07_数据查询语言DQL(多表)

2024全新MySQL企业开发版

目录
Contents

1 DQL概述和回顾

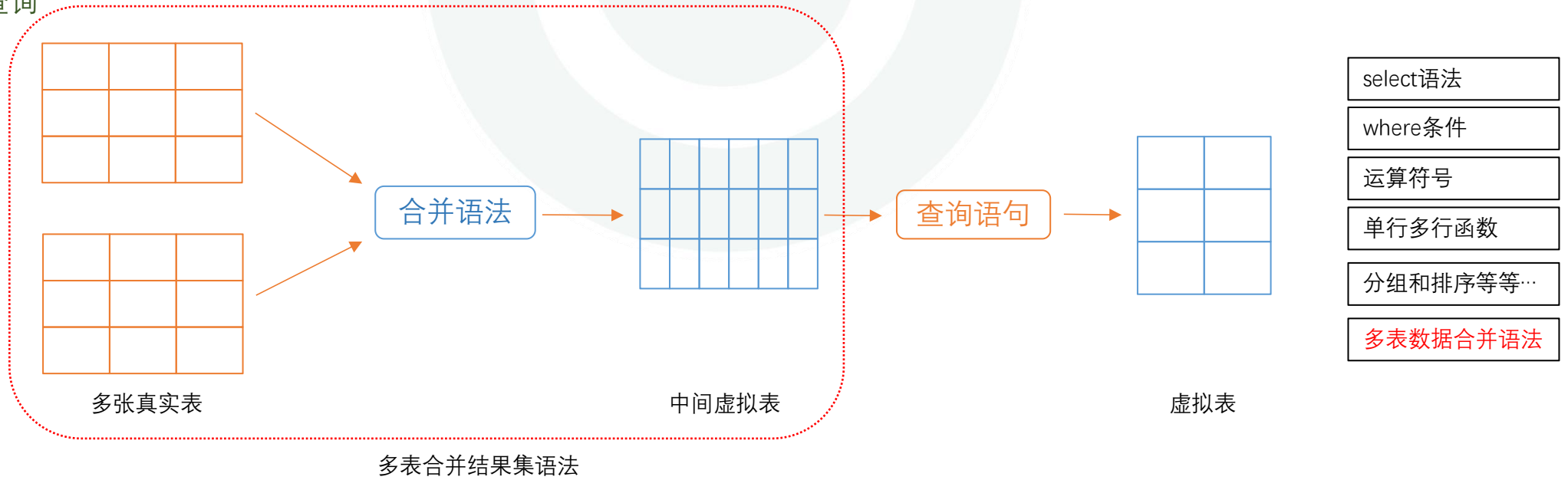
1 DQL概述和回顾

数据查询语句（DQL）语法分类

单表查询

- select语法
- where条件
- 运算符号
- 单行多行函数
- 分组和排序等等…

多表查询

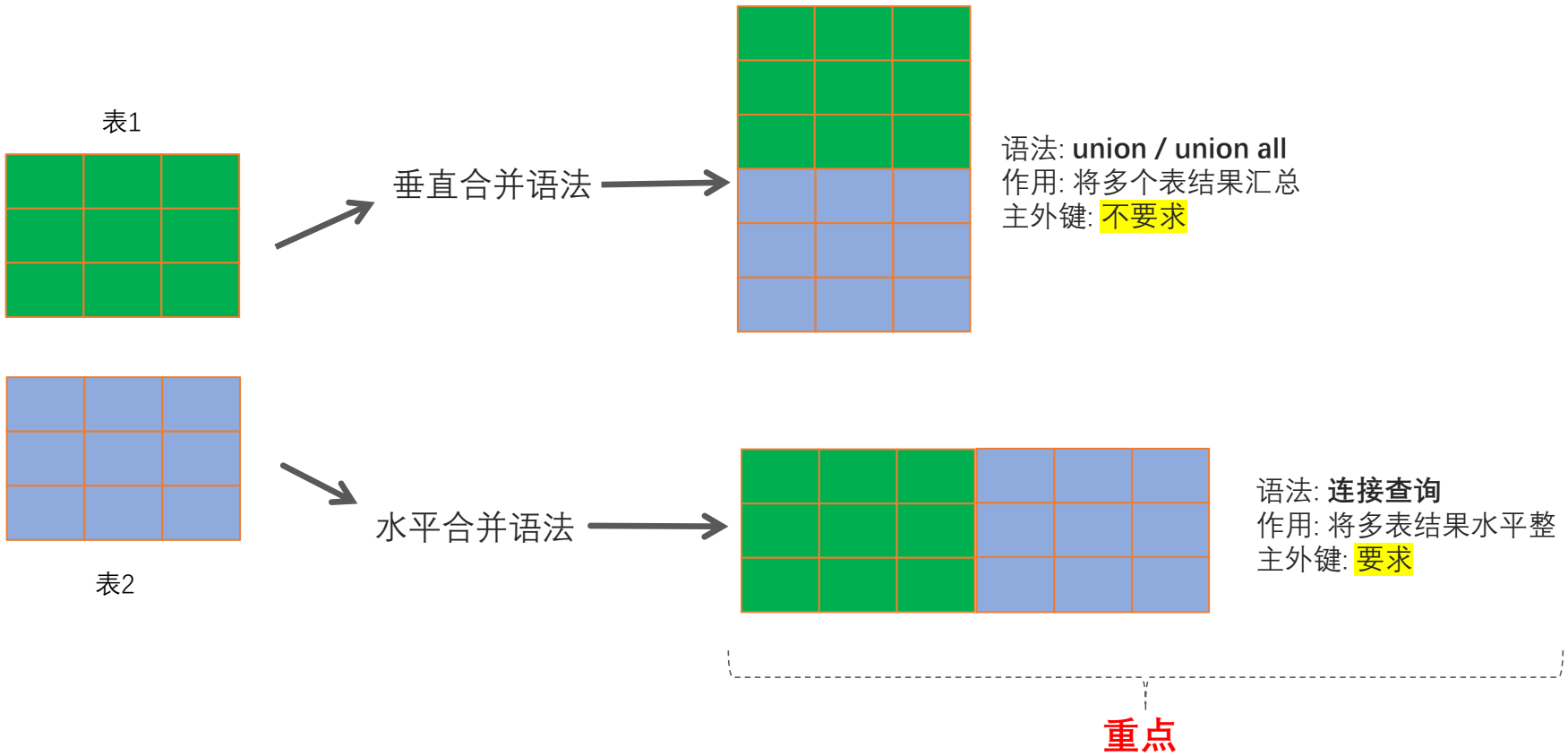


- select语法
- where条件
- 运算符号
- 单行多行函数
- 分组和排序等等…
- 多表数据合并语法

1 DQL概述和回顾

多表查询语句（DQL）语法理解

多表查询的重点是将多张表数据利用`多表查询语法`合并成单张虚拟表
按照多表结果合并的方向,可以分为:**水平合并语法**和**垂直合并语法**

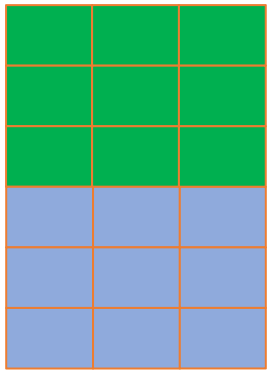


1 DQL概述和回顾

多表查询语句（DQL）语法理解

多表查询的重点是将多张表数据利用`多表查询语法`合并成单张虚拟表
按照多表结果合并的方向,可以分为:**水平合并语法**和**垂直合并语法**

垂直合并语法 →



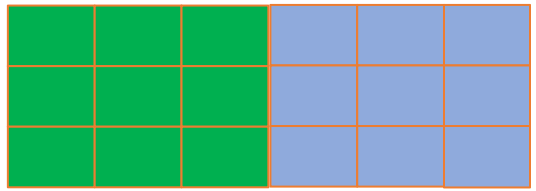
语法: union / union all
