SQL开发

讲师： 宋红康

微博：尚硅谷-宋红康

邮箱：shkstart@126.com

第1章：SQL概述

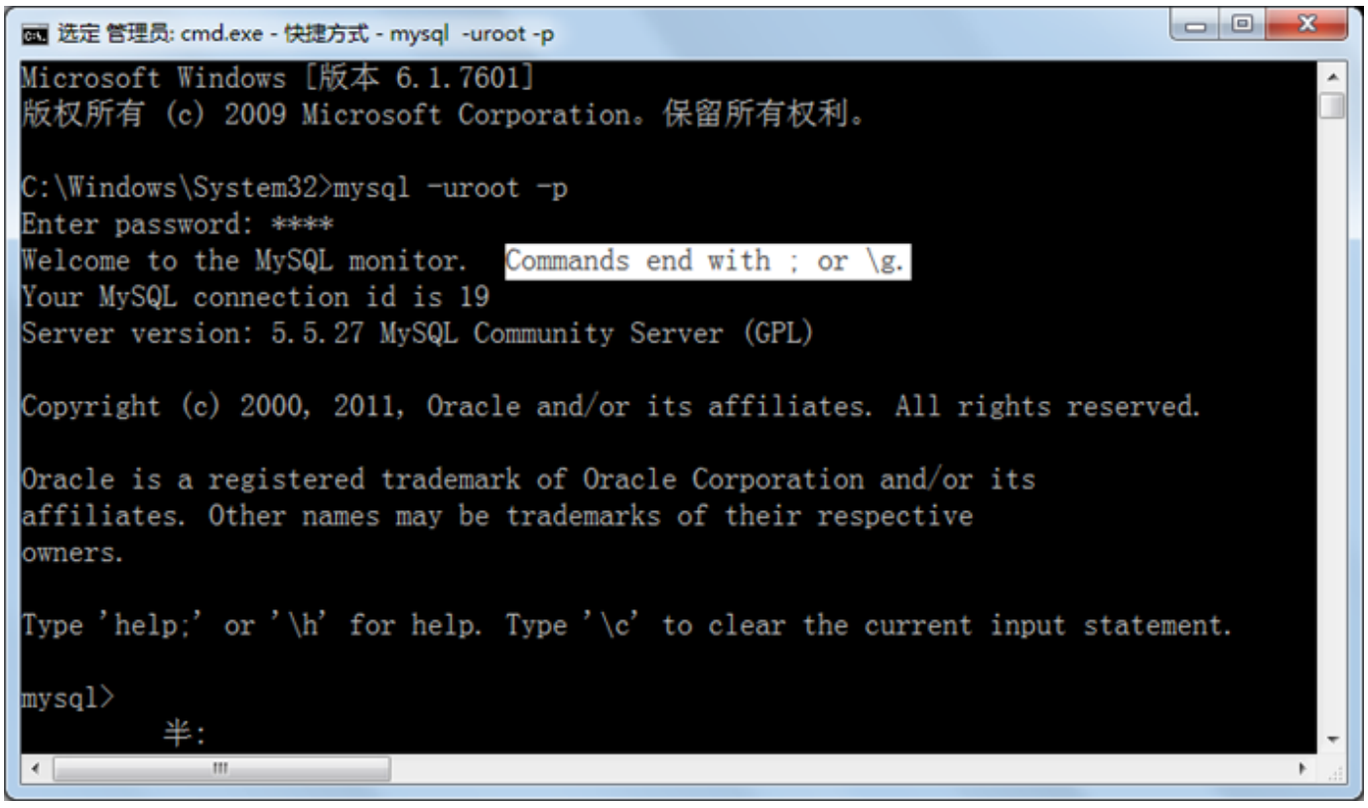
SQL：Structured Query Language结构化查询语言，它是使用关系模型的数据库应用语言，由IBM上世纪70年代开发出来。后由美国国家标准局（ANSI）开始着手制定SQL标准，先后有SQL-86，SQL-89，SQL-92，SQL-99等标准。

