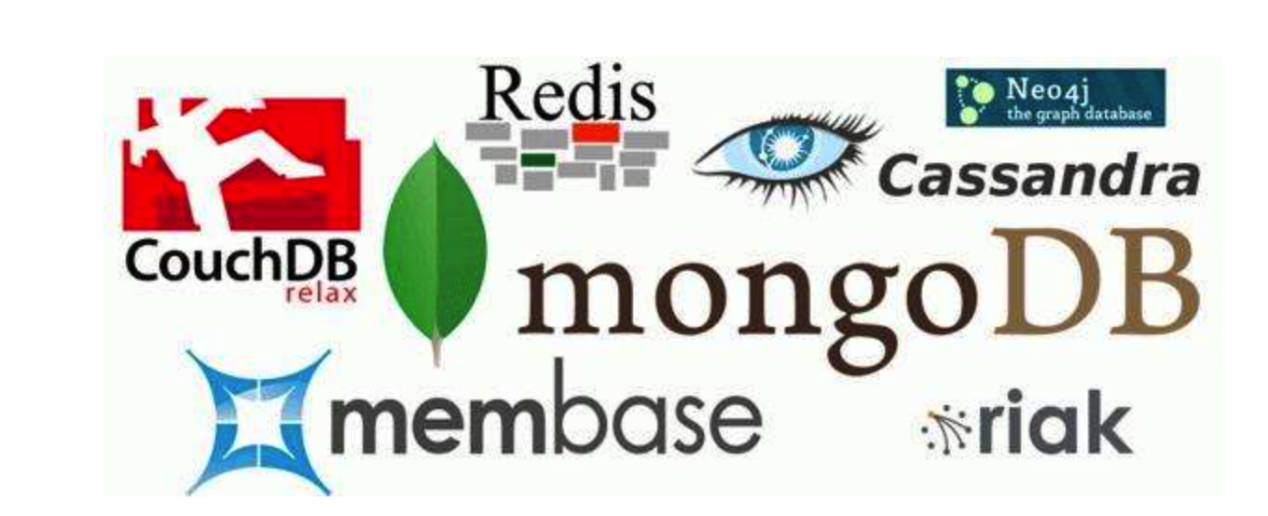
<https://mp.weixin.qq.com/s/ojnoM0BMUiUCLNCUE7WFag>

# 9 道 MySQL 面试题



## 非关系型数据库和关系型数据库区别，优势比较



非关系型数据库（感觉翻译不是很准确）称为 NoSQL，也就是 Not Only SQL，不仅仅是 SQL。非关系型数据库不需要写一些复杂的 SQL 语句，其内部存储方式是以 key-value 的形式存在可以把它想象成电话本的形式，每个人名（key）对应电话（value）。常见的非关系型数据库主要有 **Hbase、Redis、MongoDB** 等。非关系型数据库不需要经过 SQL 的重重解析，所以性能很高；非关系型数据库的可扩展性比较强，数据之间没有耦合性，遇见需要新加字段的需求，就直接增加一个 key-value 键值对即可。



关系型数据库以表格的形式存在，以行和列的形式存取数据，关系型数据库这一系列的行和列被称为表，无数张表组成了数据库，常见的关系型数据库有 **Oracle、DB2、Microsoft SQL Server、MySQL**等。关系型数据库能够支持复杂的 SQL 查询，能够体现出数据之间、表之间的关联关系；关系型数据库也支持事务，便于提交或者回滚。

它们之间的劣势都是基于对方的优势来满足的。

## MySQL 事务四大特性

一说到 MySQL 事务，你肯定能想起来四大特性：原子性、一致性、隔离性、持久性，下面再对这事务的四大特性做一个描述

* 原子性(Atomicity): 原子性指的就是 MySQL 中的包含事务的操作要么全部成功、要么全部失败回滚，因此事务的操作如果成功就必须要全部应用到数据库，如果操作失败则不能对数据库有任何影响。

**“**

这里涉及到一个概念，什么是 MySQL 中的事务？

事务是一组操作，组成这组操作的各个单元，要不全都成功要不全都失败，这个特性就是事务。

在 MySQL 中，事务是在引擎层实现的，只有使用 innodb 引擎的数据库或表才支持事务。

* 一致性(Consistency)：一致性指的是一个事务在执行前后其状态一致。比如 A 和 B 加起来的钱一共是 1000 元，那么不管 A 和 B 之间如何转账，转多少次，事务结束后两个用户的钱加起来还得是 1000，这就是事务的一致性。
* 持久性(Durability): 持久性指的是一旦事务提交，那么发生的改变就是永久性的，即使数据库遇到特殊情况比如故障的时候也不会产生干扰。
* 隔离性(Isolation)：隔离性需要重点说一下，当多个事务同时进行时，就有可能出现脏读(dirty read)、不可重复读(non-repeatable read)、幻读(phantom read) 的情况，为了解决这些并发问题，提出了隔离性的概念。

**“**

脏读：事务 A 读取了事务 B 更新后的数据，但是事务 B 没有提交，然后事务 B 执行回滚操作，那么事务 A 读到的数据就是脏数据

不可重复读：事务 A 进行多次读取操作，事务 B 在事务 A 多次读取的过程中执行更新操作并提交，提交后事务 A 读到的数据不一致。

幻读：事务 A 将数据库中所有学生的成绩由 A -> B，此时事务 B 手动插入了一条成绩为 A 的记录，在事务 A 更改完毕后，发现还有一条记录没有修改，那么这种情况就叫做出现了幻读。

SQL的隔离级别有四种，它们分别是读未提交(read uncommitted)、读已提交(read committed)、可重复读(repetable read) 和 串行化(serializable)。下面分别来解释一下。