作用: 将多个表结果汇总
主外键: 不要求

1 DQL概述和回顾

多表查询语句（DQL）语法理解

多表查询的重点,就是将多张表结果集利用`语法`合并成单张虚拟表,然后再进行条件和高级查询处理等
按照多表结果合并的方向,可以分为:**水平合并语法**和**垂直合并语法**

水平合并语法 →



语法: 连接查询
作用: 将多表结果水平整
主外键: 要求

重点

A表

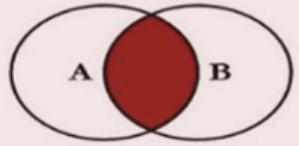
员工编号	姓名	薪资	部门编号
S1001	张三	15000	D1001
S1002	李四	13000	D1001
S1003	如意	14000	D1002
S1004	吉祥	15000	D1003
S1005	王五	16000	NULL

B表

部门编号	部门名称	部门职责
D1001	技术部	负责技术研发工作
D1002	人事部	负责人事管理工作
D1003	市场部	负责市场推广工作
D1004	测试部	负责测试检查工作

A∩B结果

员工编号	姓名	薪资	部门编号	部门名称	部门职责
S1001	张三	15000	D1001	技术部	负责技术研发工作
S1002	李四	13000	D1001	技术部	负责技术研发工作
S1003	如意	14000	D1002	人事部	负责人事管理工作
S1004	吉祥	15000	D1003	市场部	负责市场推广工作