1. SQL的语言规范

- mysql对于SQL语句不区分大小写，SQL语句关键字尽量大写
- SQL 可以写在一行或者多行。为了提高可读性，各子句分行写，必要时使用缩进
- 关键字不能被缩写也不能分行
- 值，除了数值型，**字符串型和日期时间类型使用单引号（' '）**
- 别名，尽量使用双引号（" "），而且不建议省略as
- 所有标点符号使用英文状态下的半角输入方式
- 必须保证所有(),单引号，双引号是成对结束的
- 可以使用（1）#单行注释 （2）--空格单行注释 （3）/* 多行注释 */

2. 命名规则

- 数据库、表名不得超过30个字符，变量名限制为29个
- 必须只能包含 A–Z, a–z, 0–9, _共63个字符
- 不能在对象名的字符间留空格
- 必须不能和用户定义的其他对象重名
- 必须保证你的字段没有和保留字、数据库系统或常用方法冲突
- 保持字段名和类型的一致性,在命名字段并为其指定数据类型的时候一定要保证一致性。假如数据类型在一个表里是整数,那在另一个表里可就别变成字符型了
- 在命令行中的要求：



> 说明：一个语句可以分开多行编写，以;或\g结束

3. SQL分类

- **DDL (Data Definition Languages)：数据定义语言**，这些语句定义了不同的数据段、数据库、表、列、索引等数据库对象。
  - 主要的语句关键字包括`CREATE`、`DROP`、`ALTER`等。
- **DML (Data Manipulation Language) ：数据操作语言**，用于添加、删除、更新和查询数据库记录，并检查数据完整性。
  - 主要的语句关键字包括`INSERT`、`DELETE`、`UPDATE`、`SELECT`等。
  - **SELECT是SQL语言的基础，最为重要。**
- **DCL (Data Control Language) ：数据控制语言**，用于控制不同数据段直接的许可和访问级别的语句。这些语句定义了数据库、表、字段、用户的访问权限和安全级别。
  - 主要的语句关键字包括`GRANT`、`REVOKE`、`COMMIT`、`ROLLBACK`、`SAVEPOINT`等。

第2章：数据处理之查询

2.1 基本的SELECT语句

2.1.1 SELECT ... FROM

```
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.

2.1.2 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.

2.1.3 去除重复行

默认情况下，查询会返回全部行，包括重复行。

```
SELECT department_id
FROM   employees;
```

DEPARTMENT_ID
90
90
90
60
60
60
50
50
50

20 rows selected.

在SELECT语句中使用关键字DISTINCT去除重复行。

```
SELECT DISTINCT department_id
FROM   employees;
```

DEPARTMENT_ID
10
20
50
60
80
90
110

8 rows selected.

2.1.4 空值参与运算

- 所有运算符或列值遇到null值，运算的结果都为null

```
SELECT employee_id,salary,commission_pct,
12 * salary * (1 + commission_pct) "annual_sal"
FROM employees;
```

2.1.5 显示表结构

使用DESCRIBE 或 DESC 命令，表示表结构。

```
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)

2.2 过滤数据

2.2.1 SELECT ... FROM ... WHERE

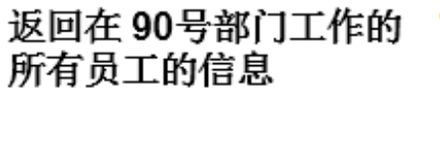
背景：

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
124	Mourgos	ST_MAN	50

...

20 rows selected.



