SQL 查询手册

一、基础查询与数据操作

1. 基础检索与过滤

- **全选与特定列:**

- `SELECT * FROM 表名;` (全选) / `SELECT 列 1, 列 2 FROM 表名;` (选特定列)

注意: 避免使用 `SELECT *`, 仅查询必要列以减少 IO 和网络传输。

- **去重与分页:**

`SELECT DISTINCT 列 FROM 表名;` (去重) / `LIMIT 5 OFFSET 10;` (从第 11 条开始取 5 条)。

- **条件过滤:**

- **运算符:** `BETWEEN` (闭区间)、`IN` (枚举值)、`LIKE` (模糊匹配)、`REGEXP` (正则表达式)。
- **空值处理:** `IS NULL` / `IS NOT NULL`, 结合 `COALESCE` (列, 默认值) 填充空值。

2. 数据操作 (CRUD)

- **插入优化:**

- **批量插入:** `INSERT INTO 表名 (列 1, 列 2) VALUES (值 1, 值 2), (值 3, 值 4);` (减少网络交互)。
- **冲突处理:** `INSERT IGNORE` (忽略重复) / `INSERT REPLACE` (替换重复)。

- **更新与删除:**

- **批量更新:** `UPDATE 表名 SET 字段=值 WHERE 条件;` (务必加 `WHERE`, 避免全表更新)。
- **清空表:** `TRUNCATE TABLE 表名;` (比 `DELETE` 更快, 不可逆)。

二、高级查询与复杂场景

1. 连接查询 (JOIN)

- **类型与场景:**

- **内连接 (INNER JOIN):** 返回交集数据, 适用于匹配关联字段 (如订单与用户)。
- **左连接 (LEFT JOIN):** 左表全显, 右表无匹配则为 `NULL`, 适用于统计缺失数据 (如未下单的用户)。
- **交叉连接 (CROSS JOIN):** 返回笛卡尔积, 需谨慎使用 (如生成测试数据)。

- **多表关联优化:**

- 调整 JOIN 顺序: 先过滤小表再关联大表, 减少中间结果集。
- 复合索引: 在 JOIN 条件字段上建立索引, 避免全表扫描。

2. 子查询与 CTE

- **子查询优化:**

- 用 EXISTS 替代 IN: EXISTS 一旦找到匹配即停止, 效率更高。
- 关联子查询转 JOIN: 将嵌套子查询重构为 JOIN, 减少重复计算。

- **CTE (公用表表达式):**

- 简化嵌套查询:

```
WITH 部门平均薪资 AS (  
    SELECT 部门 ID, AVG(薪资) AS 平均薪资 FROM 员工 GROUP  
    BY 部门 ID  
)
```

```
SELECT 员工. 姓名, 部门平均薪资. 平均薪资  
FROM 员工 JOIN 部门平均薪资 ON 员工. 部门 ID = 部门平均薪  
资. 部门 ID;
```

- 递归 CTE: 处理层级数据 (如部门架构、目录树)。

3. 窗口函数与聚合分析

- **窗口函数语法:**

函数名(列) OVER (PARTITION BY 分组列 ORDER BY 排序列 ROWS
BETWEEN 范围);

- **常用函数:**

- 排名: RANK() (相同值同排名, 有空缺)、DENSE_RANK()
(相同值同排名, 无空缺)。
- 累计计算: SUM(金额) OVER (ORDER BY 日期) (按日期累
计金额)。
- 移动平均: AVG(销量) OVER (ROWS BETWEEN 2 PRECEDING
AND CURRENT ROW) (前两行 + 当前行的平均)。

三、性能优化与安全

1. 索引优化

- **索引类型选择:**

- **B-Tree 索引:** 适用于范围查询 (如 WHERE 年龄 > 18) 和排序
(ORDER BY)。

- **哈希索引**：仅支持等值查询（如 WHERE ID = 1001），内存表（MEMORY 引擎）常用。
- **复合索引设计**：
 - 遵循“最左匹配原则”：索引列顺序需与查询条件顺序一致。
 - 覆盖索引：包含查询所需的所有列，避免回表（如 CREATE INDEX idx_覆盖 ON 订单(用户 ID, 金额, 日期)）。

2. 执行计划分析 (EXPLAIN)

- **关键指标解读**：
 - **type**：连接类型，从优到劣为 system > const > eq_ref > ref > range > index > ALL。
 - **rows**：扫描行数，越小越好。
 - **Extra**：关注 Using filesort（文件排序）、Using temporary（临时表）等低效操作，需优化索引。

3. 事务与锁机制

- **事务隔离级别**：
 - **读未提交 (Read Uncommitted)**：可能出现脏读，性能最高。
 - **读提交 (Read Committed)**：避免脏读，可能出现不可重复读（MySQL 默认级别）。
 - **可重复读 (Repeatable Read)**：避免脏读和不可重复读，可能出现幻读（InnoDB 默认级别）。
 - **串行化 (Serializable)**：最高一致性，性能最差，适用于金融场景。
- **死锁处理**：
 - 检测死锁：SHOW ENGINE INNODB STATUS; 查看死锁日志。
 - 预防策略：减少事务大小、按相同顺序访问资源、降低隔离级别。

