MySQL数据库

主讲： 郭鑫

北京动力节点教育科技有限公司
动力节点课程讲义
DONGLIJIEDIANKECHENGJIANGYI
www.bjpowernode.com

第15章 存储引擎

15.1. 存储引擎是：MySQL特有的，其它数据库没有

15.2. 存储引擎的本质

- 通过采用不同的技术将数据存储在文件或内存中；
- 每一种技术都有不同的存储机制，不同的存储机制提供不同的功能和能力；
- 通过选择不同的技术，可以获得额外的速度或功能，改善我们的应用；

15.3. MySQL支持很多种，查看存储引擎，命令如下

15.3.1. show engines\G

```
*************************** 1. row ***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
  Savepoints: NULL
*************************** 2. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 9. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
9 rows in set (0.00 sec)
```

1） 在创建表时，可使用< **ENGINES**> 选项为CREATE TABLE语句显示指定存储引擎

a） 例如：

```
CREATE TABLE table_name(
        NO INT
)ENGINE = MyISAM;
```

2） 如果创建表时没有指定存储引擎，则使用当前默认的存储引擎；

3） 默认的存储引擎可在 my.ini 配置文件中使用 **default-storage-engin** 选项指定；

4） 修改表的存储引擎使用：**ALTER TABLE** 表名 **ENGINE** = 存储引擎名称;

5） 查看表使用的存储引擎，命令如下：

a） **SHOW CREATE TABLE** emp\G;

b） **SHOW TABLE STATUS LIKE** 'emp'\G;

15.4. 常用的存储引擎

15.4.1. MyISAM存储引擎

- MyISAM引擎是MySQL数据库最常用的；
- 它管理的表具有以下特性：
- 使用三个文件表示每个表：
  a) 格式文件 — 存储表的结构（mytable.frm）
  b) 数据文件 — 存储表的数据（mytable.MYD）
  c) 索引文件 — 存储表的索引（mytable.MYI）
- 可转换为压缩、只读表来节省空间

15.4.2. InnoDB存储引擎

- InnoDB存储引擎是MySQL数据库的缺省引擎；
- 它管理的表具体有以下特征：
  a) 每个InnoDB表在数据库目录中以.frm格式文件表示
  b) InnoDB表空间tablespace被用于存储表的内容
  c) 提供一组用来记录事务性活动的日志文件
  d) 用COMMIT（提交）、SAVEPOINT及ROLLBACK（回滚）支持**事务处理**
  e) 提供**全部ACID兼容**

f) 在MySQL服务器崩溃后提供自动恢复

g) 多版本（MVCC）和行级锁定

h) **支持外键及引用的完整性，包括级联更新和删除**

15.4.3. MEMORY存储引擎

- 使用MEMORY存储引擎的表，因为数据存储在内存中，且行的长度固定，所以使得MEMORY存储引擎非常快；
- MEMORY存储引擎管理的表具有下列特征：
  a) 在数据库目录内，每个表均以.frm格式文件表示；
  b) 表数据及索引被存储在内存中；
  c) 表级锁机制；
  d) 字段属性不能包含TEXT或BLOB字段；
- MEMORY存储引擎以前被称为HEAP引擎；

15.5. 选择合适的存储引擎

- MyISAM表最适合于**大量的数据读而少量数据更新**的混合操作。MyISAM表的另一种适用情形是使用**压缩的只读表**。
- 如果查询中包含较多的数据更新操作，应使用InnoDB。其行级锁机制和多版本的支持为数据读取和更新的混合提供了良好的并发机制。
- 使用MEMORY存储引擎存储非永久需要的数据，或者是能够从基于磁盘的表中重新生成的数据。

第16章 事务Transaction

16.1. 事务对应的英文单词：Transaction

16.2. 事务是什么(5点)：

- 一个最小的不可再分的工作单元；
- 通常一个事务对应一个完整的业务；（如：银行转账业务）
- 而一个完整的业务需要批量的DML（insert、update、delete）语句共同完成；
- 事务只和DML语句有关系，或者说只有DML语句才有事务；
- 以上所描述的批量DML语句共有多少DML语句，这个和业务逻辑有关系，业务逻辑不同DML语句个数不同 ；

