**主键**

**Sql语句/规则**

**表关联**

**引擎**

**Mysql逻辑结构**

**范式and反范式**

**数据库定义**: 数据库是一个以某种有组织的方式存储的数据集合

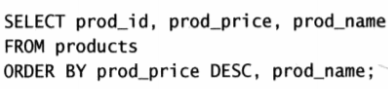
**主键：**



没有主键增删特定行很困难，因为不能保障只涉及相关的行

**SQL语句**（结构化查询语言）:与数据库通信

语句： SELECT DISTINCT 列名 FROM 表名(若多个列则是多个列是否全部不同，否则显示)

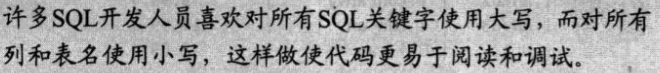
升序ASC



**语句规则：**

AND的优先级比OR高

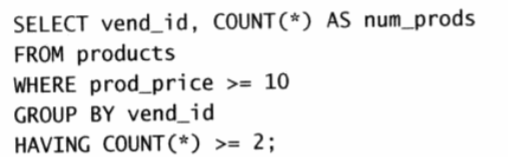
单引号(与双引号无区别)用来限定字符串，内容有单引号则用双引号，数值不需要



数据的所有存储，检索，管理和处理实际上是由数据库软件-DBMS（数据库管理系统）完成的。Mysql是一种DBMS

数据库是通过DBMS创建和操作的容器（文件或者其他），我们不直接访问数据库，通过DBMS访问

Where在数据分组前过滤，having在分组后过滤



Exists 用法

select \* from TableIn where exists(select BID from TableEx where BNAME=TableIn.ANAME)  
select \* from TableIn where ANAME in(select BNAME from TableEx)

将外查询表的每一行，代入内查询作为检验，如果内查询返回的结果存在，则EXISTS子句返回TRUE，这一行可作为外查询的结果行，否则不能作为结果

GROUP BY子句定义的每个组各返回一个结果(去掉重复)

求出最高成绩的学生姓名

姓名和成绩分为两组，然后降序（因为会选取第一行信息）

全表扫描

是在数据库搜寻表的每一条记录，直到符合给定条件的记录返回为止

内连接指的是把表连接时表与表之间匹配的数据行查询出来，就是两张

表之间数据行匹配时，要同时满足ON语句后面的条件才行

外部（左）连接的意思是，无论是否符合ON语句后面的表连接条件都会把左边那张表的记录全部查询出来，右边的那张表只匹配符合条件的数据行

**表之间关联**

都是使用forgine key来完成

一对一

从表（子表）需要主表（父表）的存在才有意义

身份证需要人的存在才有意义

因此人为主表，身份证为从表

要在身份证表中建立外键

两个具有关联关系的表中删除数据时，要先删除从表中的数据，然后在删除主表中的数据

一对多

外键不设唯一

多对多

中间表

学生表（s\_id，s\_name）

老师表（t\_id，t\_name）

中间表（t\_id，s\_id，中间表id）

**创建第三张表（中间表）在中间表中至少两个字段，这两个字段分别指向多对多两个表的主键（外键指向主键）**

**引擎**

MYISAM：

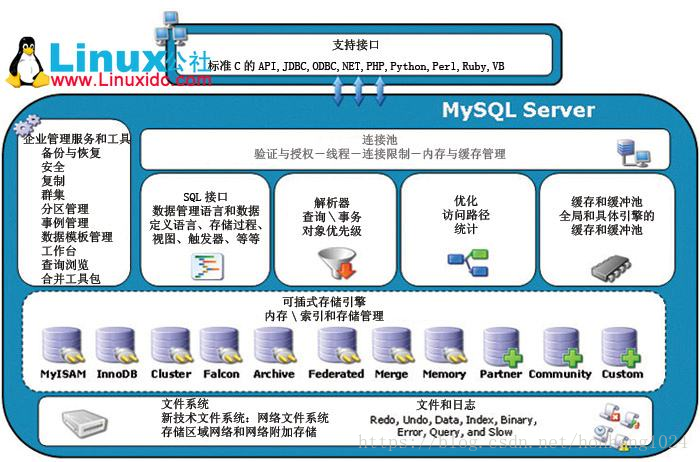
1. 不支持事务/外键，使用表锁（读锁（共享）和写锁（独占）
2. 查行数时不需要扫描全表
3. 全文索引、压缩、空间函数(GIS)等

表锁：它会锁定整张表，一个用户在对表进行写操作前需要先获得写锁，这将会阻塞其他用户对该表的所有读写操作。

INNODB：

1. 支持事务/外键，默认使用行锁（共享锁S和排他锁X）
2. 查表行数要扫描全表
3. 被设计用来处理大量的短期(short-lived)事务

**mysql内部逻辑结构**分成四层，分别是连接层、服务层、引擎层、存储层



mysql内部逻辑结构分成四层，分别是连接层、服务层、引擎层、存储层

**各层的用途说明**

**连接层**

**最上层是一些客户端和连接服务，包含本地sock通信和大多数基于客户端/服务端工具实现的类似于tcp/ip的通信。主要完成一些类似于连接处理、授权认证、及相关的安全方案。**在该层上引入了线程池的概念，为通过认证安全接入的客户端提供线程。同样在该层上可以实现基于SSL的安全链接。服务器也会为安全接入的每个客户端验证它所具有的操作权限。

