来保证数据库的安全性和完整性,但故障仍不可避免,这会影响甚至破坏数据库,造成数据错误或丢失。通过备份和恢复数据库,可以防止因为各种原因而造成的数据破坏和丢失,并使数据库继续正常工作。


5 备份及恢复管理之故障的种类

5.3.1 故障的种类

数据库系统中常见的故障种类包括事务内部故障、系统故障、介质故障和计算机病毒。

1. 事务内部故障

事务内部故障是指在当前事务内部操作执行过程中可能发生的故障,分为预期故障和非预期故障。

(1) 预期故障。

预期故障是事务内部语句执行错误而引起事务异常终止的故障。它发生在单个事务内部,编写事务的程序员应该预先估计到这个错误并加以处理。例如:校园超市中,学生购买的商品数量比商品库存量要大时,如果继续操作就会出现问题。这种情况可以事先在事务内部语句中增加判断和ROLLBACK语句。当事务执行到ROLLBACK语句时,系统会对事务进行撤销操作(UNDO操作),撤销事务对数据库的一切影响,保证事务的原子性。

(2) 非预期故障。

非预期故障是事务内部语句执行过程中发生的无法预估并不能预处理的错误。事务内部故障大部分属于非预期故障。例如,并发执行事务发生死锁、运算溢出、违反完整性约束而被提前终止等。这些故障无法预估,必须由数据库管理系统强行执行UNDO操作,使数据库恢复到该事务运行之前的状态。

2. 系统故障

系统故障又称软故障,是指造成系统停止运行并要求系统重新启动的事件。导致系统故障的原因很多,例如,CPU故障、操作系统出错、数据库管理系统错误、系统断电等。这类故障不会破坏数据库,但会影响正在运行的所有事务,导致事务以非正常的方式终止。发生系统故障后,系统必须重新启动,内存中数据库工作区内的数据会被丢失。

发生系统故障时,一些尚未完成的事务的结果可能已经写入数据库,从而造成数据库可

能处于不正确的状态；系统重新启动时，应对未完成的事务进行 UNDO 操作。对于已经完成的事务，可能存在部分更改数据仍留在内存中，尚未写入数据库，这也会使数据库处于不正确的状态；系统重新启动时，应对已经完成的事务进行重新执行操作（REDO 操作）。

3. 介质故障

介质故障也称硬故障，是指外部存储介质故障，例如，磁盘损坏、强磁场干扰等。这类故障会破坏数据库或部分数据库，并影响正在存取这部分数据的所有事务。相比前两类故障发生的可能性低很多，但破坏性很强。处理这类故障的主要技术是数据备份，当遇到介质故障时，加载最近的备份文件，重新执行该备份之后提交的所有事务。

4. 计算机病毒

计算机病毒是一种能够自我复制传播的计算机指令或程序代码，它们会破坏计算机的功能或者破坏数据，影响计算机包括数据库系统的使用。对于已经感染计算机病毒的数据库文件应使用杀毒软件进行查杀，如果无法查杀，则只能使用数据库备份文件进行恢复，从而达到数据库一致性状态。

5.3.2 数据备份概述

5 备份及恢复管理之数据备份概述

数据备份是指定期或不定期地对数据库及其相关信息进行复制，在本地机器或其他机器上创建数据库的副本。数据库备份记录了在进行备份这一操作时数据库中所有数据的状态，当数据库因意外被损坏时，副本就可在数据库恢复时用来恢复数据库。因此，数据备份是保证系统安全的一项重要措施。

1. 备份类型

根据备份数据的大小，可以把备份分成以下四种。

（1）完全备份。

完全备份又称完全数据库备份，是数据备份常用的方式之一。完全备份将备份整个数据库，不仅包括用户表、系统表、索引、视图、存储过程等所有数据库对象，还包括事务日志部分。完全备份代表备份完成时的数据库，通过包括在备份中的事务日志可以使用备份恢复到备份完成时的数据库。

完全备份操作简单，便于使用。通常情况对于规模较小的数据库而言，可以快速完成完全备份，但随着数据库规模不断增大，进行一次完全备份，需要花费更多的时间和空间。因此，需要根据备份计划安排完全备份，对于大型数据库可以使用差分备份来补充完全备份。

（2）差分备份。

差分备份也称增量备份，与完全备份不同，它仅备份自上次完全备份以来对数据进行改变的内容。差分备份相比完全备份而言备份速度更快，空间更节省，简化了数据备份操作，减少丢失数据的可能性。为了减少还原频繁修改数据库的时间，可以执行差分备份。