16.2.1. 关于银行转账业务

16.2.1.1. 银行转账业务：是一个完整的业务，最小的单元，不可再分，也就是说银行转账业务是一个完整的事务。

16.2.1.2. 示例：账户转账

1） t_act 账户表：actno,balance

actno	balance
act-001	50000.0
act-002	10000.0

2） act-001转10000.0给act-002，操作如下：

```
update t_act  set  balance = 40000.0  where  actno = 'act-001';
update t_act  set  balance = 20000.0  where  actno = 'act-002';
```

16.2.1.3. 分析：

1） 以上两条DML语句必须同时成功或者同时失败，因为它为最小业务单元，不可拆分；

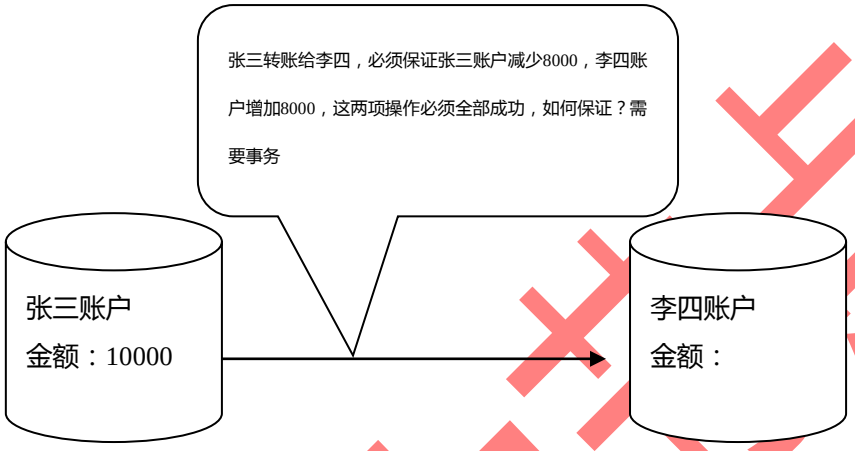
2） 当第一条DML语句执行成功之后，并不能将底层数据库中的第一个账户的数据修改，只是将操作记录了一下，这个记录是在内存中完成的

3） 当第二条DML语句执行成功之后，和底层数据库文件中的数据完成同步。

4） 若第二条DML语句执行失败，将清空所有的历史操作记录。

结论：要完成以上功能，必须借助**事务** transaction。

16.3. 概述



事务可以保证多个操作原子性，要么全成功，要么全失败。对于数据库来说，事务保证批量的DML要么全成功，要么全失败。

16.3.1. 事务具有四个特征ACID：

1） 原子性（Atomicity）

a） 事务是**最小单元**，**不可再分**；

2） 一致性（Consistency）

a） 事务要求所有的DML语句操作的时候，必须保证**同时成功**或**同时失败**；

3） 隔离性（Isolation）

a） 一个事务**不会影响其它事务**的运行；

4） 持久性（Durability）

a） 在事务完成之后，该事务对数据库所作的更改将**持久地保存在数据库**中，并不会被回滚；

16.3.2. 事务中的一些概念

1） 开启事务：**start transaction**

2）结束事务：**end transaction**

3）提交事务：**commit transaction**

4）回滚事务：**rollback transaction**

16.3.3. 和事务有关的两条SQL语句【TCL】

1）COMMIT； 提交

2）ROLLBACK； 回滚

16.3.4. 事务开启和结束的标志是什么？

16.3.4.1. 开启的标志

1）任何一条DML语句执行，标志事务的开启；

16.3.4.2. 结束的标志

1）提交（commit）或者回滚（rollback）

a）提交：成功的结束，将所有的DML语句**操作记录**和**底层**硬盘**文件中数据**进行一次**同步**；

b）回滚：失败的结束，将所有DML语句**操作记录全部清空**；

16.3.5. 重点

1）在事务进行过程中，未结束之前，DML语句是不会修改底层数据库文件中的数据。

2）只是将历史操作记录一下，在内存中完成记录。

3）只有在事务结束的，而且是成功结束的时候才会修改底层硬盘文件中的数据。

16.3.6. MySQL事务的提交和回滚的演示

16.3.6.1. MySQL默认事务：自动提交 show variables like '%commit%';

1）在MySQL数据库管理系统中，默认情况下，事务是自动提交的；也就是说，只要执行一条DML语句，就开启了事务，并且提交了事务；

16.3.6.2. 第一种：关闭MySQL事务自动提交

1） 事务成功用法：start transaction；commit;