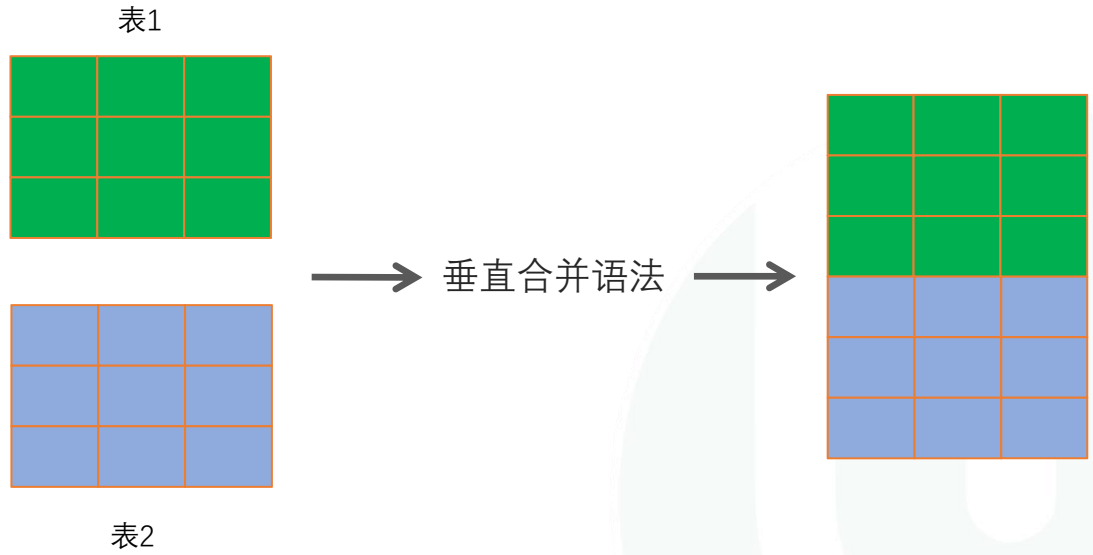
A∩B

A表 INNER JION B表

目录
Contents

2 合并结果集语法(垂直)

2 合并结果集语法(垂直)



语法:

```
union  # 合并记录同时去掉重复数据
union all  # 合并记录,且不去掉重复数据
```

细节:

实现要求：只要求合并的结果集之间的**列数和对应列的类型相同**即可

主外键要求：union只是结果集垂直汇总,不涉及行数据水平连接,**不要求有主外键**

重复数据认定：一行中的所有列值都相同,认定为重复行

场景示例:

场景1: 合并多个相似表的数据

假设一个电子商务网站在不同的数据库中存储了不同类型的商品信息，例如电子产品、服装和家居用品，每个类别对应一个表。

可以使用 UNION ALL 将这些表中的数据合并为一个统一的商品列表，以便进行产品推荐或展示。

场景2: 统计不同源的数据

一个网站可能同时从不同的渠道获取用户数据，如注册信息、访问日志等，这些数据可能存储在不同的表中。

使用 UNION 或 UNION ALL 将不同来源的用户数据合并为一个结果集，然后可以进行用户行为分析、流量统计等。

2 合并结果集语法(垂直)

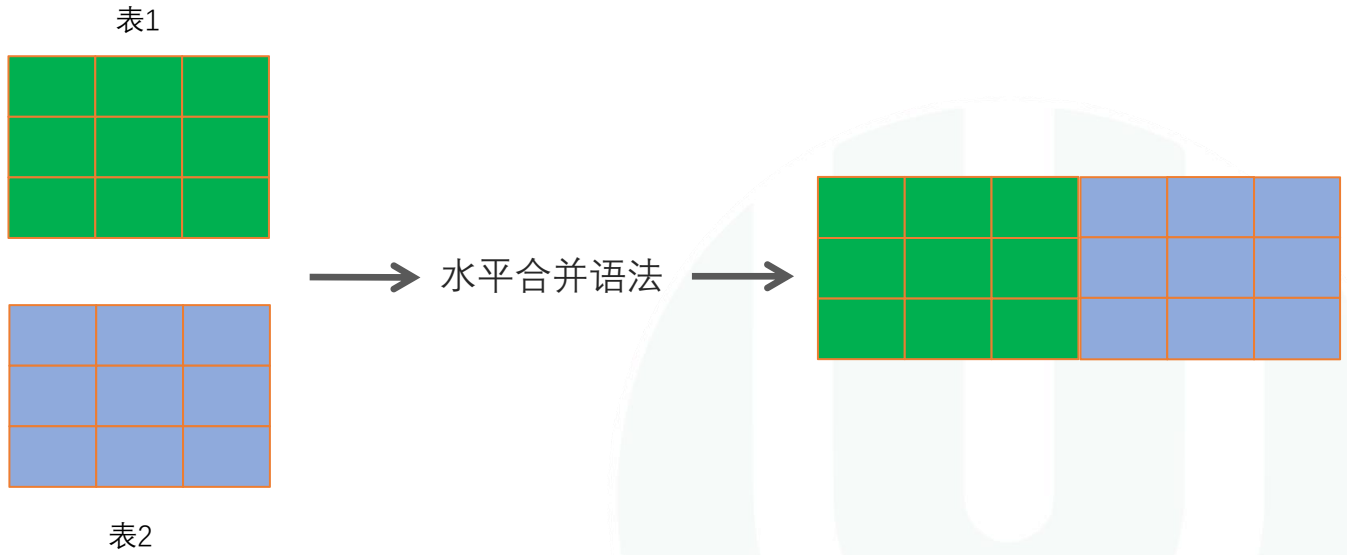
union和union all语法实战:

```
# 数据准备
CREATE TABLE a(
   aid INT,
   aname VARCHAR(10)
);

CREATE TABLE b(
   bid INT,
   bname VARCHAR(10)
);


INSERT INTO a VALUES(1,'aaaa'),(2,'bbbb'),(3,'cccc');
INSERT INTO b VALUES(4,'aaaa'),(2,'bbbb'),(3,'cccc');

# 去重复合并
SELECT aid,aname FROM a
UNION
SELECT bid ,bname FROM b;

# 不去重复合并
SELECT aid,aname FROM a
UNION ALL
SELECT bid ,bname FROM b;
```

