总结

🧩 一、表连接（JOIN）

> **核心概念**：将两张（或多张）表按指定条件“拼接”在一起，返回符合条件的行。

连接类型	定义	使用场景	示例代码	常见陷阱 & 性能建议
INNER JOIN	只保留在两张表中同时存在匹配行	1. 两表完全匹配；2. 只关心交集数据。	`SELECT * FROM A JOIN B ON A.id = B.id;`	- 未匹配行被丢弃；- 大表 join 小表性能受网络依赖。
LEFT JOIN	保留左表所有行，右表无匹配则补 NULL	1. 左表主表，需展示所有左表记录；2. 查缺失数据（无关联）。	`SELECT * FROM A LEFT JOIN B ON A.id = B.id;`	- 需配合 `WHERE B.id IS NULL` 查无关联；- 大量 NULL 列可能影响网络传输。
RIGHT JOIN	保留右表所有行，左表无匹配补 NULL（较少使用）	当右表为主表时，可换成 `LEFT JOIN` 简化。	`FROM A RIGHT JOIN B ON A.id = B.id;`	- 可用 `LEFT JOIN` 代替，保持统一风格。
FULL JOIN	保留两表所有行，不匹配的补 NULL（部分 DBMS 支持）	需要两表所有数据，不论匹配与否。	`FROM A FULL JOIN B ON A.id = B.id;`	- MySQL 不原生支持；可用 `UNION` + 两个 `LEFT JOIN` 实现。
SELF JOIN	表自身与自身连接，用别名区分两份实例	1. 查“员工-经理”；2. 比较同表不同行。	`SELECT e.name mgr.name FROM Emp e JOIN Emp m ON e.mgrId=m.id;`	- 别名命名要清晰；- 注意避免 Cartesian Product。

🔍 二、子查询（Subquery）

> **核心概念**：在一个查询内部嵌套另一个查询，可用于计算条件、筛集成集合或构造临时表。

子查询类型	定义	使用场景	示例代码	常见陷阱 & 建议
标量子查询	返回单个值（单行单列），用于 `SELECT` 或 `WHERE` 条件	1. 求最大/最小/平均值；2. 计算阈值后再过滤主表。	`SELECT * FROM E WHERE salary > (SELECT MAX(salary) FROM E);`	- 子查询必须只返回一行，否则报错；- 可用聚合 + `LIMIT 1`。
表子查询（派生表）	返回多行多列，可在 `FROM` 或 `JOIN` 中当做临时表使用	构造临时视图，拆分复杂逻辑。	`SELECT t.dept, COUNT(*) FROM (SELECT dept FROM Emp WHERE ...) t GROUP BY t.dept;`	- 必须指定别名；- 临时表大小影响性能。
IN / NOT IN	判断某列值是否在子查询的集合中	1. 包含/排除列表；2. 简单多值匹配。	`SELECT * FROM C WHERE id IN (SELECT custId FROM O);`	- 若子查询含 NULL，则 `NOT IN` 可能不返回任何行；- 推荐用 `EXISTS`。
EXISTS / NOT EXISTS	判断子查询是否至少返回一行，多用于反向排除	1. 反向查询；2. 性能优于 `IN`（大数据下）。	`SELECT * FROM C c WHERE EXISTS(SELECT 1 FROM O o WHERE o.custId = c.id);`	- `EXISTS` 只检测存在性，可利用索引；- 内层 `SELECT` 内容常写 `1` 或 `*`。

📊 三、分组与聚合（GROUP BY / HAVING）

> **核心概念：**将多行数据根据某些字段分组后，对每组执行聚合计算。

功能	定义	示例	易错点 & 建议
GROUP BY	指定分组字段后，其他非分组字段必须被聚合或出现在 `GROUP BY` 中	`SELECT dept, COUNT(*) FROM Emp GROUP BY dept;`	- `SELECT` 中字段要么在 `GROUP BY`，要么包裹在聚合函数里；- 大量分组字段影响性能。
HAVING	对 `GROUP BY` 后的结果再次筛选	`... GROUP BY dept HAVING COUNT(*) > 5;`	- 不能写在 `WHERE`；- `HAVING` 执行在分组之后，`WHERE` 在分组前。
COUNT(*)	统计组内记录总数	`COUNT(*)`	- 统计所有行，包括 NULL；- 速度快。

功能	定义	示例	易错点 & 建议
COUNT(DISTINCT)	去重计数	`COUNT(DISTINCT user_id)`	- 对去重计数性能稍差；- MySQL 在大表下可使用 HyperLogLog。
SUM、AVG	分别求和、求平均	`SUM(price)`、`AVG(score)`	- NULL 自动忽略；- 可加 `ROUND()` 保留精度。
MAX、MIN	求极值	`MAX(salary)`、`MIN(date)`	- 忽略 NULL；- 支持索引加速查询。

⏱️ 四、排序与限制（ORDER BY / LIMIT / OFFSET）

> **核心概念**：对结果集进行有序输出，并限制返回行数。

功能	定义	示例	注意事项
ORDER BY	按指定字段排序，默认升序，可加 `DESC` 降序	`SELECT * FROM E ORDER BY hire_date DESC;`	- 未指定 `DESC` 则升序；- 可多字段排序。
LIMIT n	限制返回前 n 行	`... LIMIT 10;`	- MySQL/SQLite；- SQL Server 用 `TOP`。
OFFSET n	跳过前 n 行，再返回 LIMIT 行（常与 LIMIT 配合）	`... LIMIT 10 OFFSET 20;`	- 性能差：往后翻页时全表扫描成本高；- 大数据可考虑 keyset pagination。
FETCH FIRST / NEXT	SQL 标准写法，部分 DBMS 支持	`FETCH FIRST 5 ROWS ONLY`	- 兼容性需注意。

