1、MySQL 字段类型

1）数值类型：整型(int)、浮点型、定点型；

2）字符串类型：char、varchar；

3）日期时间类型：datatime；

2、整数类型的 unsigned 属性有什么用？

表示*非负数的无符号整数*，可以将正整数的上限提高一倍（因为它不用存储负数值，避免使用符号位，节省内存空间）；适合从 0 开始递增的 ID 列；如 tinyint(-128~127)→0~255

3、CHAR 和 VARCHAR 的区别是什么？

无论字母/数字/中文，每个都只占用一个字符。

CHAR 是定长字符串，在*存储时*会在右边填充空格以达到指定的长度，*检索时*去掉空格；

VARCHAR 是变长字符串，在*存储时*需要使用 1 或 2 个*额外字节*记录字符串的长度，*检索时*不需要处理。

4、VARCHAR(100)和 VARCHAR(10)的区别

都是变长字符串，表示最多存储 100 和 10 个字符；

VARCHAR(100)在*内存操作（排序）时*分配固定大小内存块，会消耗更多的内存；

VARCHAR(10)要存储*超过 10 个字符时*，需要修改表结构；

5、DECIMAL 和 FLOAT/DOUBLE 的区别

decimal 是*定点数*，存储*精确的*小数值；（货币）对应 Java 中的 java.math.*BigDecimal 类*；

float/double 是*浮点数*，存储*近似的*小数值；

6、为什么不推荐使用 TEXT 和 BLOB？

TEXT 和 varchar 类似，但能存储更多字符；

1）不能有默认值；

2）无法使用内存临时表，只能在*磁盘上*创建；

3）检索效率低；

4）不能直接创建索引，需要指定前缀长度；

5）可能会消耗大量*网络 IO 带宽*；

7、DATETIME 和 TIMESTAMP 的区别

DATETIME 类型没有时区信息（占 8 字节）

TIMESTAMP 和时区有关，但时间范围小（4）

【补充】数值时间戳也可以表示时间，

优：日期排序以及对比等操作效率高，跨系统；

缺：可读性差；

8、NULL 和' '的区别是什么？

1）NULL 代表一个不确定的值；两个 NULL 比较时并不相等，但 distinct/groupby/orderby 中，NULL 是相等的。

2）NULL 需要*占用空间，存储 NULL 个数*；' '的长度是 0，*不占用空间*；

3）NULL 会影响聚合函数的结果，sum/avg 等聚合函数会忽略 NULL 值；count(*列*)会忽略 NULL，count(*)*不会忽略* NULL；

4）查询 NULL 值时，必须用 ***IS NULL*** 或 ***IS NOT NULL*** 来判断；而' '可使用比较运算符。

9、Boolean 类型如何表示？

没布尔类型，用 ***TINYINT(1)类型***表示布尔值

10、MyISAM 和 InnoDB 有什么区别？

MySQL 默认 InnoDB，只 InnoDB 支持事务；

1）MySQL 5.5.5 之前，MyISAM 是 MySQL 的默认存储引擎；5.5.5 之后，InnoDB 是默认；

2）InnoDB 支持行级锁；事务；外键；MVCC；

3）索引实现不同；

4）InnoDB 支持数据库异常崩溃后的安全恢复，依赖 redo log；MyISAM 不支持。

11、MySQL 日志---- redo log（重做日志）

记录的是*事务提交时数据页的物理修改*，用来实现*事务的持久性*；

内存中*重做日志缓冲*和磁盘中*重做日志文件*，当事务提交后，所有修改信息都在该日志文件中，当*刷新脏页到磁盘发生错误*时，用于数据恢复。

12、MySQL 日志----binlog（重做）

记录数据更新的语句，逻辑日志；用来实现*灾后数据恢复*和*主从复制*；

13、MySQL 日志----undo log（回滚）

记录数据被修改前的信息，逻辑日志；用来实现*事务的原子性*；（数据回滚，MVCC）

delete 时记录 insert 语句，回滚时直执行即可；

14、MySQL 数据库事务

事务是将多个数据库操作构成逻辑上的整体，要么都执行成功，要么都不执行。

【数据库事务的四大特性 ACID】

1）**原子性**：事务执行的最小单位；(要么要么)