如果数据库中的部分对象频繁更改，差分备份特别有用。在这种情况下，使用差分备份可以频繁地执行备份，并且不会产生完全备份的开销。

对于规模大的数据库来讲，完全备份需要大量的磁盘空间。为了节省备份时间和存储空间，可以在一次完全备份后安排多次差分备份。

（3）事务日志备份。

数据库事务日志是单独的文件，它记录了数据库的改变。事务日志备份是对事务日志

进行备份,备份时复制自上次备份以来对数据库所做的改变,仅需要很少的时间,因此建议频繁备份事务日志,从而减少丢失数据的可能性。

用户可以使用事务日志备份将数据库恢复到特定的即时点或恢复到故障点。

事务日志备份和差分备份有所不同。差分备份无法将数据库恢复到出现故障前某一个指定的时刻,它只能将数据库恢复到上一次差分备份结束的时刻。

(4) 文件或文件组备份。

数据库由磁盘上的许多文件构成。如果数据库非常大,执行完全备份是不可行的,则可以使用文件备份或文件组备份来备份数据库的一部分。

2. 备份设备

在创建备份时,必须选择存放备份数据库的备份设备。数据备份是可以将数据库备份到磁盘设备或磁带设备上。磁盘备份设备就是硬盘或其他磁盘上的文件,可以像操作系统文件一样进行管理,也可以将数据库备份到远程计算机的磁盘上。

3. 备份计划

创建备份的目的是恢复已损坏的数据库。但是,备份和恢复数据需要使用一定的资源,在特定的环境中进行。因此,在备份数据库之前需要对备份内容、备份频率以及数据备份存储介质等进行合理的计划。

(1) 备份内容。

备份数据库应备份数据库中的表、数据库用户、用户定义的数据库对象及数据库中的全部数据。表包括系统表、用户定义的表,还应该备份数据库日志等内容。

(2) 备份频率。

确定备份频率需要考虑的因素:存储介质出现故障时,允许丢失的数据量的大小;数据库的事务类型,以及事故发生的频率。

不同的数据库备份频率通常不一样。一般情况下,数据库可以每周备份一次,事务日志可以每日备份一次。对于一些重要的联机数据库,数据库可以每日备份一次,事务日志甚至可以每隔数小时备份一次。

(3) 备份存储介质。

常用的备份存储介质包括硬盘、磁带和命令管道等。具体使用哪种介质,要考虑用户的成本承受能力、数据的重要程度、用户的现有资源等因素。在备份中使用的介质确定以后,一定要保持介质的持续性,一般不要轻易地改变。


5 备份及恢复管理之数据恢复概述

5.3.3 数据恢复概述

数据恢复是指当系统运行过程中发生故障时,利用数据库的备份副本和日志文件将数据库恢复到故障前的某个一致性状态。不同故障有不同的恢复策略和恢复方法。

1. 事务内部故障的恢复

事务内部故障的恢复通常是对非预期事务故障的恢复。非预期事务内部故障是指事务在运行到正常终止点前被中止,这时可以利用事务操作的日志文件撤销该事务对数据库进行的修改。事务故障恢复的步骤如下。

(1) 反向扫描事务操作的日志文件,查找该事务的更新操作。

(2) 对事务的更新操作执行反向操作。也就是对已经插入的新记录执行删除操作;对

已经删除的记录执行插入操作；对已经修改的数据恢复旧值。

(3) 这样从后到前逐个扫描该事务的所有更新操作，按同样的方式进行处理，直到扫描到该事务的开始标记为止，事务故障就恢复完毕。

事务故障的恢复工作由数据库管理系统自动完成，不需要用户干预。

2. 系统故障的恢复

系统故障造成数据库数据不一致状态有两种情况：一是未完成事务对数据库的更新可能已写入数据库，这种情况需要强行撤销所有未完成的事务并清除事务对数据库所做的修改；二是已提交事务对数据库的更新可能还留在缓冲区，没有来得及写入磁盘上的物理数据库中，这种情况应将事务提交的更新结果重新写入数据库。因此系统故障恢复步骤如下。

(1) 正向扫描日志文件，找出在故障发生前已提交的事务，将其事务标记记入重做队列，同时找出故障发生时未完成的事务，将该事务标记记入撤销队列。

(2) 对撤销队列中的各个事务进行撤销处理，其方法同事务故障恢复一致。也就是对已经插入的新记录执行删除操作；对已经删除的记录执行插入操作；对已经修改的数据恢复旧值。

(3) 对重做队列中的各个事务进行重做处理，方法是正向扫描日志文件，按照日志文件中所登记的操作内容重新执行事务操作，使数据库恢复到最近的某个可用状态。

系统故障恢复仍由数据库管理系统自动完成的，不需要用户干预。

3. 介质故障的恢复

发生介质故障后，磁盘上的物理数据和日志文件被破坏，这是最严重的一种故障，可能会造成数据无法恢复。其恢复方法是重装数据库，然后重做已完成的事务。具体步骤如下。

(1) 装入最新的数据库备份副本，使数据库恢复到最近一次存储时的一致性状态。

(2) 装入最新的日志文件副本，根据日志文件中的内容重做已完成的事务。

介质故障恢复需要数据库管理员来操作，但数据库管理员只需要重装最近存储的数据库副本和有关的日志文件副本，然后执行系统提供的恢复命令即可，其余的恢复操作仍由数据库管理系统自动完成。