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

- 使用WHERE 子句，将不满足条件的行过滤掉
- **WHERE子句紧随 FROM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

2.2.2 比较运算符

① 基本操作符

操作符	含义
=	等于(不是==)
>	大于
>=	大于、等于
<	小于
<=	小于、等于
<> 或 !=	不等于

> 说明：赋值符号使用 :=

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

LAST_NAME	SALARY
Matos	2600
Vargas	2500

② 其它比较运算符

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

1) BETWEEN ... AND

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;
```

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

2) 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.

3) 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%';
```

- '%'和'_'可以同时使用。

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

LAST_NAME
Kochhar
Lorentz
Mourgos

ESCAPE

- 回避特殊符号的：**使用转义符。**例如：将[%]转为[$%]、[]转为[$]，然后再加上[ESCAPE'$']即可。

```
SELECT job_id
FROM   jobs
WHERE  job_id LIKE 'IT\_%';
```

如果使用\表示转义，要省略ESCAPE。如果不是\，则要加上ESCAPE。

```
SELECT job_id
FROM   jobs
WHERE  job_id LIKE 'IT$_%' escape '$';
```

4) 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	

2.2.3 逻辑运算符

操作符	含义
&&（或AND）	逻辑且
||（或OR）	逻辑或
NOT	逻辑否
XOR	逻辑异或

1）&&（或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

2）||（或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.

3) 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.

4) XOR

```
select last_name,department_id,salary
from employees
where department_id in (10,20) XOR salary > 8000;
```

2.2.4 算术运算符

运算符	说明
+	加法
-	减法
*	乘法
/ (或div)	除法
% (或mod)	取模

```
SELECT employee_id,salary,department_id
FROM employees
WHERE department_id MOD 2 = 0;
```

2.3 排序数据和分页

2.3.1 排序数据

- 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.

```
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.

```
1  SELECT employee_id, last_name, salary*12 annsal
2  FROM   employees
3  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.

```
1  SELECT last_name, department_id, salary
2  FROM   employees
3  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。

2.3.2 分页

- **MySQL中使用limit实现分页**
- 背景：查询返回的记录太多了，查看起来很不方便，怎么样能够实现分页查询呢？
- 分页原理

  所谓分页显示，就是将数据库中的结果集，一段一段显示出来需要的条件。

```
--前10条记录：
SELECT * FROM table LIMIT 0,10;

--第11至20条记录：
SELECT * FROM table LIMIT 10,10;

