MySQL面试

事务

1、快速/照读跟当前读的区别

- **快照读**：获取的是事务开始时的数据版本，不会读取到其他事务未提交的更改。它提供一致性的非锁定读取，不会对其他事务造成阻塞或锁竞争，通常用于实现多版本并发控制（MVCC）。
- **当前读**：获取的是数据的最新版本，可能会读取到其他事务已经提交的更改。它会使用排他锁（写锁）来保证数据的一致性，可能会对其他事务的并发操作造成影响。当前读通常用于需要最新数据并确保数据一致性的场景。

2、详细说一下Mysql的事务

MySQL事务是一组数据库操作的集合，要么全部成功，要么全部失败，确保数据一致性。

1. **ACID特性**：
   - **原子性 (Atomicity)**：事务中的操作要么全部完成，要么全部不执行。
   - **一致性 (Consistency)**：事务执行前后，数据库状态必须保持一致。
   - **隔离性 (Isolation)**：事务之间的操作相互独立，一个事务的操作不会被其他事务干扰。
   - **持久性 (Durability)**：一旦事务提交，其结果是永久性的，即使系统崩溃也不会丢失。
2. **隔离级别**：
   - **读未提交 (READ UNCOMMITTED)**：允许读取未提交的事务数据。
   - **读已提交 (READ COMMITTED)**：只允许读取已提交的数据。
   - **可重复读 (REPEATABLE READ)**：确保在一个事务中多次读取相同的数据结果一致。
   - **序列化 (SERIALIZABLE)**：强隔离级别，确保事务完全串行执行，避免幻读。
3. **锁机制**：
   - **行级锁**：锁定行数据，减少并发冲突，提高性能。
   - **表级锁**：锁定整个表，适用于低并发场景。
4. **错误处理**：如果事务中的操作出现错误，系统会自动回滚，确保数据库不会处于不一致状态。
5. **事务的开始和结束**：

- 事务可以通过`START TRANSACTION`或`BEGIN`语句开始。
- 可以使用`COMMIT`来提交事务，保存所有更改；或使用`ROLLBACK`来回滚事务，撤销所有更改。

3、事务是如何实现的？（MySQL是如何保证事务的？）

事务的实现涉及多个层面，包括存储引擎、日志管理和锁机制

- **原子性**：通过undo log（回滚日志）实现。事务开始时，所有操作都会记录到undo log中，以便在事务失败时进行回滚。
- **持久性**：通过redo log（重做日志）实现。事务提交时，所有修改都会写入redo log，并确保即使系统故障也能恢复这些修改。
- **隔离性**：通过锁机制和事务隔离级别实现。隔离级别决定了事务之间的可见性和干扰程度。
- **一致性**：事务的执行结果必须使数据库从一个一致性状态转换到另一个一致性状态，这通过数据库的约束和规则来保证

4、MySQL隔离级别，默认级别是什么？导致什么问题

默认级别：Repeatable Read（可重复读取）
ACID中的I，隔离性衍发出来

- **读未提交**（Read Uncommitted）：导致脏读。
- **读已提交**（Read Committed）：避免脏读，但可能导致不可重复读。
- **可重复读**（Repeatable Read）：避免脏读和不可重复读，但可能导致幻读。
- **序列化**（Serializable）：避免所有这些问题，但可能影响性能和并发性。
- **脏读**：读取未提交的数据。
- **不可重复读**：在同一事务内多次读取的数据不同。
- **幻读**：在同一事务内读取的数据集因其他事务的插入或删除而变化。

5、详细讲一下可重复读，两个事务读冲突了怎么办

确保在一个事务内多次读取同一数据时，每次读取的结果都是一致的，避免脏读和不可重复读，但可能导致幻读
读读冲突：当两个事务同时读取同一数据项时，这通常不会导致冲突，因为读取操作不会改变数据的状态。

- **处理**：读取操作本身不造成数据冲突，因此系统通常允许多个事务同时读取同一数据。
  写冲突：当一个事务正在读取数据，而另一个事务试图修改同一数据项时，就会出现读写冲突。

- 处理：
  锁机制：数据库通常使用行级锁或表级锁来解决读写冲突。
  锁定策略：数据库系统会锁定正在读取的数据项，防止其他事务对其进行修改。
  写写冲突：当两个事务同时对同一数据项进行修改时，就会出现写写冲突。
- 处理：
  事务序列化：数据库系统通常使用锁机制来保证事务的顺序性。
  提交或回滚：如果事务A和事务B同时对数据项X进行修改，系统会确保这两个事务按某个顺序完成，后提交的事务可能会导致前提交的事务回滚，以维持数据的一致性。

6、可重复读是怎么解决幻读的?

