MySQL 高频面试题（精简版）

一、基础与存储引擎（3题）

1. InnoDB 和 MyISAM 核心区别（高频）

维度	InnoDB	MyISAM
事务支持	支持 ACID	不支持
锁机制	行锁+表锁	仅表锁
外键	支持	不支持
索引类型	聚簇索引（主键即数据）	非聚簇索引（索引+数据分离）
崩溃恢复	支持（redo/undo log）	不支持
适用场景	读写频繁（订单系统）	只读查询（日志/报表）

2. MySQL 存储引擎及适用场景

- **InnoDB**：默认，支持事务/行锁/外键，适用于需数据一致性的场景（电商、金融）。
- **MyISAM**：查询快，不支持事务，适用于只读报表、日志存储。
- **Memory**：数据存内存，断电丢失，适用于临时缓存（会话表）。
- **Archive**：高压缩，仅支持插入/查询，适用于归档历史数据（操作日志）。

3. InnoDB 为何是默认存储引擎

1. 支持事务，满足多数业务数据一致性需求；
2. 行锁粒度小，高并发下锁冲突少；
3. 支持外键，维护表间数据完整性；
4. 崩溃可恢复，稳定性强。

二、索引核心（7题）

1. MySQL 索引类型

- 按结构：B+树（默认，支持范围/排序）、哈希（等值快，不支持范围）、全文（文本检索）；
- 按存储：聚簇索引（InnoDB 主键，索引=数据）、非聚簇索引（索引+数据地址）；
- 按功能：主键索引（唯一非空）、辅助索引（普通索引）、联合索引（多列组合）。

2. 聚簇 vs 非聚簇索引（高频）

维度	聚簇索引	非聚簇索引
存储结构	索引+数据在一起	索引、数据分离
查询效率	主键查询直接拿数据	需回表（除非覆盖索引）
InnoDB 体现	主键索引是聚簇索引	辅助索引是聚簇索引

3. InnoDB 用 B+树的原因（高频）

- 对比 B 树：B+树叶子页链表连接，支持范围查询；非叶子页仅存索引，单次加载更多索引，减少 IO；
- 对比哈希：不支持范围查询和排序；
- 对比红黑树：多叉树（扇出约1000），百万数据深度仅3，减少 IO（红黑树深度约20）。

4. 回表查询及避免（高频）

- 定义：InnoDB 用辅助索引查询时，需通过主键值查聚簇索引拿数据（二次查询）；
- 避免：用**覆盖索引**（查询字段全在索引中，如联合索引包含需查字段）。

5. 联合索引与最左前缀（高频）

- 联合索引：多列组合索引（如 `(a,b)`）；
- 最左前缀：查询需从左到右匹配，不跳过中间列（支持 `(a)`、`(a,b)`，不支持 `(b)`）。

6. 索引失效场景（高频，记6点）

1. 索引列用函数/运算（如 `SUBSTR(name,1,3)`、`age+1=20`）；
2. 类型不匹配（int 字段用字符串查询，如 `id='123'`）；
3. 违反最左前缀（联合索引 `(a,b)`，查 `where b=2`）；
4. or 连接非索引列（`where a=1 or b=2`，b 无索引）；
5. 左模糊（`like '%xxx'`，右模糊 `like 'xxx%'` 不失效）；
6. `is not null`（`is null` 不失效）。

7. 索引设计原则

1. 为 `where`/`join`/`order by` 列建索引；
2. 控制数量（单表≤5个，避免拖慢写操作）；
3. 用小字段（如 int 主键，减少索引页大小）；
4. 优先联合索引（替代多单列索引）；
5. 避免重复索引（如 `(a,b)` 覆盖 `(a)`）。

三、事务与隔离级别（5题）

1. 事务 ACID 及实现（高频）

- **原子性**：undo log（记录反向操作，失败回滚）；
- **一致性**：原子性+隔离性+持久性+业务逻辑；
- **隔离性**：锁机制（控制修改）+ MVCC（控制读取）；
- **持久性**：redo log（写操作先落盘，崩溃恢复）。

2. MySQL 事务隔离级别（高频）

- 读未提交（Read Uncommitted）：脏读；
- 读已提交（Read Committed）：解决脏读，不可重复读；
- **可重复读（Repeatable Read）**：默认，解决不可重复读，InnoDB 解决幻读；
- 串行化（Serializable）：完全隔离，性能低。

3. 隔离级别解决的问题（高频）

隔离级别	脏读	不可重复读	幻读
读未提交	✅	✅	✅
读已提交	❌	✅	✅
可重复读（默认）	❌	❌	❌
串行化	❌	❌	❌

4. InnoDB 解决幻读（高频）

- 快照读（普通 `select`）：MVCC 读快照数据，看不到新增数据；
- 当前读（`update`/`delete`/`select for update`）：间隙锁锁定数据间隙，阻止插入。

5. redo/undo/binlog 区别（高频）

日志类型	作用	存储内容	引擎支持
redo log	崩溃恢复，保障持久化	物理日志（改位置）	仅 InnoDB
undo log	回滚，支持 MVCC	逻辑日志（反向操作）	仅 InnoDB
binlog	主从复制，备份恢复	逻辑日志（SQL语句）	所有引擎

四、锁机制（4题）

1. InnoDB 锁类型

- 行锁：S 锁（读锁，多事务可加）、X 锁（写锁，排他）；
- 表锁：无索引时触发（如 `update` 无索引字段）；
- 意向锁：IS（计划加 S 锁）、IX（计划加 X 锁），避免表锁冲突；
- 间隙锁：锁定数据间隙，防幻读；
- 临键锁：行锁+间隙锁，默认锁算法。