读未提交：读未提交指的是一个事务在提交之前，它所做的修改就能够被其他事务所看到。

读已提交：读已提交指的是一个事务在提交之后，它所做的变更才能够让其他事务看到。

可重复读：可重复读指的是一个事务在执行的过程中，看到的数据是和启动时看到的数据是一致的。未提交的变更对其他事务不可见。

串行化：顾名思义是对于同一行记录，写会加写锁，读会加读锁。当出现读写锁冲突的时候，后访问的事务必须等前一个事务执行完成，才能继续执行。

这四个隔离级别可以解决脏读、不可重复读、幻象读这三类问题。总结如下



其中隔离级别由低到高是：读未提交 < 读已提交 < 可重复读 < 串行化

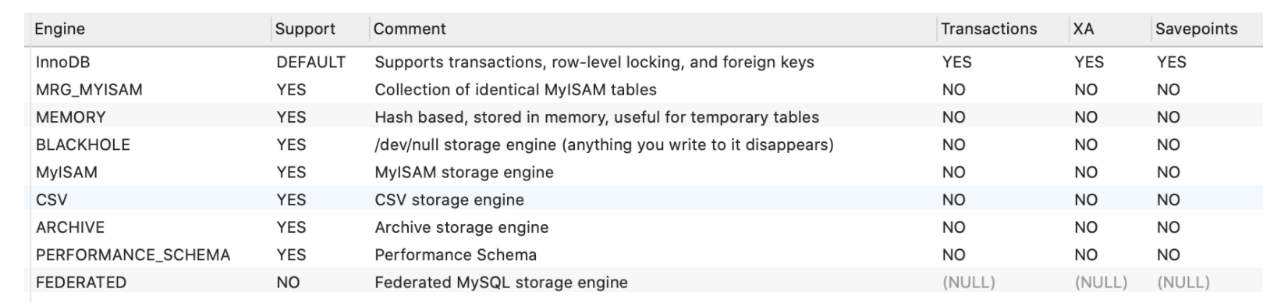
隔离级别越高，越能够保证数据的完整性和一致性，但是对并发的性能影响越大。大多数数据库的默认级别是读已提交(Read committed)，比如 Sql Server、Oracle ，但是 MySQL 的默认隔离级别是 可重复读(repeatable-read)。

**MySQL 常见存储引擎的区别**

MySQL 常见的存储引擎，可以使用

SHOW ENGINES

命令，来列出所有的存储引擎



可以看到，InnoDB 是 MySQL 默认支持的存储引擎，支持**事务、行级锁定和外键**。

### MyISAM 存储引擎的特点

在 5.1 版本之前，MyISAM 是 MySQL 的默认存储引擎，MyISAM 并发性比较差，使用的场景比较少，主要特点是

* 不支持事务操作，ACID 的特性也就不存在了，这一设计是为了性能和效率考虑的。
* 不支持外键操作，如果强行增加外键，MySQL 不会报错，只不过外键不起作用。
* MyISAM 默认的锁粒度是表级锁，所以并发性能比较差，加锁比较快，锁冲突比较少，不太容易发生死锁的情况。
* MyISAM 会在磁盘上存储三个文件，文件名和表名相同，扩展名分别是 .frm(存储表定义)、.MYD(MYData,存储数据)、MYI(MyIndex,存储索引)。这里需要特别注意的是 MyISAM 只缓存索引文件，并不缓存数据文件。
* MyISAM 支持的索引类型有 全局索引(Full-Text)、B-Tree 索引、R-Tree 索引

Full-Text 索引：它的出现是为了解决针对文本的模糊查询效率较低的问题。

B-Tree 索引：所有的索引节点都按照平衡树的数据结构来存储，所有的索引数据节点都在叶节点

R-Tree索引：它的存储方式和 B-Tree 索引有一些区别，主要设计用于存储空间和多维数据的字段做索引,目前的 MySQL 版本仅支持 geometry 类型的字段作索引，相对于 BTREE，RTREE 的优势在于范围查找。

