数据库知识篇

数据库范式

第一范式

数据库表中的每一列都是不可再分的原子项。

第二范式

在第一范式的基础上，消除了非主属性对码的部分依赖。

第三范式

在第二范式的基础上，消除了传递依赖。

MySQL基础

视图

一种虚拟存在的表，对于使用视图的用户来说基本上是透明的。视图并不实际存在，行和列数据来自定义视图的查询中的使用表，并且是在使用视图时动态生存。

视图的优势：简单、安全、独立

存储过程和函数

存储过程和函数是事先经过编译并存储在数据库中的一段SQL语句的集合，调用存储过程和函数可以简化应用开发人员的很多工作，减少数据在数据和应用服务器之间的传输，对于提高数据处理的效率是有好处的。

存储过程和函数的区别在于函数必须有韩惠址，而存储过程没有，存储过程参数可以水用IN、OUT、INOUT类型，而函数的参数只能是IN类型的。库迁移到MySQL，那么久可能因此需要讲数据改造成存储过程。

触发器

触发其是与表有关的数据库对象，在满足定义条件时触发，并执行处罚器中定义的语句结合。触发其德这种特性可以协助应用在数据库端确保数据的完整性。

创建触发器的语法如下：

```
CREATE TRIGGER trigger_name trigger_time trigger_event
        ON tbl_name FOR EACH ROW trigger_stmt
```

> 注意：触发器只能创建在永久表（Permanent Table）上，不能对临时表（Temporary Table）创建触发器。

其中 trigger_time 是触发器的触发时间，可以是 BEFORE 或者 AFTER，BEFORE 的含义指在检查约束前触发，而 AFTER 是在检查约束后触发。
而 trigger_event 就是触发器的触发事件，可以是 INSERT、UPDATE 或者 DELETE。
对同一个表相同触发时间的相同触发事件，只能定义一个触发器。例如，对某个表的不同字

删除触发器

一次可以删除一个触发程序，如果没有指定 schema_name，默认为当前数据库,具体语法如下：

```
DROP TRIGGER [schema_name.]trigger_name
```

例如，要删除 film 表上的触发器 ins_film，可以使用以下命令：

```
mysql> drop trigger ins_film;
Query OK, 0 rows affected (0.00 sec)
```

1.1.MySQL逻辑架构

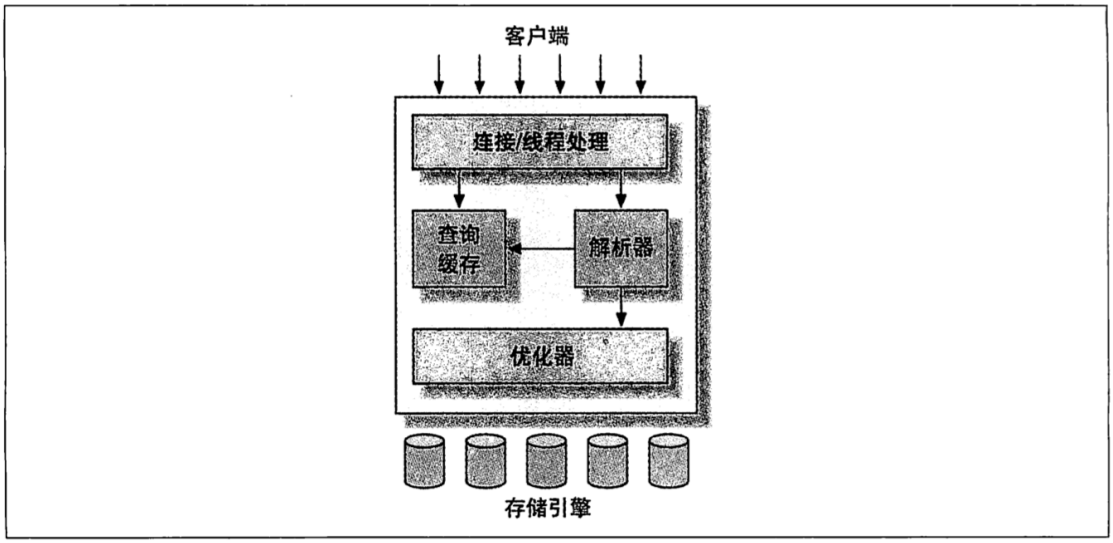

图1-1：MySQL服务器逻辑架构图

第一层给予网络的客服端/服务器的工具，连接处理、授权、安全

第二层：核心服务功能，查询解析、分析、优化、缓存，主要是存储过程、触发器、视图。

第三层：存储引擎，负责MySQL数据的存储和提取，支持事务。

1.1.1连接管理与安全性

每一个客户端连接购回在服务器进程中拥有一个线程，这个线程的查询只会在这个单独的线程中执行。服务器会缓存这些线程，因此不需要为每一个新建的连接创建或者销毁线程。连接基于SSL