除了上述针对各类故障的恢复方法外，数据库还有其他恢复技术，如检查点恢复技术、数据库镜像技术等。

5.3.4 MySQL 的数据库备份与恢复

5 MySQL的数据库备份与恢复(1)

MySQL 的备份和恢复功能为存储在 MySQL 数据库中的数据提供了重要的保护手段。MySQL 数据库管理系统实现备份和恢复的方法有多种：使用图形工具如 Navicat 的相关功能来实现、直接复制数据库相关物理文件、使用语句等。这里介绍两种常用的备份与恢复语句，包括 mysqldump 和 mysql 语句、SELECT…INTO OUTFILE 和 LOAD DATA 语句。

1. mysqldump 和 mysql 语句

mysqldump 是 MySQL 自带的一个重要的客户端工具，用于实现数据库的备份功能。mysqldump 存放在 MySQL 安装路径下的 bin 目录中。

假设 MySQL 采用默认路径安装。通过 CMD 命令或打开命令提示符进入 DOS 窗口，输入 CD C:\Program Files\MySQL\MySQL Server 8.0\bin，进入安装 MySQL 的 bin 目录，就可以使用 mysqldump 语句进行数据库备份操作。

1) mysqldump 数据备份

mysqldump 完成数据备份的基本思路：首先生成数据表结构、存储过程、自定义函数、触发器等数据库对象的 CREATE 语句，接着生成数据表记录对应的 INSERT 语句，然后将这些语句导出到 SQL 脚本文件中，从而完成数据备份工作。因此，mysqldump 语句是将数据库备份成一个脚本文件，可用于备份数据表、备份数据库和备份整个数据库系统。其语法格式如下。

(1) 备份数据表。

```
mysqldump -h hostname -u username -p password dbname table_name… > filename.sql
```

说明：hostname 指定备份的主机名，如果是本地服务器，-h 选项可以省略。username 指定合法数据库用户名。password 指定用户的密码，为了安全起见，可以在使用命令备份时，省略 password，在命令运行后再输入密码。dbname 指定数据库名称。table_name 指定备份表的名称，可以是多个数据表。filename 指定备份文件的名称，备份文件名称前可以加一个绝对路径指定文件保存的位置、扩展名通常是 sql 的文件。

【例 5-27】 使用 mysqldump 备份数据库 SuperMarket 中 Student 表和 Goods 表到 D 盘 db_bk 目录下。

```
mysqldump -u root -p SuperMarket Student Goods > D:\db_bk\bk_tables.sql
```

运行结果如图 5-6 所示。

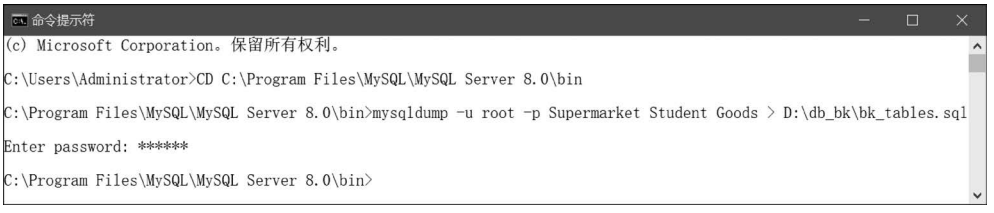


图 5-6　mysqldump 备份数据库 SuperMarket 中的表

输入 root 用户的密码后，MySQL 对 SuperMarket 中的 Student 表和 Goods 表进行备份。随后进入 D 盘查看 bk_tables.sql 文件，其内容包括创建 Student 表、Goods 表的 CREATE 语句和插入数据的 INSERT 语句。

(2) 备份数据库。

```
mysqldump -h hostname -u username -p password dbname
                                | --databases [dbname [dbname… ]] > filename.sql
```

说明：dbname 指定备份的数据库名称。--databases [dbname [dbname…]]指定备份的多个数据库名称。

【例 5-28】 使用 mysqldump 备份数据库 SuperMarket 和 School 到 D 盘 db_bk 目录下。

```
mysqldump -u root -p --databases SuperMarket school > D:\db_bk\bk_dababases.sql
```

运行结果如图 5-7 所示。

(3) 备份整个数据库系统。

```
mysqldump -h hostname -u username -p password --all-databases > filename.sql
```

