第29章：数据库维护

1. **备份与还原数据库**
2. **使用MYSQLDUMP命令备份数据库**

MYSQLDUMP是MYSQL中一个常用的备份命令，执行此命令会将包含数据的表结构与数据内容转换成相应的CREATE语句和INSERT INTO语句保存在文本文件中，将来如果需要还原数据，只需要执行该文本文件中的SQL语句即可。

MYSQLDUMP是DOS系统下的命令，在使用时无须进入 MYSQL命令行，否则将无法执行。

Mysqldump --help

导出备份数据：

mysqldump -u用户名 -h 主机名 -p 密码 数据库[表名...]

* > [路径]dump\_name.sql；

# 备份整个数据库到dump\_name.sql文件中

mysqldump -u用户名 -p 数据库 数据表1,数据表2 > [路径]dump\_name.sql；

# 备份数据库中的某个表到dump\_name.sql文件中

mysqldump -u用户名 -p --databases 数据库1,数据库2 > [路径]dump\_name.sql；

# 备份多个数据库到dump\_name.sql文件中

mysqldump -u用户名 -p --all-databases > [路径]databases\_name.sql；

# 备份系统中所有数据库到databases\_name.sql文件中

1. **使用MYSQL命令恢复数据库**

对于备份数据库后生成的包含有建库、建表、插入数据等SQL语句的文本文件，可以通过MYWSQL命令还原到新的数据库中，实现数据库的恢复。

导入备份数据：

mysql -u用户名 -p 数据库 < [路径]dump\_name.sql；

**注意：**

在执行上面的语句之前，必须在MYSQL服务器中创建新数据库，如果不存在新数据库，恢复过程将会出错。

MYSQL命令是DOS环境下的恢复数据库的命令，如果已经登录了MYSQL服务器，也可以使用SOURCE命令恢复数据库。语法格式如下：

SOURCE filename；

其中，filename为数据库备份文件。在执行命令之前，同样需要先创建新数据库，并且使用USE命令打开该数据库，否则将会出现错误。

1. **通过复制文件实现数据备份和恢复**

MYSQL服务器中的数据在磁盘上是以文件形式保存的，所以可以直接复制MYSQL数据库的存储目录及文件进行备份。MYSQL默认的数据库文件存储目录在不同的操作系统下有所不同，通常默认的存储目录 如下：

C:\programdata\mysql\mysql server 5.5\data

**注意：**

**在复制数据文件之前，需要先停止MYSQL服务。一般情况下，MYSQL服务在使用过程中不允许被停止，这种方法并不是最好的方法，而且对INNODB存储引擎的表不适用。使用该方法备份的数据最好还原到相同版本的服务器中，不同的版本可能不兼容。**

**使用此种方法备份数据库时，为了保证所备份数据的完整性，在停止MYSQL数据库服务器之前，需要先执行flush tables语句将所有数据写入到数据文件中。**

1. **表数据导出到文本文件**

通过对表数据的导出和导入，可以实现在MYSQL数据库服务器与其他数据库服务器间移动数据。导出操作是指将数据从MYSQL数据表复制到文本文件。

导出数据语法格式如下 ：

Select [字段] from 表名 where 条件 into outfile ‘文件名’[option]

示例：

Select \* from result where subjectNo=(select subjectNo from subject where subjectName=’logic java’)

Into outfile d:/backup/result\_java.txt;

若需设置导出文件的显示格式，每条数据记录为一行，每行数据记录以“>”开头，字段间以“、”分隔，各数值使用引号“”括起来，则修改后的SQL语句如下：

Use myschool；

Select \* from result

where subjectNo=

(select subjectNo from subject where subjectName=’logic java’)

Into outfile ‘d:/backup/result\_java.txt’

Fields

Terminated by ‘\、’

Enclosed by ‘\“’

Lines

Starting by ‘\>’

Terminated by ‘\r\n’

