**MySQL数据库的三范式是什么：**

第一范式：强调的是列的原子性，即数据库表的每一列都是不可分割的原子数据项。

第二范式：要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性。

第三范式：任何非主属性不依赖于其它非主属性。

**oracle 索引** ：<https://www.cnblogs.com/iliuyuet/p/4431464.html>

①oracle创建主键时会自动在该列上创建索引。

②创建了索引并不一定就会使用，oracle自动统计表的信息后，决定是否使用索引，表中数据很少时使用全表扫描速度已经很快，没有必要使用索引。

B+tree索引

HASH索引

（bitmap）位图索引

聚族索引

非聚族索引

**truncate 和 deleted 区别：**

①truncate 只能对TABLE，delete 可以是table,view,synonym。

②truncate、drop是DLL，操作立即生效，原数据不放到rollback segment中，不能回滚；delete是DML，需要提交，操作会被放到rollback segment中并在日志中保存，支持回滚。

③truncate与不带where条件的delete语句相比执行速度快，不会触发与表相关的delete触发器。

④truncate表操作后，表结构，约束、索引保持不变，表和索引所占用的空间会恢复到初始大小；delete表操作后不会减少表或索引所占用的空间；drop语句会删除表结构及其约束、触发器、索引，但依赖于该表的存储过程会变为invalid状态；

**char 和 varchar 的区别：**

char(n) ：固定长度类型，比如订阅 char(10)，当你输入"abc"三个字符的时候，它们占的空间还是 10 个字节，其他 7 个是空字节。

char 优点：效率高；缺点：占用空间；适用场景：存储密码的 md5 值，固定长度的，使用 char 非常合适。

varchar(n) ：可变长度，存储的值是每个值占用的字节再加上一个用来记录其长度的字节的长度。

所以，从空间上考虑 varcahr 比较合适；从效率上考虑 char 比较合适，二者使用需要权衡。

**float 和 double 的区别：**

float 最多可以存储 8 位的十进制数，并在内存中占 4 字节。

double 最可可以存储 16 位的十进制数，并在内存中占 8 字节。

**mysql数据库引擎选择：**

查看MySQL数据库默认数据库引擎命令：show variables like '%storage\_engine%';

查看MySQL数据库创建表的语句的命令：show create table tablename;

mysql数据库引擎是表级别的，在创建表时指定。默认的mysql数据库引擎InnoDB。

原文：<https://blog.csdn.net/len9596/article/details/80206532>

**CREATE TABLE user (**

**id int(11) NOT NULL AUTO\_INCREMENT,**

**name varchar(255) DEFAULT NULL,**

**PRIMARY KEY (id)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8;**

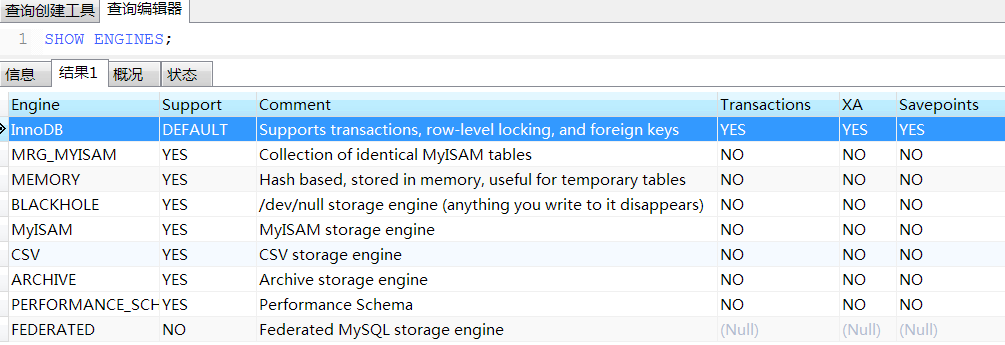
①InnoDB：如果要提供提交、回滚、崩溃恢复能力的事物安全（ACID兼容）能力，并要求实现并发控制，InnoDB是一个好的选择

②MyISAM：如果数据表主要用来插入和查询记录，则MyISAM引擎能提供较高的处理效率

③Memory：如果只是临时存放数据，数据量不大，并且不需要较高的数据安全性，可以选择将数据保存在内存中的Memory引擎，MySQL中使用该引擎作为临时表，存放查询的中间结果

