MySQL

点滴学习，随时记录

mysql日期和时间类型

mysql有5种表示时间值的日期和时间类型，

分别为、`DATE, TIME, YEAR, DATETIME, TIMESTAMP`。

注意：

`TIMESTAMP`类型有专有的自动更新特性

类型	大小 (字节)	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 00:00:00/2037 年某时	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

TIMESTAMP时间戳在创建的时候可以有多重不同的特性，如：

```
1.在创建新记录和修改现有记录的时候都对这个数据列刷新：

TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

2.在创建新记录的时候把这个字段设置为当前时间，但以后修改时，不再刷新它：

TIMESTAMP DEFAULT CURRENT_TIMESTAMP

3.在创建新记录的时候把这个字段设置为0，以后修改时刷新它：

TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

4.在创建新记录的时候把这个字段设置为给定值，以后修改时刷新它：

TIMESTAMP DEFAULT 'yyyy-mm-dd hh:mm:ss' ON UPDATE CURRENT_TIMESTAMP
```

TIMESTAMP列类型：

TIMESTAMP值可以从1970的某时的开始一直到2037年，精度为一秒，其值作为数字显示。
TIMESTAMP值显示尺寸的格式如下表所示：

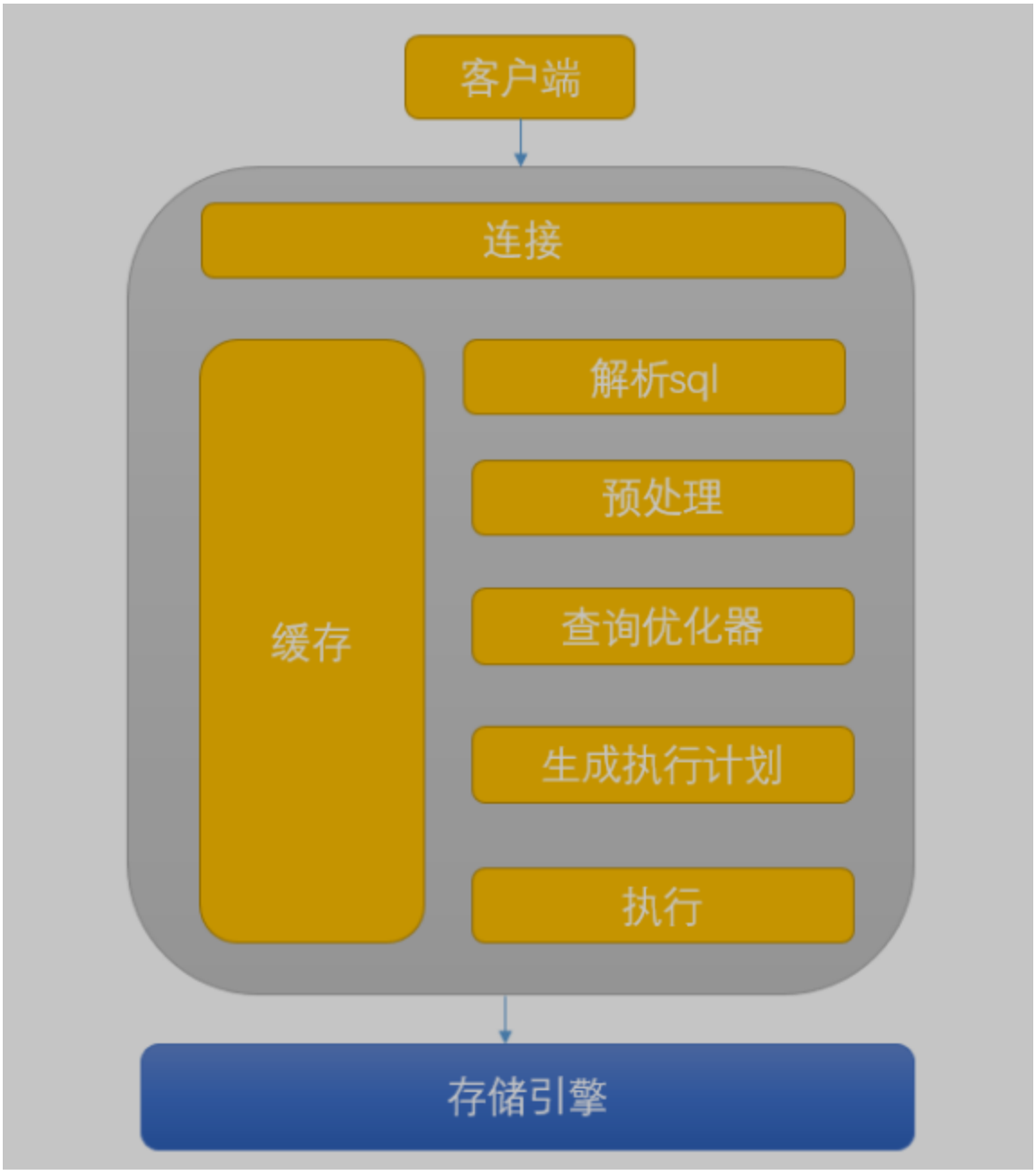
```
+---------------+----------------+
| 列类型         | 显示格式         |
| TIMESTAMP(14) | YYYYMMDDHHMMSS |
| TIMESTAMP(12) | YYMMDDHHMMSS   |
| TIMESTAMP(10) | YYMMDDHHMM     |
| TIMESTAMP(8)  | YYYYMMDD       |
| TIMESTAMP(6)  | YYMMDD         |
| TIMESTAMP(4)  | YYMM           |
| TIMESTAMP(2)  | YY             |
+---------------+----------------+
```

```

“完整”TIMESTAMP格式是14位，但TIMESTAMP列也可以用更短的显示尺寸，创造最常见的显示尺寸是6、8、12、和14。
你可以在创建表时指定一个任意的显示尺寸，但是定义列长为0或比14大均会被强制定义为列长14。
列长在从1～13范围的奇数值尺寸均被强制为下一个更大的偶数。
```

