

# 数据的处理

# 目 标

通过本章学习，您将可以：

- 基本的SELECT语句
- 过滤和排序数据
- 分组函数
- 分组查询
- 多表查询
- 分页查询

# 1-基本 SELECT 语句

```
SELECT      * | { [DISTINCT] column | expression [alias] , ... }  
FROM        table;
```

- SELECT          标识选择哪些列。
- FROM            标识从哪个表中选择。

# 选择全部列

```
SELECT *  
FROM departments ;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 rows selected.

# 选择特定的列

```
SELECT department_id, location_id  
FROM departments;
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

## 注 意

- SQL 语言**大小写不敏感**。
- SQL 可以写在一行或者多行
- **关键字不能被缩写也不能分行**
- 各子句一般要分行写。
- 使用缩进提高语句的可读性。

# 列的别名

列的别名：

- 重命名一个列。
- 便于计算。
- 紧跟列名，也可以在列名和别名之间加入关键字 **'AS'**，别名使用**双引号**，以便在别名中包含空格或特殊的字符并区分大小写。

# 使用别名

```
SELECT last_name AS name, commission_pct comm  
FROM employees;
```

NAME	COMM
King	
Kochhar	
De Haan	

...

20 rows selected.

```
SELECT last_name "Name", salary*12 "Annual Salary"  
FROM employees;
```

Name	Annual Salary
King	288000
Kochhar	204000
De Haan	204000

...

20 rows selected.



# 字符串

- 字符串可以是 SELECT 列表中的一个字符,数字,日期。
- **日期和字符只能在单引号中出现。**
- 每当返回一行时,字符串被输出一次。

# 显示表结构

使用 DESCRIBE 命令，表示表结构

**DESC[RIBE]** *tablename*

**DESCRIBE** employees

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

## **2—过滤和排序数据**

# 过滤

- 使用WHERE 子句，将不满足条件的行过滤掉。

```
SELECT    * | { [DISTINCT] column | expression [alias], ... }  
FROM      table  
[WHERE    condition(s)];
```

- WHERE 子句紧随 FROM 子句。

# 在查询中过滤行

## EMPLOYEES

EMPLOYEE ID	LAST NAME	JOB ID	DEPARTMENT ID
100	King	AD PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
104	Ernst	IT_PROG	60
107	Lorentz	IT_PROG	60

20 rows selected.

返回在 **90**号部门工作的  
所有员工的信息

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

# WHERE 子句

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

# 比较运算

操作符	含义
=	等于 (不是 ==)
>	大于
>=	大于、等于
<	小于
<=	小于、等于
<>	不等于 (也可以是 !=)

赋值使用 := 符号

# 比较运算

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000;
```

LAST_NAME	SALARY
Matos	2600
Vargas	2500



# 其它比较运算

操作符	含义
<b>BETWEEN</b> <b>...AND...</b>	在两个值之间 ( <b>包含边界</b> )
<b>IN (set)</b>	等于值列表中的一个
<b>LIKE</b>	模糊查询
<b>IS NULL</b>	空值

# BETWEEN

使用 BETWEEN 运算来显示在一个区间内的值

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```

Lower limit

Upper limit

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

# IN

使用 IN运算显示列表中的值。

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201);
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

# LIKE

- 使用 LIKE 运算选择类似的值
- 选择条件可以包含字符或数字：
  - % 代表零个或多个字符(任意个字符)。
  - \_ 代表一个字符。

```
SELECT    first_name  
FROM      employees  
WHERE     first_name LIKE 'S%';
```

# LIKE

- ‘%’和 ‘-’ 可以同时使用。

```
SELECT last_name  
FROM employees  
WHERE last_name LIKE '_o%';
```

LAST_NAME
Kochhar
Lorentz
Mourgos

# NULL

使用 IS (NOT) NULL 判断空值。

```
SELECT last_name, manager_id  
FROM employees  
WHERE manager_id IS NULL;
```

LAST_NAME	MANAGER_ID
King	

# 逻辑运算

操作符	含义
AND	逻辑并
OR	逻辑或
NOT	逻辑否

# AND

AND 要求并的关系为真。

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >=10000
AND job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000



# OR

OR 要求或关系为真。

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
OR job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

8 rows selected.

# NOT

```
SELECT last_name, job_id
FROM employees
WHERE job_id
      NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

10 rows selected.

