SQL基础查询，关联查询

SQL基础查询

基础查询语句

FROM

SQL查询语句的语法如下：

```
SELECT <*, column [alias], ......> FROM TABLE;
```

其中SELECT用于指定要查询的列，FROM指定要查询的表，如果要查询所有列，可以在SELECT后使用 *，如果只查询特定的列，可以直接在SELECT后指定列名，列名之间中逗号隔开，例子：

```
SELECT * FROM dept;
```

使用别名

在SQL语句中可以通过使用列的别名改变标题的显示样式，或者表示计算结果的含义，使用语法是列的别名跟在列名后面，中间可以加或者不加"AS"关键字，例子：

```
SELECT empno AS id, ename "NAME", sal*12 "Annual Salary" FROM emp;
```

别名可以直接写，不必用双引号引起来，但是如果希望别名中区分大小写字符，或者别名中包含字符或空格，那么就必须用双引号引起来

WHERE字句

在SELECT语句中，可以在WHERE字句中使用比较操作符，限制查询结果，是可选的，当查询条件和数据比较，可以使用单引号引起，也可以不用，当和字符以及日期类型的数据比较时，则必须用单引号引起来，例子：

```
SELECT * FROM emp WHERE deptno = 10;
SELECT ename, sal, job FROM emp WHERE job = 'SALESMAN'
```

SELECT字句

```
SELECT empno, ename, sal, job FROM emp;
```

查询条件

使用>，<，>=，<=，!=，<>，=

在WHERE字句中的查询条件，可以使用比较运算符来做查询，例子：

```
SELECT ename, sal FROM emp WHERE sal < 2000;
```

查询职员表中不属于部门10的员工信息（!= 等价于 <>）：

```
SELECT ename, sal, job FROM emp WHERE deptno != 10;
```

查询职员表中在2002年1月1号以后入职的职员信息，比较日期类型数据：

```
SELECT ename, sal, hiredate FROM emp WHERE hiredate > to_date('2002-1-1', 'YYYY-MM-DD');
```

使用AND，OR关键字

在SQL操作中，如果希望返回的结果必须满足多个条件，应该使用AND逻辑操作符连接这些条件，如果希望返回的结果满足多个条件之一即可，应该使用OR逻辑操作符连接这些条件，例子：

```
SELECT ename, sal, job FROM emp WHERE sal>1000 AND job='CLERK';
SELECT ename, sal, job FROM emp WHERE sal 1000 OR job='CLERK';
```

使用LIKE条件（模糊查询）

当用户在执行查询时，不能完全确定某些信息的查询条件，或者只知道信息的一部分，可以借助LIKE来实现模糊查询，LIKE需要借助两个通配符：

- %：表示0到多个字符
- _：标识单个字符

例子：

```
SELECT ename, job FROM emp WHERE ename LIKE '_A%';
```

使用IN，NOT IN

在WHERE字句中可以用比较操作符IN(list)来取出符合列表范围中的数据，其中的参数list表示值列表，当列或表达式匹配于列表中的任何一个值时，条件为TRUE，该条记录则被显示出来。

IN也可以理解为一个范围比较操作符，只不过这个范围是一个指定的值列表，NOT IN(list)取出不符合此列表中的数据记录，例子：

```
SELECT ename, job FROM emp WHERE jon IN ('MANAGER', 'CLERK');
SELECT ename, job, FROM emp WHERE deptno NOT IN (10,20);
```

BETWEEN ... AND ...

BETWEEN ... AND ... 操作符用来查询符合某个值域范围条件的数据，最常见的是使用在数字类型的数据范围上，但对字符类型和日期类型数据也同样适用，例子：

```
SELECT ename, sal FROM emp WHERE sal BETWEEN 1500 AND 3000;
```

使用IS NULL 和 IS NOT NULL

空值NULL是一个特殊的值，比较的时候不能使用 “=”号，必须使用IS NULL，否则不能得到正确的结果，例子：

```
SELECT ename, sal, comm FROM emp WHERE comm IS NULL;
```

使用ANY和ALL条件

在比较运算符中，可以出现ALL和ANY，表示 ‘全部’ 和 ‘任一’ ，但是ALL和ANY不能单独使用，需要配合单行比较操作符>，< .....其中：

- > ANY ： 大于最小

- < ANY：小于最大
- > ALL ：大于最大
- < ALL ：小于最小

例子：

```
SELECT empno, ename, job, sal, deptno
FROM emp
WHERE sal>ANY(
  SELECT sal FOROM emp WHERE job='SALESMAN');
```