MySql优化建议总结和注意事项

MySQL逻辑架构：



有三层结构：

> 第一层：客户端通过连接服务，将要执行的sql指令传输过来

> 第二层 （分为两种方式）

> 方式一： 服务器解析并优化sql，生成最终的执行计划并执行

> 方式二： 服务器从缓存中获取查询结果

> 第三层：存储引擎，负责数据的储存和提取

优化建议总结：

- 对于查询缓存
  查询缓存的空间不要设置的太大

  原因： 因为查询缓存是靠一个全局锁操作保护的，如果查询缓存配置的内存比较大且里面存放了大量的查询结果，当查询缓存失效的时候，会长时间的持有这个全局锁。因为查询缓存的命中检测操作以及缓存失效检测也都依赖这个全局锁，所以可能会导致系统僵死的情况 对于写密集型的应用，直接禁用查询缓存

  原因：如果一个表被更改了，那么使用那个表的所查询缓存将不再有效，并且从缓冲区中移出。频繁移除缓存会消耗大量时间
  mysql缓存相关知识如 缓存配置， 使用监控，优化等有兴趣可自己查一下，包括很多内容
- 对于数据类型
  尽量使用对应的数据类型。比如，不要用字符串类型保存时间，用整型保存IP。
  选择更小的数据类型。能用TinyInt不用Int。
- 对于具体查询
  避免查询无关的列，如使用Select * 返回所有的列。

> 从数据库里读出越多的数据，那么查询就会变得越慢。并且，如果你的数据库服务器和WEB服务器是两台独立的
> 服务器的话，这还会增加网络传输的负载。所以，你应该养成一个需要什么就取什么的好的习惯。

- 避免查询无关的行
  切分查询。将一个对服务器压力较大的任务，分解到一个较长的时间中，并分多次执行。如要删除一万条数据，可以分10次执行，每次执行完成后暂停一段时间，再继续执行。过程中可以释放服务器资源给其他任务。
  分解关联查询。将多表关联查询的一次查询，分解成对单表的多次查询。可以减少锁竞争，查询本身的查询效率也比较高。因为MySql的连接和断开都是轻量级的操作，不会由于查询拆分为多次，造成效率问题。