* 数据库所在主机如果宕机，MyISAM 的数据文件容易损坏，而且难以恢复。
* 增删改查性能方面：SELECT 性能较高，适用于查询较多的情况

### InnoDB 存储引擎的特点

自从 MySQL 5.1 之后，默认的存储引擎变成了 InnoDB 存储引擎，相对于 MyISAM，InnoDB 存储引擎有了较大的改变，它的主要特点是

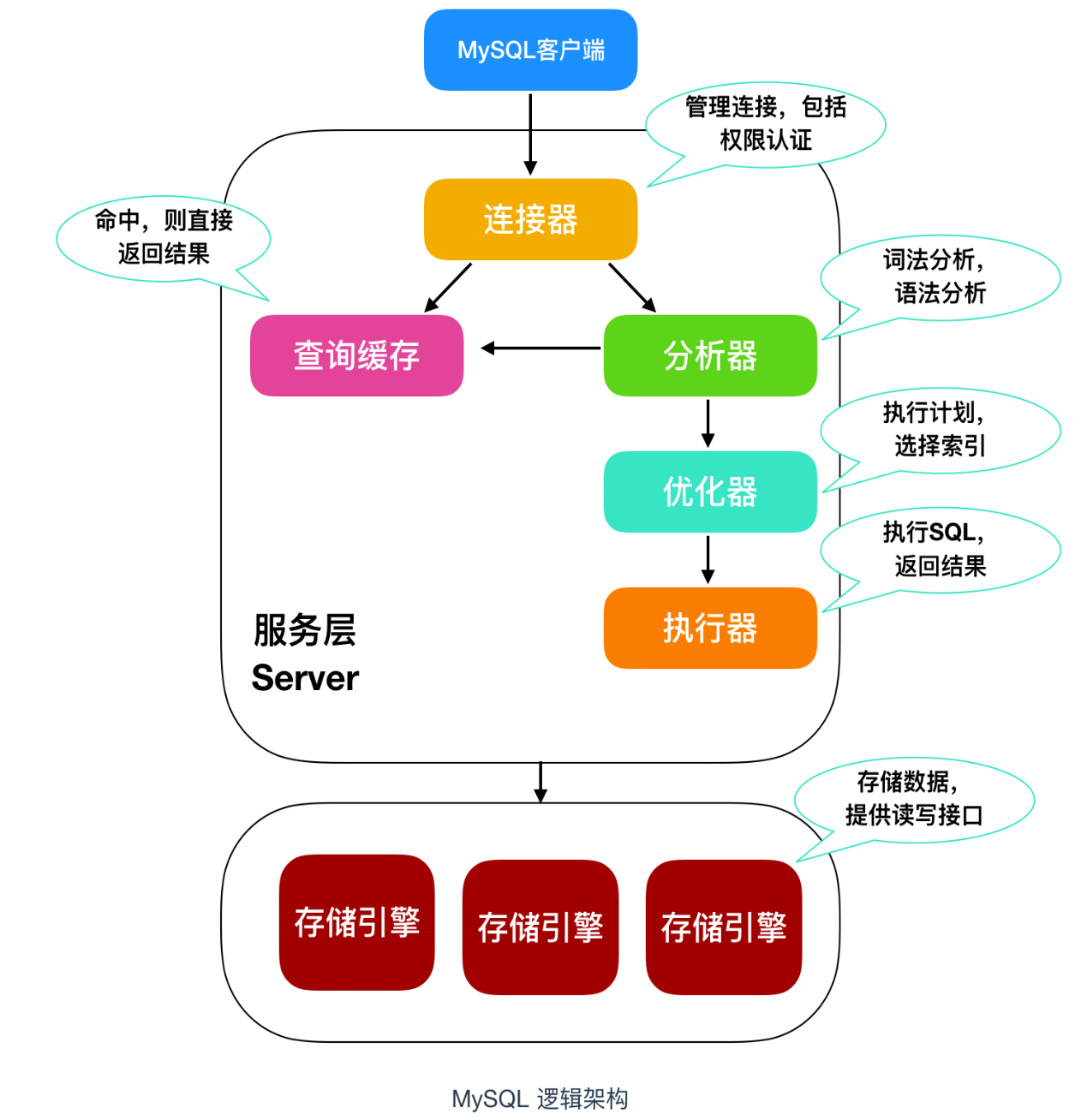
* 支持事务操作，具有事务 ACID 隔离特性，默认的隔离级别是可重复读(repetable-read)、通过MVCC（并发版本控制）来实现的。能够解决脏读和不可重复读的问题。
* InnoDB 支持外键操作。
* InnoDB 默认的锁粒度行级锁，并发性能比较好，会发生死锁的情况。
* 和 MyISAM 一样的是，InnoDB 存储引擎也有 .frm文件存储表结构 定义，但是不同的是，InnoDB 的表数据与索引数据是存储在一起的，都位于 B+ 数的叶子节点上，而 MyISAM 的表数据和索引数据是分开的。
* InnoDB 有安全的日志文件，这个日志文件用于恢复因数据库崩溃或其他情况导致的数据丢失问题，保证数据的一致性。
* InnoDB 和 MyISAM 支持的索引类型相同，但具体实现因为文件结构的不同有很大差异。
* 增删改查性能方面，如果执行大量的增删改操作，推荐使用 InnoDB 存储引擎，它在删除操作时是对行删除，不会重建表。

### MyISAM 和 InnoDB 存储引擎的对比

* 锁粒度方面：由于锁粒度不同，InnoDB 比 MyISAM 支持更高的并发；InnoDB 的锁粒度为行锁、MyISAM 的锁粒度为表锁、行锁需要对每一行进行加锁，所以锁的开销更大，但是能解决脏读和不可重复读的问题，相对来说也更容易发生死锁
* 可恢复性上：由于 InnoDB 是有事务日志的，所以在产生由于数据库崩溃等条件后，可以根据日志文件进行恢复。而 MyISAM 则没有事务日志。
* 查询性能上：MyISAM 要优于 InnoDB，因为 InnoDB 在查询过程中，是需要维护数据缓存，而且查询过程是先定位到行所在的数据块，然后在从数据块中定位到要查找的行；而 MyISAM 可以直接定位到数据所在的内存地址，可以直接找到数据。
* 表结构文件上：MyISAM 的表结构文件包括：.frm(表结构定义),.MYI(索引),.MYD(数据)；而 InnoDB 的表数据文件为:.ibd和.frm(表结构定义)；

## MySQL 基础架构

这道题应该从 MySQL 架构来理解，我们可以把 MySQL 拆解成几个零件，如下图所示



大致上来说，MySQL 可以分为 Server层和 存储引擎层。

Server 层包括连接器、查询缓存、分析器、优化器、执行器，包括大多数 MySQL 中的核心功能，所有跨存储引擎的功能也在这一层实现，包括 **存储过程、触发器、视图等**。

存储引擎层包括 MySQL 常见的存储引擎，包括 **MyISAM、InnoDB 和 Memory** 等，最常用的是 InnoDB，也是现在 MySQL 的默认存储引擎。存储引擎也可以在创建表的时候手动指定，比如下面

CREATE TABLE t (i INT) ENGINE = <Storage Engine>;

