以mysql为例：

**事务：**

事务的特性：原子性、一致性、隔离性、持久性。

事务不隔离带来的问题：更新丢失、脏读、不可重复读、虚读。

事务的隔离级别：读未提交（read uncommitted）、读已提交（read committed）、可重复读（repeatable read）、串行化（serializable）。依次解决上述4个问题。

更新丢失：

1.事务1和事务2同时更新同一条数据，事务1更新完数据A后，事务2又对A进行更新。此时事务1出现错误，回滚操作会覆盖事务2对A的更新。

2.事务1和事务2同时更新同一条数据，先更新的会被后更新的覆盖。

脏读：

指在一个事务处理过程里读取了另一个未提交的事务中的数据。事务1在更新完数据A后，事务2读取数据A。此时事务1出现错误进行回滚，事务2读取到的数据A则变为脏数据。

不可重复读：

事务1在读取数据A后，事务2立即对数据A进行了更新，事务再次对数据A读取时，得到了与第一次读取不一样的数据。

虚（幻）读：

事务1对表A的所有数据执行了将i列由"1"更新为"2"的操作，随后事务2向表A插入了一条i列为"1"的数据。此时事务1再次查询表A的数据时，发现还有一行数据的i列没有变为"2"。

读未提交：

事务写时，禁止其它事务写。

读已提交：

事务写时，禁止其它事务读写。

可重复读：

事务写时，禁止其它事务读写。事务读时，禁止其它事务”写”（其它事务可以写，当前事务读到的值一直为第一次所读到的值）。

串行化：

事务只能一个一个执行，不能并发执行（一个事务未执行完成时，另一事务开始执行会报错并返回）。

Mysql默认事务隔离级别为可重复读。Oracle、sqlserver默认事务隔离级别为读已提交。

SELECT @@tx\_isolation;查看当前事务隔离级别。

**索引：**

索引采用的数据结构：B+树。

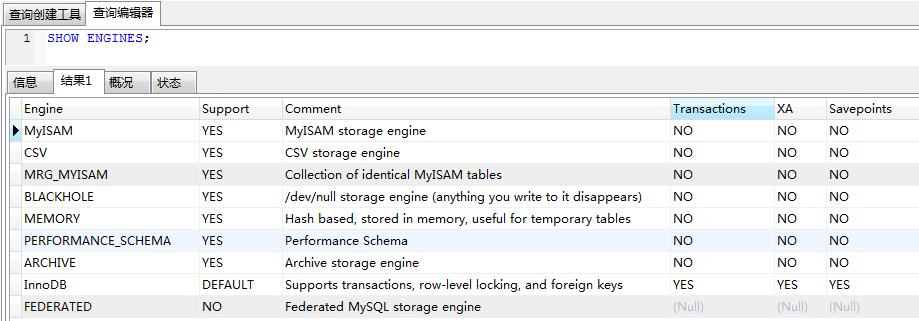
建立索引的原则：

1. 最左匹配原则：mysql会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d的顺序可以任意调整。
2. =和in可以乱序，比如a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式。
3. 尽量选择区分度高的列作为索引。若区分度较低，查询速度无法得到较大提升的同时，数据的插入更新操作效率会被拉低。
4. 索引列不能参与计算，保持列“干净”，比如from\_unixtime(create\_time) = '2014-05-29'就不能使用到索引，因为B+树中存的都是数据表中的字段值，但进行检索时，需要把所有元素都应用函数才能比较，成本太大。所以语句应该写成create\_time = unix\_timestamp('2014-05-29')。
5. 尽量扩展索引，不要新建索引。比如表中已经有a的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可。