1. **文本文件导入到数据表**

所谓导入操作，是指将数据从文本文件加载到MYSQL数据库表里。同样，导入数据的方式有多种。在导入数据之前，必须先创建

导入数据语法格式如下：

Load data infile 文件名 into table 表名 [option]

示例：

Use myschoolDB；

Create table result（

StudentNo int(4) not null comment ‘学号’,

subjectNo int(4) not null comment ‘课号’，

Examdate datetime not null comment ‘考试日期’，

Studentresult int(4) not null comment ‘考试成绩’,

Primary key (studentNo,subjectNo,examdate)

)engine-=innoDB default charset=utf8;

Load data infile ‘d:/back/result\_java.txt’ into table result;

Select \* from result;

若已经设置相应的参数，则对应的语句如下：

Delete from result；

Load data infile ‘d:/backup/result\_java.txt’into table result

Fields

Terminated by ‘\、’

Enclosed by ‘\“’

Lines

Starting by ‘\>’

Terminated by ‘\r\n’

1. **数据库维护**
2. **分析表**(更新索引的统计信息)：

ANALYZE TABLE 数据表名称[,数据表名称..]

* 分析并存储表的键值分布统计信息
* 为执行查询提供更好的选择
* 对使用InnoDB、NDB和MyISAM存储引擎的表生效
* 支持分表
* ANALYZE TABLE选项：

ANALYZE [ NO\_WRITE\_TO\_BINLOG | LOCAL  ] TABLE 数据表名称[,数据表名称..]；不记录二进制日志

1. **检查表**(检查表的完整性)；

CHECK TABLE 数据表名称[,数据表名称..]

* 检查表结构和内容的完整性
* 验证视图定义
* 支持分区表
* 对使用InnoDB、CSV、MyISAM和ARCHIVE存储引擎的表生效
* CHECK TABLE选项：

CHECK TABLE 数据表名称[,数据表名称..] FOR UPGRADE；

检查当前服务器的表是否工作

CHECK TABLE 数据表名称[,数据表名称..] QUICK；

不扫描不正确链接的行

* 如果CHECK TABLE发现使用InnoDB存储引擎的表有错误：
  + 服务器将会被关闭以防止错误传播
  + MySQL将会记录到错误到错误日志

1. **校验表**(报告表数据的一致性检测结果)；

CHECKSUM TABLE 数据表名称[,数据表名称..]

* 报告表的checksum
* (该语句)用来验证表备份，回滚或者其它操作之前或之后内容是否一致
* 读取整个表逐行进行校验
* 默认选项EXTENDED提供这种行为
* QUICK选项在MyISAM表上可用
* MyISAM默认选项上为设置CHECKSUM=1；

1. **修复表**；

REPAIR TABLE 数据表名称[,数据表名称..]

* 修复可能已经损坏的MyISAM或ARCHIVE表
* 不支持InnoDB
* 优化过程中会锁定表
* 支持分区表
* REPAIR TABLE选项：
  + REPAIR TABLE 数据表名称 QUICK；仅仅修复索引树
  + REPAIR TABLE 数据表名称 EXTENDED；逐行创建索引（替代以排序方式一次创建一个索引）
  + REPAIR TABLE 数据表名称 USE\_FRM；使用.FRM文件重新创建.MYI文件
  + REPAIR { NO\_WRITE\_TO\_BINLOG | LOCAL }TABLE 数据表名称；不记录二进制日志

1. **优化表**；

OPTIMZE TABLE 数据表名称[,数据表名称..]

