目录

[**一、** **数据库设计** 3](#_Toc23849651)

[**（一）范式** 3](#_Toc23849652)

[**（二）约束条件** 3](#_Toc23849653)

[**二、** **MySQL介绍** 3](#_Toc23849654)

[**（一）** **MySQL结构** 3](#_Toc23849655)

[**（二）四层结构** 3](#_Toc23849656)

[**1、网络接入层** 3](#_Toc23849657)

[**2、服务层（核心层）** 4](#_Toc23849658)

[**3、存储引擎层** 4](#_Toc23849659)

[**4、系统文件层** 4](#_Toc23849660)

[**三、** **数据定义语言（DDL）** 4](#_Toc23849661)

[**（一）数据库操作** 4](#_Toc23849662)

[**1、创建数据库** 4](#_Toc23849663)

[**2、删除数据库** 4](#_Toc23849664)

[**3、切换数据库** 4](#_Toc23849665)

[**（二）创建表** 4](#_Toc23849666)

[**（三）修改表** 5](#_Toc23849667)

[**1、添加列** 5](#_Toc23849668)

[**2、删除列** 5](#_Toc23849669)

[**3、删除表** 5](#_Toc23849670)

[**四、** **数据操作语句（DML）** 5](#_Toc23849671)

[**五、** **数据查询语句（DQL）** 5](#_Toc23849672)

[**六、** **数据控制语句（DCL）** 5](#_Toc23849673)

[**七、** **数据事务语句（TPL）** 5](#_Toc23849674)

[**八、** **数据类型** 5](#_Toc23849675)

[**九、** **视图** 5](#_Toc23849676)

[**（一）** **创建视图** 5](#_Toc23849677)

[**（二）** **修改视图** 5](#_Toc23849678)

[**（三）** **删除视图** 5](#_Toc23849679)

[**（四）** **查看视图** 5](#_Toc23849680)

[**十、** **索引** 5](#_Toc23849681)

[**十一、** **MySQL数据库引擎** 6](#_Toc23849682)

[**（四）** **CSV存储引擎** 7](#_Toc23849683)

[**十二、** **事务** 7](#_Toc23849684)

[**十三、** **存储过程和函数** 7](#_Toc23849685)

[**十四、** **触发器** 7](#_Toc23849686)

[**十五、** **数据库优化** 7](#_Toc23849687)

MySQL

1. **数据库设计**

**（一）范式**

第一范式(1NF)：字段值具有原子性,不能再分(所有关系型数据库系统都满足第一范式);

第二范式(2NF)：一个表必须有主键,即每行数据都能被唯一的区分;

第三范式(3NF)：一个表中不能包涵其他相关表中非关键字段的信息,即数据表不能有沉余字段;

巴斯-科德范式（BCNF）：任何非主属性不能对主键子集依赖（在3NF基础上消除对主码子集的依赖）

第四范式 （4NF）：限制关系模式的属性之间不允许有非平凡且非函数依赖的多值依赖。

第五范式（5NF）：表必须可以分解为较小的表，除非那些表在逻辑上拥有与原始表相同的主键。

备注：，三范式以后只做了解即可。

**（二）约束条件**

主键约束（primary key）

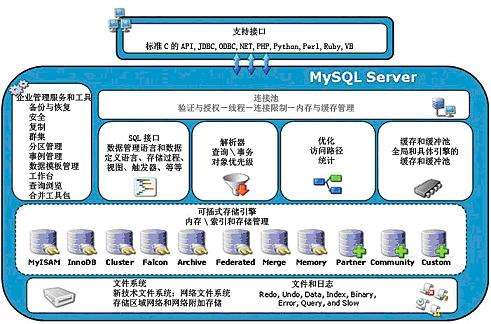
非空约束(not null)

唯一约束（unique）

默认值约束（default）

外键约束（foreign key）

1. **MySQL介绍**
2. **MySQL结构**



MySQL基架大致包括如下几大模块组件：

1.MySQL向外提供的交互接口（Connectors）：Connectors组件，是MySQL向外提供的交互组件，如java,.net,php等语言可以通过该组件来操作SQL语句，实现与SQL的交互。

2.管理服务组件和工具组件(Management Service & Utilities)：提供对MySQL的集成管理，如备份(Backup),恢复(Recovery),安全管理(Security)等。

3.连接池组件(Connection Pool)：负责监听对客户端向MySQL Server端的各种请求，接收请求，转发请求到目标模块。

每个成功连接MySQL Server的客户请求都会被创建或分配一个线程，该线程负责客户端与MySQL Server端的通信，接收客户端发送的命令，传递服务端的结果信息等。