第一步：start transaction; 手动开启事务

第二步：DML语句…. 执行批量DML语句 .

第三步：commit; 手动提交事务【事务成功结束】

a） 演示例子：在t_user表中插入数据：事务成功提交

准备：窗口1—创建 t_user表：id为自增主键，name varchar(32)

```
drop table if exists t_user;
create table t_user(
        id int(10) primary key auto_increment,
        name varchar(32)
);

insert into t_user(name) values('jack');
insert into t_user(name) values('lucy');
insert into t_user(name) values('lily');
```

第一步：窗口1，查询原表数据

```
DOS窗口1第一步：查询原表数据
mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
+----+----------+
5 rows in set (0.00 sec)
```

第二步：窗口1，手动开启事务 start transaction，插入数据，查询数据

```
DOS窗口1第二步：开启事务，插入数据，查询数据
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_user(name) values('wangwu');
mysql> insert into t_user(name) values('zhaoliu');
Query OK, 1 row affected (0.04 sec)

mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
|  7 | wangwu   |
|  8 | zhaoliu  |
+----+----------+
6 rows in set (0.00 sec)
```

开启事务

当前窗口可以查询到插入的数据

第三步：窗口2，查询 t_user表数据

```
DOS窗口2第一步：查询t_user表数据
mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
+----+----------+
5 rows in set (0.00 sec)
```

窗口2查询不到窗口1插入的2条数据

第四步：窗口1，手动提交事务commit

```
DOS窗口1第三步：手动提交事务commit
mysql> commit;  手动提交事务
Query OK, 0 rows affected (0.06 sec)

mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
|  7 | wangwu   |
|  8 | zhaoliu  |
+----+----------+
7 rows in set (0.00 sec)
```

第五步：窗口2，查询 t_user 表

```
DOS窗口2第二步：查询t_user表
mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
|  7 | wangwu   |
|  8 | zhaoliu  |
+----+----------+
7 rows in set (0.00 sec)
```

窗口1提交之后可以查询到之前插入的数据

2） 回滚提交用法：start transaction; rollback;

第一步：start transaction;　　手动开启事务

第二步：DML语句…..　　批量DML语句

第三步：rollback;　　手动回滚事务【事务失败结束】

a） 演示例子：在 t_user表中插入数据，事务失败提交

准备：窗口1—创建 t_user表：id为自增主键，name varchar(32)

```
drop table if exists t_user;
create table t_user(
        id int(10) primary key auto_increment,
        name varchar(32)
);

insert into t_user(name) values('jack');
insert into t_user(name) values('lucy');
insert into t_user(name) values('lily');
```

第一步：窗口1，查询 t_user 表数据

```
窗口1第一步：查询t_user表中数据
mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
|  7 | wangwu   |
|  8 | zhaoliu  |
+----+----------+
7 rows in set (0.00 sec)
```

第二步：窗口1，开启事务start transaction，并向t_user表插入数据

窗口1第二步：开启事务，并向t_user表中插入数据

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_user(name) values('wangpeng');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_user(name) values('lijing');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_user(name) values('liubei');
Query OK, 1 row affected (0.00 sec)

mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
|  7 | wangwu   |
|  8 | zhaoliu  |
|  9 | wangpeng |
| 10 | lijing   |
| 11 | liubei   |
+----+----------+
```

开启事务

向表中插入三条数据

第三步：窗口2，查询 t_user表数据