# ORDER BY子句

- 使用 ORDER BY 子句排序
  - **ASC** (ascend) : 升序
  - **DESC** (descend) : 降序
- **ORDER BY 子句在SELECT语句的结尾。**

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY  hire_date ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91

...

20 rows selected.

# 降序排序

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY  hire_date DESC ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Zlotkey	SA_MAN	80	29-JAN-00
Mourgos	ST_MAN	50	16-NOV-99
Grant	SA_REP		24-MAY-99
Lorentz	IT_PROG	60	07-FEB-99
Vargas	ST_CLERK	50	09-JUL-98
Taylor	SA_REP	80	24-MAR-98
Matos	ST_CLERK	50	15-MAR-98
Fay	MK_REP	20	17-AUG-97
Davies	ST_CLERK	50	29-JAN-97

...

20 rows selected.

# 按别名排序

```
SELECT employee_id, last_name, salary*12 annsal  
FROM employees  
ORDER BY annsal;
```

EMPLOYEE_ID	LAST_NAME	ANNSAL
144	Vargas	30000
143	Matos	31200
142	Davies	37200
141	Rajs	42000
107	Lorentz	50400
200	Whalen	52800
124	Mourgos	69600
104	Ernst	72000
202	Fay	72000
178	Grant	84000

...

20 rows selected.

# 多个列排序

- 按照ORDER BY 列表的顺序排序。

```
SELECT last_name, department_id, salary
FROM employees
ORDER BY department_id, salary DESC;
```

LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Mourgos	50	5800
Rajs	50	3500
Davies	50	3100
Matos	50	2600
Vargas	50	2500

...

20 rows selected.

- 可以使用不在SELECT 列表中的列排序。

## 3 — 分组函数

# 什么是分组函数

分组函数作用于一组数据，并对一组数据返回一个值。

## EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
	7000
10	4400

...

20 rows selected.

表 EMPLOYEES  
中的工资最大值

MAX(SALARY)
24000



# 组函数类型

- **AVG ( )**
- **COUNT ( )**
- **MAX ( )**
- **MIN ( )**
- **SUM ( )**

# 组函数语法

```
SELECT      [column,] group_function(column), ...  
FROM        table  
[WHERE      condition]  
[GROUP BY   column]  
[ORDER BY   column];
```

# AVG（平均值）和 SUM（合计）函数

可以对**数值型数据**使用AVG 和 SUM 函数。

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
FROM employees  
WHERE job_id LIKE '%REP%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

# MIN ( 最小值 ) 和 MAX ( 最大值 ) 函数

可以对任意数据类型的数据使用 MIN 和 MAX 函数。

```
SELECT MIN(hire_date), MAX(hire_date)  
FROM employees;
```

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

# COUNT ( 计数 ) 函数

COUNT (\*) 返回表中记录总数, 适用于任意数据类型。

```
SELECT COUNT(*)  
FROM employees  
WHERE department_id = 50;
```

COUNT(\*)

5

# COUNT ( 计数 ) 函数

- COUNT ( *expr* ) 返回 *expr* 不为空的记录总数。

```
SELECT COUNT (commission_pct)
FROM   employees
WHERE  department_id = 50;
```

COUNT(COMMISSION\_PCT)

# 分组数据

**EMPLOYEES**

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
80	10500
80	8600
80	11000
90	24000
90	17000

...

20 rows selected.

4400

9500

3500

6400

10033

求出  
**EMPLOYEES**  
表中各  
部门的  
平均工资

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

# 分组数据： GROUP BY 子句语法

可以使用GROUP BY子句将表中的数据分成若干组

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

明确：WHERE一定放在FROM后面



# GROUP BY 子句

在SELECT 列表中所有未包含在组函数中的列都应该包含在 GROUP BY 子句中。

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY  department_id ;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

8 rows selected.