4.SQL接口组件(SQL Interface)：接收用户SQL命令，如DML,DDL和存储过程等，并将最终结果返回给用户。

5.查询分析器组件(Parser)：首先分析SQL命令语法的合法性，并尝试将SQL命令分解成数据结构，若分解失败，则提示SQL语句不合理。

6.优化器组件（Optimizer）：对SQL命令按照标准流程进行优化分析。

7.缓存主件（Caches & Buffers）：缓存和缓冲组件

8.插件式存储引擎（Pluggable Storage Engines）

9.物理文件（File System）

**（二）四层结构**

**1、网络接入层**

主要负责连接管理、授权认证、安全等等。每个客户端连接都对应着服务器上的一个线程。服务器上维护了一个线程池，避免为每个连接都创建销毁一个线程。当客户端连接到MySQL服务器时，服务器对其进行认证。可以通过用户名与密码认证，也可以通过SSL证书进行认证。登录认证后，服务器还会验证客户端是否有执行某个查询的操作权限。这一层并不是MySQL所特有的技术。

**TCP/IP（传输控制协议/Internet 协议，Transmission Control Protocol/Internet Protocol）：**该通信协议套件用于连接 Internet 上的主机。在 Linux 操作系统中，TCP/IP 是内置的，供 Internet 使用，从而使其成为通过网络传输数据的标准。这是适用于 Windows 的最佳连接类型。

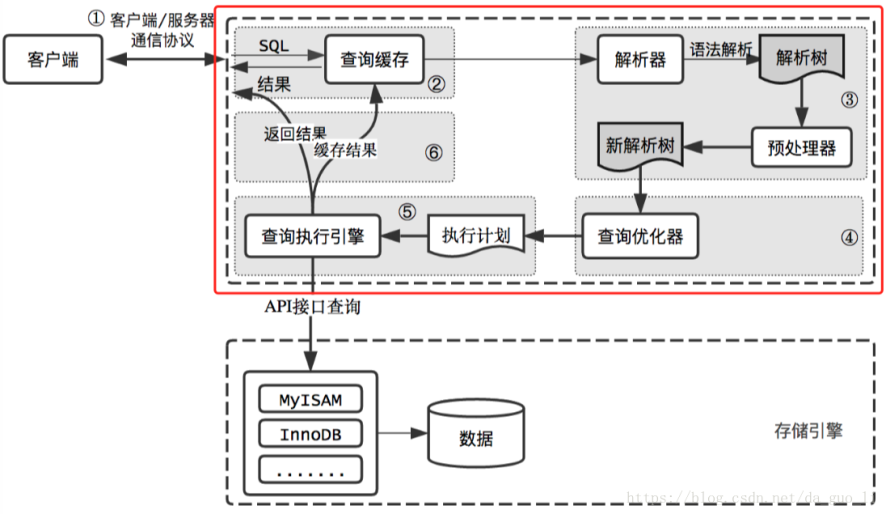
**UNIX 套接字：**一种进程间通信形式，用于在同一台计算机上的程之间形成双向通信链路的一端。套接字需要本地系统上的物理文件，这是适用于 Linux 的最佳连接类型。

**共享内存：**一种在程序之间传递数据的有效方式。一个程序创建其他进程（如果允许）可以访问的内存部分。此 Windows 显式“被动”模式仅适用于单台 (Windows) 计算机。默认情况下，共享内存处于禁用状态。要启用共享内存连接，必须使用--shared-memory 选项启动服务器。

**命名管道：**命名管道的使用偏向于客户机/服务器通信，其工作方式与套接字非常相似。命名管道支持读/写操作，以及服务器应用程序的显式“被动”模式。此协议仅适用于单台 (Windows) 计算机。默认情况下，命名管道处于禁用状态。要启用命名管道连接，必须使用 --enable-named-pipe 选项启动服务器。

**2、服务层（核心层）**

MySQL的核心服务层都在这一层，查询解析，SQL执行计划分析，SQL执行计划优化，查询缓存。以及跨存储引擎的功能都在这一层实现：存储过程，触发器，视图等。通过下图来观察服务层的内部结构：



**查询缓存：**在解析查询之前，服务器会检查查询缓存，如果能找到对应的查询，服务器不必进行查询解析、优化和执行的过程，直接返回缓存中的结果集。

**解析器与预处理器：**MySQL会解析查询，并创建了一个内部数据结构（解析树）。这个过程解析器主要通过语法规则来验证和解析。比如SQL中是否使用了错误的关键字或者关键字的顺序是否正确等等。预处理会根据MySQL的规则进一步检查解析树是否合法。比如要查询的数据表和数据列是否存在等。

**查询优化器：**优化器将其转化成查询计划。多数情况下，一条查询可以有很多种执行方式，最后都返回相应的结果。优化器的作用就是找到这其中最好的执行计划。优化器并不关心使用的什么存储引擎，但是存储引擎对优化查询是有影响的。优化器要求存储引擎提供容量或某个具体操作的开销信息来评估执行时间。

