MySQL管理

启动及关闭MySQL服务器
启动：mysqld --console
关闭：mysqladmin -uroot shutdown

MySQL连接

命令行连接mysql -u root -p　　输入密码（区分大小写）

MySQL创建/删除数据库

创建： CREATE DATABASE 数据库名;
删除：Drop命令：drop database <数据库名>;

MySQL选择数据库

选取RUNOOB数据库：use RUNOOB;

MySQL数据类型

大致可以分为三类：数值、日期/时间和字符串(字符)类型
数值类型

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 byte	(-128，127)	(0，255)	小整数值
SMALLINT	2 bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 bytes	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 bytes	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 bytes	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对DECIMAL(M,D)，如果M>D，为M+2否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

日期和时间类型

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

字符串类型

类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串
TINYBLOB	0-255 bytes	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

MySQL创建数据表

创建MySQL数据表需要以下信息：
表名
表字段名
定义每个表字段

通用语法：CREATE TABLE table_name (column_name column_type);

实例：

```
CREATE TABLE IF NOT EXISTS `runoob_tbl`(
   `runoob_id` INT UNSIGNED AUTO_INCREMENT,
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` DATE,
   PRIMARY KEY ( `runoob_id` )
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

MySQL删除数据表

通用语法：DROP TABLE table_name ;

MySQL插入数据

通用语法：
```
INSERT INTO table_name ( field1, field2,...fieldN )
                       VALUES
                       ( value1, value2,...valueN );
```

MySQL查询数据

通用语法：
```
SELECT column_name,column_name
FROM table_name
[WHERE Clause]
[LIMIT N][ OFFSET M];
```

说明：
可以使用星号（*）来代替其他字段，SELECT语句会返回表的所有字段数据
可以使用 WHERE 语句来包含任何条件
可以使用 LIMIT 属性来设定返回的记录数
可以通过OFFSET指定SELECT语句开始查询的数据偏移量。默认情况下偏移量为0

MySQL where子句

使用 WHERE 子句从数据表中读取数据的通用语法：
```
SELECT field1, field2,...fieldN FROM table_name1, table_name2...
[WHERE condition1 [AND [OR]] condition2.....
```

说明：
可以使用一个或者多个表，表之间使用逗号, 分割，并使用WHERE语句来设定查询条件。
可以在 WHERE 子句中指定任何条件。
可以使用 AND 或者 OR 指定一个或多个条件。
WHERE 子句也可以运用于 SQL 的 DELETE 或者 UPDATE 命令。
WHERE 子句类似于程序语言中的 if 条件，根据 MySQL 表中的字段值来读取指定的数据。

MySQL UPDATE 更新

通用语法：
```
UPDATE table_name SET field1=new-value1, field2=new-value2
[WHERE Clause];
```

MySQL DELETE 语句

通用语法：
DELETE FROM table_name [WHERE Clause];

说明：
如果没有指定 WHERE 子句，MySQL 表中的所有记录将被删除。
可以在 WHERE 子句中指定任何条件
可以在单个表中一次性删除记录。

MySQL like子句

LIKE 子句中使用百分号 %字符来表示任意字符，类似于UNIX或正则表达式中的星号 *。
如果没有使用百分号 %, LIKE 子句与等号 = 的效果是一样的。

通用语法：
SELECT field1, field2,...fieldN
FROM table_name
WHERE field1 LIKE condition1 [AND [OR]] filed2 = 'somevalue';

实例：我们将 runoob_tbl 表中获取 runoob_author 字段中以 COM 为结尾的的所有记录：