然后我们就可以探讨 MySQL 的执行过程了

### 连接器

首先需要在 MySQL 客户端登陆才能使用，所以需要一个连接器来连接用户和 MySQL 数据库，我们一般是使用

mysql -u 用户名 -p 密码

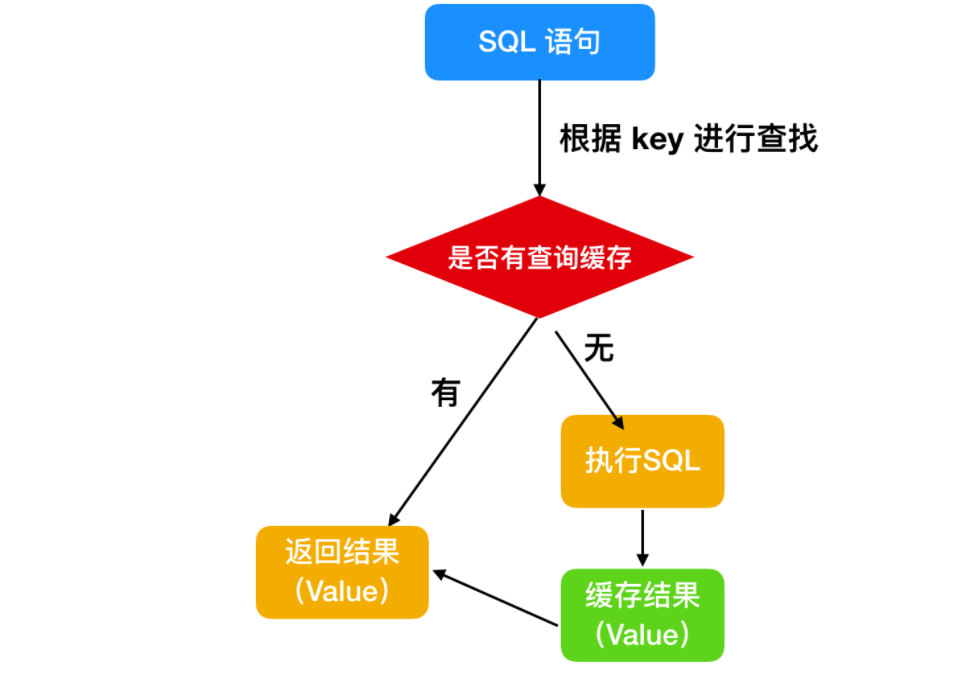
来进行 MySQL 登陆，和服务端建立连接。在完成 TCP 握手 后，连接器会根据你输入的用户名和密码验证你的登录身份。如果用户名或者密码错误，MySQL 就会提示 **Access denied for user**，来结束执行。如果登录成功后，MySQL 会根据权限表中的记录来判定你的权限。

### 查询缓存

连接完成后，你就可以执行 SQL 语句了，这行逻辑就会来到第二步：查询缓存。

MySQL 在得到一个执行请求后，会首先去 查询缓存 中查找，是否执行过这条 SQL 语句，之前执行过的语句以及结果会以 key-value 对的形式，被直接放在内存中。key 是查询语句，value 是查询的结果。如果通过 key 能够查找到这条 SQL 语句，就直接返回 SQL 的执行结果。

如果语句不在查询缓存中，就会继续后面的执行阶段。执行完成后，执行结果就会被放入查询缓存中。可以看到，如果查询命中缓存，MySQL 不需要执行后面的复杂操作，就可以直接返回结果，效率会很高。



**但是查询缓存不建议使用**

为什么呢？因为只要在 MySQL 中对某一张表执行了更新操作，那么所有的查询缓存就会失效，对于更新频繁的数据库来说，查询缓存的命中率很低。

### 分析器

如果没有命中查询，就开始执行真正的 SQL 语句。

* 首先，MySQL 会根据你写的 SQL 语句进行解析，分析器会先做 词法分析，你写的 SQL 就是由多个字符串和空格组成的一条 SQL 语句，MySQL 需要识别出里面的字符串是什么，代表什么。
* 然后进行 语法分析，根据词法分析的结果， 语法分析器会根据语法规则，判断你输入的这个 SQL 语句是否满足 MySQL 语法。如果 SQL 语句不正确，就会提示 **You have an error in your SQL syntax**

### 优化器

经过分析器的词法分析和语法分析后，你这条 SQL 就合法了，MySQL 就知道你要做什么了。但是在执行前，还需要进行优化器的处理，优化器会判断你使用了哪种索引，使用了何种连接，优化器的作用就是确定效率最高的执行方案。

### 执行器

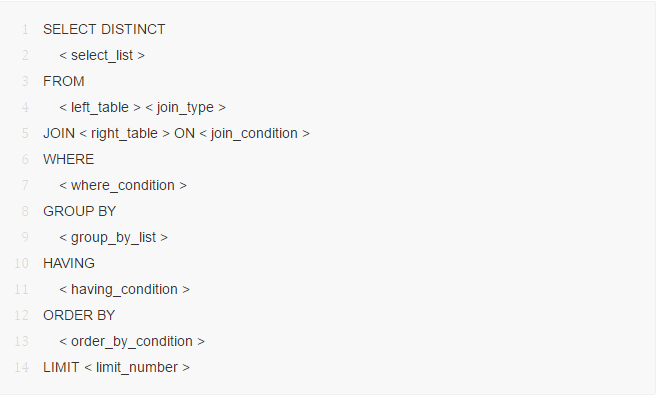
MySQL 通过分析器知道了你的 SQL 语句是否合法，你想要做什么操作，通过优化器知道了该怎么做效率最高，然后就进入了执行阶段，开始执行这条 SQL 语句

在执行阶段，MySQL 首先会判断你有没有执行这条语句的权限，没有权限的话，就会返回没有权限的错误。如果有权限，就打开表继续执行。打开表的时候，执行器就会根据表的引擎定义，去使用这个引擎提供的接口。对于有索引的表，执行的逻辑也差不多。

至此，MySQL 对于一条语句的执行过程也就完成了。

## SQL 的执行顺序

我们在编写一个查询语句的时候



它的执行顺序你知道吗？这道题就给你一个回答。

### FROM 连接

首先，对 SELECT 语句执行查询时，对FROM 关键字两边的表执行连接，会形成笛卡尔积，这时候会产生一个虚表VT1(virtual table)