**查询引擎：**在完成解析和优化阶段以后，MySQL会生成对应的执行计划，查询执行引擎根据执行计划给出的指令调用存储引擎的接口得出结果。

**3、存储引擎层**

负责MySQL中数据的存储与提取。 服务器中的查询执行引擎通过API与存储引擎进行通信，通过接口屏蔽了不同存储引擎之间的差异。MySQL采用插件式的存储引擎。MySQL为我们提供了许多存储引擎，每种存储引擎有不同的特点。我们可以根据不同的业务特点，选择最适合的存储引擎。如果对于存储引擎的性能不满意，可以通过修改源码来得到自己想要达到的性能。

特点：

    存储引擎是针对于表的而不是针对库的（一个库中不同表可以使用不同的存储引擎），服务器通过API与存储引擎进行通信，用来屏蔽不同存储引擎之间的差异。

**4、系统文件层**

该层主要是将数据库的数据存储在文件系统之上，并完成与存储引擎的交互。

1. **数据定义语言（DDL）**

**（一）数据库操作**

**1、创建数据库**

CREATE database 数据库名;

CREATE database if not exists数据库名DEFAULT CHARSET utf8 COLLATE utf8\_general\_ci;

创建数据库，该命令的作用：

如果数据库不存在则创建，存在则不创建。

创建数据库，并设定编码集为utf8

**2、删除数据库**

drop database <数据库名>;

使用数据库

use数据库;

**3、切换数据库**

Database changed

**（二）创建表**

CREATE TABLE mytable (

# int 类型，不为空，自增

id INT NOT NULL AUTO\_INCREMENT,

# int 类型，不可为空，默认值为 1，不为空

col1 INT NOT NULL DEFAULT 1,

# 变长字符串类型，最长为 45 个字符，可以为空

col2 VARCHAR(45) NULL,

# 日期类型，可为空

col3 DATE NULL,

# 设置主键为 id

PRIMARY KEY (`id`)

# ENGINE 设置存储引擎，CHARSET设置编码。

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

**（三）修改表**

**1、添加列**

alter table mytable add col char(20);

**2、删除列**

alter table mytable drop column col;

**3、删除表**

drop table mytable;

1. **数据操作语句（DML）**

insert update delete

1. **数据查询语句（DQL）**

select

1. **数据控制语句（DCL）**

grant revoke

1. **数据事务语句（TPL）**

commit collback savapoint

1. **数据类型**
2. **视图**

视图是虚拟表，本身不存储数据，而是按照指定的方式进行查询,不能创建索引。

可以通过视图插入数据，但是只能基于一个基础表进行插入，不能跨表更新数据。

优点：

简单化，数据所见即所得

安全性，用户只能查询或修改他们所能见到得到的数据

逻辑独立性，可以屏蔽真实表结构变化带来的影响

缺点：

性能相对较差，简单的查询也会变得稍显复杂

修改不方便，特变是复杂的聚合视图基本无法修改

1. **创建视图**

create view 视图名(列1，列2...) as select (列1，列2...) from 表名;

1. **修改视图**

create or replace view 视图名 as select [ ] from [ ];

1. **删除视图**

drop view if exists 视图名;（if exist参数用来判断视图是否存在）

1. **查看视图**

desc 视图名;

show fields from 视图名;

1. **索引**

索引（Index）是帮助MySQL高效获取数据的数据结构，通俗来讲索引就好比书本的目录，加快数据库的查询速度。

**（一）索引类型**

**主键索引 （PRIMARY KEY）：**数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。 在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。

**唯一索引 （UNIQUE）：**表明此索引的每一个索引值只对应唯一的数据记录，对于单列惟一性索引，这保证单列不包含重复的值。对于多列惟一性索引，保证多个值的组合不重复。

**普通索引 （INDEX） ：**最基本的索引，它没有任何限制，用于加速查询。

**组合索引 （INDEX） ：**指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。使用组合索引时遵循最左前缀集合。

**全文索引 （FULLTEXT）：**主要用来查找文本中的关键字，而不是直接与索引中的值相比较。

**（二）实现方式**

**1、B+树**

B+树索引构造B+树，根据键值快速找到数据，通过找到被查数据所在的页，然后将页读入内存，再在内存中进行查找，最后得到查找的数据。

我们经常听到B+树就是这个概念，用这个树的目的和红黑树差不多，也是为了尽量保持树的平衡，当然红黑树是二叉树，但B+树就不是二叉树了，节点下面可以有多个子节点，数据库开发商会设置子节点数的一个最大值，这个值不会太小，所以B+树一般来说比较矮胖，而红黑树就比较瘦高了。