1.1.2优化与执行

MySQL会解析查询，并创建内部数据结构(解析树)，然后对其进行各种有害，包括充血查询、决定表的读取顺序，以及选择合适的索引。

1.2并发控制

MySQL在两个层面的兵法控制：服务器和存储引擎层。

1.2.1读写锁

在处理兵法读和写时，可以通过实现一个由两种类型的锁组层的锁系统来解决问题。这两种类型的锁通常是共享锁和排他锁，也叫做读锁和写锁。

读锁是共享的，相互不阻塞，写锁是排他的。

1.2.2锁粒度

各种MySQL存储引擎都可以实现自己的锁策略和锁粒度。

表锁

MySQL基本策略，锁定整张表。写锁比读锁有更高的优先级。

行级锁

在数据存储层实现。

1.3事务

事务就是一组原子性的SQL查询，或者说是一个独立的单元。事务内的语句要么全部执行成功，要么全部执行失效。

ACID：

原子性：事务被视为一个不可分割的最小单元，要么全部成功，要么全部失败回滚。

一致性：数据库总是从一个一致性状态转化到另一个一致性状态。

隔离性：一个事务所做的修改在最终提交以前，对其他事务是不可见的。

持久性：一旦事务提交，则其所做的修改就会永久保存到数据库中。

1.3.1隔离级别

四种隔离级别

未提交读:事务可以读物为提交的数据，也即是脏读。

提交读：出现不可重复读问题

可重复读：幻读，指的是某个事务在读取某个范围内的记录，另一个事务又在该范围内插入新的记录，当前的事务读取该范围内的记录，会产生幻行。MySQL默认隔离级事务。

可窜性读：每一行数据加锁。导致大量的超时和锁争用。

!image-20180503091311083

1.3.2死锁

死锁是两个或者多个事务在同一资源是哪个互相占用，请求锁定对方占用的资源，从而导致恶心循环的现象。

InnoDB处理死锁的方法是，将持有最少行级排他锁的事务进行回滚。

1.3.3事务日志

使用事务日志，存储引擎在修改表的数据时只需要修改内存拷贝，再把修改该行的行为记录到持久在硬盘上的事务日志中。事务日志采用追加的方式。

1.3.4MySQL中的事务

自动提交

设置自动提交，只有显示遇到commit，才会结束一个事务。

在事务中混合使用存储引擎

事务是由下层的存储引擎实现的。

隐式和显示锁定

InnoDB采用两阶段锁定。

1.4多版本并发控制

为了提高性能，数据库都提供了多版本并发控制(MVCC)，避免了加锁操作；是通过保存数据在莫讵歌时间点的快找来实现的。

MVCC的实现有很多，最典型的有乐观并控制和悲观并发控制。

MVCC在可重复读中的实现，在每行记录中保存两个隐藏的实现，一个保存行的创建时间(系统版本号)，一个是过期时间(删除时间)。

SELECT语句：

1.InnoDB只查找版本早于当前事务版本的数据行

2.行的删除版本要么未定义要么大雨当亲啊事务版本号

INSERT：

InnoDB为新插入的每一行保存当前系统版本号座位行版本号

DELETE：

InnoDB为删除的每一行保存当前版本号作为行删除标识。

UPDATE：

InnoDB插入一行新纪录，保存当亲版本号为行版本号和行删除标识。

只有不可重复读和可重复读支持MVCC

1.5MySQL存储引擎

1.5.1InnoDB存储引擎

处理大量的短期事务。

InnoDB的数据存储在表空间，基于聚簇索引建立，存储格式是平台独立，采用可预测预读，在内存创建hash索引加速读操作的自适应哈希索引。

1.5.2MyISAM存储引擎

不支持事务和行级锁，存储在在数据文件和索引文件

特性：

加锁与并发：针对整个表加锁，读取加共享锁，写入加排他锁。

修复：手工或者自动执行检查和修复操作，可能导致数据丢失。

索引特性：支持全文索引，基于分词叉棍见索引，支持复杂查询。

延迟更新索引键：先跟心内存的键缓冲区没然后再将对应索引块写入磁盘。

MyISAM压缩表不支持修改，极大的减少空间占有，还有 就是磁盘IO，提高性能。

2.1索引基础

索引可以包含一个后者多个列的值。MySQL只能高效的使用索引最左边的前缀列。

2.1.1索引的类型

B—Tree索引

所有的值都是按顺序存储的，并且每一个叶子页到根的距离相同。

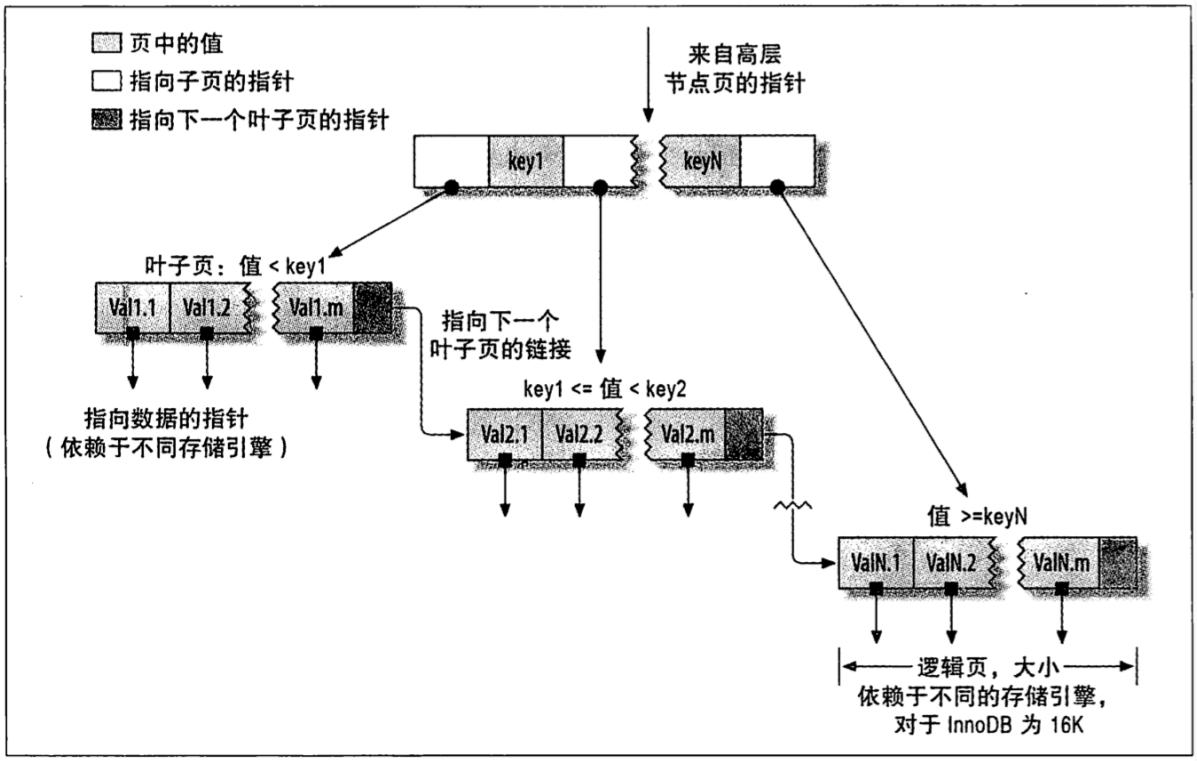

图5-1：建立在B-Tree结构（从技术上来说是B+Tree）上的索引

加速数据访问速度，索引对多个值进行排序依据是CREATE TABLE语句中定义索引时序的顺序。