**“**

首先先来解释一下什么是笛卡尔积

现在我们有两个集合 A = {0,1} , B = {2,3,4}

那么，集合 A \* B 得到的结果就是

A \* B = {(0,2)、(1,2)、(0,3)、(1,3)、(0,4)、(1,4)};

B \* A = {(2,0)、{2,1}、{3,0}、{3,1}、{4,0}、(4,1)};

上面 A \* B 和 B \* A 的结果就可以称为两个集合相乘的笛卡尔积

我们可以得出结论，A 集合和 B 集合相乘，包含了集合 A 中的元素和集合 B 中元素之和，也就是 A 元素的个数 \* B 元素的个数

再来解释一下什么是虚表

**“**

在 MySQL 中，有三种类型的表

一种是永久表，永久表就是创建以后用来长期保存数据的表

一种是临时表，临时表也有两类，一种是和永久表一样，只保存临时数据，但是能够长久存在的；还有一种是临时创建的，SQL 语句执行完成就会删除。

一种是虚表，虚表其实就是视图，数据可能会来自多张表的执行结果。

### ON 过滤

然后对 FROM 连接的结果进行 ON 筛选，创建 VT2，把符合记录的条件存在 VT2 中。

### JOIN 连接

第三步，如果是 OUTER JOIN(left join、right join) ，那么这一步就将添加外部行，如果是 left join 就把 ON 过滤条件的左表添加进来，如果是 right join ，就把右表添加进来，从而生成新的虚拟表 VT3。

### WHERE 过滤

第四步，是执行 WHERE 过滤器，对上一步生产的虚拟表引用 WHERE 筛选，生成虚拟表 VT4。

WHERE 和 ON 的区别

* 如果有外部列，ON 针对过滤的是关联表，主表(保留表)会返回所有的列;
* 如果没有添加外部列，两者的效果是一样的;

应用

* 对主表的过滤应该使用 WHERE;
* 对于关联表，先条件查询后连接则用 ON，先连接后条件查询则用 WHERE;

### GROUP BY

根据 group by 字句中的列，会对 VT4 中的记录进行分组操作，产生虚拟机表 VT5。果应用了group by，那么后面的所有步骤都只能得到的 VT5 的列或者是聚合函数（count、sum、avg等）。

### HAVING

紧跟着 GROUP BY 字句后面的是 HAVING，使用 HAVING 过滤，会把符合条件的放在 VT6

### SELECT

第七步才会执行 SELECT 语句，将 VT6 中的结果按照 SELECT 进行刷选，生成 VT7

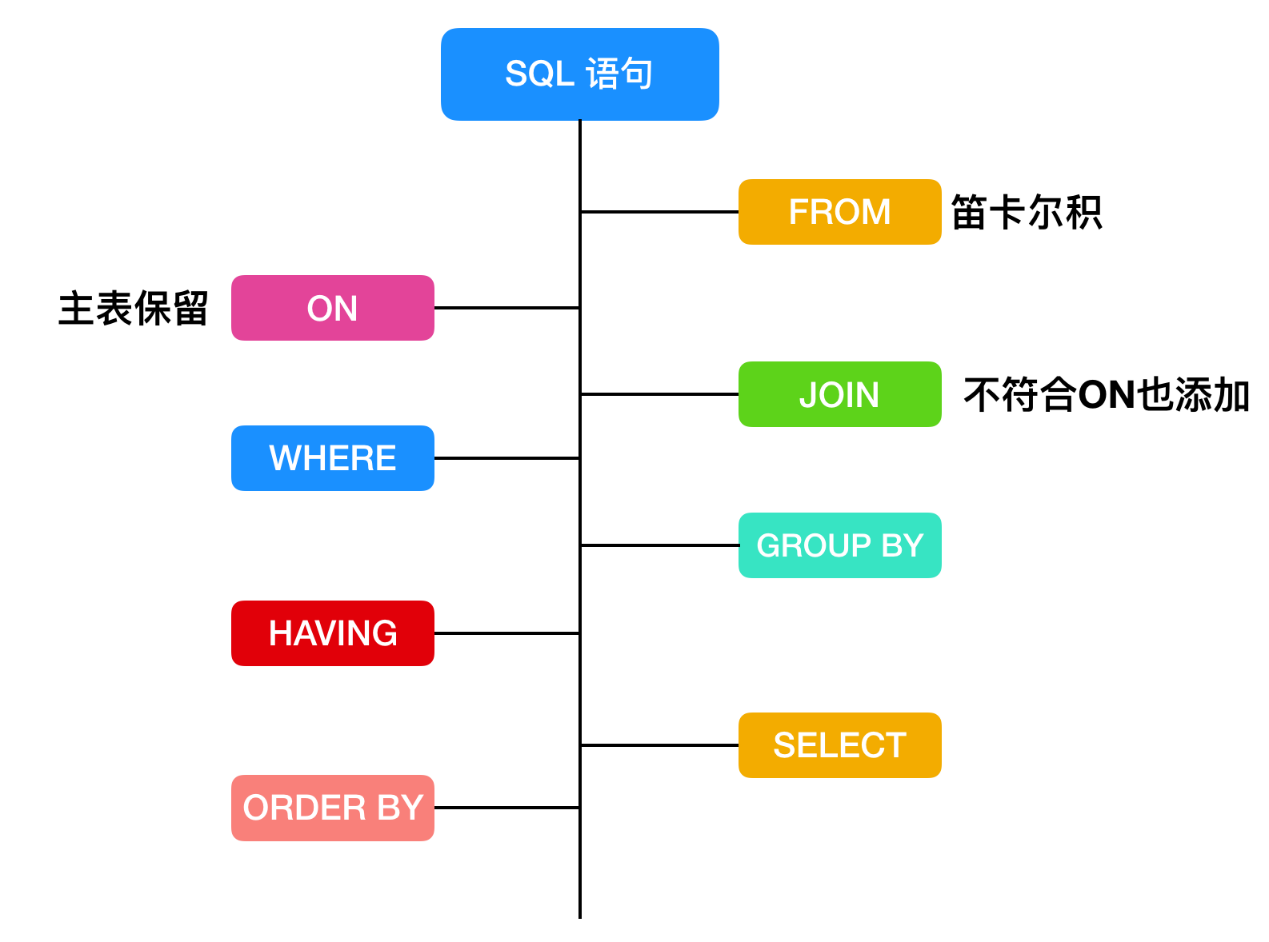
### DISTINCT

在第八步中，会对 TV7 生成的记录进行去重操作，生成 VT8。事实上如果应用了 group by 子句那么 distinct 是多余的，原因同样在于，分组的时候是将列中唯一的值分成一组，同时只为每一组返回一行记录，那么所以的记录都将是不相同的。