目录
Contents

3 连接查询语法(水平)

3 连接查询语法(水平)



语法:

内连接: from 表1 [inner] join 表2 on 主 = 外 (标准)
from 表1,表2 where 主 = 外 (非标准)
外连接: from 表1 left | right [outer] join 表2 on 主 = 外
自然连接: from 表1 natural [left | right] join 表2

细节:

核心要求: 水平连接是行和行数据连接,要求两个表必须有关系(主外键)
正确连接: 为了正确的将行数据之间连接,水平连接需要额外判定主外键相等
拆表产物: 因为关系型数据库特点,数据进行拆表存储,所以水平连接语法非常重要
语法理解: 别看水平连接有多种语法,但是基本大同小异,只要理解内外区别即可

3 连接查询语法(水平)

3.1 内连接查询语法

内连接（inner join）是一种用于从两个或多个表中检索数据的查询方式。内连接会根据指定的连接条件，将两个表中满足条件的行进行匹配，并返回匹配成功的行!

语法:

```
select * from 表1 [ inner ] join 表2 on 表1.主键 = 表2.外键 (标准)
select * from 表1,表2 where 表1.主键 = 表2.外键 (非标准)
```

细节1: 以上两种语法功能和效果相同, 推荐标准语法

标准的内连接更通用! 两者效果一致, 内连接的特点就是两个表必须满足主外键相等,方可返回数据!

细节2: 为了避免错误的数据连接,连接查询必须添加主外键相等条件

连接查询就是将所有数据行相互拼接一次! 例如: A和B连接查询, A有3条数据,B有4条数据,会出现12条数据的结果,但是数据行不一定正确连接,这就是经典的笛卡尔积问题! 我们可以通过添加主外键相等,进行正确数据筛选!

细节3: 多表查询要考虑不同表存在相同字段名的问题

多表中很大概率存在相同的名称的列(主外键一般命名相同),在select后或者条件比较列的时候,需使用表名.列名! 但是表名可能较长,可以给表名起别名,语法为: from 表 别名 | from as 别名! 这样就就可以通过: 表别名.列名

3 连接查询语法(水平)

3.1 内连接查询语法

准备数据:

部门表: 部门编号、部门名称、部门简介
岗位表: 岗位编号、岗位名称、岗位简介
员工表: 员工编号、姓名、薪资、奖金比例、生日、性别、电话、邮箱、住址、工作地、
入职日期、岗位编号、领导编号、部门编号

情景1: 基础语法和笛卡尔积

```
# 查询员工编号、姓名以及所属部门的编号和部门名称
SELECT COUNT(1) FROM t_employee e;
SELECT COUNT(1) FROM t_department d ;
SELECT COUNT(1) FROM t_employee e , t_department d ;
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e , t_department d ;
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e  inner join  t_department d ;
# 结果出现“笛卡尔积”现象，  A表记录 * B表记录,但是数据连接是错误的!
```

情景2: 主外键条件和正确连接

```
# 查询员工编号、姓名以及所属部门的编号和部门名称
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e , t_department d where
e.did = d.did ; # 主外键相等
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e  inner join  t_department d
on e.did = d.did; # 主外键相等
```

3 连接查询语法(水平)

3.1 内连接查询语法

情景3: 标准inner join语法优化

```
# 查询员工编号、姓名以及所属部门的编号和部门名称
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e  join  t_department d on
e.did = d.did ;
# inner 关键字可以省略, inner join  == join
# on 只能在连接语法的场景下使用, 配合join进行正确数据筛选
# on 绝大情况下可放置的就是主外键相等,其他条件建议使用where筛选
```