SELECT * from runoob_tbl WHERE runoob_author LIKE '%COM';

MySQL UNION 操作符

用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。多个 SELECT 语句会删除重复的数据。
语法格式：
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
UNION [ALL | DISTINCT]
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions];

参数：
expression1, expression2, ... expression_n: 要检索的列。
tables: 要检索的数据表。
WHERE conditions: 可选， 检索条件。
DISTINCT: 可选，删除结果集中重复的数据。默认情况下 UNION 操作符已经删除了重复数据，所以 DISTINCT 修饰符对结果没啥影响。

ALL: 可选，返回所有结果集，包含重复数据。

MySQL排序

可以使用 MySQL 的 ORDER BY 子句来设定你想按哪个字段哪种方式来进行排序，再返回搜索结果
SELECT field1, field2,...fieldN FROM table_name1, table_name2...
ORDER BY field1 [ASC [DESC][默认 ASC]], [field2...] [ASC [DESC][默认 ASC]];

说明：
可以使用 ASC 或 DESC 关键字来设置查询结果是按升序或降序排列。 默认情况下，它是按升序排列。
你可以添加 WHERE...LIKE 子句来设置条件。

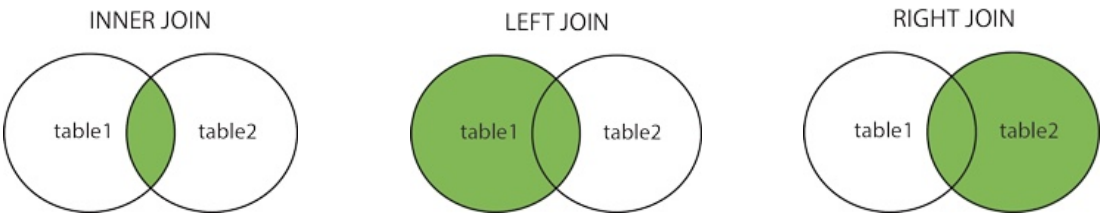
MySQL GROUP BY 语句

GROUP BY 语句根据一个或多个列对结果集进行分组。
在分组的列上我们可以使用 COUNT, SUM, AVG,等函数。
语法：
SELECT column_name, function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;

MySQL连接的使用

使用 MySQL 的 JOIN 在两个或多个表中查询数据。
可以在 SELECT, UPDATE 和 DELETE 语句中使用 Mysql 的 JOIN 来联合多表查询。
JOIN 按照功能大致分为如下三类：
INNER JOIN（内连接,或等值连接）： 获取两个表中字段匹配关系的记录。
LEFT JOIN（左连接）： 获取左表所有记录，即使右表没有对应匹配的记录。
RIGHT JOIN（右连接）： 与 LEFT JOIN 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。

实例：
使用MySQL的INNER JOIN(也可以省略 INNER 使用 JOIN，效果一样)来连接以上两张表来读取runoob_tbl表中所有runoob_author字段在tcount_tbl表对应的runoob_count字段值：
SELECT a.runoob_id, a.runoob_author, b.runoob_count FROM runoob_tbl a INNER JOIN tcount_tbl b ON a.runoob_author = b.runoob_author;



MySQL NULL 值处理

MySQL提供了三大运算符:
IS NULL: 当列的值是 NULL,此运算符返回 true。
IS NOT NULL: 当列的值不为 NULL, 运算符返回 true。
<=>: 比较操作符（不同于 = 运算符），当比较的的两个值相等或者都为 NULL 时返回 true。

MySQL 中处理 NULL 使用 IS NULL 和 IS NOT NULL 运算符。
select * , columnName1+ifnull(columnName2,0) from tableName;
columnName1，columnName2 为 int 型，当 columnName2 中，有值为 null 时，columnName1+columnName2=null， ifnull(columnName2,0) 把 columnName2 中 null 值转为 0。

MySQL正则表达式

MySQL中使用 REGEXP 操作符来进行正则表达式匹配。

模式	描述
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
$	匹配输入字符串的结束位置。如果设置了RegExp 对象的 Multiline 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。
.	匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用像 '[.\n]' 的模式。
[...]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
[^...]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'。
p1|p2|p3	匹配 p1 或 p2 或 p3。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,m}	m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多 m 次。

实例：
查找name字段中以'st'为开头的所有数据：
mysql> SELECT name FROM person_tbl WHERE name REGEXP '^st';