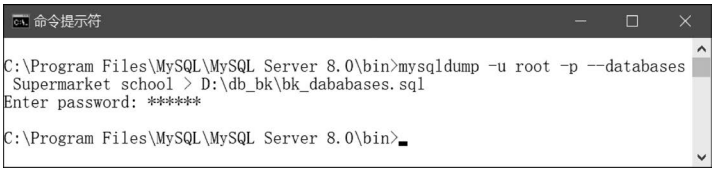


图 5-7　mysqldump 备份数据库 SuperMarket 和 School

说明：--all-databases 指定备份整个数据库系统。

【例 5-29】 使用 mysqldump 备份 MySQL 服务器上的所有数据库到 D 盘 db_bk 目录下。

```
mysqldump -u root -p --all-databases > D:\db_bk\bak_alldababase.sql
```

运行结果如图 5-8 所示。

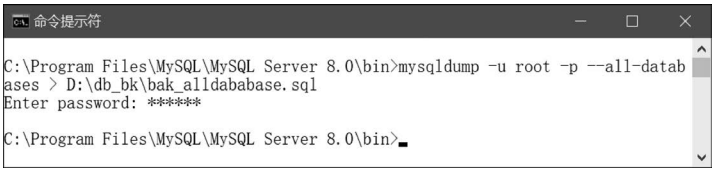


图 5-8　mysqldump 备份服务器上的所有数据库

注意：使用 mysqldump 语句进行数据备份之前，数据库管理员(DBA)需要明确备份哪些数据库对象(数据表结构、数据表记录、存储过程、自定义函数、触发器等)。确定了备份的数据库对象后，再选择对应的 mysqldump 参数列表，最后执行完成备份操作。由于篇幅的原因，这里仅罗列部分常用的参数。

--no-create-info：只导出数据，不添加 CREATE TABLE 语句。

--no-data 或-d：不导出任何数据，只导出数据表结构。

--routines 或-r：导出存储过程和自定义函数。

--triggers：备份触发器。该选项默认启用，用--skip-triggers 忽略触发器。

【例 5-30】 使用 mysqldump 备份数据库 SuperMarket，包括存储过程和自定义函数到 D 盘 db_bk 目录下。

```
mysqldump -u root -p SuperMarket --routines > D:\db_bk\bk_databases.sql
```

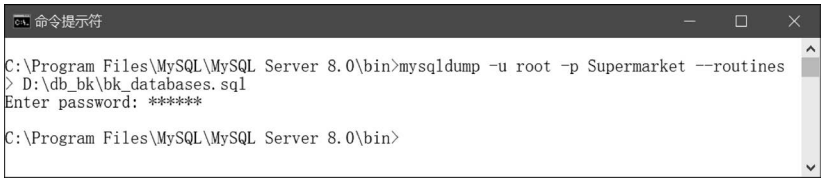


图 5-9　mysqldump 备份数据库 SuperMarket，包括存储过程和自定义函数

2) mysql 数据恢复

mysqldump 数据备份产生的文件是 SQL 脚本文件，简单的数据恢复方法就是执行 SQL 脚本 SOURCE filename.sql，也可以使用客户端工具 mysql.exe 完成数据恢复。需要注意，如果 mysqldump 语句仅备份了数据表或单个数据库时，则需要先创建数据库，然后执行 MySQL 语句，否则会出错。MySQL 语句恢复备份数据的格式如下：

```
mysql -h hostname -u username -p password [dbname] < filename.sql
```

说明：dbname 为可选项，用于指定需要恢复的数据库名称；当进行数据备份时，是对多个数据库或整个数据库系统进行备份的，恢复时不需要指定数据库。filename 表示备份文件的名称。

【例 5-31】 使用备份文件 bk_databases.sql 恢复 SuperMarket。

```
mysql -u root -p SuperMarket < D:\db_bk\bk_databases.sql
```

运行结果如图 5-10 所示。

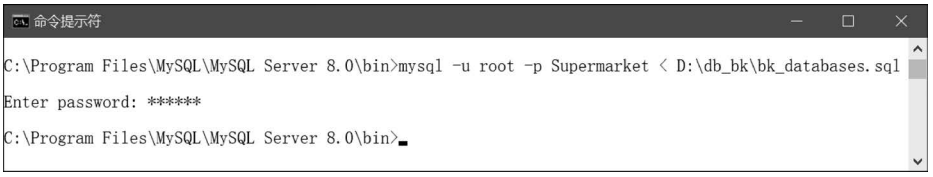


图 5-10 mysql 恢复数据库 SuperMarket

5 MySQL的数据库备份与恢复(2)

2. SELECT…INTO OUTFILE 和 LOAD DATA 语句

MySQL 数据库管理系统可以通过数据导出方法实现数据备份操作，数据导入方法实现数据恢复操作。语句 SELECT…INTO OUTFILE 和 LOAD DATA 实现文本文件的导出和导入。使用 SELECT…INTO OUTFILE 语句导出表数据的文本文件，LOAD DATA 恢复导出的表数据，但只能导出和导入表的数据内容，不包括数据表结构。

(1) SELECT…INTO OUTFILE 数据导出。

数据导出的语法格式如下：

```
SELECT column_list FROM table_name WHERE condition
    INTO OUTFILE txt_name [OPTIONS]
```