通过快照隔离、行级锁或更高级的序列化读等机制来避免幻读。
快照隔离确保事务读取到的是开始时的数据视图，而行级锁通过防止其他事务对数据的修改来实现数据的一致性。

引擎、索引

1、MYSQL的存储引擎包括哪些？InnoDB和MYISAM有什么区别?

存储引擎包括**InnoDB、MyISAM、Memory、Archive、CSV等**。
InnoDB和MyISAM的主要区别如下：

- **事务支持**：InnoDB支持事务处理，具有ACID特性，保证数据的一致性和可靠性；而MyISAM不支持事务处理。
- **锁定机制**：InnoDB支持行级锁定，适合高并发的写操作；MyISAM使用表级锁，并发写性能较差。
- **外键约束**：InnoDB支持外键约束，保证数据的完整性；MyISAM不支持外键约束。
- **崩溃恢复**：InnoDB具有自动崩溃恢复功能；MyISAM在崩溃后可能无法安全恢复。
- **适用场景**：InnoDB适用于需要事务支持和高并发读写的场景；MyISAM适用于读操作较多、写操作较少的场景。

2、InnoDB的索引类型有哪些？innodb的实现？innodb是表锁还是行锁?

- **B+树索引**：这是InnoDB默认的索引类型，包括聚集索引（主键索引）和非聚集索引（普通索引、唯一索引、联合索引等）。聚集索引的叶子节点直接对应数据页，存储整行记录数据；非聚集索引的叶子节点存储索引列值和主键值，通过主键值回表查询完整数据。
- **全文索引**：用于对文本内容进行搜索，通过倒排索引实现，可以快速查找文本中的关键词。
- **哈希索引**：InnoDB会根据表的使用情况自动为表生成哈希索引，但用户不能直接创建或干预。这种索引对于等值查询非常快速。

实现：

- **B+树索引**：
  - **聚簇索引**：表的数据按照主键顺序存储在一个 B+ 树中。叶子节点存储完整的行数据。
  - **非聚簇索引**：存储索引列值和主键值，非叶子节点中存储索引列的值，叶子节点中存储索引列的值和对应的主键值。
- **全文索引**：
  - 使用倒排索引来处理文本数据的搜索，支持快速的全文检索。
- **空间索引**：
  - 用于存储和检索地理空间数据。

行锁

3、mysql为什么使用b+树/查询快的原因，有别的代替吗？b+树的搜索时间复杂度

MySQL选择B+树作为其索引结构是因为：
数据可以存储在单个节点中，提高了数据密度和访问效率、
B+树的非叶子节点不存储数据，减少磁盘I/O次数、
B+树的所有数据都存储在叶子节点，范围查询时只需遍历两个节点，范围查询效率更高

根据场景，B树、哈希表、红黑树等
O(logn)

4、MySQL查看索引使用情况的命令是什么?

```
SHOW INDEX FROM table_name;
```

`EXPLAIN`

5、mysql怎么查看是否走了索引

在`SELECT`语句前加上`EXPLAIN`关键字

6、MySQL的索引优化方式有哪些?

选择合适的索引列、组合多列创建索引、使用前缀索引、删除重复和不必要的索引、使用覆盖索引、优化查询语句、定期分析表和优化索引

7、mysql索引失效有哪些场景/联合索引失效

索引失效的情况包括：

- **数据类型不匹配。**
- **对索引列使用函数或表达式。**

- **索引列的顺序不正确**（联合索引不满足最左前缀原则）。
- **like查询以%开头。**
- **or连接条件中存在非索引列。**
- **查询条件未使用索引列。**

8、MySQL是索引建立越多越好吗?

索引可以加速数据库的查询操作，但过多的索引也会带来性能上的开销和空间占用。

9、使用索引查询快的原因（MySQL底层数据结构是什么，有什么优势？）

MySQL选择B+树作为其索引结构是因为：
数据可以存储在单个节点中，提高了数据密度和访问效率
B+树的非叶子节点不存储数据，减少磁盘I/O次数
B+树的所有数据都存储在叶子节点，范围查询时只需遍历两个节点，范围查询效率更高

10、什么是联合索引

联合索引是指使用两个或更多个列作为索引键的一种索引方法，提高对多列查询的性能。

11、介绍一下MySQL索引匹配的最左原则

最左原则指的是在使用联合索引时，查询条件需要从索引的最左列开始匹配，才能有效利用该索引

12、如果给一个联合索引a和b，用where条件查询，什么情况下，索引会失效，什么情况不失效?