关于B+树的插入，删除，会涉及到一些算法以保持树的平衡，这里就不详述了。ORACLE的默认索引就是这种结构的。

如果经常需要同时对两个字段进行AND查询,那么使用两个单独索引不如建立一个复合索引，因为两个单独索引通常数据库只能使用其中一个，而使用复合索引因为索引本身就对应到两个字段上的，效率会有很大提高。

**2、散列索引**

优点：通过字段的值计算的hash值，定位数据非常快

缺点：不支持范围查询

第二种索引叫做散列索引，就是通过散列函数来定位的一种索引，不过很少有单独使用散列索引的，反而是散列文件组织用的比较多。

散列文件组织就是根据一个键通过散列计算把对应的记录都放到同一个槽中，这样的话相同的键值对应的记录就一定是放在同一个文件里了，也就减少了文件读取的次数，提高了效率。

散列索引呢就是根据对应键的散列码来找到最终的索引项的技术，其实和B树就差不多了，也就是一种索引之上的二级辅助索引，我理解散列索引都是二级或更高级的稀疏索引，否则桶就太多了，效率也不会很高。

**3、位图索引**

位图索引与B树索引不同，位图索引不存储rowid值（数据实际物理地址），也不存储键值。

在特殊的列上创建位图 索引.

特殊的列是指该列的基数很低的列（基数：列值的数量比列的行数来的小）。

位图索引是一种针对多个字段的简单查询设计一种特殊的索引，适用范围比较小，只适用于字段值固定并且值的种类很少的情况，比如性别，只能有男和女，或者级别，状态等等，并且只有在同时对多个这样的字段查询时才能体现出位图的优势。

基数很低的列不适合B树索引，适合位图索引，oracle建议，当一个列的所有取值数量与该列的行数比小于1%时，该列不适合建立B树索引，适合位图索引。

在表中放置单独的位图索引是没有意义的，只有多个列建立位图索引，系统才能有效的利用位图索引提高查询速度.

因为位图索引不能是唯一索引，也不能对其进行键压缩

位图索引的作用源于与其他位图索引的结合，当位图索引的多个列进行查询时，oracle对这些上的位图索引进行布尔and和or运行，最终返回结果.

1. **MySQL数据库引擎**
2. **InnoDB存储引擎**

该存储引擎提供了具有提交、回滚和崩溃恢复能力的事务安全。但是对比MyISAM引擎，写的处理效率会差一些，并且会占用更多的磁盘空间以保留数据和索引。

InnoDB存储引擎的特点：支持自动增长列，支持外键约束

1. **MyISAM存储引擎**

不支持事务、也不支持外键，优势是访问速度快，对事务完整性没有 要求或者以select，insert为主的应用基本上可以用这个引擎来创建表

支持3种不同的存储格式，分别是：静态表；动态表；压缩表

静态表：表中的字段都是非变长字段，这样每个记录都是固定长度的，优点存储非常迅速，容易缓存，出现故障容易恢复；缺点是占用的空间通常比动态表多（因为存储时会按照列的宽度定义补足空格）ps：在取数据的时候，默认会把字段后面的空格去掉，如果不注意会把数据本身带的空格也会忽略。

动态表：记录不是固定长度的，这样存储的优点是占用的空间相对较少；缺点：频繁的更新、删除数据容易产生碎片，需要定期执行OPTIMIZE TABLE或者myisamchk-r命令来改善性能

压缩表：因为每个记录是被单独压缩的，所以只有非常小的访问开支

1. **Memory存储引擎**

Memory存储引擎使用存在于内存中的内容来创建表。每个memory表只实际对应一个磁盘文件，格式是.frm。memory类型的表访问非常的快，因为它的数据是放在内存中的，并且默认使用HASH索引，但是一旦服务关闭，表中的数据就会丢失掉。

MEMORY存储引擎的表可以选择使用BTREE索引或者HASH索引，两种不同类型的索引有其不同的使用范围

Hash索引优点：

Hash 索引结构的特殊性，其检索效率非常高，索引的检索可以一次定位，不像B-Tree 索引需要从根节点到枝节点，最后才能访问到页节点这样多次的IO访问，所以 Hash 索引的查询效率要远高于 B-Tree 索引。

Hash索引缺点： 那么不精确查找呢，也很明显，因为hash算法是基于等值计算的，所以对于“like”等范围查找hash索引无效，不支持；

Memory类型的存储引擎主要用于哪些内容变化不频繁的代码表，或者作为统计操作的中间结果表，便于高效地对中间结果进行分析并得到最终的统计结果，。对存储引擎为memory的表进行更新操作要谨慎，因为数据并没有实际写入到磁盘中，所以一定要对下次重新启动服务后如何获得这些修改后的数据有所考虑。

1. **CSV存储引擎**
2. **Archive**
3. **事务**
4. **存储过程和函数**
5. **触发器**
6. **数据库优化**