说明：txt_name 指定导出的文本文件名称。OPTIONS 指定数据行在文本文件中的存放格式，包括 FIELDS 子句和 LINES 子句。FIELDS 子句包括 TERMINATED BY、[OPTIONALLLY] ENCLOSED BY 和 ESCAPED BY 三个选项，一旦指定了 FIELDS 子句，至少要指定一个选项；LINES 子句包括 STARTING BY 和 TERMINATED BY 两个选项。

TERMINATED BY 用于指定字段值之间的符号，例如，TERMINATED BY ';'指定了分号作为两个字段值之间的标记。[OPTIONALLLY] ENCLOSED BY 用于指定包裹文件中字符值的符号，例如，ENCLOSED BY '" '指定了文件中字符值放在双引号之间，加上 OPTIONALLLY 关键字表示所有的值都放在双引号之间。ESCAPED BY 用于指定转义字符，例如，ESCAPED BY ' * '指定“*”为转义字符，取代“\”。

STARTING BY 用于指定一行记录开始的标记。TERMINATED BY 用于指定一行记录结束的标记。例如，TERMINATED BY '?'指定问号作为一行记录结束标记。

如果省略 FIELDS 子句和 LINES 子句，则默认采用以下子句：

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\ '
LINES TERMINATED BY '\n '
```

注意：使用 SELECT…INTO OUTFILE 和 LOAD DATA 语句时有权限限制，需要对

指定目录进行操作，默认的指定目录为 C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/。如果执行语句时提示 1290 错误，这是由于 MySQL 数据库的安全设置所引起的，这时需要更改 MySQL 的安全设置；MySQL 服务器有一个名为 secure-file-priv 的选项，它指定了可从哪个目录中读取和写入文件；如果尝试从不在这个目录中的位置读取或写入文件，就会出现 MySQL 1290 错误；可以通过修改 secure-file-priv 选项的值来实现在不同目录中进行文件的读取和写入；设置值为空，则 MySQL 将允许从任何位置读取和写入文件；通过“SHOW VARIABLES LIKE 'secure_file_priv';”语句可以查看 secure-file-priv 选项的值。

【例 5-32】 备份数据库 SuperMarket 中的 SaleBill 表中的数据到默认目录中，要求字段值用双引号标记，字段值之间用分号隔开，每行以＃号为结束标记。

```
SELECT * FROM SaleBill
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/sal_content.txt'
FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '#';
```

执行语句后，数据表 SaleBill 中的数据以语句中指定的标记导出到 sal_content.txt 文件中。打开文件的内容，如图 5-11 所示。

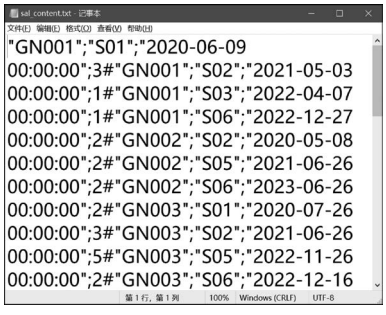


图 5-11 sal_content.txt 文件的备份结果

(2) LOAD DATA 数据导入。

数据导入的语法格式如下：

```
LOAD DATA [LOCAL] INFILE txt_name
INTO TABLE table_name(column_list) [OPTIONS];
```

说明：txt_name 用于指定待导入的数据备份文件名。table_name(column_list)用于指定需要导入数据的表名和对应的字段名。[LOCAL]是可选项，用于指定从客户端还是服务器中导入文件；指定 LOCAL 关键字，则用客户端的文件进行数据导入；省略关键字，则文件必须位于服务器中，同时需要有该文件的操作权限。OPTIONS 与 SELECT…INTO OUTFILE 语句中的含义一致，这里不再赘述。

【例 5-33】 使用例 5-32 中的备份文件 sal_content.txt 将数据导入 SuperMarket 中的 SaleBill 表中。为了能执行导入语句，先将 SaleBill 中的数据清空，即 DELETE FROM SaleBill。

```
LOAD DATA INFILE
    'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/sal_content.txt'
    INTO TABLE SaleBill
    FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
```