□ 五、窗口函数（Window Functions）

> **核心概念**：在结果集中对每一行保留原始行，然后基于某种“窗口”对其前后行或同组行进行计算。

函数	定义 & 场景	示例	建议 & 陷阱
ROW_NUMBER()	按排序后为每行生成唯一序号	`ROW_NUMBER() OVER(PARTITION BY dept ORDER BY salary DESC)`	- 并列数据无并列号；- 可与子查询结合取 Top N。
RANK()	并列排名会跳号	`RANK() OVER(ORDER BY score DESC)`	- 并列后下一个排名跳过并列数。
DENSE_RANK()	并列排名不跳号	`DENSE_RANK() OVER(PARTITION BY dept ORDER BY salary DESC)`	- 并列后下一个排名紧跟；- 常用于 Top N 并列场景。
LAG()/LEAD()	向前/向后取第 n 行的数据，用于与当前行比较	`LAG(temp,1) OVER(ORDER BY date)`	- 返回 NULL 时需用 `COALESCE()`；- 对时序数据尤为常用。
FIRST_VALUE()	返回窗口中第一个值	`FIRST_VALUE(event_date) OVER(PARTITION BY user ORDER BY event_date)`	- 保留原行，可获取分组内第一个值。
PARTITION BY	在指定分区内执行窗口函数	同上	- 类似 `GROUP BY`，但不压缩行；- 适合保留行上下文。

🔁 六、更新与删除（UPDATE / DELETE）

> **核心概念**：修改或删除表中数据，可结合条件、子查询、连接等实现批量操作。

操作	场景	示例	注意事项 & 建议
UPDATE + CASE	条件更新，不同条件赋不同值	`UPDATE Salary SET sex = CASE sex WHEN 'm' THEN 'f' ELSE 'm' END;`	- 一条语句完成多条件更新；- 其他 DBMS 可用 `IIF()` / `DECODE()`。
DELETE ... JOIN	MySQL 特有，删除与其他表或自身连接后符合条件的行	`DELETE p1 FROM Person p1 JOIN Person p2 ON ...;`	- 其他 DBMS 需子查询；- 批量删大量数据时注意事务日志。
DELETE WHERE IN	删除满足子查询条件的行	`DELETE FROM Person WHERE id NOT IN (SELECT MIN(id) FROM`	- `NOT IN` 对 NULL 敏感；- 推荐用 `NOT EXISTS`。

操作	场景	示例	注意事项 & 建议
		`Person GROUP BY email);`	

🧮 七、日期与数学函数

> **核心概念**：对日期、数值进行特殊计算。

函数 / 表达式	含义 & 示例	注意事项
DATEDIFF(d1, d2)	计算两个日期间相差天数	`DATEDIFF(day, '2025-06-19', '2025-06-20') = 1`
ROUND(x, n)	四舍五入保留 n 位小数	`ROUND(sum(s), 2)`
MOD(col, m) 或 `col % m`	取模（判断奇偶、分组）	`id % 2 = 1`
COALESCE(a, b, ...)	返回第一个非 NULL 值	`COALESCE(LEAD(x), x)`
DATEADD/ADDDATE	日期加减（MySQL `DATE_ADD` / SQL Server `DATEADD`）	`DATE_ADD(recordDate, INTERVAL 1 DAY)`

📋 八、其他技巧 & 高频考点

> **核心概念**：一些常见但容易忽略的小技巧和考点。

技巧	用途	示例 & 建议
DISTINCT	去重	`SELECT DISTINCT email`；`COUNT(DISTINCT x)`
组合匹配 `(col1, col2) IN`	同时匹配多个列	`(dept, salary) IN (SELECT dept, MAX(salary) ...)`
CTE（WITH）	分步构造临时结果，提高可读性	`WITH t AS(...) SELECT ... FROM t;`
键集分页（Keyset Pagination）	大表翻页时用索引字段代替 LIMIT+OFFSET，性能更优	`WHERE id > 上次最大ID LIMIT N`

🧠 九、常见题型 & 解题模板

题型	核心思路	模板示例
缺失关联	`LEFT JOIN … WHERE … IS NULL` / `NOT EXISTS`	`SELECT * FROM A LEFT JOIN B ON … WHERE B.id IS NULL;`
重复值查找	`GROUP BY … HAVING COUNT(*)>1` / `EXISTS`	`SELECT val FROM T GROUP BY val HAVING COUNT(*)>1;`
连续数据	`LAG()/LEAD()` / `id-ROW_NUMBER()` 差分分组	`LAG(x) OVER(ORDER BY id)` 或 `id - ROW_NUMBER() OVER(...)`
第 N 高 / 排名	`LIMIT … OFFSET …` / `RANK()/DENSE_RANK()`	`SELECT * FROM T ORDER BY x DESC LIMIT 1 OFFSET N-1;`
首次/次日行为	`MIN(date)` / `FIRST_VALUE()` + `DATEDIFF() = 1`	`MIN(event_date)` 或 `DATEDIFF(a.date,b.min_date)=1`
Top N 并列	`DENSE_RANK() <= N` / 子查询计数	`DENSE_RANK() OVER(PARTITION BY dept ORDER BY sal DESC)<=3`
删除重复行	`DELETE JOIN` / `DELETE WHERE NOT IN (SELECT MIN...)`	`DELETE p1 FROM P p1 JOIN P p2 ON … AND p1.id>p2.id;`
条件更新	`UPDATE … SET col = CASE…END`	`UPDATE T SET col = CASE WHEN… END;`

> Tip：在刷题过程中，遇到性能问题时，**先确认数据量级、检查索引、适当分批**，再考虑更高级的优化（如物化视图、分区表等）。