--第21至30条记录：
SELECT * FROM table LIMIT 20,10;
```

- 公式：**（当前页数-1）*每页条数，每页条数**

```
SELECT * FROM table
LIMIT(PageNo - 1)*PageSize,PageSize;
```

- **注意：limit子句必须放在整个查询语句的最后！**

2.4 多表查询

2.4.1 笛卡尔积错误

举例：

EMPLOYEES表

employee_id
first_name
last_name
email
phone_number
job_id
salary
commission_pct
manager_id
department_id

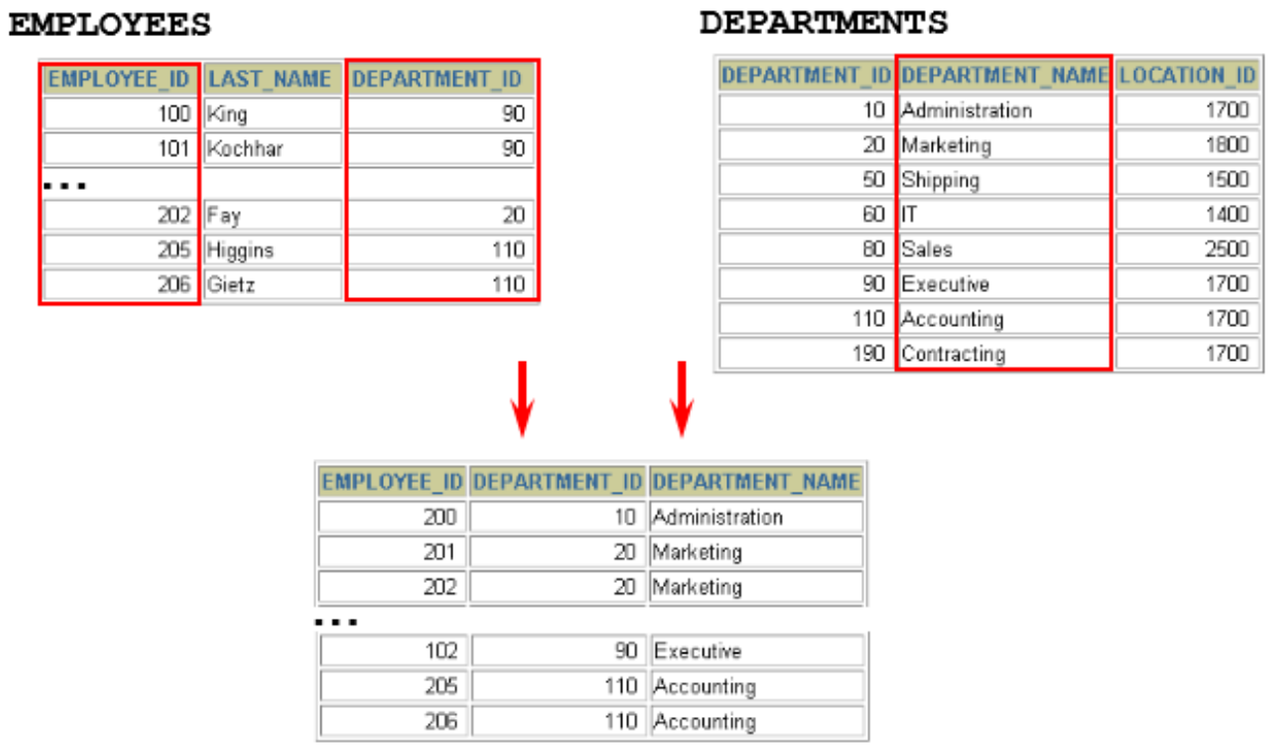
DEPARTMENTS表

department_id
department_name
manager_id
location_id

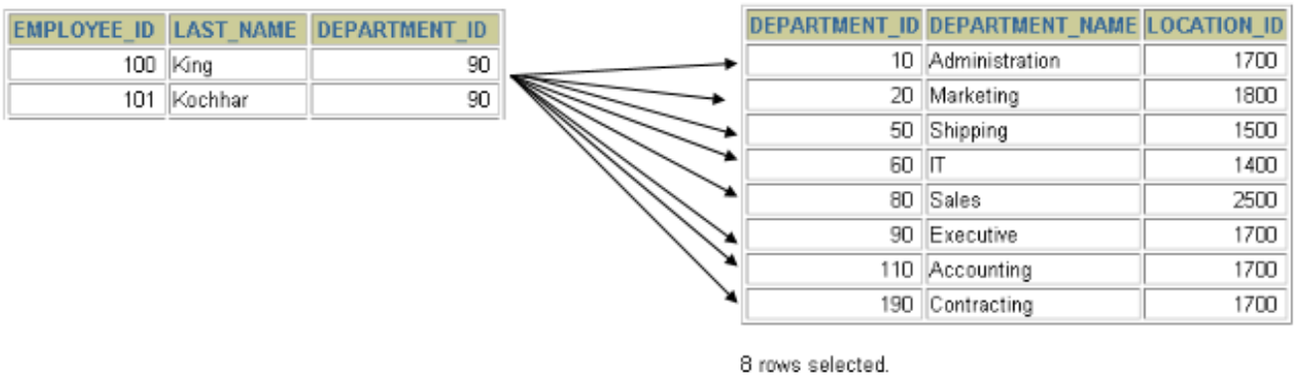
LOCATIONS表

location_id
street_address
postal_code
city
state_province
country_id

从多个表中获取数据：




```sql
select last_name, department_name
from employees, departments;
```



演示笛卡尔积的错误情况：