# GROUP BY 子句

包含在GROUP BY 子句中的列不必包含在SELECT 列表中

```
SELECT    AVG(salary)
FROM      employees
GROUP BY department_id ;
```

AVG(SALARY)	
	4400
	9500
	3500
	6400
	10033.3333
	19333.3333
	10150
	7000

# 使用多个列分组

## EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
90	AD_PRES	24000
90	AD_VP	17000
90	AD_VP	17000
60	IT_PROG	9000
60	IT_PROG	6000
60	IT_PROG	4200
50	ST_MAN	5800
50	ST_CLERK	3500
50	ST_CLERK	3100
50	ST_CLERK	2600
50	ST_CLERK	2500
80	SA_MAN	10500
80	SA_REP	11000
80	SA_REP	8600

...

20	MK_REP	6000
110	AC_MGR	12000
110	AC_ACCOUNT	8300

20 rows selected.

使用多个列  
进行分组

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

# 在GROUP BY子句中包含多个列

```
SELECT    department_id dept_id, job_id, SUM(salary)
FROM      employees
GROUP BY  department_id, job_id ;
```

DEPT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

# 非法使用组函数

- 不能在 **WHERE** 子句中使用组函数。
- 可以在 **HAVING** 子句中使用组函数。

```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
```

```
WHERE     AVG(salary) > 8000
          *
```

**ERROR at line 3:**

**ORA-00934: group function is not allowed here**

**WHERE** 子句中不能使用组函数

# 过滤分组

## EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
...	
20	6000
110	12000
110	8300

20 rows selected.

部门最高工资  
比 10000 高的  
部门

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

# 过滤分组：HAVING 子句

使用 HAVING 过滤分组:

1. 行已经被分组。
2. 使用了组函数。
3. 满足HAVING 子句中条件的分组将被显示。

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column];
```

# HAVING 子句

```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY  department_id
HAVING    MAX(salary)>10000 ;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000



## 4 — 多表查询

## beauty表

id	name	sex	boyfriend_id
1	柳岩	女	(NULL)
2	苍老师	女	(NULL)
3	Angelababy	女	3
4	热巴	女	2
5	周冬雨	女	(NULL)
6	周芷若	女	1
7	岳灵珊	女	(NULL)
8	小昭	女	1
9	双儿	女	(NULL)
10	王语嫣	女	4
11	夏雪	女	(NULL)
12	赵敏	女	1

## boys表

id	boyName	userCP
1	张无忌	100
2	鹿晗	800
3	黄晓明	50
4	段誉	300

- 如果想查询女神名称和对应男神名称，肿么办？



## 多表连接

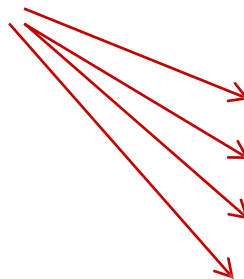
语法：select name,boyName from beauty,boys;

**beauty表**

id	name	sex	boyfriend_id
1	柳岩	女	8
2	苍老师	女	9
3	Angelababy	女	3
4	热巴	女	2
5	周冬雨	女	9
6	周芷若	女	1
7	岳灵珊	女	9
8	小昭	女	1
9	双儿	女	9
10	王语嫣	女	4
11	夏雪	女	9
12	赵敏	女	1

**boys表**

id	boyName	userCP
1	张无忌	100
2	鹿晗	800
3	黄晓明	50
4	段誉	300



笛卡尔集的错误情况：

select count(\*) from beauty;

假设输出12行

select count(\*)from boys;

假设输出4行

最终结果：12\*4=48行

# 笛卡尔集

- 笛卡尔集会在下面条件下产生：
  - 省略连接条件
  - 连接条件无效
  - 所有表中的所有行互相连接
- 为了避免笛卡尔集， 可以在 WHERE 加入有效的连接条件。

# Mysql 连接

使用连接在多个表中查询数据。

```
SELECT    table1.column, table2.column  
FROM      table1, table2  
WHERE     table1.column1 = table2.column2;
```