如果查询条件使用了联合索引中的最左侧部分，比如`a = x`或者`a = x AND b = y`，那么索引是有效的
如果查询条件跳过了联合索引的最左侧部分，比如`b = x`或者`b = x AND a = y`，那么索引可能会失效

13、mysql建表如果没有主键会怎么样

无主键表：表没法唯一标识每一行的列，可能导致数据的唯一性无法保证。
性能影响：查询操作影响性能，因为没有主键时，MySQL无法优化某些查询操作，例如索引的建立和使用。
数据完整性：没有主键可能导致重复数据的插入，因为没有约束来保证行的唯一性。
外键约束：如果表没有主键，无法创建外键约束，因为外键需要参考主键以确保数据一致性。
隐式主键：如果表中没有明确的主键，MySQL会创建一个隐式的唯一索引（仅在

InnoDB存储引擎中），这个隐式索引基于表的行ID。但这可能不是一个明确的用户定义的主键。

14、mysql主键自增和非自增哪个好，除了自增主键还有什么？

自增主键的优点包括：**自动编号，速度快，占用空间小，易排序，插入时无需指定主键值，减少了出错的可能性。**此外，自增主键还能提高插入和查询性能，特别是在高并发场景下。
雪花ID的设计目标是在分布式环境下，生成全局唯一的ID，并具有一定的有序性，适用于分布式数据库、分布式缓存等场景

15、主键和联合索引的区别

主键是一种特殊的索引，用于保证数据的唯一性，而联合索引则用于提高多列查询的性能

- 定义与功能：主键是一种特殊的唯一索引，用于唯一标识表数据。而联合索引是对多个字段同时建立的索引，用于提高基于这些字段的查询效率。
- 唯一性与空值：主键的值必须是唯一的，且不允许为空。联合索引则没有这些限制，其索引的列值组合需要唯一（如果是作为主键的联合索引），但单个列的值可以重复，且某些列的值可以为空（取决于索引创建时的设置）。
- 数量限制：一张表只能有一个主键，但可以有多个联合索引。这允许根据不同的查询需求优化索引策略。
- 使用场景：主键主要用于唯一标识记录，而联合索引则更多地用于优化涉及多个字段的查询性能。

16、说一下主键索引和唯一索引有什么区别

唯一性约束：主键索引要求列的值必须唯一且不为NULL；而唯一索引也要求列的值唯一，但可以包含NULL值。
数量限制：一个表只能有一个主键索引，但可以有多个唯一索引。
物理存储：在InnoDB存储引擎中，主键索引是聚集索引；而唯一索引是非聚集索引
外键引用：主键索引可以被其他表作为外键引用，以建立表之间的关系；而唯一索引则不能。

17、问聚簇索引跟非聚簇索引的区别？回表是什么情况？什么时候会回表？非聚簇索引一定会回表吗？你知道自增id有时候有影响吗？

聚簇索引与非聚簇索引的区别：
数据存储方式：聚簇索引将索引和数据行物理上存储在一起，而非聚簇索引则将索引和数据行分开存储
唯一性：聚簇索引在InnoDB中默认是主键，因此必须是唯一的；非聚簇索引则可以是唯一的，也可以不是唯一的。

数量限制：一个表只能有一个聚簇索引，因为数据只能按照一种顺序存储；而可以有多个非聚簇索引，以满足不同的查询需求。
查询效率：聚簇索引由于数据与索引物理顺序相同，查询效率通常较高；非聚簇索引可能需要额外的回表操作，查询效率相对较低。
插入效率：聚簇索引插入新数据时可能需要移动已有数据，效率较低；非聚簇索引插入数据时只需更新索引，效率较高。

回表：

- 回表是指在使用非聚簇索引查询时，为获取其他列数据需再次访问原始数据的操作。
- 当查询的列不在非聚簇索引中时，或者查询需要返回非索引列的数据时，会发生回表。
- 非聚簇索引不一定会回表，如果查询能够使用覆盖索引（即索引包含了查询所需的所有列），则无需回表。
  自增ID在某些情况下会有影响，如**高并发插入时可能导致锁争用**，且自增ID是局部唯一的，不适用于分布式系统。