```
窗口2第一步：查询t_user表数据
mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
|  7 | wangwu   |
|  8 | zhaoliu  |
+----+----------+
7 rows in set (0.00 sec)
```

窗口2中查询不到窗口1插入有三条数据

第四步：窗口1，rollback 回滚事务，并查询t_user表数据

```
窗口1第三步：回滚事务，并查询t_user表数据
mysql> rollback;
Query OK, 0 rows affected (0.10 sec)

mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
|  7 | wangwu   |
|  8 | zhaoliu  |
+----+----------+
7 rows in set (0.00 sec)
```

事务回滚

回滚之后之前插入的数据查询不到

第五步：窗口2，查询t_user表数据

窗口2第二步：查询t_user表数据

```
mysql> select * from t_user;
+----+----------+
| id | name     |
+----+----------+
|  1 | jack     |
|  2 | jack     |
|  4 | jack     |
|  5 | zhangsan |
|  6 | lisi     |
|  7 | wangwu   |
|  8 | zhaoliu  |
+----+----------+
7 rows in set (0.00 sec)
```

窗口1事务回滚之后，窗口2中查询不到数据

16.3.6.3. 第二种：关闭MySQL事务自动提交：只对当前会话有效

1）两种关闭自动提交事务

a）set autocommit = off

b）set session autocommit = off

2）两种打开自动提交事务

a）set autocommit = on

b）set session autocommit = on

注：以上打开或关闭事务只对当前窗口有效；

3）查询事务状态： show variables like '%commit%';

```
mysql> show variables like '%commit%';
+--------------------------------+-------+
| Variable_name                  | Value |
+--------------------------------+-------+
| autocommit                     | ON    |
| innodb_commit_concurrency      | 0     |
| innodb_flush_log_at_trx_commit | 1     |
+--------------------------------+-------+
```

16.3.7. 事务的隔离级别

16.3.7.1. 事务四个特性ACID之一：隔离性（isolation）

16.3.7.2. 隔离性有四个隔离级别：

1） read uncommitted 读未提交

2） read committed 读已提交

3） repeatable read 可重复读

4） serializable 串行化

16.3.7.3. read committed 读未提交（级别最低）

1） 事务A和和事务B，事务A未提交的数据，事务B可以读取

2） 这里读取到的数据可以叫做“脏数据”或“脏读 Dirty Read”

3） 读未提交隔离级别最低，这种级别一般只在理论上存在，数据库默认隔离级别一般都高于该隔离级别；

16.3.7.4. read committed 读已提交

1） 事务A和事务B，事务A提交的数据，事务B才可读取到；

2） 该隔离级别高于“读未提交”级别

3） 换句话说：对方事务提交之后的数据，当前事务才可读取到。

4） 该隔离级别可以避免脏数据；

5） 该隔离级别能够导致“不可重复读取”

6） **Oracle数据库管理系统默认隔离级别为“可重复读”**

16.3.7.5. repeatable read 可重复读

1） 事务A和事务B，事务A提交之后的数据，事务B读取不到

2） 事务B是可重复读到数据的

3） 这种隔离级别高于“读已提交”

4）换句话说，对方提交之后的数据，还是读取不到

5）这种隔离级别可以避免“脏读和不可重复读”，达到“重复读取”；

6）**MySQL数据库管理系统默认隔离级别为：可重复读**

7）虽然可以达到“可重复读”效果，但是会导致“幻象读”

16.3.7.6. serializable 串行化

1）事务A和事务B，事务A在操作数据库表中数据的时候，事务B只能**排队等待**；

2）这种事务隔离级别一般很少使用，**吞吐量太低**，**用户体验不好**；

3）这种隔离级别可以避免“幻象读”，每一次读取都是数据库表中真实的记录；

4）事务A和事务B不再并发；

16.3.7.7. 查看隔离级别

1）查看当前会话级隔离级别

```
select @@tx_isolation;
select @@session.tx_isolation;
```

```
select @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
或
select @@session.tx_isolation;
+------------------------+
| @@session.tx_isolation |
+------------------------+
| REPEATABLE-READ        |
+------------------------+
```

2）查看当前全局隔离级别：@@global.tx_isolation;