2）**一致性**：执行事务前后，数据保持一致；

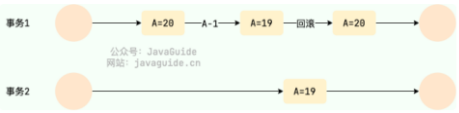
3）**隔离性**：并发访问时，不被其他事务干扰；

4）**持久性**：提交后，DB 发生故障也不受影响

15、并发事务带来的问题

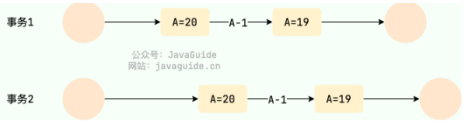
1）脏读：

事务 1 读取某表中数据 A=20，事务 1 修改 A=A-1，事务 2 读取到 A=19,事务 1 回滚导致对 A 修改并未提交到数据库，A 值还是 20。



2）丢弃修改：

事务 1 读取某表中的数据 A=20，事务 2 也读取 A=20，事务 1 先修改 A=A-1，事务 2 后来也修改 A=A-1，最终结果 A=19，事务 1 的修改被丢失。（时间戳或版本号）



3）不可重复读：

事务 1 读取某表中的数据 A=20，事务 2 也读取 A=20，事务 1 修改 A=A-1，事务 2 再次读取 A =19，此时读取的结果和第一次读取的结果不同。

4）幻读：

事务 2 读取某个范围的数据，事务 1 在这个范围插入了新的数据，事务 2 再次读取这个范围的数据发现相比于第一次读取的结果多了新的数据。

【补充：不可重复读和幻读区别】

不可重复读：*多次读取一条记录*发现其中某些记录的值被修改；

幻读：多*次执行同一条查询语句*时，发现查到的记录条数增加了。

16、并发事务的控制方式（锁和 MVCC）

1）锁（悲观控制）：通过*读写锁*显式*控制共享资源*来实现并发控制；

2）MVCC（乐观控制）：多版本并发控制方法，对一份数据会存储多个版本，通过事务的可见性来保证事务能看到自己应该看到的版本。

17、SQL 标准定义了四个事务隔离级别

1）读取未提交：允许读取未提交的数据变更；

2）读取已提交：允许读取并发事务已经提交的数据；（阻止脏读）

3）可重复读：对同一字段多次读取结果都是一致的，*除非*数据是*被本身事务所修改*（阻止脏读和不可重复读）（MySQL 默认）

4）可串行化：所有的事务依次逐个执行，完全服从 ACID 隔离级别；（阻止脏读、不可重复读和幻读）

18、MySQL 的隔离级别是怎么实现的？

MySQL 的隔离级别基于锁和 MVCC 机制共同实现的。

可串行化隔离级别是通过锁来实现的；读已提交和可重复读是基于 MVCC 实现的，但可重复读在当前读的情况下要加锁，防止幻读。

19、表级锁和行级锁了解吗？有什么区别？

1）表级锁：

1>锁定粒度大；

2>针对*非索引字段*加的锁，对*整张表*加锁；

3>触发锁冲突的概率最高，高并发下效率低；

4>资源消耗也比较少，加锁快，不会出现死锁；

5>表级锁和存储引擎无关，MyISAM 和 InnoDB 引擎都支持表级锁。

2）行级锁：

1>锁定粒度最小；

2>针对*索引字段*加的锁，只针对*行记录*加锁；

3>行级锁能*减少数据库操作冲突*，并发度高；

4>加锁的开销大，加锁慢，会出现死锁；

5>行级锁和存储引擎有关，在存储引擎实现。

20、行级锁的使用有什么注意事项？

当我们执行 UPDATE、DELETE 语句时，如果 WHERE 条件中字段没有命中唯一索引或者索引失效的话，就会导致扫描全表对表中的所有行记录进行加锁。

21、InnoDB 有哪几类行锁？

记录锁（Record Lock）：锁定单个行记录；

间隙锁（Gap Lock）：锁定一个范围，不包括记录本身；

临键锁（Next-Key Lock）：Record Lock+Gap Lock，锁定一个范围，包含记录本身，主要目的是为了解决幻读问题。记录锁只能锁住已经存在的记录，为了避免插入新记录，需要依赖间隙锁。