```sql
select count(employee_id) from employees;
输出107行
select count(department_id)from departments;
输出27行
select 107*27 from dual;
```

- **笛卡尔积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
- 为了避免笛卡尔积， 可以**在 WHERE 加入有效的连接条件。**

MySQL连接

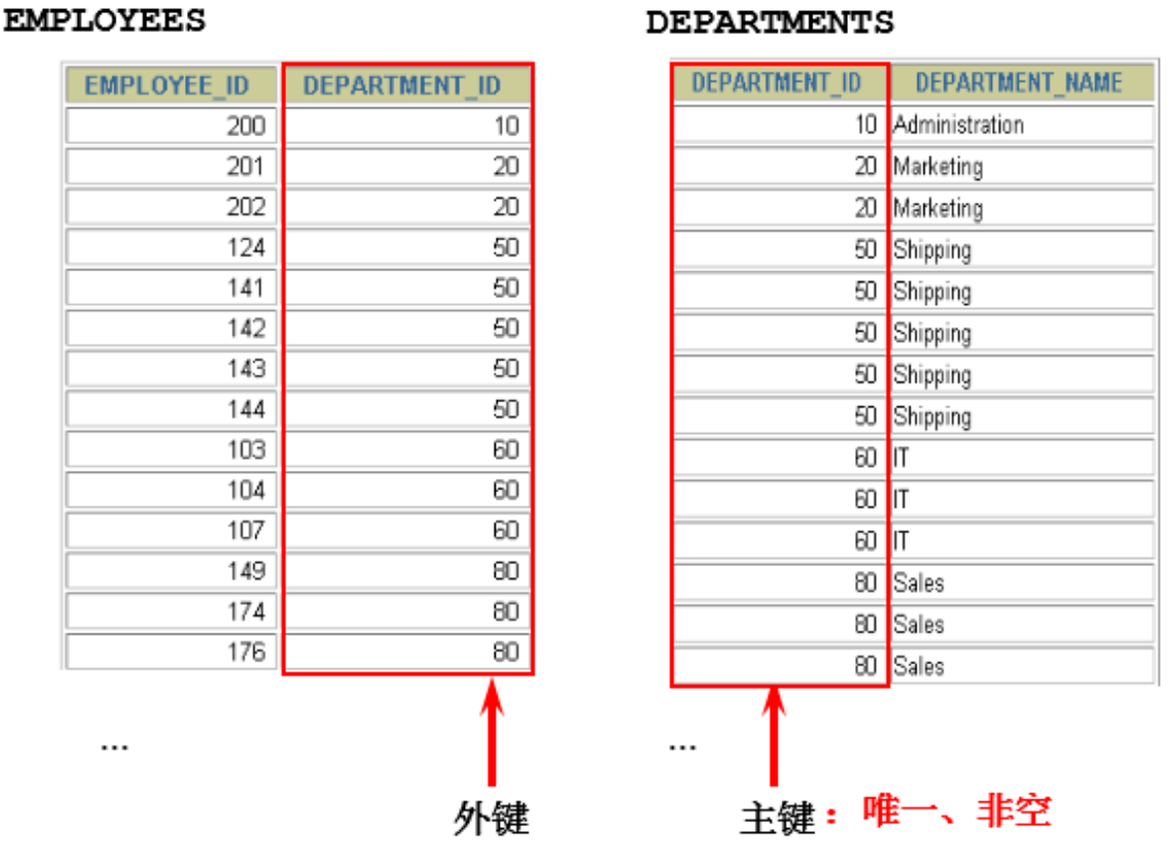
使用连接在多个表中查询数据。