```
select @@global.tx_isolation;
```

```
select @@global.tx_isolation;
+-----------------------+
| @@global.tx_isolation |
+-----------------------+
| REPEATABLE-READ       |
+-----------------------+
```

16.3.7.8. 设置服务器缺省隔离级别

1） 第一种：修改my.ini配置文件

在 my.ini 文件中的[mysqld]下面添加：

----------my.ini----------

[mysqld]

transaction-isolation = READ-COMMITTED

----------my.ini----------

a） 隔离级别可选项为：

- READ-UNCOMMITTED
- READ-COMMITTED
- REPEATABLE-READ
- SERIALIZABLE

2） 第二种：通过命令方式设置事务隔离级别

a） SET TRANSACTION ISOLATION LEVEL isolation-level;

b） isolation-level 可选值：

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

16.3.7.9. 设置隔离级别作用的范围

1） 事务隔离级别的作用范围分为两种：会话级、全局级

a） 会话级（session）：只对当前会话有效

b） 全局级（global）：对所有会话有效

2） 使用方法如下

a ） 会话级：

SET TRANSACTION ISOLATION LEVEL ISOLATION-LEVEL;
SET SESSION TRANSACTION ISOLATION LEVEL ISOLATION-LEVEL;

b ） 全局级

SET GLOBAL TRANSACTION ISOLATION LEVEL<ISOLATION-LEVEL>;

16.3.7.10. 隔离级别与一致性问题的关系

隔离级别	脏读	不可重复读	幻象读
读未提交	可能	可能	可能
读已提交	不可能	可能	可能
可重复读	不可能	不可能	可能
串行化	不可能	不可能	不可能

16.3.7.11. 并发事务与隔离级别示例

1 ） 读未提交 READ UNCOMMITTED

会话一	会话二
set global transaction isolation level read uncommitted;	
退出DOS窗口	
use bjpowernode; Drop table if exists t_user; create table t_user(no int(3) primary key auto_increment, name varchar(32));	
	use bjpowernode;
start transaction;	
	start transaction;
insert into t_user(name) values('king');	
	select * from t_user;

2） 读已提交 READ COMMITTED

会话一	会话二
set global transaction isolation level read committed;	
退出DOS窗口	
use bjpownode; Drop table if exists t_user; create table t_user(no int(3) primary key auto_increment, name varchar(32));	
	use bjpownode;
start transaction;	
	start transaction;
insert into t_user(name) values('allen');	
	select * from t_user;
commit;	
	select * from t_user;

3） 可重复读 REPEATABLE READ

会话一	会话二
set global transaction isolation level repeatable read;	
退出DOS窗口	
use bjpownode; Drop table if exists t_user; create table t_user(no int(3) primary key auto_increment, name varchar(32));	
	use bjpownode;
start transaction;	
	start transaction;
select * from t_user;	

	select * from t_user;
insert into t_user(name) values('allen'); commit;	
	select * from t_user;

4） 串行化 SERIALIZABLE

会话一	会话二
use bjpowernode;	

Drop table if exists t_user;
create table t_user(
no int(3) primary key auto_increment,
name varchar(32)
);
insert into t_user(name) values('zhangsan');
insert into t_user(name) values('lisi');	

set global transaction isolation level serializable;	
退出DOS窗口	
use bjpownode;	
	use bjpownode;
start transaction;	
	start transaction;
select * from t_user;	
	select * from t_user;
insert into t_user(name) values('allen');	
	select * from t_user;
commit;	

第17章 索引（了解）

17.1.索引原理

17.1.1. 什么是索引

17.1.1.1. 索引对应的英语单词：index

17.1.1.2. 索引作用：

1） 相当于一本字典目录，提高程序的检索 / 查询效率；表中每一个字段都可添加索引

17.1.1.3. **主键自动添加索引**：

1） 能够通过主键查询的尽量通过主键查询，效率较高；

17.1.1.4. 索引和表相同，存储在硬盘文件中

1） 索引和表相同，都是一个对象，表是存储在硬盘文件中的，那么索引也是表的一部分，索引也存储在硬盘文件中；

17.1.1.5. MySQL数据库中表的检索方式有2种：

1） 第一种：全表扫描（效率较低）

a） 举例：查询ename='KING'

假设有一张表：emp 员工表，select * from emp where ename = 'KING';

若ename没有添加索引，那么通过ename过滤数据的时候，ename字段会全表扫描；

假设有一张表：dept 部门表，select * from dept where dname = 'ACCOUNTING'；

若dname没有添加索引，那么通过dname过滤数据的时候，dname字段会全表扫描；

2） 第二种：通过索引检索（提高查询效率）

17.1.2. **什么情况下适合给表中字段添加索引

1） 该字段数据量庞大

2） 该字段很少的DML操作（由于索引也需要维护，DML操作多的话，也影响检索效率）

3） 该字段经常出现在where条件中

注：实际开发中会根据项目需求或客户需求等综合因素来做调整

17.1.3. 索引的应用

17.1.3.1. 创建索引

1） 语法结构：

a） create index 索引名 on 表名（列名）

b） create unique index 索引名 on 表名（列名）

注：添加unique表示在该表中的该列添加一个唯一性约束

示例：create index dept_dname_index on dept(dname);

17.1.3.2. 查看索引

1） 语法结构：

a） show index from 表名

示例：show index from dept;

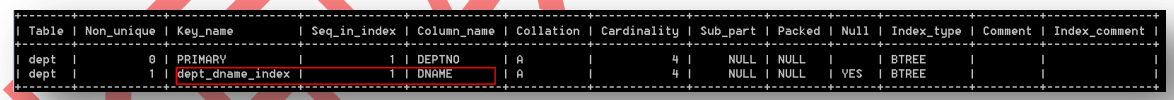


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
dept	0	PRIMARY	1	DEPTNO	A	4	NULL	NULL		BTREE		
dept	1	dept_dname_index	1	DNAME	A	4	NULL	NULL	YES	BTREE		

17.1.3.3. 删除索引

1） 语法结构：

a） drop index 索引名 on 表名;

示例：drop index dept_dname_index on dept;

第18章 视图

18.1.什么是视图

1） 视图对应英文单词：view

2） 视图在数据库管理系统中也是一个对象，也是以文件形式存在的

3） 视图也对应了一个查询结果，只是从不同的角度查看数据

18.2.创建视图

18.2.1. 语法结构：create view 视图名称 as 查询语句；

18.2.2. 例子：从emp表查询empno,ename,sal，结果当作视图展示