不会使用索引的情况：

1. 查询条件中有not in，or，not exist。
2. like查询以%开头。
3. 小表查询会全表扫描，而不会使用索引。

**数据库引擎：**



InnoDB行级锁：

锁定单行数据（若sql查询出多条数据则所有数据都加锁），行级锁是基于索引的，若该sql没有用到索引，将不会使用行级锁，而是直接锁住整表。

InnoDB间隙锁：

间隙锁同样是基于索引的。可以用于在repeatable read下解决幻读问题。

SHOW VARIABLES LIKE 'innodb\_locks\_unsafe\_for\_binlog';查看间隙锁是否开启（默认为关闭）。

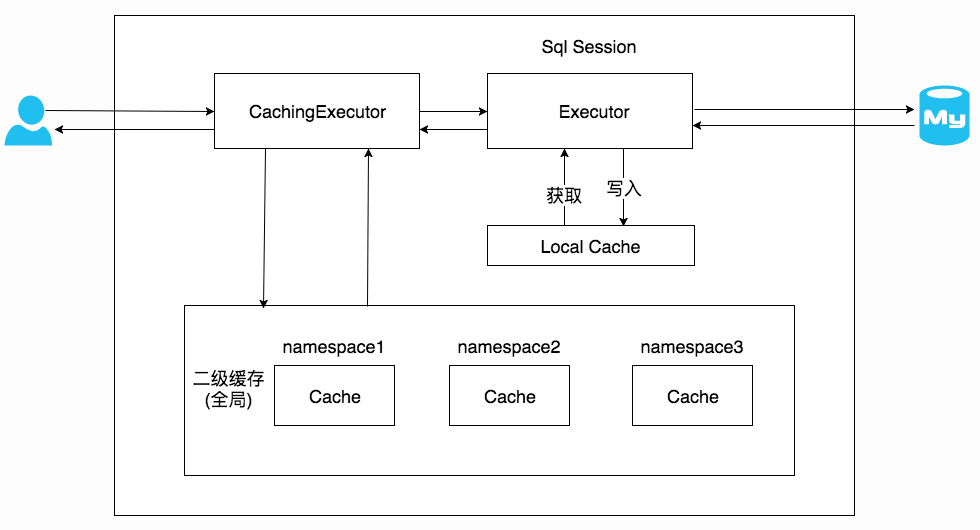
对于指定查询某一条记录的加锁语句，如果该记录不存在，会产生行锁和间隙锁，如果记录存在，则只会产生行锁。

在普通索引列上，不管是何种查询，只要加锁，都会产生间隙锁。



**MySQL缓存：**

mysql查询流程：二级缓存->一级缓存->数据库



**一级缓存：**

<setting name="localCacheScope" value="SESSION"/>

默认值为session，即在一个sqlSession（一次数据库会话(事务)）中的缓存会共享。

另一个值为STATEMENT，即仅在单个statement（sql语句）中缓存。

当值为session时，由于一级缓存仅在单个sqlSession内部中有效，在有多个sqlSession或者分布式的情况下，会出现脏数据的问题。

一级缓存内部设计简单，只是一个没有容量限定的HashMap，在缓存的功能性上有所欠缺。

**二级缓存：**

<setting name="cacheEnabled" value="true"/>

在Mybatis配置文件中开启mysql二级缓存。

<cache blocking="" eviction="" flushInterval="" readOnly="" size="" type=""/>

在mybatis映射文件中使用cache标签声明这个namespace使用二级缓存，并且可以自定义配置。

type：cache使用的类型，默认是PerpetualCache。

SynchronizedCache：同步Cache，实现比较简单，直接使用synchronized修饰方法。

LoggingCache：日志功能，装饰类，用于记录缓存的命中率，如果开启了DEBUG模式，则会输出命中率日志。

SerializedCache：序列化功能，将值序列化后存到缓存中。该功能用于缓存返回一份实例的Copy，用于保存线程安全。

LruCache：采用了LRU算法的Cache实现，移除最近最少使用的Key/Value。

PerpetualCache： 作为为最基础的缓存类，底层实现比较简单，直接使用了HashMap。

eviction： 定义回收的策略，常见的有FIFO，LRU。

flushInterval： 配置一定时间自动刷新缓存，单位是毫秒。

size： 最多缓存对象的个数。

readOnly： 是否只读，若配置可读写，则需要对应的实体类能够序列化。

blocking： 若缓存中找不到对应的key，是否会一直blocking，直到有对应的数据进入缓存。

<cache-ref namespace=""/>

在mybatis映射文件中使用cache-ref标签代表引用其它namespace的cache配置，并且两个namespace使用同一个cache。

二级缓存相对于一级缓存来说，实现了SqlSession之间缓存数据的共享，同时粒度更加的细，能够到namespace级别，通过Cache接口实现类不同的组合，对Cache的可控性也更强。

在分布式环境下，由于默认的MyBatis Cache实现都是基于本地的，分布式环境下必然会出现读取到脏数据，需要使用集中式缓存将MyBatis的Cache接口实现，有一定的开发成本，直接使用Redis、Memcached等分布式缓存可能成本更低，安全性也更高。

遇到的问题：

User:{ id:1, userName: "张三", age: 16}

User user = userMapper.getById(userId);

System.out.print(user.getUserName());

user.setUserName("李四");

User userNew = userMapper.getById(userId);

System.out.print(userNew.getUserName());

输出结果：张三李四

Id为1的user没有更新，正常情况下两次输出应该都为“张三”，但是因为mybatis缓存的原因，导致第二次输出结果为“李四”。

**锁：**