【补充】行锁默认使用的是临键锁(,]；

优化 1：索引上等值查询，给唯一索引加锁时，临键锁退化成记录锁；

优化 2：索引上等值查询，向右遍历且最后一个值不满足等值条件时，退化成间隙锁；

22、共享锁和排他锁

表级锁和行级锁，都存在共享锁和排它锁；

共享锁：事务在读取记录的时候获取共享锁，允许多个事务同时获取；（锁兼容）

排他锁：事务在修改记录的时候获取排他锁，不允许多个事务同时获取；（锁不兼容）

23、意向锁有什么作用？

表级锁，快速判断是否可以对某表使用表锁；

意向共享锁：事务有意向对表中某些记录加共享锁，加共享锁前必须先取得该表的 IS 锁；

意向排他锁：事务有意向对表中某些记录加排他锁，加排他锁前必须先取得该表的 IX 锁；

24、快照读和当前读有什么区别？

1）快照读（一致性非锁定读）：单纯的 select 语句使用快照读；当读取的记录正在执行更新或删除时，读取操作不会等，而是直接读取一个快照；（MVCC；适合一致性要求不高的）

1>读取已提交：被锁定行的最新一份快照；

2>可重复读：本事务开始时的行数据快照；

2）当前读（一致性锁定读）：给行记录加 X 锁或 S 锁；select 语句最后加 **for update/share**；

25、InnoDB 对 MVCC 的实现

维护一个数据的多个版本，读写没有冲突；

1）隐藏字段：最新操作的事务 ID、回滚指针（指向该记录的上一个版本）、**rowid 主键**；

2）undo log 版本链：链表头最新，尾最旧；

3）readview 读视图：记录当前活跃的（未提交）事务 id，MVCC 提取数据的依据；

（当前活跃事务 ID 集合，最小活跃事务 ID，预分配事务 ID<max+1>，readview 创建者 ID）

trx_id=创建者 ID→当前事务更改的，可访问；

trx_id<min_trx_id→该事务已提交，可访问；

trx_id>max_trx_id→该 tx 在读视图后，不可；

min<tx<max，tx 不在 ids→该 tx 已提交，可；

26、能用 MySQL 直接存储文件（如图片）吗？

可以，存储文件对应的二进制数据；但是会消耗存储空间，影响数据库性能。

【注意】数据库只存储*文件地址*信息，文件由*文件存储服务*负责存储。

27、MySQL 如何存储 IP 地址？

将 IP 地址转换成整形数据存储，性能更好，占用空间也更小；两种方法：

1）INET_ATON()：IP→无符号整型（4-8 位）

2）INET_NTOA()：无符号整型→IP 地址

28、什么是读写分离？如何实现读写分离？

读写分离主要是将对数据库的读写操作分散到不同的数据库节点上。一般一主（写）多从（读），*主从复制*实现数据同步保持数据一致性；适用于读多写少的情况，大幅度提高读性能，小幅度提高写性能；

【补充】代理层负责所有请求的读写分离，路由到对应数据库；（MySQL Router 自动分辨）

第三方组件：shardingsphere；

29、主从复制原理是什么？

binlog（二进制日志文件）主要记录了 MySQL 数据库中数据的所有变化；

1）*主库*将数据库中数据的变化写入到 binlog；

2）*从库*连接主库；

3）*从库*会创建一个 I/O 线程向主库请求更新的 binlog；

4）*主库*会创建一个 binlog dump 线程来发送 binlog，从库中的 I/O 线程负责接收；

5）*从库*的 I/O 线程将接收的 binlog 写入到 relay log 中；

6）*从库*的 SQL 线程读取 relay log 同步数据到本地（即再执行一遍 SQL）。

【补充 canal】模拟 MySQL 主从复制的过程，解析 binlog 将数据同步到其他的数据源(ES)；

30、如何避免主从延迟？

主从延迟：主库写完，主从同步到从库的时间；

1）强制将那些必须获取最新数据的读请求路由到主库处理，使用 Sharding-JDBC 的分片键值管理器；

2）延迟读取：设计业务流程时，多加一个页面跳转，避免立即响应读请求。

31、什么情况下会出现主从延迟？如何尽量减少延迟？

主要原因：1）*从库 I/O 线程接收* binlog 的速度跟不上*主库写入* binlog 的速度；2）从库 SQL 线程*执行* relay log 的速度跟不上*从库 I/O 线程接收* binlog 的速度。