**服务层**

**2.1 Management Serveices & Utilities： 系统管理和控制工具**

**2.2 SQL Interface: SQL接口接受用户的SQL命令，并且返回查询结果。比如select from就是调用SQL Interface。**

**2.3 Parser: 解析器：SQL命令传递到解析器的时候会被解析器验证和解析。**

**2.4 Optimizer: 查询优化器：SQL语句在查询之前会对查询进行优化。**

**用一个例子就可以理解： select uid,name from user where gender= 1;**

**优化器来决定先投影还是先过滤。**

**2.5 Cache和Buffer： 查询缓存。**

**如果查询缓存有命中的查询结果，查询语句就可以直接去查询缓存中取数据。**

这个缓存机制是由一系列小缓存组成的。比如表缓存，记录缓存，key缓存，权限缓存等

缓存是负责读，缓冲负责写。

**引擎层**

**存储引擎层，存储引擎真正的负责了MySQL中数据的存储和提取，服务器通过API与存储引擎进行通信。**

**由图可看到，Mysql有多种存储引擎，它之所以在多种不同场景中应用并发挥良好作用就主要体现在存储引擎的架构上。**

**插件式的存储引擎架构将查询处理和其它的系统任务以及数据的存储提取相分离。这种架构可以根据业务的需求和实际需要选择合适的存储引擎。**

**存储层**

**主要是将数据存储在运行于裸设备的文件系统之上，并完成与存储引擎的交互。**

**范式and反范式**

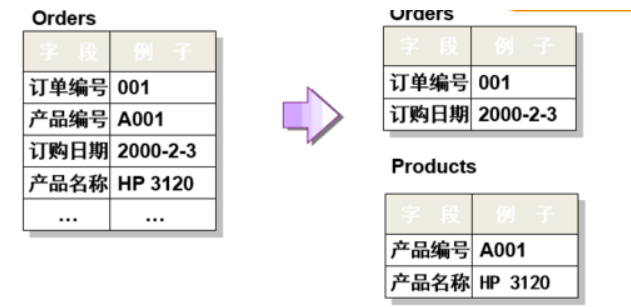
**第一范式**

**数据库表中的每一列都是不可分割的基本数据项**

例如，由“职工号”“姓名”“电话号码”组成的表(一个人可能有一部办公电话和一部移动电话)，这时将其规范化为1NF可以将电话号码分为“办公电话”和“移动电话”两个属性，即职工(职工号，姓名，办公电话，移动电话)。

**第二范式**

**满足第一范式的前提下，确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关**（主要针对联合主键而言）**,也就是说一个表只描述一件事情**



**第三范式**

**满足第三的前提下，表中的每一非主属性只与候选键直接相关而不是间接相关**

**(第二可以间接(传递)依赖第三直接依赖主键)**

订单表中需要有客户相关信息，在分离出客户表之后，订单表中只需要有一个用户id即可，而不能有其他的客户信息

因为其他的客户信息直接关联于用户id，而不是直接与订单id直接相关

**BC范式**

**BCNF需要符合3NF，并且，**不存在任何字段对任一候选关键字段的传递函数依赖

假设有表结构： 仓库id、管理员id、物品id、物品数量。其中管理员和仓库的关系是一对一，仓库和物品的关系是一对多；

这里的主要问题是，仓库id和管理员id这两个关键字段之间的关系被耦合到每一个实例中了，这导致：

(仓库ID, 存储物品ID) →(管理员ID, 数量)

(管理员ID, 存储物品ID) → (仓库ID, 数量)

但

(仓库ID) → (管理员ID)

(管理员ID) → (仓库ID)

a) 表中没有数据的时候，无法描述仓库和管理员之间的关系

b) 一个仓库的管理员替换后，表中所有含有该仓库的实例中的管理员id都要被修改

解决办法是将二者的关系提出来单独建表。

则原表结构改为：仓库id 物品id 物品数量

增加表的结构： 仓库id 管理员id

**范式的优点和缺点**

**优点：更新操作通常比反范式化快；表更小，没有冗余数据。**

**缺点：通常需要关联，代价昂贵，也可能是一些索引策略无效。**

**反范式的优点和缺点**

**优点：减少关联，更有效的使用索引，数据量大时效率较好**

超键(super key):在关系中能唯一标识元组的属性集称为关系模式的超键（元组就是行）

候选键(candidate key):不含有多余属性的超键称为候选键(删掉其中一个属性不在是超键)

主键(primary key): 候选键里面随便挑一个

外键(foreign key)如果关系模式R1中的某属性集不是R1的主键，而是另一个关系R2的主键则该属性集是关系模式R1的外键

那么对于某个属性来说，如果这个属性存在于任一候选码中，它就称之为主属性。

非主属性 不包含在任何候选码中的属性称为非主属性。