查询条件中使用表达式和函数

当查询需要对选出的字段进行进一步计算，可以在数字列上使用算术表达式（+，-，*，/），表达式符合四则运算法则和默认优先级，如果要改变优先级可以使用括号。

算术运算主要针对数字类型的数据，对日期类型的数据可以做加减操作，表示在一个日期值上加或减一天，例子：

```
SELECT ename, sal, job FROM emp WHERE ename = UPPER('rose');
SELECT ename, sal, job FROM emp WHERE sal*12 > 10000;
```

使用DISTINCT过滤重复

数据表中有可能存储相同数据的行，当执行查询操作时，默认情况会显示所有行，不管查询结果是否有重复的数据，当重复的数据没有实际意义，经常会需要去掉重复值，使用DISTINCT实现，例子：

```
SELECT deptno FROM emp;
SELECT DISTINCT deptno FROM emp;
SELECT DISTINCT deptno, job FROM emp;  -- depeno，job联合起来不重复
```

排序

使用ORDER BY子句

对查询出的数据按一定对着进行排序操作，使用ORDER BY子句，例子：

```
SELECT <*, column [alias],.....>
FROM table
[WHERE condition(s)]
[ORDER BY column [ASC | DESC]];
```

注意：ORDER BY必须出现在SELECT 中最后一个子句，例子：

```
SELECT ename, sal
FROM emp
ORDER BY sal;
```

ASC和DESC

排序时默认按升序排列，ASC用来指定升序排序，DESC用来指定降序排序，因为NULL值视作最大，则升序排序时，排在最后，降序排序，排在最前，例子：

```
SELECT empno, ename, mgr FROM emp WHERE deptno = 10 ORDER BY mgr;
SELECT ename, sal FROM emp ORDER BY sal DESC;
```

多个列排序

当以多个列作为排序标准时，首先按照第一列进行排序，如果第一列数据相同，再以第二列排序，多个列排序时，不管正序还是倒序，每个列都需要单独设置排序方式。例子：

```
SELECT ename, deptno, sal FROM emp ORDER BY deptno ASC, sal DESC;
```

聚合函数

什么是聚合函数

查询时需要做一些数据统计，例如：查询职员表中各部门职员的平均薪水，各部门的员工人数，当需要统计的数据并不能再职员表里直观列出，而是需要根据现有的数据计算出结果，这种功能可以使用聚合函数来实现，即将表的全部数据划分为几组数据，每组数据统计出一个结果。

因为是多行数据参与运算返回一行结果，也成分组函数，多行函数，集合函数，用到的关键字：

- GOURP BY 按什么分组
- HAVING 进一步限制分组结果

MAX和MIN

用来取得列或表达式的最大，最小值，可以用来统计任何数据类型，包括数字，字符和日期

```
SELECT MAX(sal) max_sal, MIN(sal) min_sal FROM emp;
```

计算最早和最晚的入职时间，参数是日期：

```
SELECT MAX(hiredate) max_hire, MIN(hiredate) min_hire FROM emp;
```

AVG和SUM

AVG和SUM函数用来统计列或表达式的平均值和和值，这两个函数只能操作数字类型，并忽略NULL值，例子：

```
SELECT AVG(sal) avg_sal, SUM(sal) sum_sal FROM emp;
```

COUNT

COUNT函数用来计算表中的记录条数，同样忽略NULL值，例子：

```
SELECT COUNT(*) total_num FROM emp;
SELECT COUNT(job) total_job FROM emp;
```

聚合函数对空值的处理

聚合函数忽略NULL值，即当emp表中的某列有NULL值，例子：

```
SELECT AVG(sal) avg_sal FROM emp;
SELECT AVG(NVL(sal, 0)) avg_sal FROM emp;
```