* 整理表的锁片
* 重建表和释放未使用的空间以整理碎片
* 优化过程中会锁定表
* 更新索引统计信息
* 在一张永久的，数据密集/完全填充？的表上是比较好的
* 在InnoDB，MyISAM和ARCHIV表上生效
* 支持分区表
* OPTIMZE TABLE选项：

OPTIMZE [ NO\_WRITE\_TO\_BINLOG | LOCAL  ] TABLE 数据表名称[,数据表名称..]；不记录二进制日志

**6）返回信息**：

1. Table：标示执行操作的表，
2. Op：操作的名字(check、repair、analyze、optimze)
3. Msg\_type：提供一个成功或失败的指示，
4. Msg\_text：提供额外的信息；
5. **数据的导入与导出（利用第三方软件）**
6. 与EXCEL数据转换
7. 与ACCESS数据库转换
8. 与.csv文件数据转换

**附：**

# **MySQL数据库中char与varchar的区别分析及使用建议**

在数据库中，字符型的数据是最多的，可以占到整个数据库的80%以上。为此正确处理字符型的数据，对于提高数据库的性能有很大的作用。在字符型数据中，用的最多的就是 Char与Varchar两种类型。前面的是固定长度，而后面的是可变长度。现在我们需要考虑的是，在什么情况下使用Char字符型数据，什么情况下采用 Varchar字符型数据。

**一、VARCHAR与CHAR字符型数据的差异**

在MySQL数据库中，用的最多的字符型数据类型就是Varchar和Char。这两种数据类型虽然都是用来存放字符型数据，但是无论从结构还是从数据的保存方式来看，两者相差很大。而且其具体的实现方式，还依赖与存储引擎。以大家最常用的MYISAM存储引擎为例，谈谈这两种数据类型的差异。在后续建议中，也是针对这种存储类型而言的。

这里首先需要明白的一点是，这两种数据类型，无论采用哪一种存储引擎，系统存储数据的方式都是不同的。正是因为如此，我们才有必要研究两者的不同。然后在合适的情况下，采用恰当的方式。了解这一点之后，我们再来看后续的内容。

Varchar往往用来保存可变长度的字符串。简单的说，我们只是给其固定了一个最大值，然后系统会根据实际存储的数据量来分配合适的存储空间。为此相比CHAR字符数据而言，其能够比固定长度类型占用更少的存储空间。不过在实际工作中，由于某些特殊的原因，会在这里设置例外。如管理员可以根据需要指定ROW\_FORMAT=FIXED选项。利用这个选项来创建MyISAM表的话，系统将会为每一行使用固定长度的空间。此时会造成存储空间的损耗。通常情况下，VARCHAR数据类型能够节约磁盘空间，为此往往认为其能够提升数据库的性能。不过这里需要注意的是，这往往是一把双刃剑。其在提升性能的同时，往往也会产生一些副作用。如因为其长度是可变的，为此在数据进行更新时可能会导致一些额外的工作。如在更改前，其字符长度是10位（Varchar规定的最长字符数假设是50位），此时系统就只给其分配10个存储的位置（假设不考虑系统自身的开销）。更改后，其数据量达到了20位。由于没有超过最大 50位的限制，为此数据库还是允许其存储的。只是其原先的存储位置已经无法满足其存储的需求。此时系统就需要进行额外的操作。如根据存储引擎不同，有的会采用拆分机制，而有的则会采用分页机制。

CHAR数据类型与VARCHAR数据类型不同，其采用的是固定长度的存储方式。简单的说，就是系统总为其分配最大的存储空间。当数据保存时，即使其没有达到最大的长度，系统也会为其分配这么多的存储空间。显然，这种存储方式会造成磁盘空间的浪费。这里需要提醒的一点是，当字符位数不足时，系统并不会采用空格来填充。相反，如果在保存CHAR值的时候，如果其后面有空值，系统还会自动过滤其空格。而在进行数据比较时，系统又会将空格填充到字符串的末尾。

显然，VARCHAR与CHAR两种字符型数据类型相比，最大的差异就是前者是可变长度，而后者则是固定长度。在存储时，前者会根据实际存储的数据来分配最终的存储空间。而后者则不管实际存储数据的长度，都是根据CHAR规定的长度来分配存储空间。这是否意味着CHAR的数据类型劣于VARCHAR呢？其实不然。否则的话，就没有必要存在CHAR字符类型了。虽然VARCHAR数据类型可以节省存储空间，提高数据处理的效率。但是其可变长度带来的一些负面效应，有时候会抵消其带来的优势。为此在某些情况下，还是需要使用Char数据类型。