18、说一下Mysql的回表是什么，以及说一下如何优化Mysql的回表

- 当执行查询时，如果非聚簇索引中包含的字段不足以满足查询需求，数据库引擎需要访问原始数据表以获取完整的数据行，这个过程被称为“回表”。
- 回表发生的情况包括：
  - 查询的字段不全在索引中时。
  - 索引只包含了部分查询条件的列，其他列的值需要从表中检索。
  - 例如，假设有一个非聚簇索引 `(A, B)`，如果查询需要列 `C`，且 `C` 不在索引中，数据库需要回表来获取 `C` 的值。
  优化：
  使用覆盖所有、优化查询字段、优化表结构、调整查询条件等

19、引擎和索引

MySQL引擎（或称存储引擎）是MySQL数据库管理系统中的一个**组件**，负责实际的数据存储、检索和管理。不同的存储引擎提供了不同的功能和特性。MySQL支持多个存储引擎，每种引擎都有其特定的用途和优缺点。主要的存储引擎包括：

- **InnoDB**：支持事务、外键约束和行级锁，是MySQL默认的存储引擎，适合需要高事务处理能力的应用。
- **MyISAM**：不支持事务和外键约束，但提供较高的查询性能和压缩功能，适合以读为主的应用。
- **Memory**：将数据存储在内存中，适用于高速访问的临时数据。

- **CSV**：将数据存储为CSV文件，适合简单的数据导入导出操作。
- **ARCHIVE**：用于存储大量历史数据，适合归档用途。
  索引是数据库表中的一个**数据结构**，用于加速数据的检索。索引可以显著提高查询速度，但会占用额外的存储空间，并可能影响数据的写入性能。索引可以是多种类型：
- **B树索引**（如MySQL中的默认索引类型）：用于加速基于范围的查询和等值查询。
- **哈希索引**：用于等值查询，快速查找特定的值。
- **全文索引**：用于支持全文搜索，适合对文本字段的复杂查询。
- **空间索引**：用于地理数据类型，支持地理空间的查询和计算。
  总结
- **MySQL引擎**：管理数据的存储和处理，提供不同的数据处理功能和特性。
- **索引**：提高数据检索效率，通过特定的数据结构加速查询操作。

锁

1、讲一下MVCC的作用，以及它是如何实现的？

多版本并发控制是一种并发控制的方法，用于实现对数据库的并发访问
MVCC（多版本并发控制）是一种并发控制的方法，用于处理多个事务同时操作数据的情况，以确保数据的一致性和隔离性，利用MVCC来判断在一个事务中，哪个数据可以被读出来，哪个数据不能被读出来。
实现原理主要是依赖隐式字段，undo log日志，Read View
1.隐藏字段：InnoDB在聚簇索引记录中包含两个隐藏列：trx_id和roll_pointer。trx_id记录了对记录进行改动的事务ID，roll_pointer指向修改之前的版本，形成一个版本链。
2.Undo日志：在对记录进行修改时，旧版本的数据会被写入到Undo日志中，roll_pointer指向这个旧版本的数据，从而形成一个版本链3。
3.ReadView（快照）：在读提交（Read Committed）和可重复读（Repeatable Read）的隔离级别下，InnoDB使用ReadView来判断读取版本链中的哪个版本。
ReadView包含了活跃事务ID列表、最小活跃事务ID和最大活跃事务ID等信息，用于版本比较。

2、update在MVCC中是如何实现的？

执行流程：
当执行 `UPDATE` 操作时，大致流程如下：

- 对目标行记录加行锁（如果是当前读，如 `SELECT ... FOR UPDATE`）。
- 将该行记录的旧版本数据复制到 undo log 中。
- 修改行记录的数据，并更新 `DB_TRX_ID` 为当前事务ID。
- 更新 `DB_ROLL_PTR` 指向新的 undo log 记录。

- 提交事务时，释放锁，并可能清理不再需要的 undo log 记录（由 purge 线程处理）

3、mysql锁，有哪些锁

- **全局锁**：锁定数据库中的所有表，使整个实例处于只读状态。主要用于全库备份、全库导出等操作，以确保数据的一致性。
- **表级锁**：每次操作锁住整张表，锁的力度大，发生锁冲突的概率高，并发度低。适用于对整个表进行操作的场景。
- **行级锁**：每次操作锁住对应的行数据，并发性高，但锁管理较复杂。主要包括Next-Key Lock、Gap Lock和Record Lock等。
- **乐观锁和悲观锁**：乐观锁假设并发操作时不会发生冲突，只在提交事务时检查数据是否被其他事务修改过；悲观锁则假设并发操作时会发生冲突，因此在操作期间持有锁来避免冲突。