B-Tree索引适合用于全键值、键值范围和键前缀查找。

B-Tree索引的限制

如果不是按照索引最左列开始查找，则无法使用索引。

不能跳过索引的列

如果查询中有某个列的范围查询，则其右边所有列无法使用索引优化查找。

Hash索引

基于hash表，只有精确匹配索引所有列的查询才有效。如果多个列hash值相同，那么就按照顺序以链表的形式存储。

```
mysql> SELECT * FROM testhash;
+--------+-----------+
| fname  | lname     |
+--------+-----------+
| Arjen  | Lentz     |
| Baron  | Schwartz  |
| Peter  | Zaitsev   |
| Vadim  | Tkachenko |
+--------+-----------+
```

假设索引使用假想的哈希函数 f()，它返回下面的值（都是示例数据，非真实数据）：

```
f('Arjen')= 2323
f('Baron')= 7437
f('Peter')= 8784
f('Vadim')= 2458
```

则哈希索引的数据结构如下：

槽（Slot）	值（Value）
2323	指向第 1 行的指针
2458	指向第 4 行的指针
7437	指向第 2 行的指针
8784	指向第 3 行的指针

所有的槽都是有顺序的

哈希索引的限制：

只含有哈希值和指针

哈希索引数据并不是按照索引值顺序存储

哈希索引不支持匹配查找

哈希索引只支持等值比较查询

hash索引非常快，除非有hash冲突

hash冲突过多，那么维护成本较高

InnoDB支持自适应哈希索引，根据使用情况建立哈希索引

空间数据索引(R-Tree)

MyISAM支持空间索引，可以用作地理数据存储。

全文索引

群文索引是一种特殊的类型索引，它查找的文本中的关键词，而不是直接比较索引中的值。

2.2索引的优点

除了加快查询之外，B-Tree索引，按照顺序存储数据，所有可以用来做OrderBy和GroupBy操作。

2.3高性能索引策略

几种优化方法

2.3.1独立的列

指索引列不能是表达式的一部分，也不是函数的参数。我们应该简化where条件的习惯，始终将索引列单独放在比较符的一侧

2.3.2前缀索引和索引选择性

有时候需要索引很长的字符列，这时可以索引开始的部分字符，这样可以大大节约索引空间。但是，这样也会降低索引的选择性。

索引的选择性是指，不重复的索引值，也称为基数和数据表记录总数(#T)的比值，范围从1/#T到1之间。

对于BOLB、TEXT或者很长的varchar类型，必须使用前缀索引

前缀索引应该足够长保证较高的选择性，同时也不能太长。

缺点是MySQL无法使用前缀索引在哦ORDER BY和GROUP BY，无法史依弘前缀索引做覆盖扫描。

2.3.3多列索引

索引机构合并策略：

将两个单列索引进行扫描后，将结果合并，有三种变种，OR条件的联合，AND条件的相交、组合其两种情况。

优化前

```
mysql> SELECT film_id, actor_id FROM sakila.film_actor
    -> WHERE actor_id = 1 OR film_id = 1;
```

优化后的SQL

```
mysql> SELECT film_id, actor_id FROM sakila.film_actor WHERE actor_id = 1
    -> UNION ALL
    -> SELECT film_id, actor_id FROM sakila.film_actor WHERE film_id = 1
    ->    AND actor_id <> 1;
```

索引合并策略有时候是一种优化结果，很湿实际上很多时候很糟糕：

1.服务器对多个索引做相交操作时

2.服务器多多个索引做联合操作时

3.优化器只关心随机页面读取

2.3.4选择合适的索引顺序

在一个多列B-Tree索引中，索引列的顺序意味着索引首先按照最左列进行排序，其次是第二列。

对于如下的查询语句

```
SELECT * FROM payment WHERE staff_id = 2 AND customer_id = 584;
```

应该如何的建立索引，选择索引的顺序呢?