分组

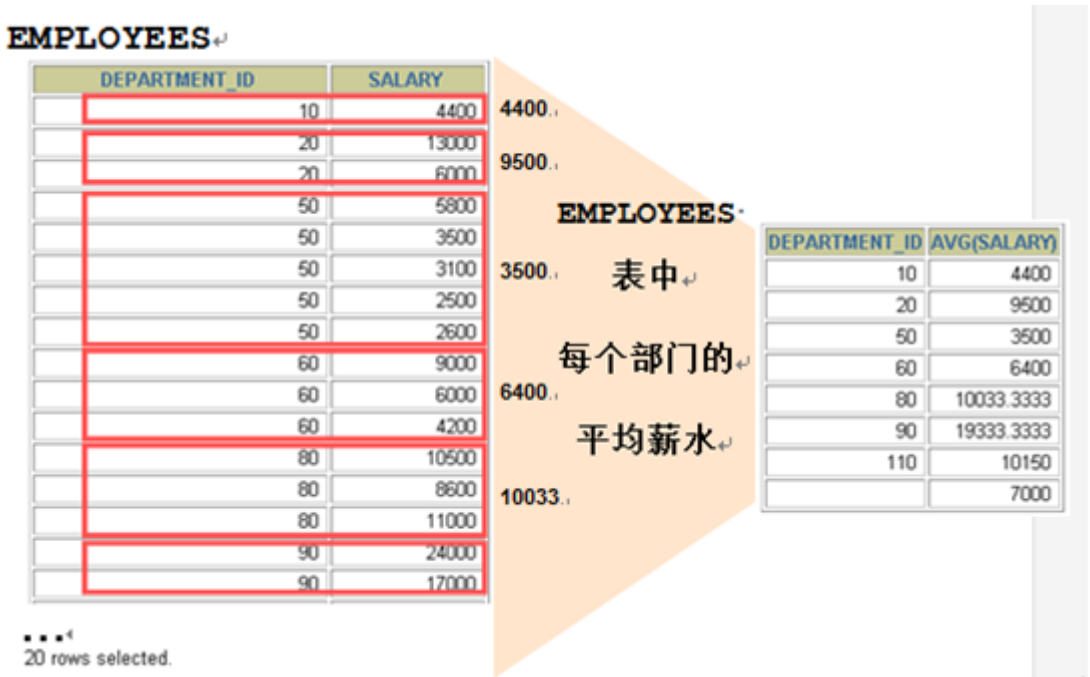
GROUP BY字句

上面的例子都是以整个表作为一组，如果希望得到每个部门的平均薪水，而不是整个机构的平均薪水，需要把整个数据表按部门划分成一个个小组，每个小组中包含一行或多行数据，在每个小组中再使用分组函数进行计算，每组返回一个结果，语法：

```
SELECT <*, column [alias], ......>
FROM table [WHERE condition(s)]
[GROUP BY group_by_expression]
[HIVING group_condition]
[ORDER BY column [ASC|DESC]];
```