查找name字段中以'ok'为结尾的所有数据：
mysql> SELECT name FROM person_tbl WHERE name REGEXP 'ok$';

查找name字段中包含'mar'字符串的所有数据：
mysql> SELECT name FROM person_tbl WHERE name REGEXP 'mar';

查找name字段中以元音字符开头或以'ok'字符串结尾的所有数据：
mysql> SELECT name FROM person_tbl WHERE name REGEXP '^[aeiou]|ok$';

MySQL事务

在 MySQL 中只有使用了 Innodb 数据库引擎的数据库或表才支持事务。
事务处理可以用来维护数据库的完整性，保证成批的 SQL 语句要么全部执行，要么全部不执行。
事务用来管理 insert,update,delete 语句。

一般来说，事务是必须满足4个条件（ACID）：：原子性（Atomicity，或称不可分割性）、一致性（Consistency）、隔离性（Isolation，又称独立性）、持久性（Durability）。
原子性：一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。
隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。
持久性：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

事务控制语句：
BEGIN 或 START TRANSACTION 显式地开启一个事务；
COMMIT 也可以使用 COMMIT WORK，不过二者是等价的。COMMIT 会提交事务，并使已对数据库进行的所有修改成为永久性的；
ROLLBACK 也可以使用 ROLLBACK WORK，不过二者是等价的。回滚会结束用户的事务，并撤销正在进行的所有未提交的修改；
SAVEPOINT identifier，SAVEPOINT 允许在事务中创建一个保存点，一个事务中可以有多个 SAVEPOINT；
RELEASE SAVEPOINT identifier 删除一个事务的保存点，当没有指定的保存点时，执行该语句会抛出一个异常；
ROLLBACK TO identifier 把事务回滚到标记点；
SET TRANSACTION 用来设置事务的隔离级别。InnoDB 存储引擎提供事务的隔离级别有READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ 和 SERIALIZABLE。

MYSQL 事务处理主要有两种方法：
1、用 BEGIN, ROLLBACK, COMMIT来实现
BEGIN 开始一个事务
ROLLBACK 事务回滚
COMMIT 事务确认
2、直接用 SET 来改变 MySQL 的自动提交模式:
SET AUTOCOMMIT=0 禁止自动提交
SET AUTOCOMMIT=1 开启自动提交

MySQL ALTER命令

修改数据表名或者修改数据表字段。

删除，添加或修改表字段：

使用ALTER 命令及 DROP 子句来删除创建表的 i 字段：
ALTER TABLE testalter_tbl DROP i;

在表 testalter_tbl 中添加 i 字段，并定义数据类型:
ALTER TABLE testalter_tbl ADD i INT;

如果需要指定新增字段的位置，可以使用MySQL提供的关键字 FIRST (设定位第一列)， AFTER 字段名（设定位于某个字段之后）。

修改字段类型及名称：

如果需要修改字段类型及名称, 可以在ALTER命令中使用 MODIFY 或 CHANGE 子句 。
例如，把字段 c 的类型从 CHAR(1) 改为 CHAR(10)，可以执行以下命令:
 ALTER TABLE testalter_tbl MODIFY c CHAR(10);

使用 CHANGE 子句, 语法有很大的不同。 在 CHANGE 关键字之后，紧跟着的是你要修改的字段名，然后指定新字段名及类型。尝试如下实例：
ALTER TABLE testalter_tbl CHANGE i j BIGINT;
ALTER TABLE testalter_tbl CHANGE j j INT;

ALTER TABLE 对 Null 值和默认值的影响：

修改字段时，可以指定是否包含值或者是否设置默认值。
以下实例，指定字段 j 为 NOT NULL 且默认值为100 。
ALTER TABLE testalter_tbl MODIFY j BIGINT NOT NULL DEFAULT 100;

修改字段默认值：

实例：
ALTER TABLE testalter_tbl ALTER i SET DEFAULT 1000;

可以使用 ALTER 命令及 DROP子句来删除字段的默认值：

ALTER TABLE testalter_tbl ALTER i DROP DEFAULT;

修改表名：

如果需要修改数据表的名称，可以在 ALTER TABLE 语句中使用 RENAME 子句来实现。
以下实例将数据表 testalter_tbl 重命名为 alter_tbl：
ALTER TABLE testalter_tbl RENAME TO alter_tbl;

MySQL索引

索引分单列索引和组合索引。单列索引，即一个索引只包含单个列，一个表可以有多个单列索引，但这不是组合索引。组合索引，即一个索引包含多个列。
创建索引时，需要确保该索引是应用在 SQL 查询语句的条件(一般作为 WHERE 子句的条件)。

普通索引

创建索引

这是最基本的索引，它没有任何限制。它有以下几种创建方式：