我们可以使用where来查看数据基数有多大

```
mysql> SELECT SUM(staff_id = 2), SUM(customer_id = 584) FROM payment\G
*************************** 1. row ***************************
     SUM(staff_id = 2): 7992
SUM(customer_id = 584): 30
```

我们可以看到customer_id基数很小，我们应该将这个索引放在前面建立

我们来看看staff_id列的选择性如何

```
mysql> SELECT SUM(staff_id = 2) FROM payment WHERE customer_id = 584\G
*************************** 1. row ***************************
SUM(staff_id = 2): 17
```

这里我们可以看出查询的结果非常依赖于选定的具体值。

2.3.5聚簇索引

是一种数据存储方式。

InnoDB的聚簇索引实际上在同一个结构中保存了B-Tree索引和数据行。

一个表只能有一个聚簇索引。

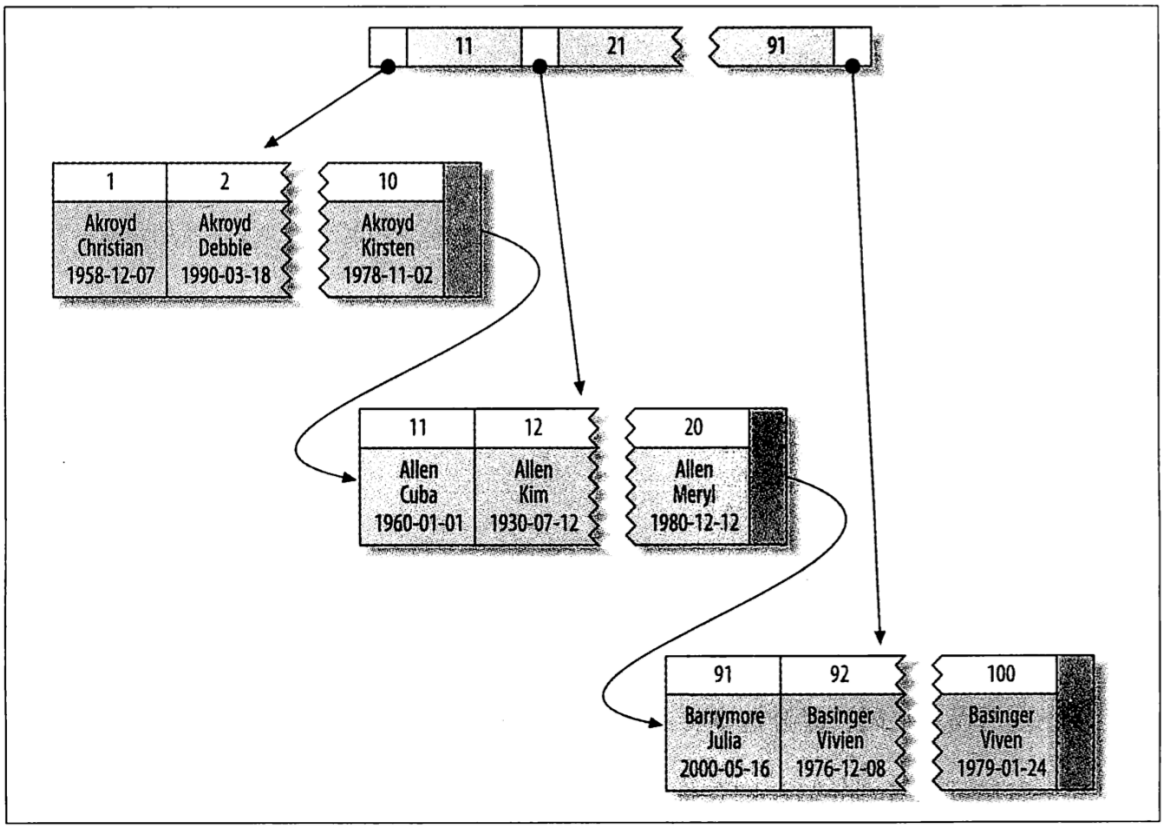

图5-3：聚簇索引的数据分布

InnoDB将通过主键聚集数据，这也就是说途中被索引的列示主键列。如果没有主键，会选择唯一非空索引代替。如果也没有上述规则，会隐式定义一个主键来座位聚簇索引。

InnoDB和MyISAM的数据分布

MyISAM可以从表的靠头跳过所需的字节找到需要的行。

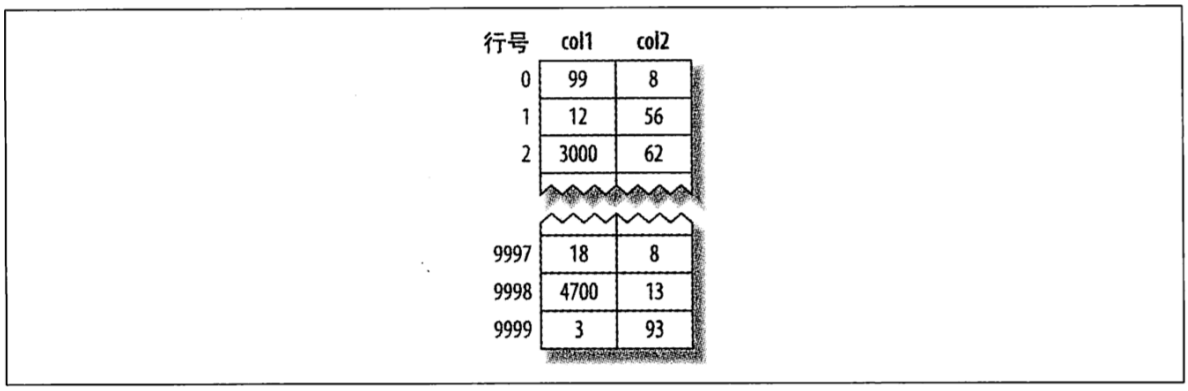

图5-4：MyISAM表layout_test的数据分布

主键分布

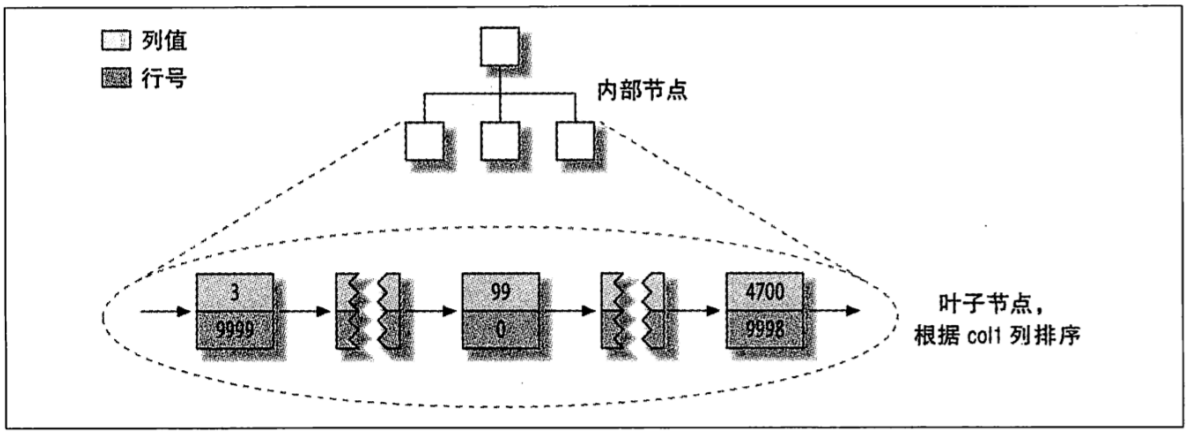