情景4: 添加额外的条件筛选

```
# 查询员工编号大于10的员工编号、姓名以及所属部门的编号和部门名称
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e , t_department d where
e.did = d.did and e.eid > 10 ; # 主外键相等
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e  inner join  t_department d
on e.did = d.did where e.eid > 10; # 主外键相等
```

情景5: 多表(3+)查询并且添加额外的条件筛选

```
# 查询员工编号大于10的员工编号、姓名以及所属部门的编号和部门名称,岗位名称
SELECT e.eid,e.ename,e.did,d.did,d.dname,j.jname FROM t_employee e , t_department
d,t_job j where e.did = d.did and e.job_id = j.jid and  e.eid > 10 ;
SELECT e.eid,e.ename,e.did,d.did,d.dname ,j.jname FROM t_employee e  inner join
t_department d on e.did = d.did inner join t_job j  on e.job_id = j.jid  where e.eid > 10;
```

3 连接查询语法(水平)

3.2 外连接查询语法

外连接（outer join）是一种用于从两个或多个表中检索数据的查询方式，与内连接不同的是，外连接会返回所有符合条件的行，同时还会返回未匹配的行。外连接分为左外连接（LEFT JOIN）、右外连接（RIGHT JOIN）！

语法:

```
select * from 表1 left [ outer] join 表2 on 表1.主键 = 表2.外键（左外）
select * from 表1 right [ outer] join 表2 on 表1.主键 = 表2.外键（右外）
```

细节1: 内连接和外连接语法效果区别

内连接: 只满足匹配条件的行数.两个表必须存在且主外键值相等才会返回
外连接: 可以通过左和右指定一个逻辑主表,逻辑主表数据一定会查询到

细节2: 外连接语法可以省略和优化

外连接的语法也可以省略 [outer]
left | right outer join = left | right join
left 和 right 就是指定左还右是逻辑主表

细节3: 外连接的连续性

建议将分析的逻辑主表放在第一个位置(最左),那么本次查询必然全部是左外连接!
from 逻辑主表 **left join** 表2 on 主 = 外 **left join** 表3 on 主 = 外 **left join** .....

3 连接查询语法(水平)

3.2 外连接查询语法

思考: 以下两个需求选择内还是外连接?

总结:
开发中绝大部分情况都是逻辑主表, 存在逻辑主表就要使用外连接!外连接登场率会更高!

3 连接查询语法(水平)

3.2 外连接查询语法

沿用数据:

部门表: 部门编号、部门名称、部门简介
岗位表: 岗位编号、岗位名称、岗位简介
员工表: 员工编号、姓名、薪资、奖金比例、生日、性别、电话、邮箱、住址、工作地、
入职日期、岗位编号、领导编号、部门编号

情景1: 基础语法和笛卡尔积 [错误演示]

```
# 查询所有员工编号、姓名以及所属部门的编号和部门名称
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e left outer join t_department d ;
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_department d  right  join  t_employee e ;
# sql语句异常,1064错误,外连接必须添加 on 主外键相等!
# 查询所有员工,就算没有部门,所以员工是逻辑主表 left | right 指向员工表
```

情景2: 主外键条件和正确连接 [正确语法]

```
# 查询所有员工编号、姓名以及所属部门的编号和部门名称
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e left outer join t_department d on e.did = d.did ;
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_department d  right  join  t_employee e on e.did = d.did ;
```

3 连接查询语法(水平)

3.2 外连接查询语法

情景3: 添加额外的条件筛选

```
# 查询员工编号大于10的员工编号、姓名以及所属部门的编号和部门名称
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e  left join  t_department d
on e.did = d.did where e.eid > 10; # 主外键相等
```

情景4: 多表(3+)查询并且添加额外的条件筛选

```
# 查询员工编号大于10的员工编号、姓名以及所属部门的编号和部门名称,岗位名称
SELECT e.eid,e.ename,e.did,d.did,d.dname ,j.jname FROM t_employee e  left join
t_department d on e.did = d.did left join t_job j  on e.job_id = j.jid  where e.eid > 10;
```

3 连接查询语法(水平)

3.3 内和外连接语法总结

1. 内外连接需要添加主外键判定,去除笛卡尔积问题。
2. 内连接属于`公平`查询法,双方都满足主外键相等方可查询。
3. 外连接属于`不公平`查询法,可以指定一方为逻辑主表,逻辑主表的数据一定会查询出来。
4. 一般情况主外键的条件添加到on后,其他条件依然使用where。
5. 如果有逻辑主表,建议放在最左侧,后续一致可以使用左外连接。
6. 多表联合查询,关联主外键条件个数为n-1。

多表查询关键字顺序:

```
SELECT ...,....,...
FROM ...,...,....
LEFT | RIGHT  JOIN ON
LEFT | RIGHT  JOIN ON
LEFT | RIGHT  JOIN ON  # 多表连接
WHERE AND
GROUP BY ...,...
HAVING
ORDER BY ... ASC/DESC
LIMIT ...,...
```

3 连接查询语法(水平)

3.3 自然连接查询语法

自然连接（natural join）是一种内连接和外连接的升级版，会自动找到两个表中相同的列名,判定相等, 可以省略on 主 = 外的语法!
但是也值得注意,除了主外键其他列名相同,它也会自动判定相等! 稍显不靠谱!