```
CREATE INDEX indexName ON table_name (column_name)
```

如果是CHAR，VARCHAR类型，length可以小于字段实际长度；如果是BLOB和TEXT类型，必须指定length。

修改表结构(添加索引)

```
ALTER table tableName ADD INDEX indexName(columnName)
```

创建表的时候直接指定

```
CREATE TABLE mytable(
ID INT NOT NULL,
username VARCHAR(16) NOT NULL,
INDEX [indexName] (username(length))
);
```

删除索引的语法

```
DROP INDEX [indexName] ON mytable;
```

唯一索引

它与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。它有以下几种创建方式：

创建索引

```
CREATE UNIQUE INDEX indexName ON mytable(username(length))
```

修改表结构

```
ALTER table mytable ADD UNIQUE [indexName] (username(length))
```

创建表的时候直接指定

```
CREATE TABLE mytable(
ID INT NOT NULL,
username VARCHAR(16) NOT NULL,
UNIQUE [indexName] (username(length))
);
```

使用ALTER 命令添加和删除索引

有四种方式来添加数据表的索引：
ALTER TABLE tbl_name ADD PRIMARY KEY (column_list): 该语句添加一个主键，这意味着索引值必须是唯一的，且不能为NULL。
ALTER TABLE tbl_name ADD UNIQUE index_name (column_list): 这条语句创建索引的值必须是唯一的（除了NULL外，NULL可能会出现多次）。
ALTER TABLE tbl_name ADD INDEX index_name (column_list): 添加普通索引，索引值可出现多次。
ALTER TABLE tbl_name ADD FULLTEXT index_name (column_list):该语句指定了索引为 FULLTEXT ，用于全文索引。
可以在 ALTER 命令中使用 DROP 子句来删除索引。以下实例删除索引:
ALTER TABLE testalter_tbl DROP INDEX c;

使用 ALTER 命令添加和删除主键

主键作用于列上（可以一个列或多个列联合主键），添加主键索引时，你需要确保该主键默认不为空（NOT NULL）。实例如下：
ALTER TABLE testalter_tbl MODIFY i INT NOT NULL;
ALTER TABLE testalter_tbl ADD PRIMARY KEY (i);
也可以使用 ALTER 命令删除主键：
ALTER TABLE testalter_tbl DROP PRIMARY KEY;

显示索引信息

可以使用 SHOW INDEX 命令来列出表中的相关的索引信息。可以通过添加 \G 来格式化输出信息。
实例:
SHOW INDEX FROM table_name; \G

MySQL临时表

临时表在我们需要保存一些临时数据时是非常有用的。临时表只在当前连接可见，当关闭连接时，Mysql会自动删除表并释放所有空间。
CREATE TEMPORARY TABLE SalesSummary ();
默认情况下，当你断开与数据库的连接后，临时表就会自动被销毁。当然你也可以在当前MySQL会话使用 DROP TABLE 命令来手动删除临时表。

MySQL复制表

完整的复制MySQL数据表，步骤如下：
使用 SHOW CREATE TABLE 命令获取创建数据表(CREATE TABLE) 语句，该语句包含了原数据表的结构，索引等。
复制以下命令显示的SQL语句，修改数据表名，并执行SQL语句，通过以上命令 将完全的复制数据表结构。
如果想复制表的内容，可以使用 INSERT INTO ... SELECT 语句来实现。

MySQL元数据

查询结果信息： SELECT, UPDATE 或 DELETE语句影响的记录数。
数据库和数据表的信息： 包含了数据库及数据表的结构信息。
MySQL服务器信息： 包含了数据库服务器的当前状态，版本号等。

获取服务器元数据

命令	描述
SELECT VERSION( )	服务器版本信息
SELECT DATABASE( )	当前数据库名 (或者返回空)
SELECT USER( )	当前用户名
SHOW STATUS	服务器状态
SHOW VARIABLES	服务器配置变量

MySQL序列使用

如果想实现其他字段也实现自动增加，就可以使用MySQL序列来实现。
使用 AUTO_INCREMENT
MySQL 中最简单使用序列的方法就是使用 MySQL AUTO_INCREMENT 来定义序列。

重置序列
如果删除了数据表中的多条记录，并希望对剩下数据的AUTO_INCREMENT列进行重新排列，那么可以通过删除自增的列，然后重新添加来实现。 不过该操作要非常小心，如果在删除的同时又有新记录添加，有可能会出现数据混乱。操作如下所示：