```
LINES TERMINATED BY '#';
```

注意：使用文本文件导入数据时，需要确保文本文件的编码和导入表的编码一致。例如，表的编码定义为 utf-8，则文本文件的编码也要调整为 utf-8，否则会报乱码错误。

小结

数据库管理与维护功能用以保证数据库中数据的正确有效和安全可靠。本章从数据库的安全性管理、并发控制和数据库的备份及恢复管理三方面进行了阐述。

数据库的安全性管理是数据库管理系统中非常重要的部分，安全性管理的好坏直接影响数据库数据的安全。与数据库相关的安全措施包括用户身份鉴别、存取控制、视图机制、数据加密、审计等。本章介绍了 MySQL 数据库管理系统的安全性管理。MySQL 主要通过权限来控制用户对数据库的访问，权限信息存放在系统数据库 mysql 中，这些权限信息根据不同的级别分别存放在 user 表、db 表、tables_priv 表、columns_priv 表和 procs_priv 表中。MySQL 的用户管理包括创建用户、删除用户、修改用户名和密码等操作，使用 CREATE USER 创建用户、DROP USER 删除用户、RENAME USER 修改用户名、SET PASSWORD 修改用户密码。MySQL 的权限管理包括授予权限和收回权限，授予权限分为授予列级权限、表级权限、存储过程级权限、数据库级权限、服务器级权限，使用 GRANT 语句授权、REVOKE 语句收回权限。逐一为用户授予权限，工作量大，MySQL 提供角色管理简化用户权限管理工作，使用 CREATE ROLE 创建角色，使用 GRANT 为角色授权、为用户授予角色，使用 DROP ROLE 删除角色。

本章接着介绍了事务和并发控制的概念。事务在数据库中是非常重要的一个概念，它是保证数据并发控制的基础。事务的特点是事务中的操作是一个完整的工作单元，这些操作，要么执行全部成功，要么执行全部不成功。只要数据库管理系统能够保证系统中一切事务的 ACID 特性，即事务的原子性、一致性、隔离性和持续性，也就保证了数据库处于一致状态。并发控制是当同时执行多个事务时，为了保证一个事务的执行不受其他事务的干扰所采取的措施。并发控制的主要方法是封锁，根据对数据操作的不同，锁可以分为共享锁和排他锁两种、当只对数据做读取操作时，加共享锁；当需要对数据进行修改操作时，需要加排他锁。在一个数据对象上可以同时存在多个共享锁，但只能同时存在一个排他锁。本章介绍了最常用的封锁方法和三级封锁协议。不同的封锁和不同级别的封锁协议所提供的系统一致性保证是不同的。对数据对象施加封锁会带来活锁和死锁问题，数据库一般采用先来先服务、死锁诊断和解除等技术来防止活锁和死锁的发生。为了保证并发执行的事务是正确的，一般要求事务遵守两阶段锁协议，即在一个事务中明显地分锁申请期和释放期，它是保证事务是可并发执行的充分条件。

本章还介绍了数据库管理与维护中很重要的工作：备份和恢复数据库。数据库系统故障不可避免，常见的故障包括事务内部故障、系统故障、介质故障和计算机病毒。发生故障后需要进行处理，确保数据库中的数据恢复到出故障之前。数据备份通常有四种方式：完全备份、差分备份、事务日志备份、文件或文件组备份。完全备份是备份整个数据库，不仅包括用户表、系统表、索引、视图、存储过程等所有数据库对象，还包括事务日志部分。差分备份仅备份自上次完全备份以来对数据进行改变的内容。事务日志备份对事务日志进行备

份，备份时复制自上次备份以来对数据库所做的改变，仅需要很少的时间。文件或文件组备份是备份磁盘上跟数据库相关的文件。数据库的备份地点可以是磁盘，也可以是磁带。在备份数据库时可以将数据库备份到备份设备上，也可以直接备份在磁盘文件上。数据库的恢复需要根据故障的类型选择合适的恢复策略和方法。在 MySQL 中可以通过 mysqldump 和 mysql 语句、SELECT…INTO OUTFILE 和 LOAD DATA 语句实现数据库的备份和恢复管理。

习题

一、单项选择题

1. MySQL 的 GRANT 和 REVOKE 语句主要用来维护数据库的(　　)。

A. 可靠性　　B. 一致性　　C. 安全性　　D. 完整性

2. 数据库的(　　)是指数据的正确性和相容性。

A. 并发控制　　B. 完整性　　C. 安全性　　D. 恢复

3. 一个事务执行过程中，其正在访问的数据被其他事务修改，导致处理结果不一致，这是由于违背了事务(　　)特性引起的。

A. 一致性　　B. 原子性　　C. 隔离性　　D. 持久性

4. 如果事务 T 对数据 R 已加 S 锁，则对数据 R(　　)。

A. 不能加 S 锁可加 X 锁　　B. 可加 S 锁不能加 X 锁

C. 可加 S 和 X 锁　　D. 不能加任何锁

5. 数据库中的封锁机制是(　　)的主要方法。

A. 完整性　　B. 并发控制　　C. 安全性　　D. 恢复

6. (　　)可以防止丢失修改，读“脏”数据和不可重复读。

A. 一级封锁协议　　B. 二级封锁协议

C. 三级封锁协议　　D. 两段锁协议

7. 如果对并发操作不加以控制，则可能会带来(　　)问题。

A. 死锁　　B. 死机　　C. 不安全　　D. 不一致

8. 四种数据库备份类型中，(　　)是指将从最近一次完全备份结束以来所有改变的数据备份到数据库。