### ORDER BY

应用 order by 子句。按照 order\_by\_condition 排序 VT8，此时返回的一个游标，而不是虚拟表。sql 是基于集合的理论的，集合不会预先对他的行排序，它只是成员的逻辑集合，成员的顺序是无关紧要的。

SQL 语句执行的过程如下



**什么是临时表，何时删除临时表**

什么是临时表？MySQL 在执行 SQL 语句的过程中，通常会临时创建一些存储中间结果集的表，临时表只对当前连接可见，在连接关闭时，临时表会被删除并释放所有表空间。

临时表分为两种：一种是内存临时表，一种是磁盘临时表，什么区别呢？内存临时表使用的是 MEMORY 存储引擎，而临时表采用的是 MyISAM 存储引擎。

**“**

MEMORY 存储引擎：memory 是 MySQL 中一类特殊的存储引擎，它使用存储在内容中的内容来创建表，而且**数据全部放在内存中**。每个基于 MEMORY 存储引擎的表实际对应一个磁盘文件。该文件的文件名与表名相同，类型为 frm 类型。而其数据文件，都是存储在内存中，这样有利于数据的快速处理，提高整个表的效率。MEMORY 用到的很少，因为它是把数据存到内存中，如果内存出现异常就会影响数据。如果重启或者关机，所有数据都会消失。因此，基于 MEMORY 的表的生命周期很短，一般是一次性的。

MySQL 会在下面这几种情况产生临时表

* 使用 UNION 查询：UNION 有两种，一种是UNION ，一种是 UNION ALL ，它们都用于联合查询；区别是 使用 UNION 会去掉两个表中的重复数据，相当于对结果集做了一下去重(distinct)。使用 UNION ALL，则不会排重，返回所有的行。使用 UNION 查询会产生临时表。
* 使用 TEMPTABLE 算法或者是 UNION 查询中的视图。TEMPTABLE 算法是一种创建临时表的算法，它是将结果放置到临时表中，意味这要 MySQL 要先创建好一个临时表，然后将结果放到临时表中去，然后再使用这个临时表进行相应的查询。
* ORDER BY 和 GROUP BY 的子句不一样时也会产生临时表。
* DISTINCT 查询并且加上 ORDER BY 时；
* SQL 用到 SQL\_SMALL\_RESULT 选项时；如果查询结果比较小的时候，可以加上 SQL\_SMALL\_RESULT 来优化，产生临时表
* FROM 中的子查询；
* EXPLAIN 查看执行计划结果的 Extra 列中，如果使用 Using Temporary 就表示会用到临时表。

**MySQL 常见索引类型**

索引是存储在一张表中特定列上的数据结构，索引是在列上创建的。并且，索引是一种数据结构。

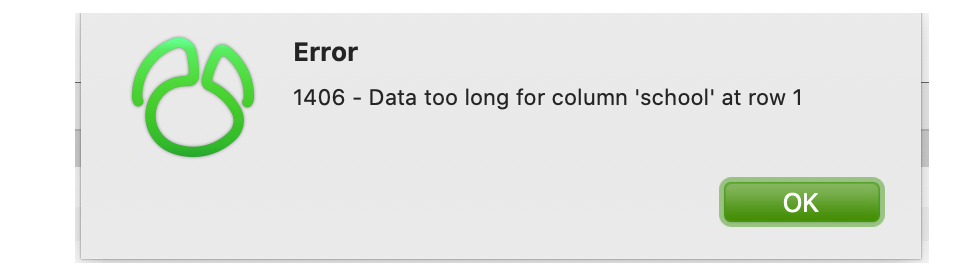
在 MySQL 中，主要有下面这几种索引

* 全局索引(FULLTEXT)：全局索引，目前只有 MyISAM 引擎支持全局索引，它的出现是为了解决针对文本的模糊查询效率较低的问题。
* 哈希索引(HASH)：哈希索引是 MySQL 中用到的唯一 key-value 键值对的数据结构，很适合作为索引。HASH 索引具有一次定位的好处，不需要像树那样逐个节点查找，但是这种查找适合应用于查找单个键的情况，对于范围查找，HASH 索引的性能就会很低。
* B-Tree 索引：B 就是 Balance 的意思，BTree 是一种平衡树，它有很多变种，最常见的就是 B+ Tree，它被 MySQL 广泛使用。
* R-Tree 索引：R-Tree 在 MySQL 很少使用，仅支持 geometry 数据类型，支持该类型的存储引擎只有MyISAM、BDb、InnoDb、NDb、Archive几种，相对于 B-Tree 来说，R-Tree 的优势在于范围查找。

## varchar 和 char 的区别和使用场景

MySQL 中没有 nvarchar 数据类型，所以直接比较的是 varchar 和 char 的区别

char ：表示的是定长的字符串，当你输入小于指定的数目，比如你指定的数目是char(6)，当你输入小于 6 个字符的时候，char 会在你最后一个字符后面补空值。当你输入超过指定允许最大长度后，MySQL 会报错



varchar：varchar 指的是长度为 n 个字节的可变长度，并且是非Unicode的字符数据。n 值是介于 1 - 8000 之间的数值。存储大小为实际大小。