④Archive：如果只有INSERT和SELECT操作，可以选择Archive，Archive支持高并发的插入操作，但是本身不是事务安全的。Archive非常适合存储归档数据，如记录日志信息可以使用Archive

⑤还有其它MySQL引擎。查看MySQL数据库引擎命令：show engines;



**InnoDB和MyISAM的区别：**

▶ InnoDB：

①索引：聚集索引，数据和索引存储在一个文件。会根据id自动生成索引，对表主键（id）采用B+TREE数据结构存储，存储该条记录。必须要有主键，如果没有设定主键或者非空唯一索引，会自动生成一个用户不可见的6字节主键。InnoDB没有保存表的总行数，如果使用select count(\*) from table，会遍历整个表，开销大。

②锁级别：InnoDB支持事务和行级锁。但是InnoDB表的行锁也不是绝对的，如果在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表，例如update table set num=1 where name like ”%aaa%”。

③数据库文件：

使用InnoDB创建表时会生成两个文件：

tableName.frm：创建表的文件，存储表的定义

tableName.ibd：数据+索引文件，存储表的数据和索引

**NOTE--** InnoDB有两种存储方式：共享表空间，独占表空间。

原文：<https://blog.csdn.net/chenjiayi_yun/article/details/45716933>

共享表空间：多个表，一个文件，默认在data目录下，默认文件名：ibdata1 初始化为10M。

独占表空间：多个表，多个.frm文件，多个.ibd文件。

▶ MyISAM：

①索引：非聚集索引，数据和索引分开存储。对表主键（id）采用B+TREE数据结构存储，存储记录对应的物理地址。数据重复度高的列不建议创建索引。允许没有任何索引和主键的表存在。MyISAM保存有表的总行数，如果使用select count(\*) form table会直接取出该值。但是加了where条件后MyISAM和InnoDB的处理方式就一样了。

②锁级别：MyISAM只支持表级锁，select，update，delete，insert语句都会给表自动加锁，不支持事务。

③数据库文件：

使用MyISAM创建表时会生成三个文件：

tableName.frm：创建表的文件，存储表的定义

tableName.myd：表的数据文件，存储表的数据

tableName.myi：表的索引文件，存储表的索引

**NOTE--** MyISAM支持三种不同的存储格式：静态表，动态表，压缩表。

---------------------------------------------------------------------------------------------------------------------------------

**SUMMARY--**

InnoDB是事务安全的，MyISAM不支持事务；InnoDB支持行级锁，MyISAM是表级锁；

InnoDB支持外键，MyISAM不支持外键；InnoDB是聚集索引，MyISAM是非聚集索引；

InnoDB不保存总行数，MyISAM保存表的总行数。

---------------------------------------------------------------------------------------------------------------------------------

~~InnoDB不支持全文索引，而MyISAM支持全文索引，查询效率上MyISAM要高。~~（mysql5.7 InnoDB已经支持）

开发注意事项：

⑴InnoDB每一条SQL语言都默认封装成事务，自动提交，这样会影响速度，所以最好把多条SQL语言放在begin和commit之间，组成一个事务。

⑵InnoDB是聚集索引，数据文件是和索引绑在一起的，必须要有主键，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。因此，主键不应该过大，因为主键太大，其他索引也都会很大。