注意count的操作只能统计不为null的列，所以统计总的行数使用count（*）。
group by 按照标识列分组效率高，分组结果不宜出行分组列之外的列。
关联查询延迟关联，可以根据查询条件先缩小各自要查询的范围，再关联。
Limit分页优化。可以根据索引覆盖扫描，再根据索引列关联自身查询其他列。

> 原理：
> 利用表的覆盖索引来加速分页查询。
> 索引查询的语句中如果只包含了那个索引列（覆盖索引），那么这种情况会查询很快。
> SELECT * FROM product a JOIN (select id from product limit 866613, 20) b ON a.ID = b.id

- Union查询默认去重，如果不是业务必须，建议使用效率更高的Union All
  只查询一条数据时，加 limit 1

> 当你查询表的有些时候，你已经知道结果只会有一条结果,在这种情况下，加上 LIMIT 1
> 可以增加性能。这样MySQL数据库引擎会在找到一条数据后停止搜索，而不是继续往后查找下一条符合记录的数据。

- 千万不要 ORDER BY RAND()

> 想打乱返回的数据行？随机挑一个数据？这样使用只让你的数据库的性能呈指数级的下降。这里的问题是：MySQL会不得 不去执行RAND()函数（很耗CPU时间），而且这是为了每一行记录去记行，然后再对其排序。就算是你用了Limit 1也无济于事（因为要排序）

- 我们应该为数据库里的每张表都设置一个ID做为其主键，而且最好的是一个INT型的（推荐使用UNSIGNED），并设置上自动增加的AUTO_INCREMENT标志。

> 就算是你 users 表有一个主键叫 “email”的字段，你也别让它成为主键。使用 VARCHAR 类型来当主键会使用得性能下降。另外，在你的程序中使用表的ID来构造你的数据结构。

- 选择正确的存储引擎

> 在 MySQL 中常用的有两个存储引擎 MyISAM 和 InnoDB，每个引擎都有利有弊。
> MyISAM 适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要update一个字段，整个表都会被锁起来，而别的进程，就算是读进程都 无法操作直到读操作完成。
> InnoDB 的趋势会是一个非常复杂的存储引擎，对于一些小的应用，它会比 MyISAM 还慢。他是它支持“行锁”，于是在写操作比较多的时候，会更优秀。并且，他还支持更多的高级应用，比如：事务。
> InnoDB的性能更加平衡。
> 老版本中mysql默认搜索引擎是MyISAM， 现在已经都改为innodb。

- 使用INNER JOIN 而不是WHERE来创建连接(多表查询)

> 这个笛卡尔连接问题， 若量表各有100数据， where 产生 100 * 100 = 10000 条数据，再根据where条件过滤
> 而 INNER JOIN 只产生 100 条结果

- mysql插入
  一条SQL语句插入多条数据

```
1 INSERT INTO `insert_table` (`datetime`, `uid`, `content`, `type`)
```

```
2 VALUES ('0', 'userid_0', 'content_0', 0), ('1', 'userid_1', 'content_1',
   1);
```

合并后日志量（MySQL的binlog和innodb的事务日志）减少了，降低日志刷盘的数据量和频率，从而提高效率，同时也能减少SQL语句解析的次数，减少网络传输的IO。

- 数据有序插入

  数据有序的插入是指插入记录在主键上是有序排列，dateid：

```
1 INSERT INTO `insert_table` (`dateid`, `uid`, `content`, `type`) VALUES
   ('0', 'userid_0', 'content_0', 0);
2
3
4 INSERT INTO `insert_table` (`dateid`, `uid`, `content`, `type`) VALUES
   ('1', 'userid_1', 'content_1', 1);
5
6
7 INSERT INTO `insert_table` (`dateid`, `uid`, `content`, `type`) VALUES
   ('2', 'userid_2', 'content_2',2);
```

由于数据库插入时，需要维护索引数据，无序的记录会增大维护索引的成本。我们可以参照innodb使用的B+tree索引，如果每次插入记录都在索引的最后面，索引的定位效率很高，并且对索引调整较小；如果插入的记录在索引中间，需要B+tree进行分裂合并等处理，会消耗比较多计算资源，并且插入记录的索引定位效率会下降，数据量较大时会有频繁的磁盘操作。 建表时最好以自增id为主键就解决了这个问题。

