**1. 避免SELECT \***

别一上来就 SELECT \*，你觉得这样写省事，但数据库得把整张表的每一列都给你搬过来，代价大得吓人。需要啥列写清楚，别偷懒。

*-- 错误写法*  
SELECT \* FROM users;  
  
*-- 正确写法*  
SELECT id, name, age FROM users;

**优化原因**：减少网络传输数据量，避免无用字段占内存。

**2. WHERE字段要有索引**

你写 WHERE 条件时，涉及到的字段要有索引。没索引，MySQL 就得一行行地扫描表，全表扫描慢得你怀疑人生。

*-- 创建索引*  
CREATE INDEX idx\_users\_age ON users(age);  
  
*-- 使用索引查询*  
SELECT \* FROM users WHERE age > 25;

**优化原因**：索引就像目录，能直接跳到你要的那页，而不是一本书从头翻到尾。

**3. 用EXPLAIN分析SQL性能**

写SQL前，别直接跑，先加个 EXPLAIN 看看执行计划。看啥？type 是不是 ALL，是不是走了全表扫描。如果有，赶紧改。

EXPLAIN SELECT \* FROM users WHERE age > 25;

**优化原因**：清楚执行计划，知道SQL跑得是不是高效。

**4. 避免WHERE里的函数操作**

别在WHERE条件里对字段用函数，索引直接失效。

*-- 错误写法*  
SELECT \* FROM users WHERE YEAR(create\_time) = 2023;  
  
*-- 正确写法*  
SELECT \* FROM users WHERE create\_time >= '2023-01-01' AND create\_time < '2024-01-01';

**优化原因**：函数操作让索引失效，回表扫描浪费时间。

**5. 避免OR条件，改用UNION**

WHERE 里写了个 OR，可能导致两个字段的索引全失效。别用 OR，用 UNION 替代。

*-- 错误写法*  
SELECT \* FROM users WHERE age = 25 OR city = 'shenzhen';  
  
*-- 正确写法*  
SELECT \* FROM users WHERE age = 25  
UNION  
SELECT \* FROM users WHERE city = 'shenzhen';

**优化原因**：OR 会让索引失效，而 UNION 可以分别利用索引。

**6. 优化LIMIT分页**

LIMIT 用得好不好直接影响性能，特别是分页到 10000 页的时候，直接卡死你。改进方法是用 ID 或时间范围过滤。

*-- 错误写法*  
SELECT \* FROM users ORDER BY create\_time LIMIT 10000, 10;  
  
*-- 正确写法*  
SELECT \* FROM users WHERE id > 10000 ORDER BY id LIMIT 10;

**优化原因**：用范围过滤减少不必要的扫描。

**7. 用覆盖索引**

覆盖索引的意思是，查询的字段全在索引里，MySQL 不需要回表。

*-- 创建覆盖索引*  
CREATE INDEX idx\_users\_age\_name ON users(age, name);  
  
*-- 查询*  
SELECT name FROM users WHERE age > 25;

**优化原因**：只从索引获取数据，不用回表，提高查询速度。

**8. 减少JOIN表数量**

JOIN 太多表会让执行计划复杂化，数据量大的表用 JOIN，性能就拉了。

*-- 复杂的多表JOIN*  
SELECT \* FROM orders  
JOIN users ON orders.user\_id = users.id  
JOIN products ON orders.product\_id = products.id;  
  
*-- 优化：拆分查询*  
SELECT \* FROM orders WHERE user\_id IN (  
    SELECT id FROM users WHERE age > 25  
);

**优化原因**：减少中间表的数据处理量，降低JOIN复杂度。

**9. 使用批量插入**

单条插入写多了，性能会被写锁拖垮，换成批量插入。

*-- 错误写法*  
INSERT INTO users (id, name) VALUES (1, 'zhangsan');  
INSERT INTO users (id, name) VALUES (2, 'lisi');  
  
*-- 正确写法*  
INSERT INTO users (id, name) VALUES (1, 'zhangsan'), (2, 'lisi');

**优化原因**：减少数据库连接和提交的次数。

**10. GROUP BY前过滤数据**

GROUP BY 本质上是对结果排序分组，数据量大时效率感人。先用 WHERE 把数据量缩小。

*-- 错误写法*  
SELECT age, COUNT(\*) FROM users GROUP BY age;  
  
*-- 正确写法*  
SELECT age, COUNT(\*) FROM users WHERE age > 25 GROUP BY age;

**优化原因**：减少GROUP BY处理的数据量。

**11. LIKE查询优化**

模糊查询用 %xxx% 的时候，索引没了，直接全表扫描。改成前缀匹配，或者用全文索引。

*-- 前缀匹配*  
SELECT \* FROM users WHERE name LIKE 'zhang%';  
  
*-- 全文索引*  
CREATE FULLTEXT INDEX idx\_name ON users(name);  
SELECT \* FROM users WHERE MATCH(name) AGAINST('zhang');

**优化原因**：用索引提高查询效率。

**12. 避免大字段频繁查询**

TEXT 和 BLOB 这种大字段会拖累查询性能，把它们单独拆到一张表。

*-- 原表*  
CREATE TABLE users (  
    id INT PRIMARY KEY,  
    name VARCHAR(50),  
    profile TEXT  
);  
  
*-- 拆表*  
CREATE TABLE user\_profiles (  
    user\_id INT PRIMARY KEY,  
    profile TEXT  
);

**优化原因**：减少主表查询时的大字段负担。

**13. 定期清理无用数据**

表里垃圾数据太多，索引也跟着膨胀，查询性能直线下降。

*-- 定期清理历史数据*  
DELETE FROM logs WHERE create\_time < '2022-01-01';

**优化原因**：保持表轻量化，避免数据膨胀。

**14. 用分区表优化大数据表**

分区表是大数据量优化的利器，把数据按范围分成多个分区。

CREATE TABLE orders (  
    id INT NOT NULL,  
    order\_date DATE NOT NULL  
) PARTITION BY RANGE (YEAR(order\_date)) (  
    PARTITION p2022 VALUES LESS THAN (2023),  
    PARTITION p2023 VALUES LESS THAN (2024)  
);

**优化原因**：查询只扫描一个分区，而不是整张表。

**15. 索引字段选用低基数的字段**

基数低的字段（比如性别）用索引没意义，能用组合索引就用。

*-- 错误：性别字段加索引*  
CREATE INDEX idx\_gender ON users(gender);  
  
*-- 正确：性别+年龄组合索引*  
CREATE INDEX idx\_gender\_age ON users(gender, age);

**优化原因**：避免低基数索引浪费存储。

**16. 控制表字段数量**

表字段太多会让数据表臃肿，查询性能也会变差。合理拆分表。

**17. 用事务优化批量更新**

大批量更新用事务，可以减少锁竞争。

START TRANSACTION;  
UPDATE users SET age = age + 1 WHERE age > 25;  
COMMIT;

**18. 用查询缓存**

MySQL 有查询缓存机制，适合静态数据高频查询场景。

*-- 开启查询缓存*  
SET GLOBAL query\_cache\_size = 1048576;  
SELECT SQL\_CACHE \* FROM users WHERE age > 25;

**19. 用PreparedStatement**

预编译可以提高性能，还能防止 SQL 注入。

**20. 定期优化表**

表用了很久会有碎片，定期优化表能提高性能。

OPTIMIZE TABLE users;

**总结**

SQL 优化是个手艺活，写好 SQL 就能少掉无数锅，尤其是团队开发时，一个烂 SQL 能拖死整个项目。