⑶在一个事务获得排他锁或者意向排他锁以后，如果后面还有需要处理的sql语句，在这两条或者多条sql语句之间程序应尽量少的进行逻辑运算和处理，减少锁的时间。

⑷尽量避免死锁。

解决死锁的方法：

1、 数据库参数

2、 应用中尽量约定程序读取表的顺序一样

3、 应用中处理一个表时，尽量对处理的顺序排序

4、 调整事务隔离级别（避免两个事务同时操作一行不存在的数据，容易发生死锁）

**一张自增表里面总共有 7 条数据，删除了最后 2 条数据，重启 MySQL 数据库，又插入了一条数据，此时 id 是几？**

表类型如果是 MyISAM ，那 id 就是 8。

表类型如果是 InnoDB，那 id 就是 6。

InnoDB 表只会把自增主键的最大 id 记录在内存中，所以重启之后会导致最大 id 丢失。

**mysql索引：**

有hash索引和b+tree索引

加快数据检索的数据结构

**mysql事务：**

①ACID属性：

1.原子性（Atomicity）：事务是一个原子操作单元，其对数据的修改，要么全都执行，要么全都不执行。

2.一致性（Consistent）：在事务开始和完成时，数据都必须保持一致状态。这意味着所有相关的数据规则都必须应用于事务的修改，以保持数据的完整性；事务结束时，所有的内部数据结构（如B树索引或双向链表）也都必须是正确的。

3.隔离性（Isolation）：数据库系统提供一定的隔离机制，保证事务在不受外部并发操作影响的“独立”环境执行。这意味着事务处理过程中的中间状态对外部是不可见的，反之亦然。

4.持久性（Durable）：事务完成之后，它对于数据的修改是永久性的，即使出现系统故障也能够保持。

②四种隔离级别：查看mysql的默认事务隔离级别：show global variables like ‘tx\_isolation’;

1.read uncommitted(未提交读)(会出现脏读，不可重复读，幻读)：所有事务都可以看到其它未提交事务的执行结果。该隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。

2.read committed(已提交读)(会出现不可重复读，幻读)：这是大多数数据库系统的隔离级别（除mysql）。一个事物只能看见已经提交事务所做的改变。

3.repeatable read (可重复读)(会出现幻读)：这是mysql默认的事务隔离级别，它确保同一事务的多个实例在并发读取数据时会看到同样的数据行。但可能会导致幻读：当用户读取某一范围的数据行时另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影”行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，multiversion concurrent control）机制解决了该问题。

4.serializable(可串行化)：这是最高的隔离级别，通过强制事务排序使之不可能互相冲突从而解决幻读问题。简而言之，它是在每个读的数据行上加上共享锁，可能导致大量的超时现象和锁竞争。

脏读 ：表示一个事务能够读取另一个事务中还未提交的数据。比如，某个事务尝试插入记录 A，此时该事务还未提交，然后另一个事务尝试读取到了记录 A。

不可重复读 ：是指在一个事务内，多次读同一数据。

幻读 ：指同一个事务内多次查询返回的结果集不一样。比如同一个事务 A 第一次查询时候有 n 条记录，但是第二次同等条件下查询却有 n+1 条记录，这就好像产生了幻觉。发生幻读的原因也是另外一个事务新增或者删除或者修改了第一个事务结果集里面的数据，同一个记录的数据内容被修改了，所有数据行的记录就变多或者变少了。

**mysql数据库锁：**

1.行级锁：共享锁（又称读锁）、排它锁（又称写锁）

共享锁（S）：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。(事务获取数据集的读锁，允许其他事务读取，不允许执行写操作)

排他锁（X)：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁。（允许获得写锁的事务更新数据，不允许其他事务获对相同数据集的读和写）

2.表级锁：意向共享锁（IS）、意向排他锁（IX）

意向共享锁（IS）：事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。

意向排他锁（IX）：事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。

InnoDB引擎的锁机制：

1）共享锁和排他锁都是行锁，意向锁都是表锁，应用中我们只会使用到共享锁和排他锁，意向锁是mysql内部使用的，不需要用户干预。

2）对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)；对于普通SELECT语句，InnoDB不会加任何锁，事务可以通过以下语句显示给记录集加共享锁或排他锁。

共享锁（S）：SELECT \* FROM table\_name WHERE ... LOCK IN SHARE MODE。

排他锁（X)：SELECT \* FROM table\_name WHERE ... FOR UPDATE。

\*\*对于锁定行记录后需要进行更新操作的应用，应该使用Select...For update 方式，获取排它锁。（用共享锁，在读了之后再写会阻塞，会导致死锁）

3.乐观锁、悲观锁：

悲观锁：正如其名，它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。实现：往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。

1）使用悲观锁，我们必须关闭mysql数据库的自动提交属性，采用手动提交事务的方式，因为MySQL默认使用autocommit模式，也就是说，当你执行一个更新操作后，MySQL会立刻将结果进行提交。