- 主从同步，读写分离

MySQL主从同步的作用：
1、可以作为一种备份机制，相当于热备份（在从备份，避免备份期间影响主服务器服务）
2、可以用来做读写分离，均衡数据库负载(主写从读)
3、当主服务器出现问题时，可以切换到从服务器。

- 分库分表

  对表进行水平划分

如果一个表的记录数太多了，比如上千万条，而且需要经常检索，那么我们就有必要化整为零了。如果我拆成100个表，那么每个表只有10万条记录。当然这 需要数据在逻辑上可以划分。一个好的划分依据，有利于程序的简单实现，也可以充分利用水平分表的优势。比如系统界面上只提供按月查询的功能，那么把表按月 拆分成12个，每个查询只查询一个表就够了。

- 对表进行垂直划分

有些表记录数并不多，可能也就2、3万条，但是字段却很长，表占用空间很大，检索表时需要执行大量I/O，严重降低了性能。这个时候需要把大的字段拆分到另一个表，并且该表与原表是一对一的关系。

> （降低了每张表的文件大小） 实例：文章信息表就可以拆分成两张表，一张记录文章信息如作者、栏目、发表日期、关键词等，另一张表记录 文章内容，摘要等。利用id进行对应。因为在大多数情况下只需要搜索第一张表即可，只有在文章展示页才需要访问两张表，如此可以提高搜索性能!

- 水平分库分表

> 水平分库分表与上面讲到的水平分表的思想相同，唯一不同的就是将这些拆分出来的表保存在不同的数据中。
> 某种意义上来讲，有些系统中使用的“冷热数据分离”（将一些使用较少的历史数据迁移到其他的数据库中。而在业务功能上，通常默认只提供热点数据的查询），也是类似的实践。在高并发和海量数据的场景下，分库分表能够有效缓解单机和单库的性能瓶颈和压力，突破IO、连接数、硬件资源的瓶颈。

- 垂直分库

> 垂直分库基本的思路就是按照业务模块来划分出不同的数据库，而不是像早期一样将所有的数据表都放到同一个数据库中。

**count() *优化：*

```
有时候某些业务场景并不需要完全精确的COUNT值，可以用近似值来代替，EXPLAIN出来的行数就是一个不错的近似值，而且执行EXPLAIN并不需要真正地去执行查询，所以成本非常低。
实例：

[SQL]select count(name) from aaa;
受影响的行: 0
时间: 2.957s

结果:
428396

[SQL]EXPLAIN select count(name) from aaa;
受影响的行: 0
时间: 0.038s

结果:
1   SIMPLE  aaa index       name_idx    767     413996  Using index
```

```
通常来说，执行COUNT()都需要扫描大量的行才能获取到精确的数据，因此很难优化，MySQL层面还能做得也就只有覆盖索引了。

```

如果不还能解决问题，只有从架构层面解决了，比如添加汇总表，或者使用redis这样的外部缓存系统。

关联查询优化：

在大数据场景下，表与表之间通过一个冗余字段来关联，要比直接使用JOIN有更好的性能。如果确实需要使用关联查询的情况下，需要特别注意的是：

1、确保ON和USING字句中的列上有索引。在创建索引的时候就要考虑到关联的顺序。当表A和表B用列c关联的时候，如果优化器关联的顺序是A、B，那么就不需要在A表的对应列上创建索引。

2、没有用到的索引会带来额外的负担，一般来说，除非有其他理由，只需要在关联顺序中的第二张表的相应列上创建索引。

3、确保任何的GROUP BY和ORDER BY中的表达式只涉及到一个表中的列，这样MySQL才有可能使用索引来优化。

union 优化：

MySQL处理UNION的策略是先创建临时表，然后再把各个查询结果插入到临时表中，最后再来做查询。因此很多优化策略在UNION查询中都没有办法很好的时候。经常需要手动将WHERE、LIMIT、ORDER BY等字句“下推”到各个子查询中，以便优化器可以充分利用这些条件先优化。