情况：1）从库机器性能比主库差；

2）从库处理过多的读请求，占用 CPU 资源，复制效率慢；（*读请求分散*到不同从库）

3）异步复制；（*半同步复制*）

4）从库太多，主库同步压力大，写慢；（从库分层）

5）长时间未提交的事务（大事务），同步很慢；（尽量分批修改数据）

32、什么是分库？

将数据库中的数据分散到不同的数据库上；

垂直分库：不同的业务使用不同的数据库；

水平分库：同一个表按一定规则拆分到不同的数据库，解决单表存储的瓶颈问题。

33、什么是分表？

分表就是对单表的数据进行拆分；

垂直分表：对数据表的列进行拆分；

水平分表：对数据表的行进行拆分；

（*水平分表*通常和*水平分库*同时出现）

34、什么情况下需要考虑分库分表？

1）单表数据千万级别以上，读写速度缓慢；

2）数据占用空间越来越大，备份时间长；（成本高，非必要不采用）

35、常见的分片算法有哪些？

1）哈希分片：根据*分片键哈希值*确定分片；（适合随机读写，不适合范围查询）

2）范围分片：根据特定的范围区间（ID 等）；（适合经常范围查询且数据分布均匀）

3）映射表分片：*单个表*存储分片键/分片位置；（需要额外维护一个表）

4）一致性哈希分片：将*哈希空间*组织成一个环形，*分片键和节点*映射到环上，按*顺序分配*；（解决哈希方法不能加数据库数量的问题）

36、分片键如何选择？

1）数据尽可能均匀分布在各分片；

2）单次查询尽量减少分片数量；

3）分片键的值要稳定，不发生变化；

4）支持分片动态增加；

37、分库分表会带来什么问题呢？

1）无法 jion 操作；（封装数据多次查询业务层）

2）单个数据库的事务不能用于分布式；（引入分布式事务）

3）数据库自增主键不适用了；（引入分布式 ID）

4）跨库聚合查询问题；（多个分片聚合，汇总后输出）

38、分库分表后，数据怎么迁移呢？

1）停机迁移：在系统使用人数非常少时，写个脚本，将老库的数据写到新库中；

2）双写方案：

1>对老库的更新同时写入新库中，若新库不存在，就从老库插入新库；

2>新库老库对比，没有更新的数据插入新库；

39、深度分页优化建议

1）范围查询

当主键连续时，可以根据 ID 范围分页；

2）延迟关联

先根据“起始页，页码”查询出主键值，再将主键值构成的表 2 与分页查询的表 1 根据 id 内连接；

3）覆盖索引，避免回表操作

40、什么是数据库冷热分离？

冷数据：不经常访问，但要长期保存的数据；

热数据：经常被访问和修改且需*要快速访问*的数据；

【区分方法】

1>时间维度：适合数据访问频率和时间有较强相关性；（如 1 年前的订单）

2>访问频率：需要记录数据的访问频率，成本较高；（如文章浏览量）

41、数据冷热分离的优缺点

优点：热数据的查询性能得到优化，节约成本；

缺点：系统复杂性和风险增加，统计效率低（统计的时候会用到冷数据）

42、冷数据如何迁移？

1）任务调度：利用 xxl-job 等分布式任务调度平台，*定时扫描数据库*，将满足条件的冷数据，批量复制到冷库中，（适合按照时间区分的）

2）监听数据库变更日志 binlog：将满足冷数据条件的数据从 binlog 中提取出来，然后复制到冷库中，并从热库中删除。

43、索引的结构

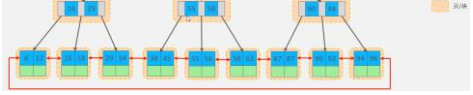
1）二叉树：顺序插入时，会形成一个链表，查询性能降低，大数据量时，层级较深；

2）红黑树：大数据量时，层级较深，检索慢；

3）B-树：每个节点都会存储数据；页大小为 16k，存储数据会占空间，导致键少，层数深；

4）B+树：所有数据都在叶子节点，叶子节点形成单向链表；（层数少）