2）需要注意的是，在事务中，只有SELECT ... FOR UPDATE 或LOCK IN SHARE MODE 同一笔数据时会等待其它事务结束后才执行，一般SELECT ... 则不受此影响。对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)。

3）补充：MySQL select…for update的Row Lock与Table Lock

使用select…for update会把数据给锁住，不过我们需要注意一些锁的级别，MySQL InnoDB默认Row-Level Lock，所以只有「明确」地指定主键（或有索引的地方），MySQL 才会执行Row lock (只锁住被选取的数据) ，否则MySQL 将会执行Table Lock (将整个数据表给锁住)。

乐观锁：相对悲观锁而言，乐观锁假设认为数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让返回用户错误的信息，让用户决定如何去做（一般是回滚事务）。实现：一般来说有以下2种方式：

1）.使用数据版本（Version）记录机制实现，这是乐观锁最常用的一种实现方式。即为数据增加一个版本标识，一般是通过为数据库表增加一个数字类型的 “version” 字段来实现。当读取数据时，将version字段的值一同读出，数据每更新一次，对此version值加一。当我们提交更新的时候，判断数据库表对应记录的当前版本信息与第一次取出来的version值进行比对，如果数据库表当前版本号与第一次取出来的version值相等，则予以更新，否则认为是过期数据。

2）.第二种实现方式和第一种差不多，同样是在需要乐观锁控制的table中增加一个字段，名称无所谓，字段类型使用时间戳（timestamp）, 和上面的version类似，也是在更新提交的时候检查当前数据库中数据的时间戳和自己更新前取到的时间戳进行对比，如果一致则OK，否则就是版本冲突。

---------------------------------------------------------------------------------------------------------------------------------

**NOTE--**

表级锁(table-level locking)：MyISAM和MEMORY存储引擎

行级锁(row-level locking) ：InnoDB存储引

页面锁(page-level-locking)：BDB存储引擎

表级锁：开销小，加锁快;不会出现死锁;锁定粒度大，发生锁冲突的概率最高,并发度最低。

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,并发度也最高。

页面锁：开销和加锁时间界于表锁和行锁之间;会出现死锁;锁定粒度界于表锁和行锁之间，并发度一般

**MySQL慢查询：**<http://www.cnblogs.com/luyucheng/p/6265594.html>

开启慢查询日志，可以让MySQL记录下查询超过指定时间的语句，通过定位分析性能的瓶颈，才能更好的优化数据库系统的性能。

①参数说明：slow\_query\_log 慢查询开启状态

slow\_query\_log\_file 慢查询日志存放位置

long\_query\_time 查询超过多少秒才记录

②查看慢查询相关参数：show variables like ‘slow\_query%’;

show variables like ‘long\_query\_time’;

③设置方法：设置好后重启mysql服务

方法一：**set global slow\_query\_log=’ON’;**

**set global slow\_query\_log\_file=’/usr/local/mysql/data/slow.log’**

**set global long\_query\_time=1;(秒)**

方法二：修改配置文件my.conf,在[mysql]下加入：

**[mysql]**

**slow\_query\_log = ON**

**slow\_query\_log\_file = /usr/local/mysql/data/slow.log**

**long\_query\_time = 1**

**优化数据库的数据查询，另外应用层上还能如何优化？**

1）数据库层面上：

•除了主键索引，唯一索引之外，对于常用的查询字段也要加索引。查询的时候尽量使用主键索引，因为MySQL 的 InnoDB 的主键索引索引的是整行数据，而普通索引索引的是主键，会有回表操作。当然索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，需要酌情考虑。