除非确实需要服务器去重，否则就一定要使用UNION ALL，如果没有ALL关键字，MySQL会给临时表加上DISTINCT选项，这会导致整个临时表的数据做唯一性检查，这样做的代价非常高。当然即使使用ALL关键字，MySQL总是将结果放入临时表，然后再读出，再返回给客户端。虽然很多时候没有这个必要，比如有时候可以直接把每个子查询的结果返回给客户端。

大表 alter table：

> 大表ALTER TABLE非常耗时，MySQL执行大部分修改表结果操作的方法是用新的结构创建一个张空表，从旧表中查出所有的数据插入新表，然后再删除旧表。

> 尤其当内存不足而表又很大，而且还有很大索引的情况下，耗时更久。当然有一些奇淫技巧可以解决这个问题，有兴趣可自行查阅。

> 比如：大表更改默认值使用alter table不重建表，直接修改.frm
> 在MySQL中执行很大部分的修改动作，都需要重建一个表，然后把数据放进去，最后删除旧的表！有时候要是有索引的列上进行大批且频繁的表的时候会导致系统的性能严重下降，这里可以在修改SQL上做部分调整，减轻相关的构建结构带来的系统压力问题！
> 例如 在修改一个表的默认值为8的时候，常规做法为：
> （1）：alter table modes modify column dept tinyint(3) not null default 8；
> 这里通过一些show status分析出，每次都要做大量的句柄的读取和插入操作，类似于从新构建了一张新表的样式，这样在一些大表，上千万的数据量会出现瓶颈问题！
> 这里我们需要灵活知道mysql的相关默认值实际是放在相关的表结构.frm文件中；我们可以不经过数据层，可以直接调整！上述的modify column会导致相关的表进行拷贝操作，不利于系统的正常稳定运行，可以使用下面方式；
> （2）：alter table modes alter column dept set default 8;
> 这里只是更改了相关的frm文件，没有改动表，因此速度很快的即可完成；

> 总结：在此我们可以注意到相关的alter table后面跟不同形式的命令，可以对数据产生了不同程度的影响，这里还有一个change column操作也是不一样的！

表命名问题

> 1、数据库表名创建时使用小写英文字母加下划线的形式，不要出现大写字母，防止大小写敏感问题（数据库、字段创建最好也遵守这一规则，保持统一）

MySQL不能远程连接解决方法

mysql不能远程连接主要从以下3个方面着手解决：

1、如果是云服务器检查其安全组规则，查看一下是否放行了3306端口。

2、在mysql库user表中添加一个用户 主机为 % 的任意主机（也可以编辑已有的用户），如果要指定IP远程访问，创建一个 主机 为你指定IP的用户即可。

```
# 创建并授权可远程访问root用户
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '111111' WITH GRANT OPTION;

# 刷新MySQL的系统权限相关表，使设置生效
FLUSH PRIVILEGES;
```

3、检查iptables，删除DROP 3306端口的规则。参考如下文章：

Here is yuque doc card, click on the link to view:https://www.yuque.com/u316337/tforki/woqd87

MySQL 共享锁&排他锁

共享锁(lock in share mode)

允许不同事务之间共享加锁读取，但不允许其它事务修改或者加入排他锁;如果有修改必须等待一个事务提交完成，才可以执行，容易出现死锁

共享锁事务之间的读取

session1:

```
1 start transaction;
2 select * from test where id = 1 lock in share mode;
```

session2:

```
1 start transaction;
2 select * from test where id = 1 lock in share mode;
```

此时session1和session2都可以正常获取结果，那么再加入session3 排他锁读取尝试

session3:

```
1 start transaction;
2 select * from test where id = 1 for update;
```

在session3中则无法获取数据，直到超时或其它事物commit

```
1 Lock wait timeout exceeded; try restarting transaction
```

共享锁之间的更新

当session1执行了修改语句：
session1:

```
1 update test set name = 'kkkk' where id = 1;
```

可以很多获取执行结果。
当session2再次执行修改id=1的语句时：
session2:

```
1 update test set name = 'zzz' where id = 1;
```