**二、项目建议**

根据上面的分析，我们知道VARCHAR数据类型是一把双刃剑，其在带来性能提升的同时，也可能会存在着一些额外的消耗。我们在评估到底是使用VARCHAR数据类型还是采用CHAR数据类型时，就需要进行均衡。在实际项目中，我们会考量如下情况。

一是根据字符的长度来判断。如某个字段，像人的名字，其最长的长度也是有限的。如我们给其分配18个字符长度即可。此时虽然每个人的名字长度有可能 不同，但是即使为其分配了固定长度的字符类型，即18个字符长度，最后浪费的空间也不是很大。而如果采用VARCHAR数据类型时，万一以后需要改名， 而原先的存储空间不足用来容纳新的值，反而会造成一些额外的工作。在这种情况下，进行均衡时，会认为采用CHAR固定长度的数据类型更好。在实际项目中， 如果某个字段的字符长度比较短此时一般是采用固定字符长度。

二是考虑其长度的是否相近。如果某个字段其长度虽然比较长，但是其长度总是近似的，如一般在90个到100个字符之间，甚至是相同的长度。此时比较 适合采用CHAR字符类型。比较典型的应用就是MD5哈希值。当利用MD5哈希值来存储用户密码时，就非常使用采用CHAR字符类型。因为其长度是相同 的。另外，像用来存储用户的身份证号码等等，一般也建议使用CHAR类型的数据。

另外请大家考虑一个问题，CHAR（1）与VARCHAR（1）两者的定义，会有什么区别呢？虽然这两个都只能够用来保存单个的字符，但是 VARCHAR要比CHAR多占用一个存储位置。这主要是因为使用VARCHAR数据类型时，会多用1个字节用来存储长度信息。这个管理上的开销CHAR 字符类型是没有的。

三是从碎片角度进行考虑。使用CHAR字符型时，由于存储空间都是一次性分配的。为此某个字段的内容，其都是存储在一起的。单从这个角度来讲，其不存在碎片的困扰。而可变长度的字符数据类型，其存储的长度是可变的。当其更改前后数据长度不一致时，就不可避免的会出现碎片的问题。故使用可变长度的字符型数据时，数据库管理员要时不时的对碎片进行整理。**如执行数据库导出导入作业，来消除碎片。**

四是即使使用Varchar数据类型，也不能够太过于慷慨。这是什么意思呢？如现在用户需要存储一个地址信息。根据评估，只要使用100个字符就可以了。但是有些数据库管理员会认为，反正Varchar数据类型是根据实际的需要来分配长度的。还不如给其大一点的呢。为此他们可能会为这个字段一次性分配200个字符的存储空间。这VARCHAR（100）与VARCHAR（200）真的相同吗？结果是否定的。虽然他们用来存储90个字符的数据，其存储空间相同。但是对于内存的消耗是不同的。对于VARCHAR数据类型来说，硬盘上的存储空间虽然都是根据实际字符长度来分配存储空间的，但是对于内存来说，则不是。而其实是使用固定大小的内存块来保存值。简单的说，就是使用字符类型中定义的长度，即200个字符空间。显然，这对于排序或者临时表（这些内容都需要通过内存来实现）作业会产生比较大的不利影响。所以如果某些字段会涉及到文件排序或者基于磁盘的临时表时，分配VARCHAR数据类型时仍然不能够太过于慷慨。还是要评估实际需要的长度，然后选择一个最长的字段来设置字符长度。如果为了考虑冗余，可以留10%左右的字符长度。千万不能认为其为根据实际长度来分配存储空间，而随意的分配长度，或者说干脆使用最大的字符长度。