•优化查询语句，尽量采用确认性查询语句，减少 or，in，not in，%xxx%语法的使用。

2）应用层面上：

•采用缓存机制，将常用的数据进行缓存，增加访问速度；

•分库分表，读写分离，将数据分开读写，提升性能

**MySQL数据库优化：**<http://www.cnblogs.com/luyucheng/p/6323477.html>

①sql语句优化：

（1）使用limit对查询结果的记录进行限定

（2）避免select \*，将需要查找的字段列出来

（3）使用连接（join）来代替子查询

（4）拆分大的delete或insert语句

②选择合适的数据类型：

（1）使用可存下数据的最小的数据类型，整型 < date,time < char,varchar < blob。

（2）使用简单的数据类型，整型比字符处理开销更小，因为字符串的比较更复杂。如，int类型存储时间类型，bigint类型转ip函数。

（3）使用合理的字段属性长度，固定长度的表会更快。使用enum、char而不是varchar。

（4）尽可能使用not null定义字段。

（5）尽量少用text，非用不可最好分表。

③选择合适的索引列：

（1）查询频繁的列，在where，group by，order by，on从句中出现的列

（2）where条件中<，<=，=，>，>=，between，in，以及like 字符串+通配符（%）出现的列。

（3）长度小的列，索引字段越小越好，因为数据库的存储单位是页，一页中能存下的数据越多越好。

（4）离散度大（不同的值多）的列，放在联合索引前面。查看离散度，通过统计不同的列值来实现，count越大，离散程度越高：

mysql> SELECT COUNT(DISTINCT column\_name) FROM table\_name;

④使用命令分析：

⑤定位慢查询：同上mysql慢查询

⑥mysql分区和分表：<http://www.cnblogs.com/luyucheng/p/6290567.html>

⑦配置优化：<http://www.cnblogs.com/luyucheng/p/6340076.html>

**jdbc连接步骤：**

1.加载驱动

mysql： **Class.forName("com.mysql.cj.jdbc.Driver");**

oracle：**Class.forName("oracle.jdbc.driver.OracleDriver");**

2.创建连接

**Connect** **connection=DriverManager.getConnection(**

**"jdbc:mysql://localhost:3306/my\_tmall\_springboot?useUnicode=true&characterEncoding=utf8&serverTimezone=UTC", "root","123456");**

3.获取statement

普通语句：**Statement statement = connection.createStatement();**

预执行语句：**PreparedStatement preparedStatement = connection.prepareStatement(sql);**

存储过程语句：**CallableStatement callableStatement= connection.prepareCall(produce);**

4.执行SQL语句

execute、executeQuery、executeUpdate

5.获取结果集

resultSet

6.jdbc资源关闭

关闭语句，关闭连接

---------------------------------------------------------------------------------------------------------------------------------

**NOTE--**

jdbc连接oracle url的三种格式：<https://blog.csdn.net/u012062455/article/details/52442838>

格式一: Oracle JDBC Thin using an SID: jdbc:oracle:thin:@host:port:SID

Example: jdbc:oracle:thin:@localhost:1521:orcl ，这种格式是最简单也是用得最多的

你的oracle的sid可以通过一下指令获得： sqlplus / as sysdba

select value from v$parameter where name=’instance\_name’;

格式二: Oracle JDBC Thin using a ServiceName: jdbc:oracle:thin:@//host:port/service\_name

Example:jdbc:oracle:thin:@//localhost:1521/orcl.city.com ，注意这里的格式，@后面有//, port后面:换成了/,这种格式是 Oracle 推荐的格式，因为对于集群来说，每个节点的SID 是不一样的，但是SERVICE\_NAME 确可以包含所有节点。

你的oracle的service\_name可以通过以下方式获得： sqlplus / as sysdba

select value from v$parameter where name=’service\_names’;

格式三：Oracle JDBC Thin using a TNSName: jdbc:oracle:thin:@TNSName

Example: jdbc:oracle:thin:@TNS\_ALIAS\_NAME ，我在谷歌上找了一些资源，要实现这种连接方式首先要建立tnsnames.ora 文件，然后通过System.setProperty指明这个文件路径。再通过上面URL中的@符号指定文件中的要使用到的资源。 这种 格式我现在水平几乎没见过，对于我来说用得到这种的情况并不多吧。当然既然是通过配置文件来读取指定资源肯定也可 以直接将资源拿出来放在URL中，直接放在URL中的URL模版是下面这样的（tnsnames.ora这个文件中放的就是@符号后 面的那一段代码，当然用文件的好处就是可以配置多个，便于管理）：

jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS\_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=hostA)(PORT=1522))(ADDRESS=(PROTOCO L=TCP)(HOST=your host)(PORT=1521)))(SOURCE\_ROUTE=yes)(CONNECT\_DATA=(SERVICE\_NAME=your service\_name)))