图5-5：MyISAM表layout_test的主键分布

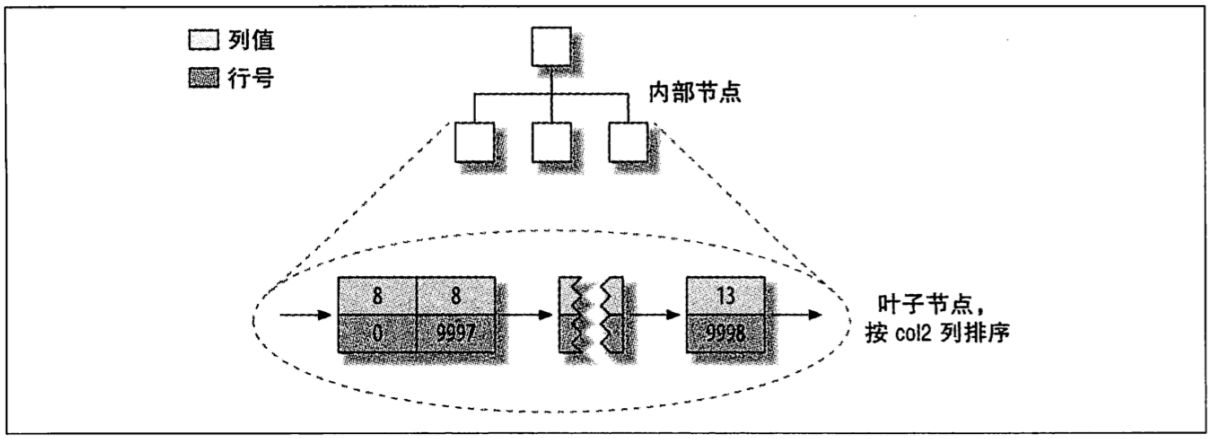

图5-6：MyISAM表layout_test的col2列索引的分布

InnoDB的数据分布

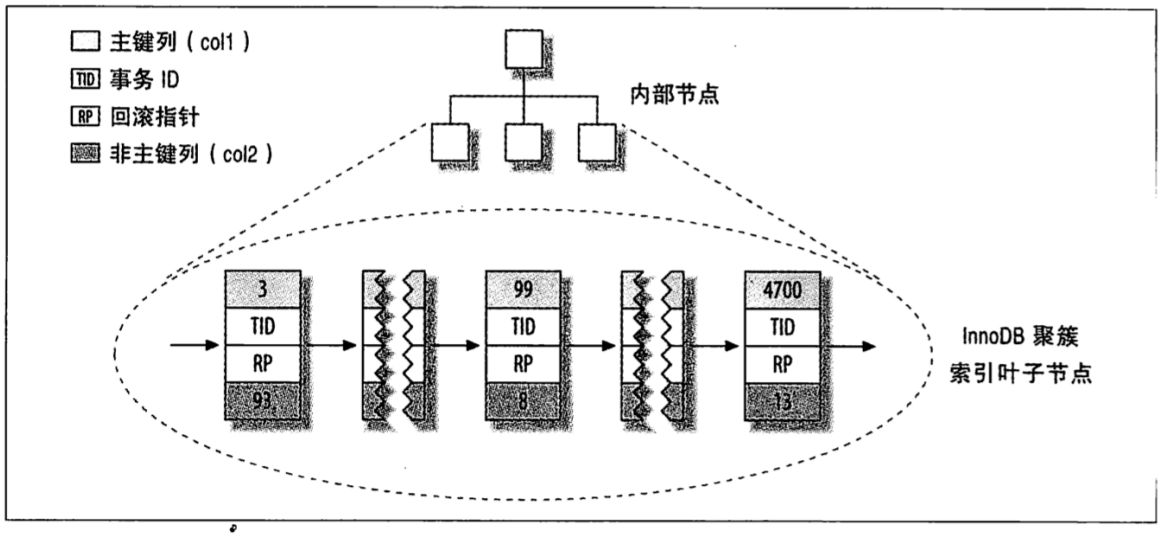

图5-7：InnoDB表layout_test的主键分布

这里看出InnoDB支持聚簇索引，存储的是整张表

与MyISAM不同的是，InnoDB的二级索引和聚簇索引很不相同。InnoDB的二级索引的叶子节点存储的不是行指针，而是主键。

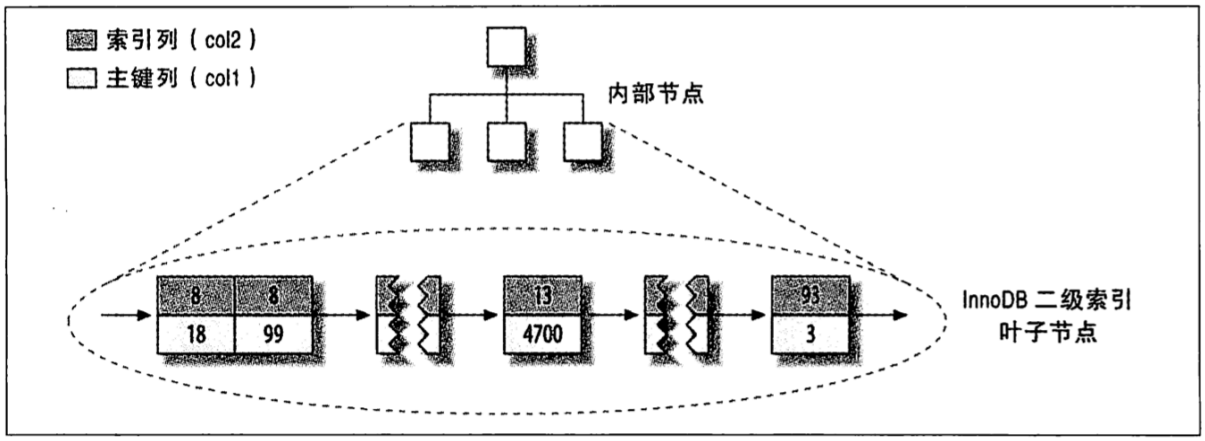

图5-8：InnoDB表layout_test的二级索引分布

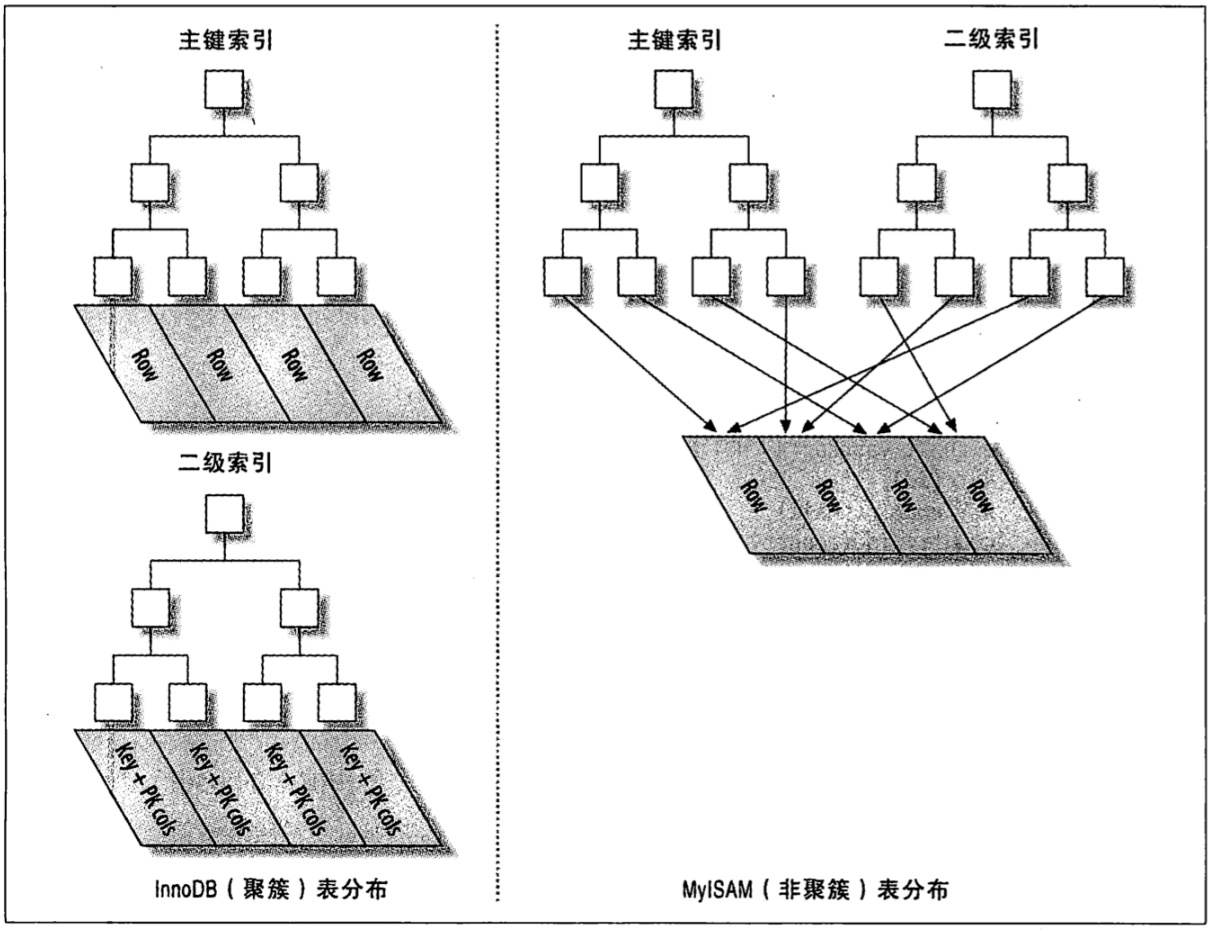

图5-9：聚簇和非聚簇表对比图

在InnoDB表中按主键顺序插入行

使用自增序列作为主键。

使用自增序列作为主键和使用UUID作为主键数据对比

表 5-1：向InnoDB表插入数据的测试结果

表名	行数	时间（秒）	索引大小（MB）
userinfo	1 000 000	137	342
userinfo_uuid	1 000 000	180	544
userinfo	3 000 000	1233	1036
userinfo_uuid	3 000 000	4525	1707

可以看出UUID作为主键时间更长，索引更大。

在高并发情况希，顺序主键可能导致明显的争用。

2.3.6覆盖索引

如果一个索引包含所需要查询的字段值，我们就称为覆盖索引。覆盖索引不需要存储索引列的值，而其然索引不存储索引列的值。MySQL只能使用B-Tree索引作为覆盖索引。

使用如下SQL语句

```
mysql> EXPLAIN SELECT * FROM products WHERE actor='SEAN CARREY'