```
将查询结果当作一个视图创建出来
create view myview as select empno,ename,sal from emp;

通过视图对象看数据
mysql> select * from myview;
+-------+--------+---------+
| empno | ename  | sal     |
+-------+--------+---------+
|  7369 | SMITH  |  800.00 |
|  7499 | ALLEN  | 1600.00 |
|  7521 | WARD   | 1250.00 |
|  7566 | JONES  | 2975.00 |
|  7654 | MARTIN | 1250.00 |
|  7698 | BLAKE  | 2850.00 |
|  7782 | CLARK  | 2450.00 |
|  7788 | SCOTT  | 3000.00 |
|  7839 | KING   | 5000.00 |
|  7844 | TURNER | 1500.00 |
|  7876 | ADAMS  | 1100.00 |
|  7900 | JAMES  |  950.00 |
|  7902 | FORD   | 3000.00 |
|  7934 | MILLER | 1300.00 |
+-------+--------+---------+
```

18.3.删除视图

18.3.1. 语法结构：drop view if exists 视图名称;

示例：删除myview视图：drop view if exists myview;

```
mysql> drop view if exists myview;
Query OK, 0 rows affected (0.00 sec)
```

18.4.修改视图

18.4.1. 语法结构：alter view 视图名称 as 查询语句

示例：将myview视图修改为deptno,ename,job,sal