```
ALTER TABLE insect DROP id;
ALTER TABLE insect
ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id);
```

设置序列的开始值
可以在表创建成功后，通过以下语句来实现：

```
ALTER TABLE t AUTO_INCREMENT = 100;
```

MySQL处理重复数据

防止表中出现重复数据
可以在 MySQL 数据表中设置指定的字段为 PRIMARY KEY（主键） 或者 UNIQUE（唯一） 索引来保证数据的唯一性。
如果我们设置了唯一索引，那么在插入重复数据时，SQL 语句将无法执行成功,并抛出错。
INSERT IGNORE INTO 与 INSERT INTO 的区别就是 INSERT IGNORE INTO 会忽略数据库中已经存在的数据，如果数据库没有数据，就插入新的数据，如果有数据的话就跳过这条数据。这样就可以保留数据库

中已经存在数据，达到在间隙中插入数据的目的。
INSERT IGNORE INTO 当插入数据时，在设置了记录的唯一性后，如果插入重复数据，将不返回错误，只以警告形式返回。 而 REPLACE INTO 如果存在 primary 或 unique 相同的记录，则先删除掉。再插入新记录。

统计重复数据
以下我们将统计表中 first_name 和 last_name的重复记录数：
```
SELECT COUNT(*) as repetitions, last_name, first_name
    FROM person_tbl
    GROUP BY last_name, first_name
    HAVING repetitions > 1;
```
一般情况下，查询重复的值，执行以下操作：
确定哪一列包含的值可能会重复。
在列选择列表使用COUNT(*)列出的那些列。
在GROUP BY子句中列出的列。
HAVING子句设置重复数大于1。

过滤重复数据
如果需要读取不重复的数据可以在 SELECT 语句中使用 DISTINCT 关键字来过滤重复数据。
```
SELECT DISTINCT last_name, first_name FROM person_tbl;
```
也可以使用 GROUP BY 来读取数据表中不重复的数据：
```
SELECT last_name, first_name FROM person_tbl GROUP BY (last_name, first_name);
```

删除重复数据
如果想删除数据表中的重复数据，可以使用以下的SQL语句：
```
CREATE TABLE tmp SELECT last_name, first_name, sex FROM person_tbl  GROUP BY (last_name, first_name, sex);
DROP TABLE person_tbl;
ALTER TABLE tmp RENAME TO person_tbl;
```
也可以在数据表中添加 INDEX（索引） 和 PRIMAY KEY（主键）这种简单的方法来删除表中的重复记录。方法如下：
```
ALTER IGNORE TABLE person_tbl
ADD PRIMARY KEY (last_name, first_name);
```

MySQL及 SQL 注入

防止SQL注入，需要注意以下几个要点：
1.永远不要信任用户的输入。对用户的输入进行校验，可以通过正则表达式，或限制长度；对单引号和双"-"进行转换等。
2.永远不要使用动态拼装sql，可以使用参数化的sql或者直接使用存储过程进行数据查询存取。
3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。
4.不要把机密信息直接存放，加密或者hash掉密码和敏感的信息。
5.应用的异常信息应该给出尽可能少的提示，最好使用自定义的错误信息对原始错误信息进行包装
6.sql注入的检测方法一般采取辅助软件或网站平台来检测，软件一般采用sql注入检测工具jsky，网站平

台就有亿思网站安全平台检测工具。MDCSOFT SCAN等。采用MDCSOFT-IPS可以有效的防御SQL注入，XSS攻击等。

MySQL导出数据

可以使用SELECT...INTO OUTFILE语句来简单的导出数据到文本文件上。
在下面的例子中，生成一个文件，各值用逗号隔开。这种格式可以被许多程序使用。

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.text'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM test_table;
```