```
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。**

2.4.2 分类1：等值连接 vs 非等值连接

① 等值连接




```
SELECT employees.employee_id, employees.last_name,
       employees.department_id, departments.department_id,
       departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
144	Vargas	50	50	1500

19 rows selected.

多个连接条件与 AND 操作符

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
Whalen	10
Hartstein	20
Fay	20
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Hunold	60
Ernst	60

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT

...

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。

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 e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e , departments d
WHERE  e.department_id = d.department_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

② 非等值连接

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

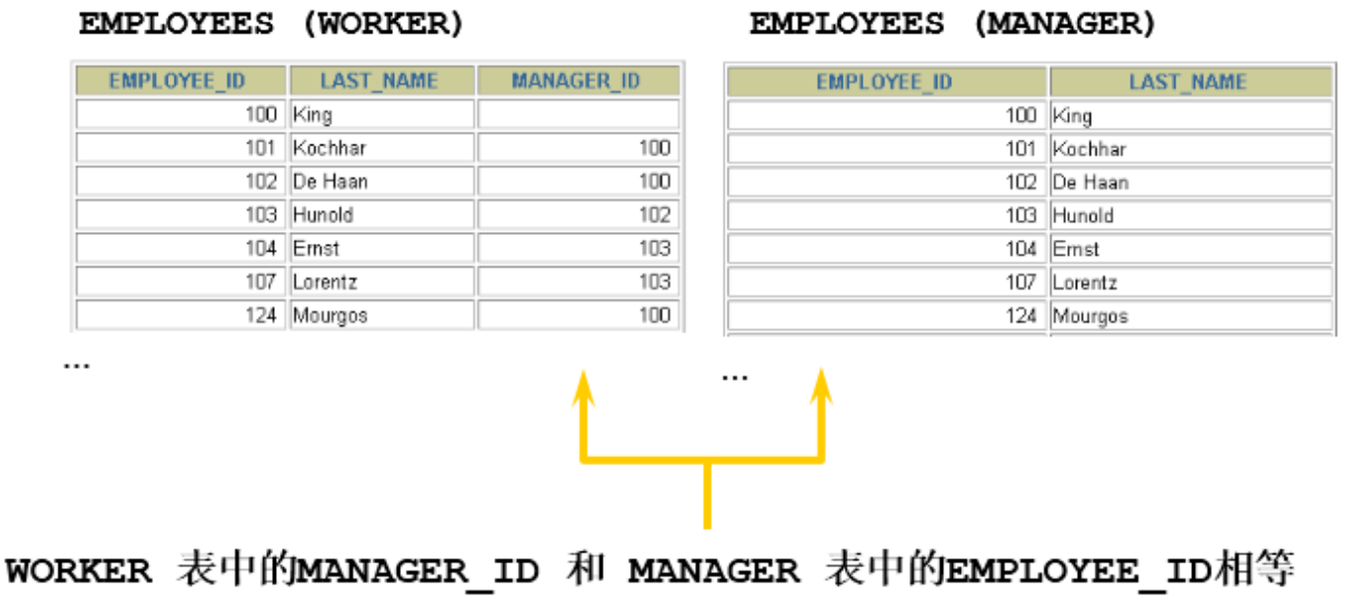
EMPLOYEES表中的列工资
应在JOB_GRADES表中的最高
工资与最低工资之间

```sql
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

20 rows selected.

2.4.3 分类2：自连接 vs 非自连接



- 当table1和table2本质上是同一张表，只是用取别名的方式虚拟成两张表以代表不同的意义。然后两个表再进行内连接，外连接等查询

题目：查询employees表，返回“Xxx works for Xxx”

```
SELECT CONCAT(worker.last_name ,' works for '
       , manager.last_name)
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id ;
```

WORKER.LAST_NAME||'WORKSFOR'||MANAGER.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Hunold works for De Haan
Ernst works for Hunold

19 rows selected.

练习：查询出last_name为 'Chen' 的员工的 manager 的信息

2.4.4 分类3：内连接 vs 外连接

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

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

...

20 rows selected.