就会出现死锁或者锁超时，错误如下：

```
1 Deadlock found when trying to get lock; try restarting transaction
```

或者：

```
1 Lock wait timeout exceeded; try restarting transaction
```

必须等到session1完成commit动作后，session2才会正常执行，如果此时多个session并发执行，可想而知出现死锁的几率将会大增。

session3则更不可能

结论：

mysql共享锁(lock in share mode)

```
1 允许其它事务也增加共享锁读取
2 不允许其它事物增加排他锁(for update)
```

```
当事务同时增加共享锁时候，事务的更新必须等待先执行的事务commit后才行，
如果同时并发太大可能很容易造成死锁
共享锁，事务都加，都能读。修改是惟一的，必须等待前一个事务commit，才可
```

排他锁(for update)

当一个事物加入排他锁后，不允许其他事务加共享锁或者排它锁读取，更加不允许其他事务修改加锁的行。

排他锁不同事务之间的读取

同样以不同的session来举例
session1:

```
start transaction;
select * from test where id = 1 for update;
```

session2:

```
start transaction;
select * from test where id = 1 for update;
```

当session1执行完成后，再次执行session2，此时session2也会卡住，无法立刻获取查询的数据。直到出现超时

```
Lock wait timeout exceeded; try restarting transaction
```

或session1 commit才会执行

那么再使用session3 加入共享锁尝试

```
select * from test where id = 1 lock in share mode;
```

结果也是如此，和session2一样，超时或等待session1 commit

```
Lock wait timeout exceeded; try restarting transaction
```

排他锁事务之间的修改
当在session1中执行update语句：

```
update test set name = 123 where id = 1;
```

可以正常获取结果

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

此时在session2中执行修改

```
update test set name = 's2' where id = 1;
```

则会卡住直接超时或session1 commit,才会正常吐出结果

session3也很明显和session2一样的结果，这里就不多赘述

总结

```
事务之间不允许其它排他锁或共享锁读取，修改更不可能
一次只能有一个排他锁执行commit之后，其它事务才可执行
不允许其它事务增加共享或排他锁读取。修改是惟一的，必须等待前一个事务commit，才可
```

转自：https://laravel-china.org/index.php/articles/12800/lock-in-share-mode-mysql-shared-lock-exclusive-lock-for-update

mysql定位和分析执行效率方法

```
1. explain(mysql),查看sql语句执行计划；

2. 启用slow query log记录慢查询；

3. 通常还要看数据库设计是否合理，需求是否合理等。
```

mysql 事务特性以及隔离级别说明

事务：

事务就是针对数据库的一组操作，它可以由一条或者多条SQL语句组成，同一个事务的操作具备同步的特点，如果其中有一条语句不能执行的话，那么所有的语句都不会执行，也就是说，事务中的语句要么都执行，要么都不执行。

事务特性

事务操作具有严格的定义，它必须满足ACID：

ACID，指数据库事务正确执行的四个基本要素的缩写。包含：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。

`原子性：` 原子性是指事务是一个不可再分割的工作单位，事务中的操作要么都发生，要么都不发生。

`一致性：` 一致性是指在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。这是说数据库事务不能破坏关系数据的完整性以及业务逻辑上的一致性。

即事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

`隔离性：` 隔离性是指并发的事务是相互隔离的。即一个事务内部的操作及正在操作的数据必须封锁起来，不被企图进行修改的事务看到 。

`持久性：` 持久性是指在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。 即使出现了任何事故比如断电等，事务一旦提交，则持久化保存在数据库中。

隔离性说明：

隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

不考虑事务的隔离性会发生什么问题?