**“**

Unicode 是一种字符编码方案，它为每种语言中的每个字符都设定了统一唯一的二进制编码，以实现跨语言、跨平台进行文本转换、处理的要求

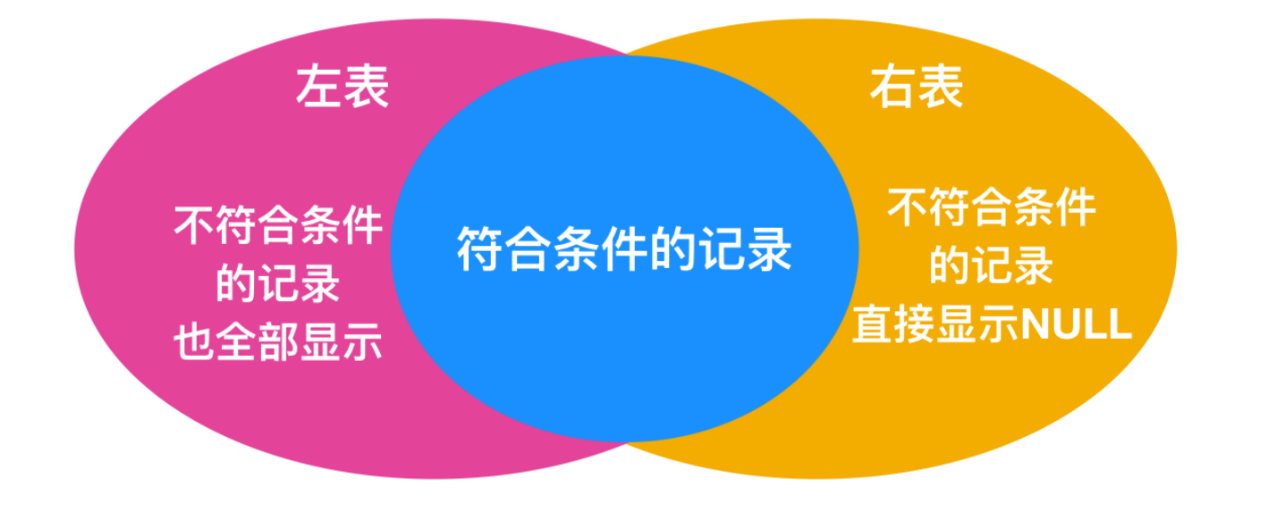
使用 char 存储定长的数据非常方便、char 检索效率高，无论你存储的数据是否到了 10 个字节，都要去占用 10 字节的空间

使用 varchar 可以存储变长的数据，但存储效率没有 char 高。

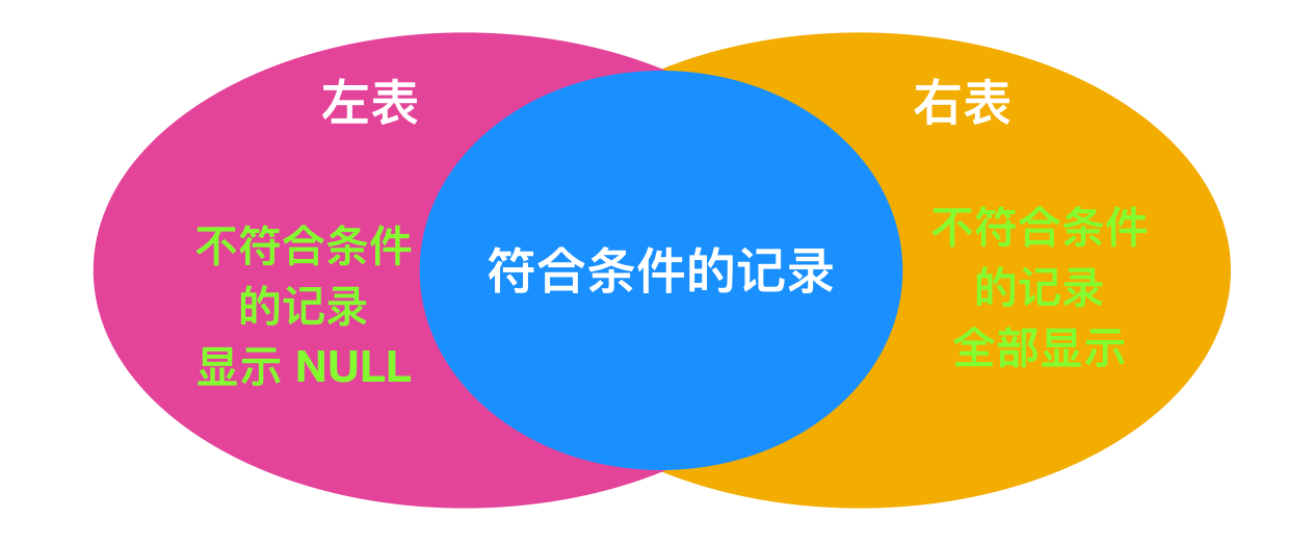
## 什么是 内连接、外连接、交叉连接、笛卡尔积

连接的方式主要有三种：**外连接、内链接、交叉连接**

* 外连接(OUTER JOIN)：外连接分为三种，分别是左外连接(LEFT OUTER JOIN 或 LEFT JOIN) 、右外连接(RIGHT OUTER JOIN 或 RIGHT JOIN) 、全外连接(FULL OUTER JOIN 或 FULL JOIN)