4、悲观锁和乐观锁、分布式锁

- **悲观锁**：假设并发操作时会发生冲突，因此在操作期间持有锁来避免冲突。在整个数据处理过程中，将数据处于锁定状态，任何事务都不能对该数据进行修改，只能等待锁被释放才可以执行。数据库中的行锁、表锁、读锁、写锁等均为悲观锁的一种实现。
- **乐观锁**：假设并发操作时不会发生冲突，只在提交事务时检查数据是否被其他事务修改过。并发的事务在处理时不会影响数据。乐观锁在数据库上的实现，一般通过在需要锁的数据上增加一个版本号或时间戳来实现。
- **分布式锁**：在分布式系统里互斥访问资源的解决方案，用于执行定时任务或处理并发请求时，确保多点系统里同时只有一个执行线程进行处理。

语法

1、mysql的数据类型

- **数值类型**：包括整数类型（如TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT）和浮点数类型（如FLOAT、DOUBLE），以及定点数类型（如DECIMAL）。
- **日期和时间类型**：包括YEAR、TIME、DATE、DATETIME和TIMESTAMP，用于表示日期和时间值。
- **字符串类型**：包括CHAR、VARCHAR、TEXT等，用于存储字符串数据。其中，CHAR是定长字符串，VARCHAR是可变长度字符串。
- **二进制类型**：包括BINARY、VARBINARY、BLOB等，用于存储二进制数据。

2、什么是视图

视图（View）是一个虚拟表，它是由一个或多个表的数据组成的查询结果集。视图本身不存储数据，而是动态地生成数据集，以便用户可以像使用普通表一样进行查询和操作。

3、mysql 里面时间是怎么存储的

1. `DATETIME`: 存储日期和时间，格式为 `YYYY-MM-DD HH:MM:SS`。
2. `DATE`: 仅存储日期，格式为 `YYYY-MM-DD`。
3. `TIME`: 仅存储时间，格式为 `HH:MM:SS`。
4. `TIMESTAMP`: 存储日期和时间，格式为 `YYYY-MM-DD HH:MM:SS`，并且会根据时区自动转换。
5. `YEAR`: 存储年份，格式为 `YYYY`。

4、explain字段

用于分析和优化 SQL 查询

5、where和 having有什么区别?

`WHERE` 和 `HAVING` 都用于筛选数据

- `WHERE` 用于在数据检索阶段过滤行数据。
- `HAVING` 用于在数据分组和聚合后过滤分组数据。

6、TRUNCATE和delete区别

`DELETE` 是一种逐行删除数据的方式，适用于需要有选择性地删除记录的情况；
而 `TRUNCATE` 是一种快速的全表删除操作，适用于需要清空整个表并重置计数器的情况

7、如果直接delete一半记录会直接空出一半空间吗?

不会
在InnoDB存储引擎中，delete操作实际上是将数据标记为已删除状态，而不是从磁盘上物理删除。

8、mysql中的聚合函数

count、avg、max、min、sum、

9、数据库如何进行去重

distinct、group by、窗口函数

10、mysql如何实现分页

limit、offset

11、ddl，dml，dcl定义

DDL (Data Definition Language)：数据定义语言，用于定义和管理数据库结构。

- **主要命令**:create、alter、drop、truncate
  DML (Data Manipulation Language): 数据操作语言，用于对数据库中的数据进行操作。
- **主要命令**:select、insert、update、delete
  DCL (Data Control Language): 数据控制语言，用于管理数据库的访问权限。
- **主要命令**:grant、revoke

12、表关联

内连接（Inner Join）、外连接（Outer Join）、自然连接（Natural Join）和交叉连接（Cross Join）
内连接返回两个表中满足连接条件的交集部分
外连接则包括左外连接（Left Outer Join）、右外连接（Right Outer Join）和全外连接（Full Outer Join），它们返回连接条件满足的结果以及其中一个表中未匹配到的行
自然连接根据两个表中相同的列名自动进行连接
交叉连接返回两个表的笛卡尔积。