190号部门没有员工

- 内连接: 合并具有同一列的两个以上的表的行, **结果集中不包含一个表与另一个表不匹配的行**
- 外连接: 两个表在连接过程中除了返回满足连接条件的行以外**还返回左（或右）表中不满足条件的行 ，这种连接称为左（或右） 外连接**。没有匹配的行时, 结果表中相应的列为空(NULL)。

① 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。**

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

19 rows selected.

```
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;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

19 rows selected.

- **USING的使用** (了解)

只能和JOIN一起使用，而且要求**两个**关联字段在关联表中名称一致，而且只能表示关联字段值相等

```
SELECT employee_id,department_name
FROM employees e JOIN departments d
USING (department_id);
```

② 左外连接

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
LEFT OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

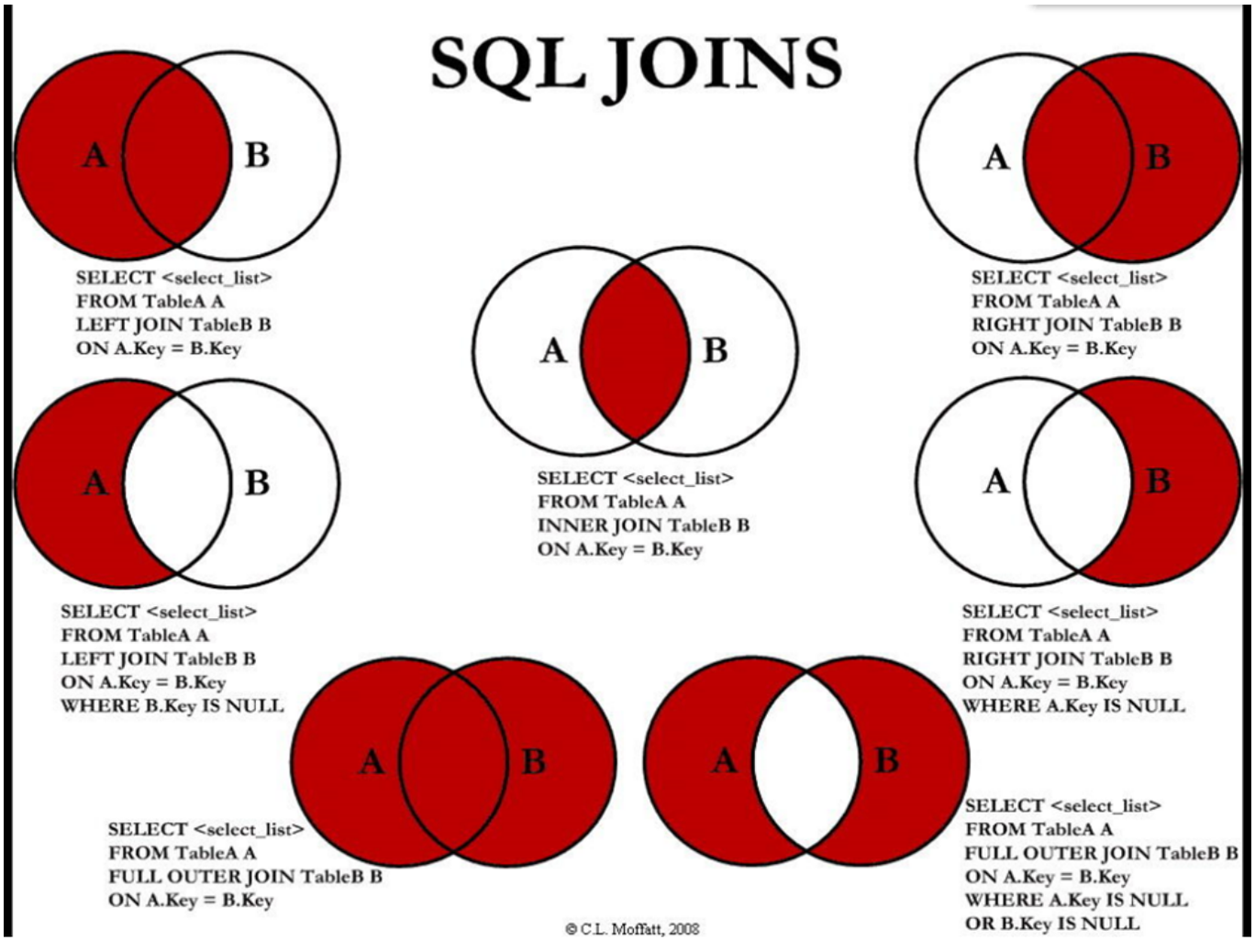
③ 右外连接