**statement和prepareStatement区别：**

prepareStatement接口继承statement接口，PreparedStatement 实例包含已编译的 SQL 语句，其执行速度要快于 Statement 对象

prepareStatement中的SQL可以带参数，使用？作为参数占位符。prepareStatement有一系列的方法用于设置参数。

**Hibernate：**

Hibernate 是对 jdbc 的封装，大大简化了数据访问层的繁琐的重复性代码。

Hibernate 是一个优秀的 ORM 实现，很多程度上简化了 DAO 层的编码功能。可以很方便的进行数据库的移植工作。提供了缓存机制，使程序执行更高效。

**Hibernate 的几种查询方式：**

对象导航查询(objectcomposition)。

HQL查询：

1、 属性查询

2、 参数查询、命名参数查询

3、 关联查询

4、 分页查询

5、 统计函数

Criteria 查询。

SQLQuery本地SQL查询。

**Hibernate 的工作流程（如何工作的）：**

读取并解析配置文件。

读取并解析映射文件，创建 SessionFactory。

打开 Session。

创建事务。

进行持久化操作。

提交事务。

关闭 Session。

关闭 SessionFactory。

**Hibernate 的缓存机制：**https://www.cnblogs.com/xiaoluo501395377/p/3377604.html

Hibernate 常用的缓存有一级缓存和二级缓存：

一级缓存：也叫 Session 缓存，可以在session范围内减少数据库的访问次数只在 Session 作用范围内有效。只要是持久化状态的对象都受session管理。不需要用户干涉，由 Hibernate 自身维护，可以通过：evict(object)清除 object 的缓存；clear()清除一级缓存中的所有缓存；flush()刷出缓存；

二级缓存：应用级别的缓存，在所有 Session 中都有效。有默认的实现，需要在Hibernate.cfg.xml中配置，还支持配置第三方的缓存，如：EhCache。二级缓存中缓存的仅仅是对象，如果查询的是字段则不会被加载到二级缓存中。

**Hibernate都支持哪些缓存策略：**

usage的属性有4种：

放入二级缓存的对象，只读(Read-only);

非严格的读写(Nonstrict read/write)

读写； 放入二级缓存的对象可以读、写(Read/write)；

基于事务的策略(Transactional)