`脏读：` 脏读就是一个事务读取到了另一个事务还未提交的数据，另一个事务中数据可能进行了回滚,此时A事务读取的数据可能和数据库中数据是不一致的，此时认为数据是脏数据，读取脏数据过程叫做脏读。

`不可重复读：` 两次读取的数据不一致(表现在数据更新，数据内容不一致，update)，当事务A第一次读取事务后，事务B对事务A读取的数据进行修改，事务A中再次读取的数据和之前读取的数据不一致，此过程称为不可重复读。

`虚读（幻读）：` 两次读取的数据一致(表现在数据新增或删除，数据量不一致，insert & delete)，事务A按照特定条件查询出结果，事务B新增了一条符合条件的数据。事务 A 中查询的数据和数据库中的数据不一致的，事务 A 好像出现了幻觉，这种情况称为幻读。 主要针对的操作是新增或删除。

`丢失更新：` 两个事务对同一条记录进行操作，后提交的事务，将先提交的事务的修改的数据覆盖了

> 为了防止出现脏读、不可重复读、幻读等情况，我们就需要根据我们的实际需求来设置数据库的隔离级别。

事务的隔离级别

隔离级别	含义
Serializable	可避免脏读、不可重复读、虚读情况的发生。（串行化）
Repeatable read	可避免脏读、不可重复读情况的发生。（可重复读）不可以避免虚读
Read committed	可避免脏读情况发生（读已提交）
Read uncommitted	最低级别，以上情况均无法保证。(读未提交)

如何使用这些隔离级别，那就需要根据业务的实际情况来进行判断了。

读未提交（Read Uncommitted）

读未提交，顾名思义，就是可以读到未提交的内容。

因此，在这种隔离级别下，查询是不会加锁的，也由于查询的不加锁，所以这种隔离级别的一致性是最差的，可能会产生“脏读”、“不可重复读”、“幻读”。

如无特殊情况，基本是不会使用这种隔离级别的。

读提交（Read Committed）

读提交，顾名思义，就是只能读到已经提交了的内容。

这是各种系统中最常用的一种隔离级别，也是SQL Server和Oracle的默认隔离级别。这种隔离级别能够有效的避免脏读，但除非在查询中显示的加锁，如：

```
select * from T where ID=2 lock in share mode;

select * from T where ID=2 for update;
```

不然，普通的查询是不会加锁的。

那为什么“读提交”同“读未提交”一样，都没有查询加锁，但是却能够避免脏读呢?

这就要说道另一个机制“快照（snapshot）”，而这种既能保证一致性又不加锁的读也被称为“快照读（Snapshot Read）”

假设没有“快照读”，那么当一个更新的事务没有提交时，另一个对更新数据进行查询的事务会因为无法查询而被阻塞，这种情况下，并发能力就相当的差。

而“快照读”就可以完成高并发的查询，不过，“读提交”只能避免“脏读”，并不能避免“不可重复读”和“幻读”。

可重复读(Repeated Read)

可重复读，顾名思义，就是专门针对“不可重复读”这种情况而制定的隔离级别，自然，它就可以有效的避免“不可重复读”。而它也是MySql的默认隔离级别。

在这个级别下，普通的查询同样是使用的“快照读”，但是，和“读提交”不同的是，当事务启动时，就不允许进行“修改操作（Update）”了，而“不可重复读”恰恰是因为两次读取之间进行了数据的修改，因此，“可重复读”能够有效的避免“不可重复读”，但却避免不了“幻读”，因为幻读是由于“插入或者删除操作（Insert or Delete）”而产生的。

`串行化（Serializable）`

这是数据库最高的隔离级别，这种级别下，事务“串行化顺序执行”，也就是一个一个排队执行。

这种级别下，“脏读”、“不可重复读”、“幻读”都可以被避免，但是执行效率奇差，性能开销也最大，所以基本没人会用。

隔离级别的设置与查询：

1、设置隔离级别：

设置隔离级别分为设置全局的隔离级别与设置当前的隔离级别

全局设置，已存在的session不会生效，以后的新session会生效（以读未提交举例）：

```
1 set global transaction isolation level read uncommitted;
```

单独设置当前连接：

```
1 set session transaction isolation level read uncommitted;
```

2 、在MySQL数据库中 查看 当前事务的隔离级别：

```
1  select @tx_isolation;
```

count(*)与group by的统计问题

sql统计示例：

统计 group by 后每个分组的个数：

```
SELECT COUNT(*) FROM search_word GROUP BY word;
```

那么得到的结果是:

```
COUNT(*)
1
10
1
2
1
1
1
15
```

统计 group by 后分组个数

```
SELECT COUNT(DISTINCT word) FROM search_word;
```

结果：

```
COUNT(DISTINCT word)
346
```

或

```
SELECT COUNT(*) FROM (SELECT COUNT(*) FROM search_word GROUP BY word) t;
```