13、左连接和内连接的区别

左连接会包含左表中的所有行，而内连接只包含两个表中匹配的行

14、Select *和 select* for update 分别加了什么锁

`SELECT *`查询语句本身不会加任何锁，而`SELECT * FOR UPDATE`除了具有查询作用外，还会加上悲观锁。

- 在使用`SELECT * FOR UPDATE`时，如果查询条件使用了索引或主键，那么它会加上行锁，锁住符合条件的一行或多行数据。
- 如果查询条件没有使用索引或主键，那么它可能会加上表锁，锁住整张表。

15、binlog的三种格式

binlog是MySQL数据库中用于记录所有修改数据库表结构和表数据操作的二进制日志

- Statement（基于SQL语句的复制，SBR）：记录每一项更改数据的SQL语句本身。优点是记录内容简洁，减少了binlog日志的大小，提升了性能。缺点是可能无法确保复制的一致性，尤其是在涉及特定存储过程、函数、触发器调用的情况下。

- Row（基于行的复制，RBR）：直接记录数据行级别的更改详情。优点是能精确反映数据变化，避免了SBR中可能出现的复制一致性问题。缺点是可能导致binlog日志量增大，占用更多存储空间和网络传输负担。
- Mixed（混合模式复制，MBR）：是对前两种格式的综合运用，MySQL会根据执行的具体SQL语句选择合适的日志记录方式，以尽量减小binlog日志大小，同时最大程度地保障主从复制的一致性。

16、三个log binlog、undolog、redolog的区别

- binlog：是MySQL的逻辑日志，记录数据库执行的写入性操作（不包括查询），以二进制形式保存在磁盘中。主要用于数据恢复和主从复制。
- undo log：是InnoDB存储引擎的回滚日志，记录事务执行前的数据状态，用于事务回滚和多版本并发控制（MVCC）。
- redo log：同样是InnoDB存储引擎的日志，记录数据页的物理修改，用于保证事务的持久性。在数据库崩溃时，可通过redo log恢复已提交的事务，确保数据一致性

17、MySQL redolog+binlog干什么用的

redo log：主要用于确保事务的持久性。它记录了InnoDB存储引擎中数据页的物理修改操作。当数据库发生崩溃时，redo log可以用来恢复已提交的事务，确保数据的一致性和完整性。redo log采用循环写的方式，存储在磁盘上，是MySQL崩溃恢复的关键组件。
binlog：是MySQL Server层维护的二进制日志，记录了所有的DDL和DML语句（除了数据查询语句），以事务的形式保存在磁盘中。binlog主要用于复制和数据恢复。在主从复制中，主服务器将binlog发送给从服务器，从服务器根据binlog进行数据的复制和同步。同时，binlog也可以用于数据恢复，通过回放binlog中的操作，可以将数据库恢复到特定的时间点或状态。

18、mysql如果占用了80%的空间，如何迁移?

评估、mysqldump备份、迁移数据导入、验证数据并更新应用

19、MySQL主从复制原理

MySQL主从复制原理简述如下：

1. **Master记录操作**：Master服务器将数据的改变记录到二进制日志（binlog）中。
2. **Slave请求并接收binlog**：Slave服务器通过I/O线程连接到Master，请求并接收binlog，写入到本地的中继日志（relay log）中。
3. **Slave重放事件**：Slave的SQL线程读取中继日志中的事件，并重放这些事件，更新Slave的数据，使其与Master保持一致。

20、MySQL如何防止写操作导致资源为负数

MySQL防止写操作导致资源为负数，通常与数据类型选择和约束设置有关，如使用无符号数据类型（UNSIGNED）和CHECK约束来限制负数的插入

优化

1、mysql是如何保证每次查询耗时差不了太多的呢?

- **索引优化**：通过在表的列上创建索引，MySQL可以快速定位匹配的行，避免扫描整个表，从而提高查询速度。
- **查询分析**：利用EXPLAIN命令分析SQL语句的执行计划，识别潜在的性能瓶颈，如是否使用了索引、查询类型等。
- **慢查询日志**：设置慢查询日志，记录执行时间较长的查询语句，便于后续分析和优化。
- **分页查询优化**：对于深度分页场景，通过子查询、范围查询等方式减少扫描行数，提高查询效率