**Hibernate 对象有哪些状态：**

临时/瞬时状态：直接 new 出来的对象，该对象还没被持久化（没保存在数据库中），不受 Session 管理。

持久化状态：当调用 Session 的 save/saveOrupdate/get/load/list 等方法的时候，对象就是持久化状态。

游离状态：Session 关闭之后对象就是游离状态。

**Hibernate的三种检索策略优缺点:**

立即检索：立即检索:lazy=false；

优点： 对应用程序完全透明，不管对象处于持久化状态，还是游离状态，应用程序都可以方便的从一个对象导航到与它关联的对象；

缺点： 1.select语句太多；2.可能会加载应用程序不需要访问的对象白白浪费许多内存空间；

延迟检索：延迟加载：lazy=true；

优点： 由应用程序决定需要加载哪些对象，可以避免可执行多余的select语句，以及避免加载应用程序不需要访问的对象。因此能提高检索性能，并且能节省内存空间；

缺点： 应用程序如果希望访问游离状态代理类实例，必须保证他在持久化状态时已经被初始化；

迫切左外连接检索：预先抓取： fetch=”join”；

优点： 1对应用程序完全透明，不管对象处于持久化状态，还是游离状态，应用程序都可以方便地从一个对象导航到与它关联的对象。2使用了外连接，select语句数目少；

缺点： 1 可能会加载应用程序不需要访问的对象，白白浪费许多内存空间；2复杂的数据库表连接也会影响检索性能；

**Hibernate sorted collection 和ordered collection的区别：**

sorted collection：是在内存中通过Java比较器进行排序的

ordered collection：是在数据库中通过order by进行排序的

对于比较大的数据集，为了避免在内存中对它们进行排序而出现 Java中的OutOfMemoryError，最好使用ordered collection。

**Hibernate 中 getCurrentSession 和 openSession 有什么区别：**

getCurrentSession 会绑定当前线程，而 openSession 则不会。

getCurrentSession 事务是 Spring 控制的，并且不需要手动关闭，而 openSession 需要我们自己手动开启和提交事务。

**get()和load()的区别：**

get()：立即查询，先查一级缓存，再查二级缓存，然后查数据库，如果没找到会返回null。

load()：懒加载，先查一级缓存，如果没有找到就创建代理对象，等需要的时候再查二级缓存和数据库，如果没有找到会抛异常。

**persist和save的区别：**

persist不保证立即执行，可能要等到flush；不更新缓存；没有任何返回值。

save 会立即执行Sql insert；返回该持久化对象的标识属性值(即对应记录的主键值)；

**Hibernate 实体类为什么必须要有无参构造函数：**

Hibernate 中每个实体类必须提供一个无参构造函数，因为 Hibernate 框架要使用 reflection api，通过调用 Class.newInstance() 来创建实体类的实例，如果没有无参的构造函数就会抛出InstantiationException异常。

**可不可以将Hibernate的实体类定义为final类:**

可以将Hibernate的实体类定义为final类，但这种做法并不好。因为Hibernate会使用代理模式在延迟关联的情况下提高性能，如果你把实体类定义成final类之后，因为 Java不允许对final类进行扩展，所以Hibernate就无法再使用代理了， 如此一来就限制了使用可以提升性能的手段。

**Hibernate中inverse的作用：**

inverse属性默认是false,就是说关系的两端都来维护关系。

比如Student和Teacher是多对多关系，用一个中间表TeacherStudent维护。如果Student这边inverse=”true”, 那么关系由另一端Teacher维护，就是说当插入Student时，不会操作TeacherStudent表（中间表）。只有Teacher插入或删除时才会触发对中间表的操作。所以两边都inverse=”true”是不对的，会导致任何操作都不触发对中间表的影响；当两边都inverse=”false”或默认时，会导致在中间表中插入两次关系。

如果表之间的关联关系是“一对多”的话，那么inverse只能在“一”的一方来配置！

**主键生成策略：**

①主键的自动生成策略：

identity 自增长(mysql,db2)

sequence 自增长(序列)， oracle中自增长是以序列方法实现

native 自增长【会根据底层数据库自增长的方式选择identity或sequence】

如果是mysql数据库, 采用的自增长方式是identity

如果是oracle数据库， 使用sequence序列的方式实现自增长

increment 自增长(会有并发访问的问题，一般在服务器集群环境使用会存在问题。)

②指定主键生成策略为手动指定主键的值

assigned

③指定主键生成策略为UUID生成的值

uuid

④foreign(外键的方式)

**jpa 和 Hibernate 有什么区别：**

jpa 全称 Java Persistence API，是 Java 持久化接口规范，Hibernate 属于 jpa 的具体实现。

**Mybatis面试题：https://www.cnblogs.com/huajiezh/p/6415388.html**

**Mybatis#{}和${}的区别：https://www.cnblogs.com/baizhanshi/p/5778692.html**

${}是Properties文件中的变量占位符，它可以用于标签属性值和sql内部，属于静态文本替换，比如${driver}会被静态替换为com.mysql.jdbc.Driver。

#{}是sql的参数占位符，Mybatis会将sql中的#{}替换为?号，在sql执行前会使用PreparedStatement的参数设置方法，按序给sql的?号占位符设置参数值，比如ps.setInt(0, parameterValue)，#{item.name}的取值方式为使用反射从参数对象中获取item对象的name属性值，相当于param.getItem().getName()。

**Mybatis Xml映射文件中，除了常见的select|insert|updae|delete标签之外，还有哪些标签：**

还有很多其他的标签，<resultMap>、<parameterMap>、<sql>、<include>、<selectKey>，加上动态sql的9个标签，trim|where|set|foreach|if|choose|when|otherwise|bind等，其中<sql>为sql片段标签，通过<include>标签引入sql片段，<selectKey>为不支持自增的主键生成策略标签。

**通常一个Xml映射文件，都会写一个Dao接口与之对应，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？**

Dao接口，就是人们常说的Mapper接口，接口的全限名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中MappedStatement的id值，接口方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement。