语法:

```
SELECT * FROM emp NATURAL JOIN dept ； #自然内连接
SELECT * FROM emp NATURAL LEFT JOIN dept; #自然左外连接
SELECT * FROM emp NATURAL RIGHT JOIN dept; #自然右外连接
```

细节1: 自然连接就是内和外连接语法升级版本

自然连接可以变成内连接或者外连接,会自动查找相同的列判定相等!
使用自然连接要求: 主外键命名要相同,除了主外键命名要不同!

细节2: 自然连接可以指定判定哪些列

例如: employees e JOIN departments d USING (department_id);
使用 USING 指定数据表里的`同名字段`进行等值连接。
因为其繁琐性,不是很推荐使用!

3 连接查询语法(水平)

3.3 自然连接查询语法

沿用数据:

```
部门表: 部门编号、部门名称、部门简介
岗位表: 岗位编号、岗位名称、岗位简介
员工表: 员工编号、姓名、薪资、奖金比例、生日、性别、电话、邮箱、住址、工作地、
        入职日期、岗位编号、领导编号、部门编号
```

情景1: 自然内连接使用

```
# 查询所有有部门的员工编号、姓名以及所属部门的编号和部门名称
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e natural join t_department
d ;
# 使用自然内连接
```

情景2: 自然外连接使用

```
# 查询所有员工编号、姓名以及所属部门的编号和部门名称
SELECT e.eid,e.ename,e.did,d.did,d.dname FROM t_employee e natural left join
t_department d;
# 自然外连接,不需要添加outer关键字
```

3 连接查询语法(水平)

3.4 自连接查询语法

自连接自连接是指在数据库中，一个表与自身进行连接的操作。它在查询中使用相同表的别名来表示两个不同的实例，然后通过连接条件将这两个实例进行连接。
自连接不是新的语法,就是单张表进行多次使用一种场景!

语法: 自连接依然使用内外连接语法实现

细节1: 自连接不是新语法,而是一种特殊的查询场景

自连接和自然连接不一样,自然连接是新的语法,自连接指定的是一张表连接自己实现特殊的多表查询的情况!

细节2: 自连接具体实现语法问题

自连接这种情况,依然需要内连接或者外连接或者自然连接具体语法实现!

细节3: 自连接的应用场景

一个表中就存在数据的引用关系, 要查询的数据关联在同一个表的其他行!!
例如: 员工表中有员工编号,领导编号, 而领导编号应用的也是同一张表的其他行数据! 如果查询员工信息和员工的领导信息,就是典型的自连接场景!

3 连接查询语法(水平)

3.4 自连接查询语法

沿用数据:

部门表: 部门编号、部门名称、部门简介
岗位表: 岗位编号、岗位名称、岗位简介
员工表: 员工编号、姓名、薪资、奖金比例、生日、性别、电话、邮箱、住址、工作地、
入职日期、岗位编号、领导编号、部门编号

情景1: 两次复用自连接

```
# 查询编号等于5号员工的编号,姓名,领导编号,领导姓名
SELECT e1.eid,e1.ename,e1.mid,e2.ename FROM t_employee e1 LEFT JOIN
t_employee e2 ON e1.mid = e2.eid
                    WHERE e1.eid = 5;
# 外连接,可能没有领导 e1充当员工 e2充当领导
```

情景2: 多次复用自连接

```
# 查询编号等于5号员工的编号,姓名,领导编号,领导姓名 ,以及领导的领导编号和姓名
SELECT e1.eid,e1.ename,e1.mid,e2.ename,e3.ename FROM
                        t_employee e1 LEFT JOIN t_employee e2 ON e1.mid = e2.eid
                                LEFT JOIN t_employee e3 ON e2.mid = e3.eid
                    WHERE e1.eid = 5;
# e1充当员工 e2充当领导 e3充当领导的领导
```

3.5 连接查询总结练习

练习题: https://leetcode.cn/problems/employees-earning-more-than-their-managers/description/