SELECT ... INTO OUTFILE 语句有以下属性:
LOAD DATA INFILE是SELECT ... INTO OUTFILE的逆操作，SELECT句法。为了将一个数据库的数据写入一个文件，使用SELECT ... INTO OUTFILE，为了将文件读回数据库，使用LOAD DATA INFILE。
SELECT...INTO OUTFILE 'file_name'形式的SELECT可以把被选择的行写入一个文件中。该文件被创建到服务器主机上，因此您必须拥有FILE权限，才能使用此语法。
输出不能是一个已存在的文件。防止文件数据被篡改。
你需要有一个登陆服务器的账号来检索文件。否则 SELECT ... INTO OUTFILE 不会起任何作用。
在UNIX中，该文件被创建后是可读的，权限由MySQL服务器所拥有。这意味着，虽然可以读取该文件，但可能无法将其删除。

MySQL导入数据

1、mysql 命令导入　语法格式：

```
mysql -u用户名　-p密码　< 要导入的数据库数据(runoob.sql)
```

2、source 命令导入
source 命令导入数据库需要先登录到数库终端：

```
mysql> create database abc;      # 创建数据库
mysql> use abc;                  # 使用已创建的数据库
mysql> set names utf8;           # 设置编码
mysql> source /home/abc/abc.sql  # 导入备份数据库
```

3、使用 LOAD DATA 导入数据
MySQL 中提供了LOAD DATA INFILE语句来插入数据。 以下实例中将从当前目录中读取文件 dump.txt ，将该文件中的数据插入到当前数据库的 mytbl 表中。

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt' INTO TABLE mytbl;
```

4、使用 mysqlimport 导入数据
mysqlimport 客户端提供了 LOAD DATA INFILEQL 语句的一个命令行接口。mysqlimport 的大多数选项直接对应 LOAD DATA INFILE 子句。
从文件 dump.txt 中将数据导入到 mytbl 数据表中, 可以使用以下命令：

```
$ mysqlimport -u root -p --local mytbl dump.txt
password *****
```

mysqlimport 命令可以指定选项来设置指定格式,命令语句格式如下：

```
$ mysqlimport -u root -p --local --fields-terminated-by=":" \
  --lines-terminated-by="\r\n"  mytbl dump.txt
```

mysqlimport 语句中使用 --columns 选项来设置列的顺序：

```
$ mysqlimport -u root -p --local --columns=b,c,a \
   mytbl dump.txt
```

mysqlimport的常用选项介绍

选项	功能
-d or --delete	新数据导入数据表中之前删除数据数据表中的所有信息
-f or --force	不管是否遇到错误，mysqlimport将强制继续插入数据
-i or --ignore	mysqlimport跳过或者忽略那些有相同唯一 关键字的行， 导入文件中的数据将被忽略。
-l or -lock-tables	数据被插入之前锁住表，这样就防止了， 你在更新数据库时，用户的查询和更新受到影响。
-r or -replace	这个选项与－i选项的作用相反；此选项将替代 表中有相同唯一关键字的记录。
--fields-enclosed-by= char	指定文本文件中数据的记录时以什么括起的， 很多情况下 数据以双引号括起。 默认的情况下数据是没有被字符括起的。
--fields-terminated-by=char	指定各个数据的值之间的分隔符，在句号分隔的文件中， 分隔符是句号。您可以用此选项指定数据之间的分隔符。 默认的分隔符是跳格符（Tab）
--lines-terminated-by=str	此选项指定文本文件中行与行之间数据的分隔字符串 或者字符。 默认的情况下mysqlimport以newline为行分隔符。 您可以选择用一个字符串来替代一个单个的字符： 一个新行或者一个回车。

mysqlimport 命令常用的选项还有 -v 显示版本（version）， -p 提示输入密码（passw
等。

MySQL函数和运算符

优先级顺序	运算符
1	:=
2	||, OR, XOR
3	&&, AND
4	NOT
5	BETWEEN, CASE, WHEN, THEN, ELSE
6	=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
7	|
8	&
9	<<, >>
10	-, +
11	*, /, DIV, %, MOD
12	^
13	- (一元减号), ~ (一元比特反转)
14	!