举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个<select>、<insert>、<update>、<delete>标签，都会被解析为一个MappedStatement对象。

Dao接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

Dao接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Dao接口生成代理proxy对象，代理对象proxy会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。

**MyBatis 有几种分页方式：**

分页方式：逻辑分页和物理分页。

逻辑分页： 使用 MyBatis 自带的 RowBounds 进行分页，它是一次性查询很多数据，然后在数据中再进行检索。这样做弊端是需要消耗大量的内存、有内存溢出的风险、对数据库压力较大。

物理分页： 自己手写 SQL 分页或使用分页插件 PageHelper，去数据库查询指定条数的分页数据的形式。弥补了一次性全部查出的所有数据的种种缺点。

**RowBounds 是一次性查询全部结果吗：**

RowBounds 表面是在“所有”数据中检索数据，其实并非是一次性查询出所有数据。因为 MyBatis 是对 jdbc 的封装，在 jdbc 驱动中有一个 Fetch Size 的配置，它规定了每次最多从数据库查询多少条数据，假如你要查询更多数据，它会在你执行 next()的时候，去查询更多的数据。就好比你去自动取款机取 10000 元，但取款机每次最多能取 2500 元，所以你要取 4 次才能把钱取完。只是对于 jdbc 来说，当你调用 next()的时候会自动帮你完成查询工作。这样做的好处可以有效的防止内存溢出。Fetch Size 官方相关文档：http://t. cn/EfSE2g3。

**MyBatis 有哪些执行器（Executor）：**MyBatis 有三种基本的Executor执行器：

SimpleExecutor：每执行一次 update 或 select 就开启一个 Statement 对象，用完立刻关闭 Statement 对象；

ReuseExecutor：执行 update 或 select，以 SQL 作为 key 查找 Statement 对象，存在就使用，不存在就创建，用完后不关闭 Statement 对象，而是放置于 Map 内供下一次使用。简言之，就是重复使用 Statement 对象；

BatchExecutor：执行 update（没有 select，jdbc 批处理不支持 select），将所有 SQL 都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个 Statement 对象，每个 Statement 对象都是 addBatch()完毕后，等待逐一执行 executeBatch()批处理，与 jdbc 批处理相同。

**MyBatis 的一级缓存和二级缓存：**https://www.cnblogs.com/happyflyingpig/p/7739749.html

一级缓存：默认开启。基于 PerpetualCache 的 HashMap 本地缓存，它的声明周期是和 SQLSession 一致的，有多个 SQLSession 或者分布式的环境中数据库操作，可能会出现脏数据。

①MyBatis在开启一个数据库会话时，会创建一个新的SqlSession对象，SqlSession对象中会有一个新的Executor对象。Executor对象中持有一个新的PerpetualCache对象；当会话结束时，SqlSession对象及其内部的Executor对象还有PerpetualCache对象也一并释放掉。

②如果SqlSession调用了close()方法，会释放掉一级缓存PerpetualCache对象，一级缓存将不可用。

③如果SqlSession调用了clearCache()，会清空PerpetualCache对象中的数据，但是该对象仍可使用。

④SqlSession中执行了任何一个update操作(update()、delete()、insert()) ，都会清空PerpetualCache对象的数据，但是该对象可以继续使用

二级缓存：基于 PerpetualCache 的 HashMap 本地缓存，不同在于其存储作用域为 Mapper 级别的，如果多个SQLSession之间需要共享缓存，则需要使用到二级缓存，并且二级缓存可自定义存储源，如 Ehcache。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态)。

开启二级缓存数据查询流程：二级缓存 -> 一级缓存 -> 数据库。

缓存更新机制：当某一个作用域(一级缓存 Session/二级缓存 Mapper)进行了C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。

**Mybatis 二级缓存回收策略：**

(1) LRU,最近最少使用的，一处最长时间不用的对象

(2) FIFO,先进先出，按对象进入缓存的顺序来移除他们

(3) SOFT,软引用，移除基于垃圾回收器状态和软引用规则的对象

(4) WEAK,弱引用，更积极的移除基于垃圾收集器状态和弱引用规则的对象。这里采用的是LRU，移除最长时间不用的对象。