2. 行锁 vs 表锁适用场景

- 行锁：高并发读写（订单修改）；
- 表锁：全表操作（全表删除）、无索引操作；
- 触发表锁：无索引字段操作、手动 `lock tables`。

3. 间隙锁及避免

- 定义：可重复读下，当前读锁定数据间隙（如 ID 1-3 间），防插入；
- 避免：隔离级别降为读已提交、用等值查询/主键查询。

4. 死锁及解决（高频）

- 产生条件：互斥、持有并等待、不可剥夺、循环等待；
- 排查：`show engine innodb status;` 看死锁日志；
- 避免：统一加锁顺序、减少锁持有时间、拆分长事务。

五、SQL 优化（5题）

1. 定位慢查询

- 开启慢查询日志：`set global slow_query_log=1;` 定义 `long_query_time=2`（2秒为慢查询）；
- 分析工具：`mysqldumpslow`（自带）、`pt-query-digest`（第三方）。

2. EXPLAIN 核心字段（高频）

- **id**：执行顺序（大的先执行）；
- **type**：索引类型（从好到差：`const` > `eq_ref` > `ref` > `range` > `index` > `ALL`，避免 `ALL`/`index`）；
- **key**：实际用的索引（NULL 表示未用）；
- **rows**：预估扫描行数（越少越好）；
- **Extra**：`Using index`（覆盖索引，好）、`Using filesort`/`Using temporary`（差，需优化）。

3. 常见 SQL 优化手段

- join：小表驱动大表，关联列建索引；
- 子查询：用 join 替代（减少临时表）；
- limit 大分页：用主键定位（`where id>100000 limit 10`）；
- order by：用索引排序（避免 `filesort`）。

4. 不建议用 SELECT *

- 增加网络传输；
- 无法用覆盖索引；
- 字段变更风险（新增/删除字段导致程序报错）。

5. 大表分页优化（`limit 100000,10`）

- 主键定位：`where id>100000 limit 10`（需知道起始主键）；
- 索引覆盖：`select t.* from table t join (select id from table limit 100000,10) temp on t.id=temp.id`。

六、主从复制与高可用（4题）

1. 主从复制原理（高频）

3个线程：

1. 主库 binlog 线程：写操作记录到 binlog；
2. 从库 IO 线程：拉取主库 binlog，写入 relay log；
3. 从库 SQL 线程：读 relay log，执行 SQL 同步数据。

2. 主从延迟原因及解决

- 原因：主库写压力大、从库读压力大、大事务、网络延迟；
- 解决：从库分流查询（级联复制）、并行复制（`slave_parallel_workers=4`）、拆分大事务、优化网络。

3. 主从架构

- 一主一从：简单，适用于小规模；
- 一主多从：主写从读，适用于读多写少；
- 级联复制：主→中间从→从从库，减少主库压力。

4. 读写分离实现

- 应用层：代码判断 SQL 类型，动态切换数据源（如 Spring `AbstractRoutingDataSource`）；
- 中间件：MyCat/Sharding-JDBC/ProxySQL，自动路由（写→主，读→从）。

七、分库分表（4题）

1. 分库分表原因及区别

- 原因：单表超 1000 万行（数据量瓶颈）、并发超连接数（并发瓶颈）；
- 区别：分库（拆数据库，解并发）、分表（拆表，解数据量）。

2. 分库分表策略（高频）

- 垂直分片：按业务拆（如用户库/订单库；订单表拆基本表/详情表）；
- 水平分片：按数据拆（范围：按时间/ID；哈希：`user_id%4` 拆4表）。

3. 分布式事务解决方案

- 2PC：两阶段提交（强一致，同步阻塞）；
- TCC：Try-Confirm-Cancel（无锁，开发成本高）；
- SAGA：拆长事务+补偿（最终一致，简单）；
- 本地消息表：业务+消息同事务，定时发消息（最终一致，依赖MQ）。

4. 全局 ID 实现

- 数据库自增（号段模式，有序，需 ID 库）；
- UUID（简单，无序，长）；
- 雪花算法（64位，有序，性能高，依赖时间）。

八、其他高频（5题）

1. varchar vs char

维度	varchar	char
存储	变长（实际字符+长度标识）	定长（不足补空格）
长度限制	utf8mb4 下 65535 字节	255 字符
空间	省	费
性能	略低	略高
适用场景	长度不固定（用户名）	长度固定（手机号）

2. NULL vs '' 及不建议 NULL 默认

- 区别：NULL 无值，'' 空字符串；NULL 需用 `is null` 判断，'' 用 `=''`；
- 不建议 NULL 默认：可能索引失效、计算异常（`NULL+1=NULL`）、逻辑复杂。

3. 数据库连接池

- 作用：复用连接，提性能；控制并发；统一管理，防泄漏；
- 核心参数：`maxActive`（最大活跃连接）、`maxIdle`（最大空闲）、`minIdle`（最小空闲）、`waitTimeout`（等待超时）。

4. InnoDB 主键建议自增 ID

- 聚簇索引顺序插入，避免页分裂；
- 整型索引页存更多键，减少 IO；
- 便于排序和范围查询。

5. 数据备份与恢复

- 逻辑备份（`mysqldump`）：备份 SQL，跨平台，慢，适用于小数据；
- 物理备份（XtraBackup）：备份数据文件，快，支持增量，适用于大数据。