2、SQL的慢查询排查

1. **分析慢查询日志**：启用并分析慢查询日志，找出执行时间过长的查询。
2. **使用性能分析工具**：利用EXPLAIN命令或第三方工具分析查询的执行计划，识别性能瓶颈。
3. **检查查询结构**：优化查询语句，如避免SELECT * ，减少不必要的JOIN操作，确保使用索引。
4. **检查数据库设计**：确保数据库设计正规化，考虑对大表进行分区以提高查询性能。
5. **优化数据库配置**：调整缓存大小，优化连接池配置，确保系统资源不是性能瓶颈

3、mysql的慢查询问题，如何解决和优化？覆盖索引

1. **定位慢查询**：
   - 确认是否开启了慢查询日志，通过`show variables like "%slow%";`查看，并设置慢查询的时间限制。
   - 查询慢查询日志，定位具体的慢SQL语句。
2. **分析慢查询**：
   - 使用`EXPLAIN`分析具体的SQL语句，查看执行计划，识别性能瓶颈。
3. **慢查询优化**：
   - 优化索引：确保在WHERE条件列建立索引，避免全表扫描；注意索引不生效的场景，如隐式类型转换、LIKE通配符以%开头等。
   - 优化查询语句：避免使用子查询，尽量使用JOIN；读取适当的记录，使用LIMIT M,N；禁止不必要的ORDER BY排序等。
   - 优化数据库结构：合理设计数据库结构，考虑数据冗余、查询和更新的速度等因素。

4、假设查询场景，有20多个字段，他们可能部分或全部作为查询条件，这种情况下用MySQL有什么缺点？应该用什么技术来处理这种场景？

1. **性能下降**：查询多个字段作为条件会增加查询的复杂度，可能导致查询速度变慢，尤其是当这些字段没有适当索引时。
2. **资源消耗**：处理大量字段的查询会消耗更多的CPU和内存资源，影响数据库的整体性能。
   索引优化、查询优化、分区表、缓存策略

5、mysql数据量大、慢怎么解决（MySQL优化方法有哪些）

1. **优化查询和索引**：使用EXPLAIN分析查询，避免SELECT *，只选择需要的列；为查询中频繁使用的列创建索引，避免过多索引；考虑使用复合索引优化多列查询。
2. **分表分库**：将数据分割到多个表或数据库中，减小单表数据量，将数据分散存储，提高查询效率。
3. **优化数据库结构**：合理设计数据库表结构，避免数据冗余，选择适当的数据类型，减少存储空间占用。
4. **查询优化**：使用`EXPLAIN`分析查询执行计划，优化SQL语句，避免全表扫描。
5. **调整数据库配置**：根据服务器硬件和查询负载，调整MySQL的配置参数，如增加innodb_buffer_pool_size以提高缓存效率。
6. **使用缓存技术**：利用缓存技术（如Redis或Memcached）存储经常查询的数据，减少查询数据库次数。

6、百亿级数据存储情况下，为啥选Clickhouse而不是关系型数据库？例如事务型关系型数据库MYSQL和PGSQL。

- **高性能查询**：ClickHouse采用列式存储，适合进行大量数据的分析和查询，查询性能远高于传统行式存储的关系型数据库。
- **资源消耗**：在处理百亿级数据时，关系型数据库如MySQL需要消耗大量资源，且索引可能占用大量空间，影响查询性能。
- **扩展性**：ClickHouse支持分布式处理，处理速度几乎可以线性扩展，更适合处理大规模数据。

7、分库分表的分片方法有哪些？如何保证唯一主键？

分片方法主要包括水平拆分和垂直拆分，其中水平拆分又可以根据数值范围、数值取模、地理位置等策略进行。垂直拆分则包括垂直分库和垂直分表，主要根据业务模块或表结构进行拆分。
保证唯一主键几种方法：

- **使用UUID**：UUID可以生成全局唯一的ID，但缺点是ID较长且不是递增的，可能影响数据库性能。
- **自增ID**：通过单库生成唯一ID，但不适用于高并发场景。可以改进为批量生成ID的方式，以提高效率。
- **Snowflake算法**：Twitter开源的分布式ID生成算法，可以生成64位的唯一ID，适用于分布式系统。

8、一个sql语句所有部分的执行顺序是什么？

FROM
ON (用于连接条件)
JOIN
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
LIMIT / OFFSET

9、es倒排索引

倒排索引主要包含两部分：单词词典（Term Dictionary）和倒排列表（Posting List）。单词词典存储所有唯一单词及其对应的倒排列表位置，而倒排列表则记录了每个单词在哪些文档中出现过，以及出现的位置、频率等信息。