```
mysql> alter view myview as select deptno,ename,job,sal from emp;
Query OK, 0 rows affected (0.06 sec)
```

18.5.视图作用

18.5.1. 面向视图查询，可以提高查询效率

18.5.1.1. 例如

1）单独执行：select e.ename,d.dname from emp e join dept d on e.deptno = d.deptno;

2）将上面执行结果当作视图对象创建

create view myview as select e.ename,d.dname from emp e join dept d on e.deptno = d.deptno;

3）面向视图查询：没有进行表连接，提高检索效率

select * from myview;

【非常重要】**隐藏表的实现细节**

1、 create view myview as select empno a,ename b,sal c from emp;

2、面向视图查询，只知道myview视图中有a,b,c三个字段，不知道该视图背后真实表的结构是什么

第19章 DBA命令

19.1.新建用户

19.1.1. CREATE USER username IDENTIFIED BY 'password';

username：你将创建的用户名,

password：该用户的登陆密码,密码可以为空,如果为空则该用户可以不需要密码登陆服务器.

示例：

create user p361 identified by '123';

--可以登录但是只可以看见一个库 information_schema

19.2.授权

19.2.1. 命令详解

示例：grant all privileges on dbname.tbname to 'username'@'login ip' identified by 'password' with grant option;

1) dbname=*表示所有数据库
2) tbname=*表示所有表
3) login ip=%表示任何ip
4) password为空，表示不需要密码即可登录
5) with grant option; 表示该用户还可以授权给其他用户

细粒度授权

1、首先以root用户进入mysql

2、然后键入命令：grant select,insert,update,delete on *.* to p361 @localhost Identified by "123";

3、如果希望该用户能够在任何机器上登陆mysql，则将localhost改为 "%" 。

粗粒度授权

我们测试用户一般使用该命令授权：

GRANT ALL PRIVILEGES ON *.* TO 'p361'@'%' Identified by "123";

注意：用以上命令授权的用户不能给其它用户授权,如果想让该用户可以授权,用以下命令:

GRANT ALL PRIVILEGES ON *.* TO 'p361'@'%' Identified by "123" WITH GRANT OPTION;

用户权限privileges包括：

1） alter：修改数据库的表

2） create：创建新的数据库或表

3） delete：删除表数据

4） drop：删除数据库/表

5） index：创建/删除索引

6） insert：添加表数据

7） select：查询表数据

8） update：更新表数据

9） all：允许任何操作

10） usage：只允许登录

19.3. 回收授权

19.3.1. 命令详解

revoke privileges on dbname[.tbname] from username;

revoke all privileges on *.* from p361;

use mysql

select * from user

进入 mysql库中

修改密码;

update user set password = password('qwe') where user = 'p646';

刷新权限;

flush privileges

19.4.导入导出

19.4.1. 导出：mysqldump

导出整个数据库

C: \Administrator> mysqldump bjpowernode>D:\bjpowernode.sql -uroot -p123

导出指定库下的指定表

C: \Administrator>mysqldump bjpowernode emp> D:\ bjpowernode.sql -uroot -p123

19.4.2. 导入：source

登录MySQL数据库管理系统之后执行：

mysql>source D:\ bjpowernode.sql

第20章 数据库设计三范式

定义：设计数据库的时候所依据的规范，共有三个规范；

20.1.第一范式：主键、字段不能再分

20.1.1. 定义：要求有主键，数据库中不能出现重复记录，每一个字段是原子性不能再分；