力扣181题: 超过经理收入的员工

Employee 表:

```
+----+-------+--------+-----------+
| id     | name   | salary | managerId |
+----+-------+--------+-----------+
| 1      | Joe      | 70000  | 3           |
| 2      | Henry  | 80000  | 4           |
| 3      | Sam     | 60000  | Null        |
| 4      | Max     | 90000  | Null        |
+----+-------+--------+-----------+
```

输出:

```
+----------+
| Employee    |
+----------+
| Joe           |
+----------+
```

解释: Joe 是唯一挣得比经理多的雇员。

目录
Contents

4 子查询语法(嵌套)

4 子查询语法(嵌套)

4.1 子查询语法介绍

子查询是指在 SQL 中嵌套另一个完整的 SELECT 查询语句，这个嵌套的查询通常被称为子查询或内部查询!

```
# 例如: 查询嵌套子查询
SELECT student_id, student_name, score FROM t_score  WHERE score > ( SELECT AVG(score) FROM t_score );
```

注意: 子查询也可以INSERT、UPDATE 或 DELETE 语句中使用，用于提供额外的条件、过滤结果或进行比较。

```
# 例如: 其他类型嵌套子查询
UPDATE t_scores
                SET score = ( SELECT AVG(score) FROM t_scores )
                WHERE student_id  = 5;
```

标量子查询：子查询返回单行单列，这个值通常用于条件判断、过滤或者作为 SELECT 查询的一部分

行子子查询：子查询返回一行多列，通常用于更新或插入数据，或者作为一个整体来比较。

列子子查询：子查询返回一列多行，通常用于 IN、ANY、ALL 等条件中，或者作为其他查询的条件。

表子子查询：子查询返回多行多列，通常不能用于查询条件，用于连接或联合查询中的虚拟表。

4 子查询语法(嵌套)

4.2 select中嵌套子查询

沿用数据:

部门表: 部门编号、部门名称、部门简介
岗位表: 岗位编号、岗位名称、岗位简介
员工表: 员工编号、姓名、薪资、奖金比例、生日、性别、电话、邮箱、住址、工作地、
入职日期、岗位编号、领导编号、部门编号

情景1: 标量子查询(单行单列)

```
# 1.1 查询研发部门的所有员工信息
#步骤1: 查询研发部门的did
SELECT did FROM t_department WHERE dname = '研发部';
#步骤2: 嵌套子查询,查询员工信息
SELECT * FROM t_employee WHERE did = (SELECT did FROM t_department WHERE dname = '研发部');

# 1.2 查询每个部门的拼接工资和公司的平均工资差
#步骤1: 查询公司的平均工资
SELECT AVG(salary) FROM t_employee;
#步骤2: 分组查询部门的平均工资
SELECT did,AVG(salary) FROM t_employee GROUP BY did;
#步骤3: 嵌套子查询,计算平均工资差
SELECT did,AVG(salary), AVG(salary)-(SELECT AVG(salary) FROM t_employee)FROM t_employee GROUP BY
did;
```

4 子查询语法(嵌套)

4.2 select中嵌套子查询

情景2: 行子子查询(单行多列)

```
# 1.1 查询和白露性别和部门相同信息的员工
#步骤1: 查询白露的性别和部门id (单行多列)
SELECT gender,did FROM t_employee WHERE ename = '白露';
#步骤2: 查询与之都相同的员工信息(in整体等于对比)
SELECT * FROM t_employee WHERE (gender,did) in (SELECT gender,did FROM t_employee WHERE ename
= '白露');
# 行子对比的时候,可以使用in关键字,注意列要和值一一对应! 等同于多个列等于+and关系
```

情景3: 列子子查询(多行单列)

```
# 1.1 查询和“白露”，“谢吉娜”同一部门的员工姓名和电话。
SELECT ename,tel,did FROM t_employee
WHERE did IN(SELECT did FROM t_employee WHERE ename='白露' || ename='谢吉娜');
SELECT ename,tel,did FROM t_employee
WHERE did =ANY(SELECT did FROM t_employee WHERE ename='白露' || ename='谢吉娜');

# 1.2 查询薪资比“白露”，“李诗雨”，“黄冰茹”三个人的薪资都要高的员工姓名和薪资。
SELECT ename,salary FROM t_employee
WHERE salary >ALL(SELECT salary FROM t_employee WHERE ename IN('白露','李诗雨','黄冰茹'));
```

多个值，那么需要用in,not in, >all,>any....形式做比较!

4 子查询语法(嵌套)

4.2 select中嵌套子查询

情景4: 表子子查询(多行多列)

```
# 1.1 查询所有部门的部门编号、部门名称、部门平均薪资
#步骤1: 查询部门的平均工资
SELECT did,AVG(salary) FROM t_employee GROUP BY did;
#步骤2: 查各部门的信息和平均工资
SELECT t_department.did ,dname,AVG(salary) FROM t_department LEFT JOIN (SELECT
did,AVG(salary) FROM t_employee GROUP BY did) temp ON t_department.did = temp.did;
#错误，from后面的t_department和temp表都没有salary字段，
#SELECT t_department.did ,dname,AVG(salary)出现AVG(salary)是错误的
```

多行多列一般用于做查询的临时虚拟表,注意虚拟表要起别名!