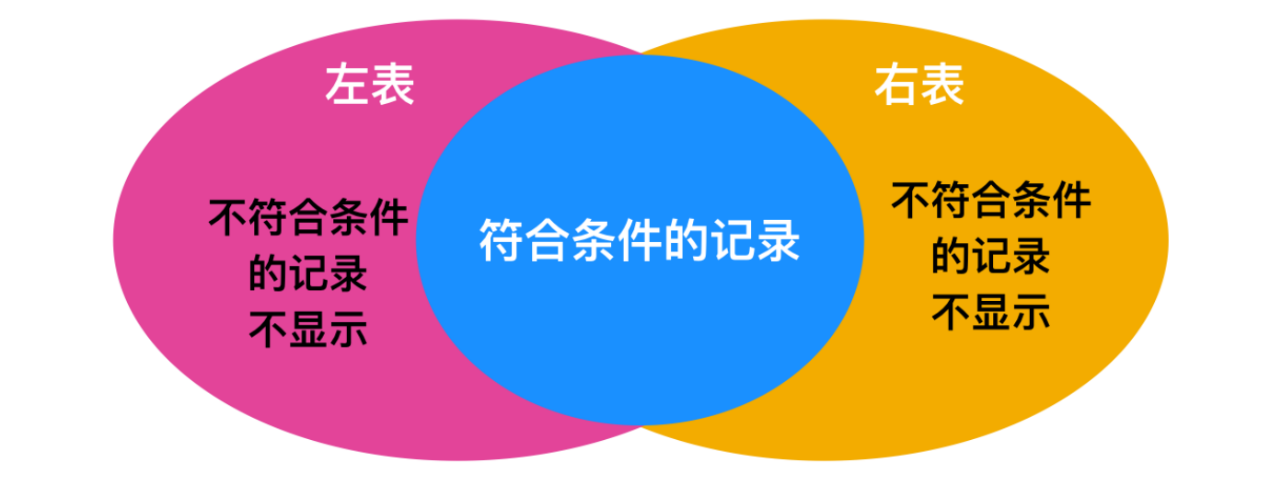
左外连接：又称为左连接，这种连接方式会显示左表不符合条件的数据行，右边不符合条件的数据行直接显示 NULL

右外连接：也被称为右连接，他与左连接相对，这种连接方式会显示右表不      符合条件的数据行，左表不符合条件的数据行直接显示 NULL



**MySQL 暂不支持全外连接**

* 内连接(INNER JOIN)：结合两个表中相同的字段，返回关联字段相符的记录。



* 笛卡尔积(Cartesian product)：我在上面提到了笛卡尔积，为了方便，下面再列出来一下。

**“**

现在我们有两个集合 A = {0,1} , B = {2,3,4}

那么，集合 A \* B 得到的结果就是

A \* B = {(0,2)、(1,2)、(0,3)、(1,3)、(0,4)、(1,4)};

B \* A = {(2,0)、{2,1}、{3,0}、{3,1}、{4,0}、(4,1)};

上面 A \* B 和 B \* A 的结果就可以称为两个集合相乘的 笛卡尔积

我们可以得出结论，A 集合和 B 集合相乘，包含了集合 A 中的元素和集合 B 中元素之和，也就是 A 元素的个数 \* B 元素的个数

* 交叉连接的原文是Cross join ，就是笛卡尔积在 SQL 中的实现，SQL中使用关键字CROSS JOIN来表示交叉连接，在交叉连接中，随便增加一个表的字段，都会对结果造成很大的影响。

SELECT \* FROM t\_Class a CROSS JOIN t\_Student b WHERE a.classid=b.classid

或者不用 CROSS JOIN，直接用 FROM 也能表示交叉连接的效果

SELECT \* FROM t\_Class a ,t\_Student b WHERE a.classid=b.classid

如果表中字段比较多，不适宜用交叉连接，交叉连接的效率比较差。

* 全连接：全连接也就是 full join，MySQL 中不支持全连接，但是可以使用其他连接查询来模拟全连接，可以使用 UNION 和 UNION ALL 进行模拟。例如

(select colum1,colum2...columN from tableA ) union (select colum1,colum2...columN from tableB )  
  
或 (select colum1,colum2...columN from tableA ) union all (select colum1,colum2...columN from tableB )；

使用 UNION 和 UNION ALL 的注意事项

**“**

通过 union 连接的 SQL 分别单独取出的列数必须相同

使用 union 时，多个相等的行将会被合并，由于合并比较耗时，一般不直接使用 union 进行合并，而是通常采用 union all 进行合并

**谈谈 SQL 优化的经验**

* 查询语句无论是使用哪种判断条件 **等于、小于、大于**， WHERE 左侧的条件查询字段不要使用函数或者表达式
* 使用 EXPLAIN 命令优化你的 SELECT 查询，对于复杂、效率低的 sql 语句，我们通常是使用 explain sql 来分析这条 sql 语句，这样方便我们分析，进行优化。
* 当你的 SELECT 查询语句只需要使用一条记录时，要使用 LIMIT 1
* 不要直接使用 SELECT \*，而应该使用具体需要查询的表字段，因为使用 EXPLAIN 进行分析时，SELECT \* 使用的是全表扫描，也就是 type = all。
* 为每一张表设置一个 ID 属性
* 避免在 WHERE 字句中对字段进行 NULL 判断
* 避免在 WHERE 中使用 != 或 <> 操作符
* 使用 BETWEEN AND 替代 IN
* 为搜索字段创建索引
* 选择正确的存储引擎，InnoDB 、MyISAM 、MEMORY 等
* 使用 LIKE %abc% 不会走索引，而使用 LIKE abc% 会走索引
* 对于枚举类型的字段(即有固定罗列值的字段)，建议使用ENUM而不是VARCHAR，如性别、星期、类型、类别等
* 拆分大的 DELETE 或 INSERT 语句
* 选择合适的字段类型，选择标准是 **尽可能小、尽可能定长、尽可能使用整数**。
* 字段设计尽可能使用 NOT NULL
* 进行水平切割或者垂直分割

**“**

水平分割：通过建立结构相同的几张表分别存储数据

垂直分割：将经常一起使用的字段放在一个单独的表中，分割后的表记录之间是一一对应关系。