结果：

```
COUNT(*)
346
```

mysql 字段类型说明和推荐

`char(M)`和`varchar(M)`的区别：

char的长度是不可变的，而varchar的长度是可变的;

char(M)定义的列的长度为固定的，M取值可以为0～255之间；
varchar(M)定义的列的长度为可变长，M取值可以为0~65535之间;

定义一个char[10]和varchar[10],如果存进去的是‘abcd’,那么char所占的长度依然为10，除了字符‘abcd’外，后面跟六个`空格`，而varchar就立马把长度变为4了.

存数据时char类型数据后面的空格会被删除，varchar原样存储；取数据的时候，char类型尾部自动添加的空格会被删除，而varchar尾部的空格仍然保留;

char的存取数度还是要比varchar要快得多，因为其长度固定，方便程序的存储与查找；但是char也为此付出的是空间的代价，因为其长度固定，所以难免会有多余的空格占位符占据空间，可谓是以空间换取时间效率，而varchar是以空间效率为首位的;

char的存储方式是，对英文字符（ASCII）占用1个字节，对一个汉字占用两个字节；而varchar的存储方式是，对每个英文字符占用2个字节，汉字也占用2个字节，两者的存储数据都非unicode的字符数据;

`何时该用char，何时该用varchar：`

varchar比char节省空间，在效率上比char会稍微差一些，即要想获得效率，就必须牺牲一定的空间，这也就是我们在数据库设计上常说的‘以空间换效率’；

varchar虽然比char节省空间，但是如果一个varchar列经常被修改，而且每次被修改的数据的长度不同，这会引起‘行迁移’(Row Migration)现象(如之前给分配的存储空间不足了)，而这造成多余的I/O，是数据库设计和调整中要尽力避免的，在这种情况下用char代替varchar会更好一些。而且varchar每次存储都要有额外的计算，得到长度等工作，如果一个非常频繁改变的，那就要有很多的精力用于计算，而这些对于char来说是不需要的。

char定长,一般用于固定长度的表单提交数据存储 ； 例如：身份证号，手机号，电话，密码等；

固定长度的。比如使用uuid作为主键，身份证号，手机号，电话，密码等，用char应该更合适。因为他固定长度，varchar动态根据长度的特性就消失了，而且还要占个长度信息。

存储很短的信息，比如门牌号码101，201……这样很短的信息应该用char，因为varchar还要占个byte用于存储信息长度，本来打算节约存储的现在得不偿失。

从空间上考虑，varchar更好，从效率上考虑，char更好。这其中的选择就需要我们根据情况自己考量。

基本在使用时首先考虑varchar,若char更好则用char;

`like` 只能用于字符串格式

注意sql语句中like只能用于字符串格式字段，若 要用于int|date等类型字段需要在sql语句中将字段数据转为char|varchar等字符串类型；

若int字段有索引，用like查询的话会使索引失效；

若某字段中存储的数据内容为数字，但业务场景中需要like查询等，则可以把该字段设置为char||varchar格式，如手机号虽然为11位数字，但最好用char(11),不要用bigint，即避免了32位系统导致的无法存储问题，又方便模糊查询。

用`int`表示时间戳字段

存储时间戳时int类型具有更大的优势(时间戳 或 HmdHis 格式)

1、仅占用4个字节，资源占用少，但只能表示到2038年，可到时候改成bigint

2、一般逻辑中经常用到时间范围查询，大小比较等，int类型在建立索引后，查询速度更快。只是在处理简单的数字

3、条件范围搜索可以使用使用between

结论：适合需要进行大量时间范围查询的数据表

mysql 字段类型使用推荐

char

手机号 | 电话 | uuid | 身份证号 | 密码 | 门牌号

int

datetime类型时间保存为时间戳 或 HmdHis 格式