嵌套子查询小练习:

```
# 1.1 显示部门平均工资比全公司的总平均工资高的部门编号、部门名称、部门平均薪资，
#并按照部门平均薪资升序排列。
```

4 子查询语法(嵌套)

4.3 update中嵌套子查询

```
# 1.1 将“测试部”部门的员工薪资改为原来薪资的1.5倍。
UPDATE t_employee SET salary = salary * 1.5
WHERE did = (SELECT did FROM t_department WHERE dname = '测试部');

# 1.2 将没有部门的员工的部门改为“测试部”部门。
UPDATE t_employee  SET did = (SELECT did FROM t_department WHERE dname = '测试部
')
WHERE did IS NULL;

# 1.3 修改“t_employee”表中“李冰冰”的薪资值等于“孙红梅”的薪资值。
UPDATE t_employee SET salary = (SELECT salary FROM t_employee WHERE ename = '孙红
梅') WHERE ename = '李冰冰';
#You can't specify target table 't_employee' for update in FROM clause'
UPDATE t_employee SET salary = (SELECT salary FROM(SELECT salary FROM t_employee
WHERE ename = '孙红梅') temp) WHERE ename = '李冰冰';
#当update的表和子查询的表是同一个表时，需要将子查询的结果用临时表的方式表示
#即再套一层子查询，使得update和最外层的子查询不是同一张表
```

4 子查询语法(嵌套)

4.4 delete中嵌套子查询

```
# 1.1 将“测试部”部门的员工删除。
DELETE FROM t_employee
WHERE did = (SELECT did FROM t_department WHERE dname = '测试部');

# 1.2 从“t_employee”表中删除和“李冰冰”同一个部门的员工记录。。
DELETE FROM t_employee WHERE did = (SELECT did FROM t_employee WHERE ename = '
李冰冰');
#You can't specify target table 't_employee' for update in FROM clause'
#删除和子查询是同一张表

DELETE FROM t_employee WHERE did = (SELECT did FROM (SELECT did FROM t_employee
WHERE ename = '李冰冰') temp);
```

4 子查询语法(嵌套)

4.5 insert中嵌套子查询

```
#1.1 演示通过子查询复制表，
#（1）复制表结构
#（2）复制一条或多条记录
#（3）同时复制表结构和记录
#仅仅是复制表结构，可以用create语句
CREATE TABLE department LIKE t_department;

#1.2 使用INSERT语句+子查询，复制数据，此时INSERT不用写values
INSERT INTO department (SELECT * FROM t_department WHERE did<=3);

#1.3 同时复制表结构+数据
CREATE TABLE d_department AS (SELECT * FROM t_department);
#如果select后面是部分字段，复制的新表就只有这一部分字段
```

目录
Contents

5 多表查询实战练习

5 多表查询实战练习

5.1 多表结构和数据

student_id	name	age	gender
1	爱丽丝	20	女
2	鲍勃	22	男
3	查理	21	男
(NULL)	(NULL)	(NULL)	(NULL)

students

student_id	course_id	score
1	101	85
1	102	78
2	101	90
2	102	85
3	101	92
3	102	88
3	103	85

scores

course_id	course_name
101	数学
102	物理
103	化学

courses

建表和插入数据语句在07_数据库DQL(多表)脚本中!

5 多表查询实战练习

```
#1. 查询每位学生的姓名、年龄、所选课程的名称以及对应的成绩，如果学生没有选择课程，则成绩为 NULL。
SELECT s.name AS 姓名, s.age AS 年龄, c.course_name AS 课程名称, sc.score AS 成绩
    FROM students s
    LEFT JOIN scores sc ON s.student_id = sc.student_id
    LEFT JOIN courses c ON sc.course_id = c.course_id;

#2. 查询没有选择任何课程的学生姓名。
SELECT name
    FROM students
    WHERE student_id NOT IN (SELECT DISTINCT student_id FROM scores);

#3. 查询每门课程的平均成绩，并列出平均成绩高于所有学生平均成绩的课程。
SELECT course_id, AVG(score) AS 平均成绩
    FROM scores
    GROUP BY course_id
    HAVING AVG(score) > (SELECT AVG(score) FROM scores);

#4. 查询每位学生所选课程的平均成绩，并按照平均成绩降序排列。
SELECT s.name AS 姓名, AVG(sc.score) AS 平均成绩
    FROM students s
    LEFT JOIN scores sc ON s.student_id = sc.student_id
    GROUP BY s.name
    ORDER BY 平均成绩 DESC;
```

感谢观看