    -> AND title like '%APOLLO%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: products
         type: ref
possible_keys: ACTOR,IX_PROD_ACTOR
          key: ACTOR
      key_len: 52
          ref: const
         rows: 10
        Extra: Using where
```

这里的索引无法覆盖改查询，原因

1，没有任何索引可以覆盖这个查询，但是可以使用where后面的语句

2.MySQL在索引中不能执行Like操作

先将索引扩展至覆盖三个数据列(arist,title,prod_id)

可以使用优化：

```
mysql> EXPLAIN SELECT *
    -> FROM products
    ->    JOIN (
    ->       SELECT prod_id
    ->       FROM products
    ->       WHERE actor='SEAN CARREY' AND title LIKE '%APOLLO%'
    ->    ) AS t1 ON (t1.prod_id=products.prod_id)\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: <derived2>
                ...omitted...
*************************** 2. row ***************************
           id: 1
  select_type: PRIMARY
        table: products
                ...omitted...
*************************** 3. row ***************************
           id: 2
  select_type: DERIVED
        table: products
         type: ref
possible_keys: ACTOR,ACTOR_2,IX_PROD_ACTOR
          key: ACTOR_2
      key_len: 52
          ref:
         rows: 11
        Extra: Using where; Using index
```

这种方式叫做延迟关联。

锁问题

3.1MySQL锁概述

MySQL支持3种锁的机制：

1.表级锁：开销小，加锁快；不会出现死锁，锁的力度大，发生冲突的概率最高并发度最低。

2.行级锁:开销大，加锁慢，会出现死锁，锁定粒度最小，发生锁冲突的概率最低，并发度最高。

3.页面锁:开销和枷锁时间介于表锁和行锁之间；会出现死锁；锁定粒度介于表锁和行锁之间，并发度一般。

3.2MyISAM表锁

MyISAM存储引擎只支持表锁。

MySQL的表级锁支持两种模式，表共享读锁和表独占写锁。

表 20-1　　MySQL 中的表锁兼容性

请求锁模式 / 是否兼容 / 当前锁模式	None	读锁	写锁
读锁	是	是	否
写锁	是	否	否

对MyISAM表的读操作，不会阻塞其他用户对同一表的读请求，但会组塞对同一表的写请求；对yISAM表的写操作，则会阻塞其他用户对同一表的读和写操作；MyISAM表的读操作与写操作之间，以及写操作之间是串行的。

MyISAM在执行查询语句之前，会自动给涉及的所有表加读锁，在执行更新操作前，会自动给涉及的表加写锁，这个过程并不需要用户单路，硬刺，用户一般不需要直接用Lock TABLE命令给MyISAM表显示加锁。