A. 完全备份　　B. 差分备份

C. 事务日志备份　　D. 文件或文件组备份

9. 下面的 SQL 语句中，用于实现数据控制命令的是(　　)。

A. COMMIT　　B. UPDATE

C. GRANT　　D. SELECT

10. 在数据库系统中，定义用户可以对哪些数据对象进行的操作被称为(　　)。

A. 授权　　B. 视图　　C. 审计　　D. 鉴别

11. 下面关于 MySQL 数据库对象的操作权限的描述正确的是(　　)。

A. 操作权限有 INSERT、DELETE、UPDATE 三种

B. 操作权限只能用于数据表对象，不能用于视图

C. 可以使用 REVOKE 语句收回权限

D. 使用 COMMIT 语句授予权限

12. 在 MySQL 中,存储用户全局权限的表是(　　)。

A. db　　B. user　　C. tables_priv　　D. procs_priv

13. 在 MySQL 中,存储用户列级权限的表是(　　)。

A. columns_priv　　B. user　　C. tables_priv　　D. procs_priv

14. 在 MySQL 中,存储用户表级权限的表是(　　)。

A. columns_priv　　B. user　　C. tables_priv　　D. procs_priv

15. 在 MySQL 中,存储用户数据库级权限的表是(　　)。

A. columns_priv　　B. user　　C. db　　D. procs_priv

16. (　　)可以防止一个用户的工作不适当地影响另一个用户的工作。

A. 完整性控制　　B. 并发控制　　C. 安全性控制　　D. 访问控制

17. 下列不属于并发操作带来的问题是(　　)。

A. 不可重复读　　B. 读"脏"数据　　C. 死锁　　D. 丢失修改

18. 数据库管理系统普遍采用(　　)方法来保证调度的正确性。

A. 授权　　B. 封锁　　C. 索引　　D. 日志

19. 如果事务 T 获得了对数据 D 上的排他锁,则 T 对 D(　　)。

A. 既能读又能写　　B. 只能读不能写

C. 只能写不能读　　D. 不能读也不能写

20. 如果有两个事务,同时对数据库中同一数据进行操作,不会引起冲突的操作是(　　)。

A. 两个都是 UPDATE　　B. 一个是 SELECT,一个是 DELETE

C. 两个都是 SELECT　　D. 一个是 DELETE,一个是 SELECT

21. 假设事务 T1 和 T2 对数据库中的数据 D 进行操作,可能有如下几种情况,(　　)操作不会发生冲突。

A. T1 正在写 D,T2 也要写 D　　B. T1 正在写 D,T2 要读 D

C. T1 正在读 D,T2 要写 D　　D. T1 正在读 D,T2 也要读 D

22. 修改用户名的语句是(　　)。

A. CREATE　　B. RENAME　　C. REVOKE　　D. INSERT

23. 四种数据库备份类型中,(　　)是备份制作数据库中所有内容的一个副本。

A. 完全备份　　B. 差分备份

C. 事务日志备份　　D. 文件或文件组备份

24. 四种数据库备份类型中,(　　)是指将从最近一次日志备份以来所有的事务日志备份到备份设备中。

A. 完全备份　　B. 差分备份

C. 事务日志备份　　D. 文件或文件组备份

25. 四种数据库备份类型中,(　　)是对数据库中的部分文件或文件组进行备份。

A. 完全备份　　B. 差分备份

C. 事务日志备份　　D. 文件或文件组备份

26. 在使用 DROP USER 语句删除用户时，未明确指出用户的主机名，则该用户的主机名默认为是(　　)。

A. localhost　　B. %　　C. root　　D. super

27. 在 MySQL 中，使用 GRANT 语句给 MySQL 用户授权时，用于指定权限授予对象的关键字是(　　)。

A. ON　　B. WITH　　C. FROM　　D. TO

28. 在 MySQL 中，使用 CREATE USER 创建用户时，设置密码的子句是(　　)。

A. IDENTIFIED BY　　B. IDENTIFIED WITH

C. PASSWORD　　D. PASSWORD BY

29. 用户刚创建后，只能登录数据库，无法执行数据库操作的原因是(　　)。

A. 用户还需要修改密码　　B. 用户尚未激活

C. 用户还没有操作数据库的权限　　D. 以上皆有可能

30. 新创建一个 MySQL 用户，还未授权，则该用户可执行的操作是(　　)。

A. SELECT　　B. INSERT

C. UPDATE　　D. 登录 MySQL 服务器

31. 在 MySQL 中，备份数据库的语句是(　　)。

A. mysqldump　　B. backup　　C. copy　　D. mysql

32. 在 MySQL 中，还原数据库的语句是(　　)。

A. mysqldump　　B. backup　　C. return　　D. mysql

33. 实现批量数据导入的语句是(　　)。

A. mysqldump　　B. backup　　C. return　　D. mysql