- 在 **WHERE** 子句中写入连接条件。
- 在表中有相同列时，在列名之前加上表名前缀

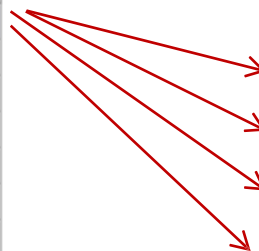
# 多表连接

beauty表

id	name	sex	boyfriend_id
1	柳岩	女	8
2	苍老师	女	9
3	Angelababy	女	3
4	热巴	女	2
5	周冬雨	女	9
6	周芷若	女	1
7	岳灵珊	女	9
8	小昭	女	1
9	双儿	女	9
10	王语嫣	女	4
11	夏雪	女	9
12	赵敏	女	1

boys表

id	boyName	userCP
1	张无忌	100
2	鹿晗	800
3	黄晓明	50
4	段誉	300



id	name	boyname
3	Angelababy	黄晓明
4	热巴	鹿晗
6	周芷若	张无忌
8	小昭	张无忌
10	王语嫣	段誉
12	赵敏	张无忌

# 等值连接

```
SELECT beauty.id,NAME,boyname FROM beauty ,boys  
WHERE beauty.`boyfriend_id`=boys.id;
```

# 区分重复的列名

- 使用表名前缀在多个表中区分相同的列。
- 在不同表中具有相同列名的列可以用表的别名加以区分。
- 如果使用了表别名，则在select语句中需要使用表别名代替表名
- 表别名最多支持32个字符长度，但建议越少越好



# 表的别名

- 使用别名可以简化查询。
- 使用表名前缀可以提高执行效率。

```
SELECT bt.id,NAME,boyname  
FROM beauty bt,boys b;  
WHERE bt.`boyfriend_id`=b.id ;
```

# 连接多个表

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Zlotkey	80
Abel	80
Taylor	80

...  
20 rows selected.

DEPARTMENTS

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

- 连接  $n$  个表, 至少需要  $n-1$  个连接条件。 例如: 连接三个表, 至少需要两个连接条件。

练习: 查询出公司员工的 last\_name, department\_name, city

# SQL99：使用ON子句创建连接

- 自然连接中是以具有相同名字的列为连接条件的。
- 可以使用 **ON子句指定额外的连接条件**。
- 这个连接条件是与其它条件分开的。
- **ON子句使语句具有更高的易读性**。

# Join连接

- 分类：
  - 内连接 `[inner] join on`
  - 外连接
    - 左外连接 `left [outer] join on`
    - 右外连接 `right [outer] join on`

# ON 子句

```
SELECT bt.id,NAME,boyname  
FROM beauty bt  
Inner join boys b  
On bt.`boyfriend id`=b.id ;
```

id	name	boyname
3	Angelababy	黄晓明
4	热巴	鹿晗
6	周芷若	张无忌
8	小昭	张无忌
10	王语嫣	段誉
12	赵敏	张无忌

# 连接多个表

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Zlotkey	80
Abel	80
Taylor	80

...  
20 rows selected.

DEPARTMENTS

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

- 连接  $n$  个表, 至少需要  $n-1$  个连接条件。 例如: 连接三个表, 至少需要两个连接条件。

练习: 查询出公司员工的 last\_name, department\_name, city

# 使用 ON 子句创建多表连接

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

# 外连接

beauty表

id	name	sex	borndate	phone	photo	▲ boyfriend_id
6	周芷若	女	1988-02-03 00:00:00	18209876577	( <input type="checkbox"/> OK	1
8	小昭	女	1989-02-03 00:00:00	18209876567	( <input type="checkbox"/> OK	1
12	赵敏	女	1992-02-03 00:00:00	18209179577	( <input type="checkbox"/> OK	1
4	热巴	女	1993-02-03 00:00:00	18209876579	( <input type="checkbox"/> OK	2
3	Angelababy	女	1989-02-03 00:00:00	18209876567	( <input type="checkbox"/> OK	3
1	柳岩	女	1988-02-03 00:00:00	18209876577	( <input type="checkbox"/> OK	8
2	苍老师	女	1987-12-30 00:00:00	18219876577	( <input type="checkbox"/> OK	9
9	双儿	女	1993-02-03 00:00:00	18209876579	( <input type="checkbox"/> OK	9
11	夏雪	女	1993-02-03 00:00:00	18209876579	( <input type="checkbox"/> OK	9
7	岳灵珊	女	1987-12-30 00:00:00	18219876577	( <input type="checkbox"/> OK	9
5	周冬雨	女	1992-02-03 00:00:00	18209179577	( <input type="checkbox"/> OK	9