其中划分的小组有多少，最终结果集行数就有多少

分组查询



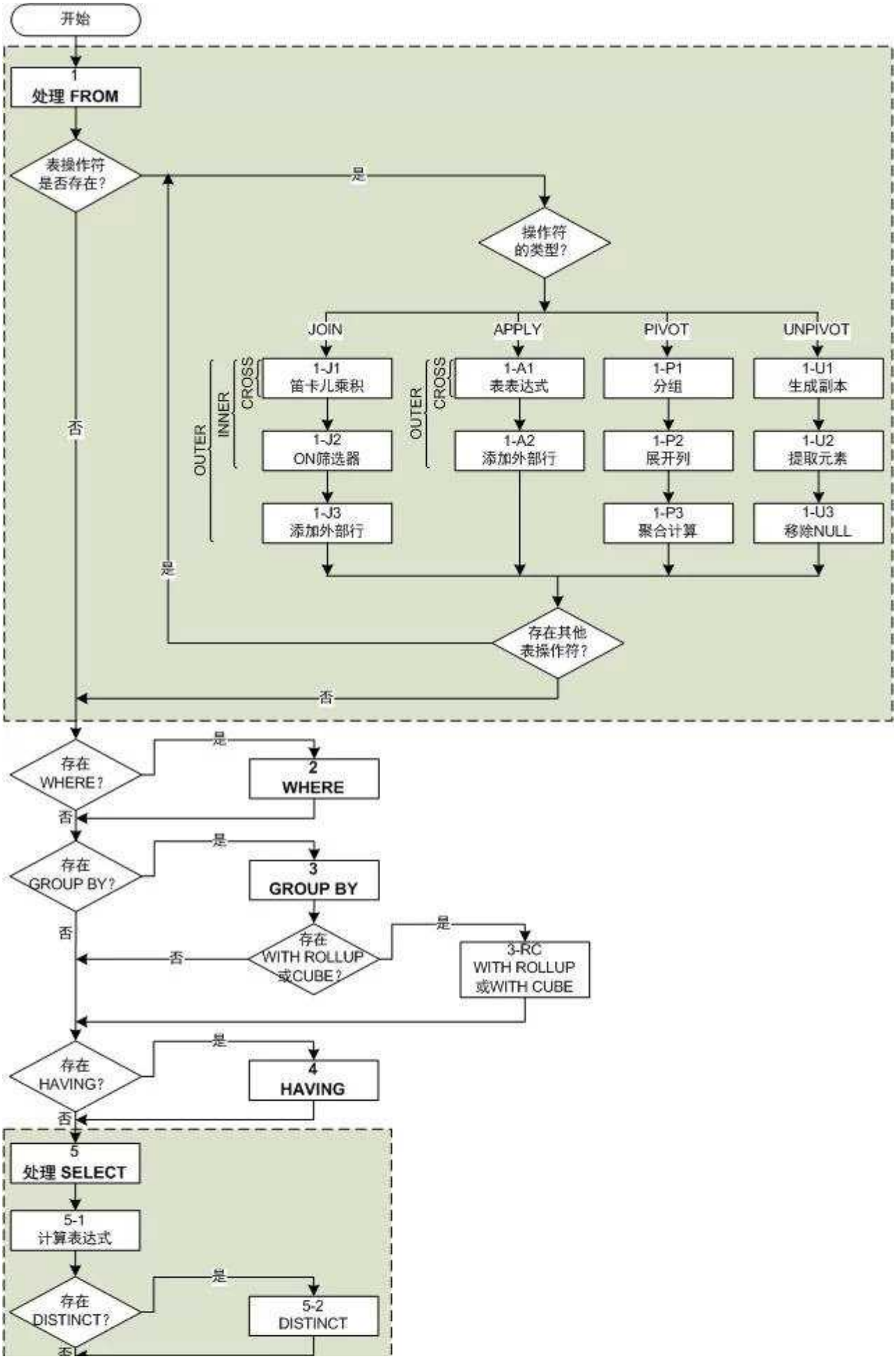
HAVING子句

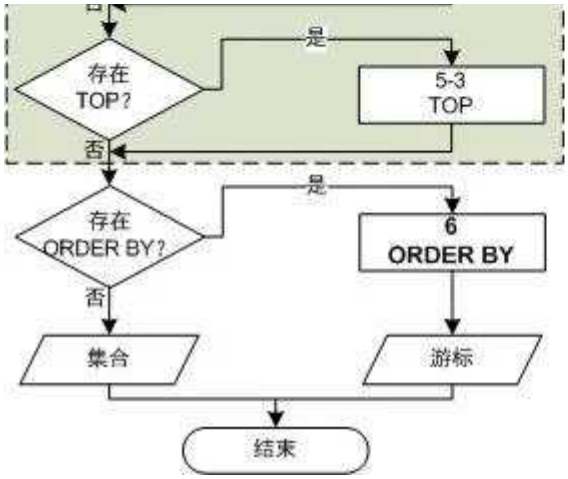
当一条查询语句中包含所有的子句，执行顺序如下：

- FROM子句：执行顺序为从后往前，从右到左，数据量较少的表尽量放在后面
- WHERE子句：执行顺序为自下而上从右到左，将能过滤掉最大数量记录的条件写在WHERE子句最右
- GROUP BY：执行顺序从左到右分组，最好在GROUP BY前使用WHERE，将不需要的记录在GROUP BY之前过滤掉
- HAVING子句：消耗资源。尽量避免使用，HAVING会在检测出所有记录之后才对结果集进行过滤，需要排序等操作
- SELECT子句：少用*号，尽量取字段名称，ORACLE在解析过程中，通过查询数据字典将*号一次转换成所有的列名，消耗时间