示例：不符合第一范式

学生编号	学生姓名	联系方式
1001	张三	zs@gmail.com,1359999999
1002	李四	ls@gmail.com,13699999999
1001	王五	ww@163.net,13488888888

分析以上设计存在的问题：

1） 数据存在重复记录，数据不唯一，没有主键

2） 联系方式可以再分，不是原子性

修改以上设计方案：

学生编号(pk)	学生姓名	email	联系电话
1001	张三	zs@gmail.com	1359999999
1002	李四	ls@gmail.com	13699999999
1003	王五	ww@163.net	13488888888

结论：关于第一范式

1、 每一行必须唯一，也就是每个表必须有主键，这是我们数据库设计的最基本要求，

2、 主键主要通常采用数值型或定长字符串表示

3、 关于列不可再分，应根据具体的情况来决定。如联系方式，为了开发上的便利可能就采用一个字段了；

20.2.第二范式：非主键字段完全依赖主键

20.2.1. 定义：第二范式是建立在第一范式基础之上，要求数据库中所有非主键字段完全依赖主键，不能产生部分依赖；（严格意义上说：尽量不要使用联合主键）

示例一：数据仍然可能重复

学生编号	学生姓名	教师编号	教师姓名
1001	张三	001	王老师
1002	李四	002	赵老师
1003	王五	001	王老师
1001	张三	002	赵老师

示例二：确定主键，学生编号、教师编号，出现冗余

学生编号(PK)	教师编号(PK)	学生姓名	教师姓名
1001	001	张三	王老师
1002	002	李四	赵老师
1003	001	王五	王老师
1001	002	张三	赵老师

综合分析：

1、以上虽然确定了主键，但此表会出现大量冗余，主要涉及到的冗余字段为“学生姓名”和“教师姓名”；

2、出现冗余的原因在于：学生姓名部分依赖了主键的一个字段学生编号，而没有依赖教师编号，而教师姓名部分依赖了主键的一个字段教师编号，这就是第二范式部分依赖。

解决方案如下：

学生信息表：

学生编号（PK）	学生姓名
1001	张三

1002	李四
1003	王五

教师信息表：

教师编号（PK）	教师姓名
001	王老师
002	赵老师

教师和学生的关系表：

学生编号(PK) fk→学生表的学生编号	教师编号(PK) fk→教师表的教师编号
1001	001
1002	002
1003	001
1001	002

结论：一种典型的“多对多”的设计

20.3.第三范式

20.3.1. 定义：建立在第二范式基础之上，要求非主键字段不能产生传递依赖于主键字段；

示例一：学生信息表

学生编号（PK）	学生姓名	班级编号	班级名称
1001	张三	01	一年一班
1002	李四	02	一年二班
1003	王五	03	一年三班
1004	六	03	一年三班

综合分析：

1、从表中看出，班级名称字段存在冗余，因为班级名称字段没有直接依赖于主键

2、班级名称字段依赖于班级编号，班级编号依赖于学生编号，那么这就是传递依赖，

解决方案：

1、 将冗余字段单独拿出来建立表

2、 如下表所示：学生信息表中班级编号设为外键FK

学生信息表：

学生编号（PK）	学生姓名	班级编号（FK）
1001	张三	01
1002	李四	02
1003	王五	03
1004	六	03

班级信息表：

班级编号（PK）	班级名称
01	一年一班
02	一年二班
03	一年三班

3、 结论：典型的一对多

以上设计是一种典型的一对多的设计，一存储在一张表中，多存储在一张表中，在多的那张表中添加外键指向一的一方的主键

20.4.三范式总结(几个比较经典的设计)

20.4.1. 一对一

20.4.1.1. 第一种方案：分两张表存储，共享主键

1） 示例：t_husband 和 t_wife两张表

丈夫表 ：t_husband

hno（PK）	hname
1	Zhangsan
2	Wangwu

3	zhaoliu

妻子表：t_wife

wno（PK）【同时也是外键FK，引用t_husband的主键】	wname
3	a
2	b
1	c

20.4.1.2. 第二种方案：分两张表存储，外键唯一

1） 示例：t_husband 和 t_wife两张表

丈夫表 ：t_husband

hno（PK）	hname	Wifeno（FK - unique）
1	Zhangsan	100
2	Wangwu	200
3	zhaoliu	300

妻子表：t_wife

wno（PK）	wname
100	a
200	b
300	c

20.4.2. 一对多

20.4.2.1. 分两张表存储，在多的一方添加外键，这个外键字段引用一的一方中的主键字段

例如：学生信息表和班级信息表，请参考：20.3.2 示例一

20.4.3. 多对多

20.4.3.1. 分三张表存储，在学生表中存储学生信息，在课程表中存储课程信息，在选课表中存储学生选课信息，请参考：20.2

20.5.实际开发中是怎样的？

1）数据库设计尽量遵循三范式

2）根据实际需求进行取舍，有时可能会拿冗余换速度，最终用目的要满足**客户需求**。