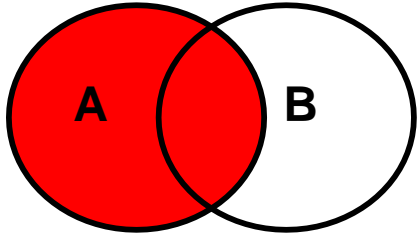
boys表

id	boyName	userC
1	张无忌	
2	鹿晗	
3	黄晓明	
4	段誉	

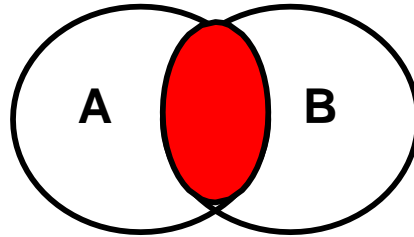
id	name	boyname
6	周芷若	张无忌
8	小昭	张无忌
12	赵敏	张无忌
4	热巴	鹿晗
3	Angelababy	黄晓明
10	王语嫣	段誉
1	柳岩	(NULL)
2	苍老师	(NULL)
5	周冬雨	(NULL)
7	岳灵珊	(NULL)
9	双儿	(NULL)
11	夏雪	(NULL)



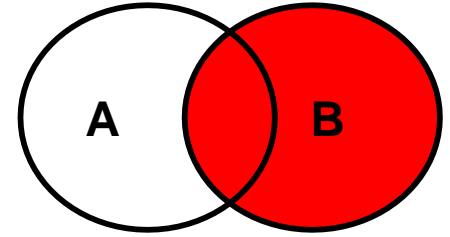
# John连接总结



```
SELECT <select_list>  
FROM A  
LEFT JOIN B  
ON A.key=B.key
```

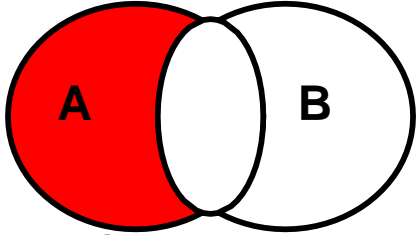


```
SELECT <select_list>  
FROM A  
INNER JOIN B  
ON A.key=B.key
```

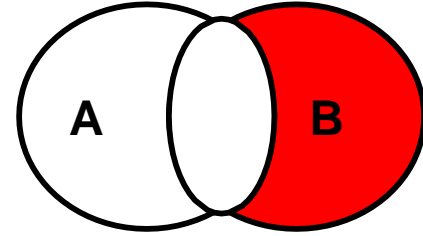


```
SELECT <select_list>  
FROM A  
RIGHT JOIN B  
ON A.key=B.key
```

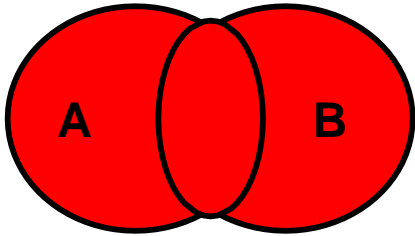
# John连接总结



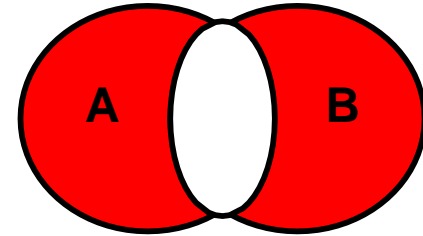
SELECT <select\_list>  
FROM A  
LEFT JOIN B  
ON A.key=B.key  
WHERE B.key is null;



SELECT <select\_list>  
FROM A  
RIGHT JOIN B  
ON A.key=B.key  
WHERE A.key is null;



SELECT <select\_list>  
FROM A  
FULL JOIN B  
ON A.key=B.key



SELECT <select\_list>  
FROM A  
FULL JOIN B  
ON A.key=B.key  
WHERE A.key is null  
OR B.key is null;