34. 导出表数据的语句是(　　)。

A. mysqldump　　B. LOAD DATA INFILE

C. mysql　　D. SELECT…INTO OUTFILE

35. 可用于备份表、备份数据库和备份整个数据库系统的语句是(　　)。

A. mysqldump　　B. LOAD DATA INFILE

C. mysql　　D. SELECT…INTO OUTFILE

二、编程题

1. 假设当前系统中不存在用户 test_user，请使用 SQL 语句创建这个用户，设置其登录密码为 test_pwd。

2. 假设当前系统中不存在角色 test_role，请使用 SQL 语句创建这个角色，为这个角色授予查询 SuperMarket 所有表的权限。

3. 授予 test_user 用户 test_role 角色。

4. 收回 test_user 用户的 test_role 角色。

5. 删除 test_role 角色和 test_user 用户。

6. 假设当前系统中不存在用户 sql_user1、用户 sql_user2，请用 SQL 语句创建用户，并授权。授予用户 sql_user1 查询 SuperMarket 中 Category 表的权限；授予 sql_user2 用户修改 SuperMarket 中 Goods 表的 SalePrice 列的权限。

7. 收回用户 sql_user2 在 SuperMarket 中修改 Goods 表的 SalePrice 列的权限。

8. 创建一个事务,将所有啤酒类商品的售价增加 2 元,将所有毛巾类的售价降低 1 元,并提交。

9. 分别实现对数据库 SuperMarket 的备份和恢复操作。

10. 备份数据库 SuperMarket 中的 Student 表和 Goods 表。

三、简答题

1. MySQL 采用哪些措施实现数据库的安全性管理?
2. MySQL 权限表存放在哪个数据库中? 有哪些权限表?
3. MySQL 可以授予的权限有哪几种?
4. 什么是事务? 事务有哪些特征?
5. 简述数据库中进行并发控制的原因。
6. 简述锁的机制及锁的类型,各类锁之间的兼容性。
7. 简述死锁及其解决办法。
8. 第一次对数据库进行备份时,必须使用哪种备份方式?
9. 差分备份方式备份的是哪段时间的哪些内容?
10. 什么是数据备份? 数据备份的类型有哪些?
11. MySQL 数据库管理系统常用的备份数据的方法有哪些?
12. 简述进行数据库备份时,应备份哪些内容?
13. MySQL 数据库管理系统常用的恢复数据的方法有哪些?
14. 什么是活锁? 简述活锁产生的原因和解决方法。
15. 什么样的并发调度是正确的调度?
16. 根据不同的故障,给出对应的恢复策略和方法。
17. MySQL 有哪几种锁的级别? 请简述各级锁的特点。
18. 简述两段锁协议的概念。
19. 并发操作可能产生哪几类数据不一致? 用什么方法能避免各种不一致的情况。
20. 使用封锁技术进行并发操作的控制会带来什么问题? 如何解决?

实验

一、实验目的

(1) 熟悉和掌握数据库安全性管理的方法。

(2) 熟悉和掌握数据库备份和恢复的方法。

二、实验平台

操作系统: Windows XP/7/8/10。

数据库管理系统: MySQL 8.0。

图形化管理工具: Navicat Premium 15。

三、实验内容

在超市管理数据库 SuperMarket 的基础上进行实验。

1. 安全性管理

(1) 创建用户 stu1,密码为 pwd1; 创建用户 stu2,密码为 pwd2; 创建用户 stu3,密码为

pwd3。

(2) 查看 MySQL 下所有的用户。

(3) 修改用户 stu1 的密码为 123456。

(4) 授予用户 stu1 查询 Goods 表的权限,修改进价、售价列的权限。

(5) 授予用户 stu2 创建表、删除表、查询数据、插入数据的权限。

(6) 授予用户 stu3 对数据库执行所有数据库操作的权限,并允许授予其他用户。

(7) 收回用户 stu1 修改进价、售价列的权限。

(8) 创建角色 role1,并授予查询 Student 表的权限。

(9) 授予 stu1 用户 role1 的角色,并查看 stu1 的权限信息。

(10) 删除 role1 角色,并查看 stu1 的权限信息。

2. 备份与恢复

(1) 备份数据库 SuperMarket 中的 SaleBill 表中的数据,要求字段值如果是字符就用双引号标注,字段值之间用逗号隔开,每行用分号结束。

(2) 使用 mysqldump 备份数据库 SuperMarket 的 Goods 表和 SaleBill 表到 D 盘 db_bak 目录下。

(3) 备份数据库 SuperMarket 到 D 盘 db_bak 下。

(4) 备份 MySQL 服务器上的所有数据库到 D 盘 db_bak 下。

(5) 删除数据库 SuperMarket 中 SaleBill 表的数据后,将(1)中的备份文件导入 SaleBill 中。

(6) 删除数据库 SuperMarket 的所有表,并使用(3)的备份文件将其恢复。