MyISAM不支持表升级。

MyISAM总是一次获得SQL语句所需要额全部锁，这也正是MyISAM表不会出现死锁的原因。

MyISAM总是写进程先获得锁。不仅如此，即使读请求先到锁等待队列，写请求后到，写锁也会插到读锁请求之前。

InnoDB锁问题

事务是由一组SQL语句组成的逻辑单元，事务具有4个属性，通常简称事务的ACID属性。

原子性

一致性

隔离行

持久性

并发事务处理带来的问题，主要有以下几个问题：

更新丢失：多个事务同时更新同一行，然后基于最初选定的值更新改行时，对于每个事务都不知道其他事务的存在，然后发生丢失更新问题。

脏读：一个事务对一条记录做下改，在这个事务完成并提交前，这条记录的数据就处于不一致状态；这时另一个事务也来读取同一条记录了如果不佳控制，第二个事务读取这些脏数据，这就是脏读。

不可重复读：一个书屋在读取某些数据后的某个时间，再次读取以前读过的数据，去发现读取的数据已经发生了改变、或某些记录已经被扇了。

幻读:一个事务按相同的条件重新读取以前检索过的数据，却发现其他事务插入满足其查询条件的新数据。

事务隔离级别

数据库实现事务隔离的方式，基本分为两种：

1.读取数据，对其加锁，阻止其他事务怼数据进行修改。

2.另一种时不加锁，通过一定机制生成一个数据请求时间点的一致性数据快照，并用这个快照来提供一定级别的一致性读取。

以上两种技术叫做数据多版本并发控制MVCC。

表 20-5　　4 种隔离级别比较

读数据一致性及允许的并发副作用 / 隔离级别	读数据一致性	脏读	不可重复读	幻读
未提交读（Read uncommitted）	最低级别，只能保证不读取物理上损坏的数据	是	是	是
已提交度（Read committed）	语句级	否	是	是
可重复读（Repeatable read）	事务级	否	否	是
可序列化（Serializable）	最高级别，事务级	否	否	否

InnoDB行锁实现方式

InnoDB行锁是通过个索引上的索引项来实现的。

InnoDB这种行锁实现的特点是只有通过索引条件检索数据，InnoDB才会使用行级锁，构造InnoDB将会使用表级锁。

需要注意的是：

1.在不通过索引条件查询的时候，InnoDB确定使用的是表锁，而不是行锁。

2.由于MySQL的行锁是针对索引加的锁，不是针对记录加的锁，所以虽然访问不同行的记录，但是如果使用相同的索引键，会出现锁冲突。

3.当表有多个索引的时候，不同的事务可以使用不同的索引锁定不同的行，宁外，不论是使用主键索引、唯一索引或普通索引，InnB都会使用行锁对数据加锁。

4.即便在条件中使用了索引字段，但是否使用索引来检索数据由MySQL通过判断不同执行计划的代价来决定，如果MySQL认为全表扫描效率更高，比如对一些很小的表，他就不会使用索引，这种情况下InnoDB将会使用表锁，而不是行锁。

间隙锁

当我们用范围条件而不是想等条件检索数据，并请求共享或排他锁时，InnoDB会给符合条件的已有数据记录的索引加锁；对于简直在数据范围内但并不存在的记录，叫做间隙，innoDB也会对这个间隙加锁，这种锁机制就是所谓的间隙锁。

在使用范围条件检索并锁定记录时，InnoDB这种加锁机制会阻塞符合条件范围内的并发插入，这往往会造成严重的锁等待。

InnoDB除了通过范围条件加锁时，使用间隙锁外，如果使用想等条件求给一个不存在的记录加锁，InnoDB也会使用间隙锁。

什么情况使用表锁

事务需要更新大部分或者全部数据，表又比较大，如果使用默认的行锁，不仅这个事务执行效率低，而且可能造成其他事务长时间锁等待和锁冲突，这种情况应该使用表锁。

事务涉及多个表，比较复杂，很可能引起死锁，造成大量事务回滚。

SQL优化

大批量插入数据

当用load命令导入数据的时候，适当的设置可以提高导入的速度。

1.导入的数据如果按照主键排列，那么可以可以有效的提高导入数据的效率。

2.在导入数据前执行set unique_checks=0关闭唯一性校验，在导入结束后执行set unique_checks =1,恢复唯一性校验，可以提高导入的效率。

3.导入数据之前关闭，自动提交，导入后打开，也可以提高导入的效率。

优化INSERT语句

1.当同一个客户端插入多行的时候，使用多个指标的insert语句进行，可以减少资源消耗。

2.如果不同客户插入很多行，可以使用insert delayed语句得到很高的速度。数据放在内存

3.索引文件和数据文件分别放在不同的磁盘存放。

优化GROUP BY语句

如故宫查询包括GROUP BY但用户想要避免排序结果的消耗，则可以指定ORDER By NULL禁止排序。

优化嵌套查询

1.不要使用 not in 可以使用join代替，比如

select * from sales2 where company_id not in (select id from company2)

优化后

select * from sales2 join on company2 on sales2.company_id=company2.id where sales2.company_id is null

优化OR查询语句

对于含有OR的查询子句，如果利用索引，则OR之间的每个条件列出必须用到索引，如果没有索引，则可以增加索引。