4. 安全与防注入

- **参数化查询**：

使用占位符（如?）预编译语句，避免拼接 SQL 字符串：

-- 危险写法

```
SELECT * FROM 表 WHERE 用户名 = '${用户输入}';
```

-- 安全写法 (Java 示例)

```
PreparedStatement stmt = conn.prepareStatement("SELECT * FROM  
表 WHERE 用户名 = ?");
```

```
stmt.setString(1, 用户输入);
```

- 权限管理:

- 最小权限原则: 仅授予用户必要权限 (如 GRANT SELECT ON 表 TO '用户'@'IP';)。

- 定期审计: 检查 mysql.user 表, 删除闲置用户。

四、数据库差异与高级功能

1. 主流数据库语法对比

功能	MySQL	SQL Server	Oracle
分页	LIMIT 10 OFFSET 20	OFFSET 20 ROWS FETCH NEXT 10 ROWS	ROWNUM <= 10 AND ROWNUM > 20
自动增长列	AUTO_INCREMENT	IDENTITY(1, 1)	序列 CREATE SEQUENCE seq;
字符串拼接	CONCAT(列 1, 列 2)	+或 CONCAT()	CONCAT()
日期函数	NOW()	GETDATE()	SYSDATE

2. 高级功能

- 存储过程 (MySQL):

```
DELIMITER //
```

```
CREATE PROCEDURE 计算平均薪资 (IN 部门 ID INT, OUT 平均薪资  
DECIMAL(10, 2))
```

```
BEGIN
```

```
SELECT AVG(薪资) INTO 平均薪资 FROM 员工 WHERE 部门 ID = 部门  
ID;
```

```
END //
```

```
DELIMITER ;
```

```
-- 调用
```

```
CALL 计算平均薪资(1, @avg);
```

```
SELECT @avg;
```

注意：存储过程可封装复杂逻辑，但需谨慎使用动态 SQL 以避免注入风险。

- **递归 CTE (PostgreSQL) :**

```
WITH RECURSIVE 部门树 AS (
```

```
    SELECT ID, 父 ID, 名称 FROM 部门 WHERE ID = 1
```

```
    UNION ALL
```

```
    SELECT 子部门.ID, 子部门.父 ID, 子部门.名称 FROM 部门树
```

```
    JOIN 部门 子部门 ON 部门树.ID = 子部门.父 ID
```

```
)
```

```
SELECT * FROM 部门树;
```

用于查询层级数据（如组织架构、目录结构）。

五、性能优化实战案例

案例：电商系统复杂查询优化

场景：查询“近 30 天内购买过 TOP10 热销商品的用户及其复购率”。

优化步骤：

1. **执行计划分析：**

```
EXPLAIN
```

```
SELECT 用户.姓名, COUNT(订单.ID) AS 购买次数
```

```
FROM 订单
```

```
JOIN 用户 ON 订单.用户 ID = 用户.ID
```

```
JOIN 商品 ON 订单.商品 ID = 商品.ID
```

```
WHERE 订单.日期 >= CURDATE() - INTERVAL 30 DAY
```

```
    AND 商品.ID IN (SELECT 商品 ID FROM 订单 GROUP BY 商品 ID
```

```
    ORDER BY COUNT(*) DESC LIMIT 10)
```

```
GROUP BY 用户.ID;
```

○ 问题：子查询重复执行，JOIN 顺序未优化。

2. **重构查询：**

-- 预计算 TOP10 商品

```
WITH 热销商品 AS (
```

```
    SELECT 商品 ID FROM 订单
```

```

WHERE 日期 >= CURDATE() - INTERVAL 30 DAY
GROUP BY 商品 ID ORDER BY COUNT(*) DESC LIMIT 10
)
SELECT 用户.姓名, COUNT(订单.ID) AS 购买次数
FROM 订单
JOIN 用户 ON 订单.用户 ID = 用户.ID
JOIN 热销商品 ON 订单.商品 ID = 热销商品.商品 ID
WHERE 订单.日期 >= CURDATE() - INTERVAL 30 DAY
GROUP BY 用户.ID;

```

3. 索引优化:

- 为订单(日期, 商品 ID)建立复合索引, 加速日期过滤和商品关联。
- 为用户.ID建立主键索引, 加速 JOIN 操作。

结果: 查询耗时从 1.2 秒降至 200 毫秒。

六、注意事项与最佳实践

1. 避免全表扫描:

- 确保 WHERE 条件、JOIN 条件使用索引, 避免 type=ALL。

2. 分批处理大查询:

- 对百万级数据, 使用 LIMIT 分批查询, 避免内存溢出:
-- 分批次删除

```
DELETE FROM 日志 WHERE 日期 < '2023-01-01' LIMIT 1000;
```

3. 监控与调优工具:

- **MySQL:** pt-query-digest 分析慢查询日志, SHOW PROFILE 追踪查询耗时。
- **Oracle:** SQL Tuning Advisor 生成优化建议, AWR 报告分析性能趋势。

七、速查表格

功能	语法示例	说明
批量插入	INSERT INTO 表 VALUES (1, 'A'), (2, 'B');	减少网络交互, 提升写入效率
范围查询	BETWEEN 10 AND 20	闭区间, 等价于 >=10 AND <=20

功能	语法示例	说明
分页	LIMIT 10 OFFSET 20 (MySQL)	其他数据库语法见 “数据库差异” 章节
临时表	CREATE TEMPORARY TABLE 临时表 AS SELECT ... ;	会话级临时表，关闭连接自动删除
递归 CTE	WITH RECURSIVE ... UNION ALL	处理层级数据（如部门树）