- ORDER BY子句：执行顺序为从左到右排序，消耗资源

SQL执行顺序图


开始
1
处理 FROM
表操作符
是否存在?
是
否
操作符
的类型?
JOIN
APPLY
PIVOT
UNPIVOT
OUTER
INNER
CROSS
1-J1
笛卡儿乘积
1-J2
ON筛选器
1-J3
添加外部行
OUTER
CROSS
1-A1
表表达式
1-A2
添加外部行
1-P1
分组
1-P2
展开列
1-P3
聚合计算
1-U1
生成副本
1-U2
提取元素
1-U3
移除NULL
存在其他
表操作符?
是
否
存在
WHERE?
是
2
WHERE
否
存在
GROUP BY?
是
3
GROUP BY
否
存在
WITH ROLLUP
或CUBE?
是
否
3-RC
WITH ROLLUP
或WITH CUBE
存在
HAVING?
是
4
HAVING
否
5
处理 SELECT
5-1
计算表达式
存在
DISTINCT?
是
5-2
DISTINCT
否



SQL关联查询

关联基础

关联的概念

实际应用中所需要的数据，经常会需要查询两个或两个以上的表，这种查询两个或两个以上数据表或者驶入的查询叫做连接查询，连接查询通常建立在存在相互关系的父子表之间，语法如下：

```
SELECT table.column table2.column
FROM table1, table2
WHERE table.column1 = table2.column2;
或
SELECT table1.column table2.column
FROM table1 JOIN table2
ON(table1.column1 = table2.column2);
```

笛卡尔积

笛卡尔积指做关联操作的每个表的每一行都和其他表的每一行做组合，假设两个表的记录条数分别是X和Y，笛卡尔积将返回X*Y条记录，当两个表关联查询时，不写连接条件，得到的记过就是笛卡尔积，例子：

```
SELECT COUNT(*) FROM emp; -- 14条记录
SELECT COUNT(*) FROM dept; -- 4条记录
SELECT emp.ename, dept.dname FROM emp, dept; -- 56条记录
```

等值连接

等值连接是连接查询中最常见的一种，通常是在有主外键关联关系的表间建立，并将连接条件设定为有关系的列，使用等号连接相关的表，例如查询职员的姓名，职位，以及所在部门的名字和所在的城市，使用两个相关的列做等值操作：例子：

```
SELECT e.ename, e.job, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

关联查询

内连接

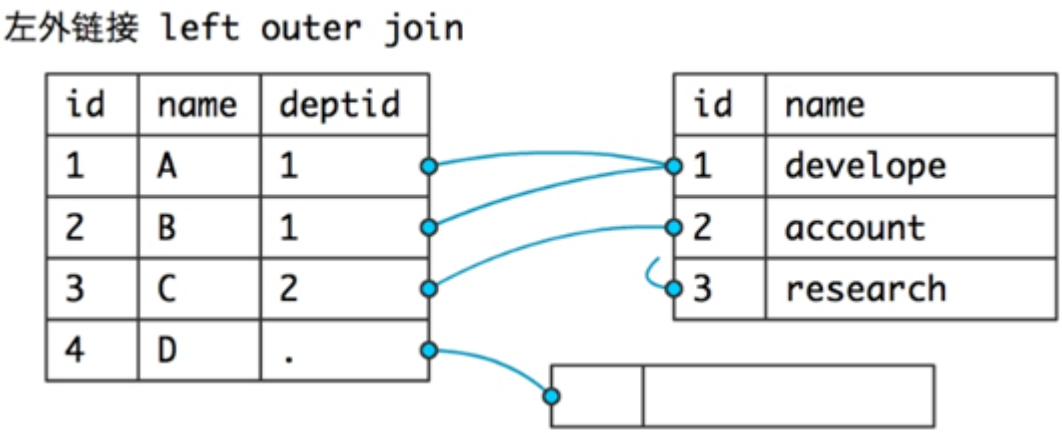
内连接返回两个关联表中所有满足连接条件的记录，例子：

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno = d.deptno
-- 或
SELECT e.ename, d.dname
FROM emp e JOIN dept d
ON(e.deptno = d.deptno);
```