MySQL 优化 B+树，增加相邻叶子节点的链表指针，提高区间访问性能。



5）哈希索引：将键值通过 hash 算法计算 hash 值，映射到 hash 表上，链表解决 hash 碰撞。

特点：只能对等比较；不能范围查询和排序；

44、索引分类

1）主键索引：针对主键创建的，只有一个；

2）唯一索引：某个字段的值不能重复；

3）常规索引：快速定位特定的数据；

4）全文索引：全文查找文本中关键词；

45、根据索引的存储形式（数据和索引分开否）

1）聚集索引：索引结构叶子节点保存*行数据*；

2）二级索引：索引结构的叶子节点*关联主键*；

【聚集索引选取规则】

1）存在主键→主键索引；

2）不存在主键→唯一索引；

3）都没有→自动生成一个 rowid 的聚集索引；

【注意】二级索引会回表查询→聚集索引

46、索引失效情况分析

1）索引的最左前缀法则：查询从索引的最左列开始，且不跳过索引列；

如果跳跃某一列，后面字段的索引会失效；（name,age,status：用 name/status, status 失效）

2）范围查询（包含>或<）右侧的列索引失效；（可改成>=或 <=）

3）索引列进行函数运算，索引会失效；

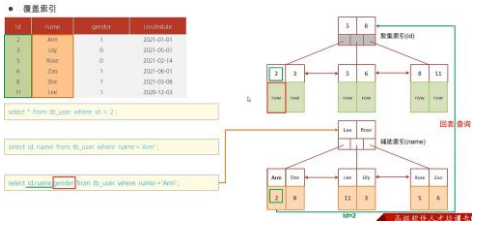
4）字符串类型字段，不加引号，索引会失效；

5）头部模糊查询(%天)，索引失效，尾部不会；

6）or 前索引列 or 后不是，则都失效；

47、覆盖索引

查询使用索引，并且需要返回的列，在该索引中已经全部能找到，避免了回表查询；



48、前缀索引

对于 varchar/text 长字符串类型字段建立索引时，查询时浪费大量磁盘 IO，查询慢；

所以，取字符串的一部分前缀，建立索引；

【前缀长度：根据索引的选择性，1 最好】

选择性=不重复索引值的数量/数据表总数量

【注意】遇到重复时，需要继续拿到全部信息比对，验证是否满足条件才行；

49、联合索引（一个索引包含多个列）

索引键值是多个列组合，遵循最左前缀法则；

50、索引设计原则

1）数据量大且查询频繁的表建立索引；

2）查询条件/排序/分组操作的字段建立索引；

3）尽量建立唯一索引，区分度高；

4）尽量联合索引，大概率可以覆盖索引；

5）长字段，前缀索引；

6）索引数量适当，维护索引结构代价大；

7）索引列不存 NULL；

51、SQL 优化

1）insert 优化：

1>批量插入；2>手动提交事务（包含多个批量插入）；3>主键顺序插入；4>超大批量用 load；

2）主键优化：

1>降低主键的长度；2>主键顺序插入；

【补充】乱序插入时，会有页分裂

页分裂：第 1 页分割一半到第 3 页，插入 50；

页合并，一页 50%被删除，会合并下一页；

3）order by 优化：（后面字段建立索引）

Using index，通过有序索引顺序扫描直接返回数据，不用额外排序。

默认全部升序排序，全部降序排序，走索引；

一个升序一个降序，需要重新创建一个索引；

4）group by 优化：（后面字段建立索引）

【注意】和 order by 一样，都要遵循最左前缀法则，可以最左索引列放 where 里再 order by；

5）count(*)：

MyISAM 引擎：磁盘记录表的总行数；（返回）

InnoDB 引擎：读取所有数据，累积计数；（烦）

count(主键)：读取主键值，按行累加（无 NULL）

count(字段)：读取字段值，有 NULL 加 0；

count(1)：遍历整张表不取值，返回每行放 1；

count(*)：MySQL 优化不取值，直接按行累加

6）update 优化：

InnoDB 的行锁是针对索引加锁，所以查询条件得加索引，不然降级为表锁。