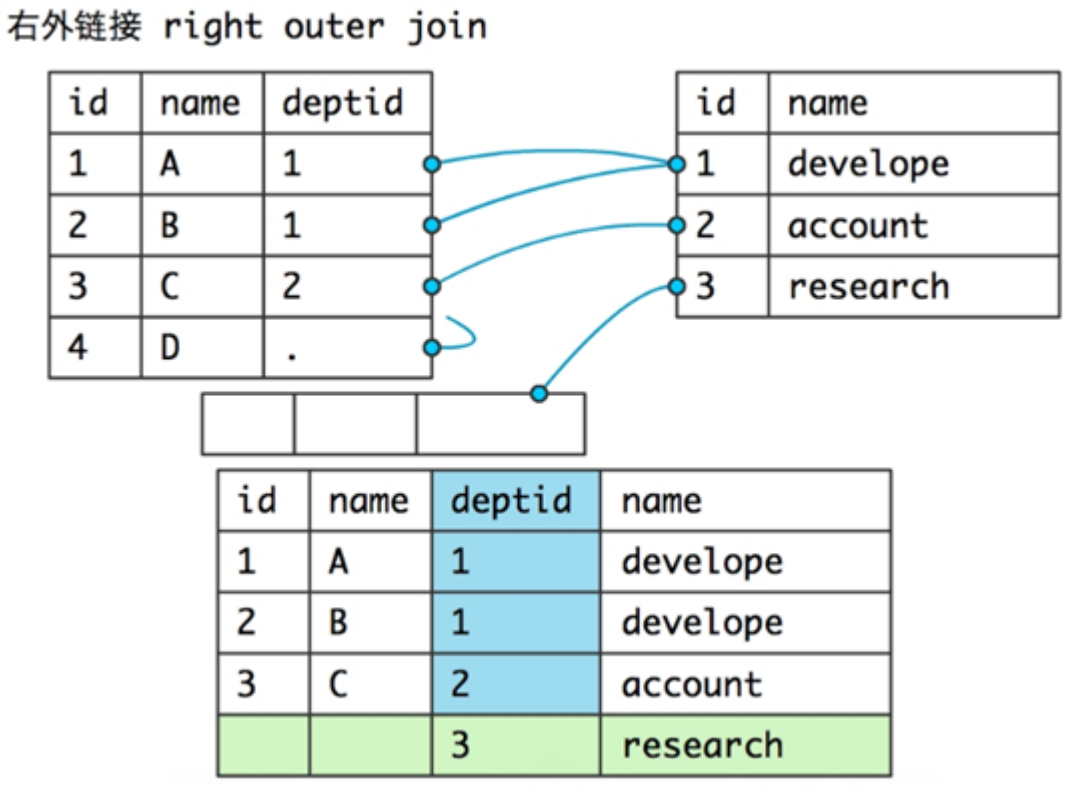
外连接

内连接返回两个表中所有满足连接条件的数据记录，在有些情况下，需要返回那些不满足连接条件的记录，需要使用外连接，即不仅返回满足连接条件的记录，还返回不满足连接条件的记录，比如把没有职员的部门和没有部门的职员查出来，例子：

```
SELECT table.column, table2.column
FROM table1 [LEFT | RIGHT | FULL] JOIN table2
ON table.column1 = table.column2
```



id	name	deptid	name
1	A	1	develope
2	B	1	develope
3	C	2	account
4	D		



外连接查询例子：

```
SELECT e.ename, d.dname
FROM emp e LEFT OUTER JOIN dept d
ON e.deptno = d.deptno
```

全连接

全外连接是指除了返回两个表中满足连接条件的记录，还会返回不满足连接条件的所有其他行，即是左外连接和右外连接查询的总和，例子：

```
SELECT e.ename, d.dname
FROM emp e FULL OUTER JOIN dept d
ON e.deptno = d.deptno;
```

全外链接 full outer join

id	name	deptid
1	A	1
2	B	1
3	C	2
4	D	.

id	name
1	develope
2	account
3	research

id	name	deptid	name
1	A	1	develope
2	B	1	develope
3	C	2	account
4	D		
		3	research

自连接

自连接值一种特殊的连接查询，数据的来源是一个表，即关联关系来自于单标中的多个列，表中的列参照同一个表中的其他列的情况乘坐自参照表。

自连接是通过将表用别名虚拟成两个表的方式实现，可以是等值或不等值连接，例如查询每个职员的经理名字，以及他们的职员编码：例子：

```
SELECT worker.empnow_empno, worker.enamew_ename, manager.empnom_empno, manager.enamem_ename
FROM emp worker JOIN emp manager
ON worker.mgr = manager.empno;
```