```sql
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
...		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Gietz	110	Accounting
		Contracting

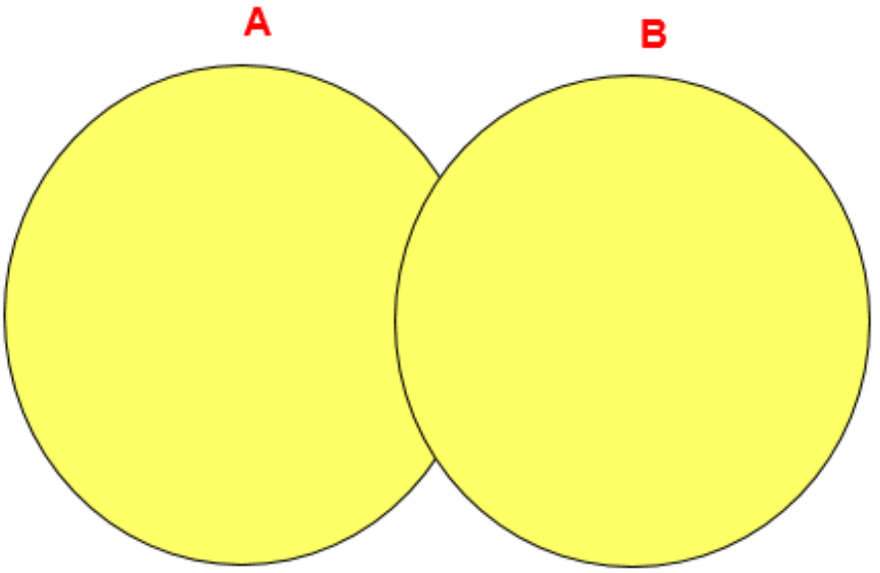
20 rows selected.

总结：SQL JOINS



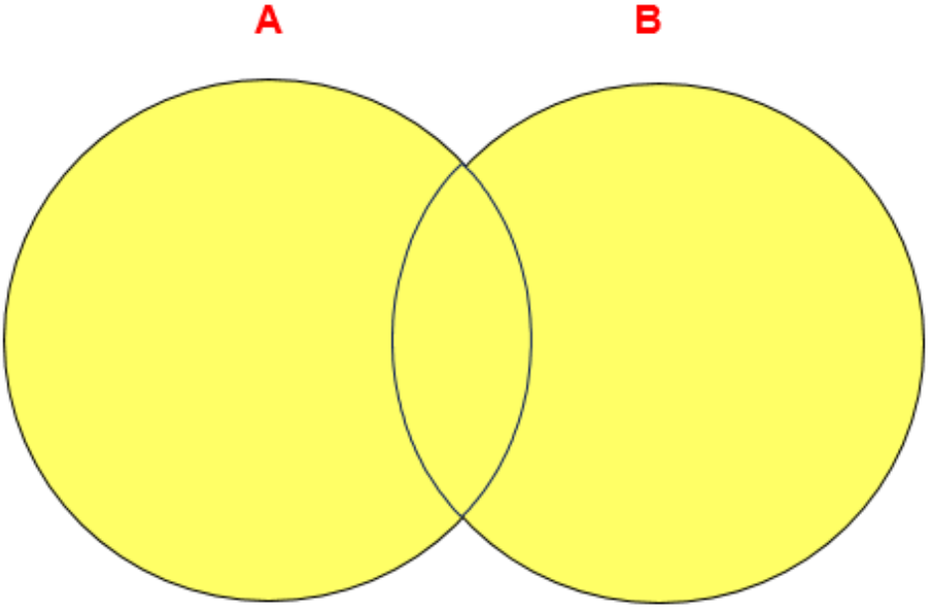
补充：

UNION操作符



UNION 操作符返回两个查询的结果集的并集。

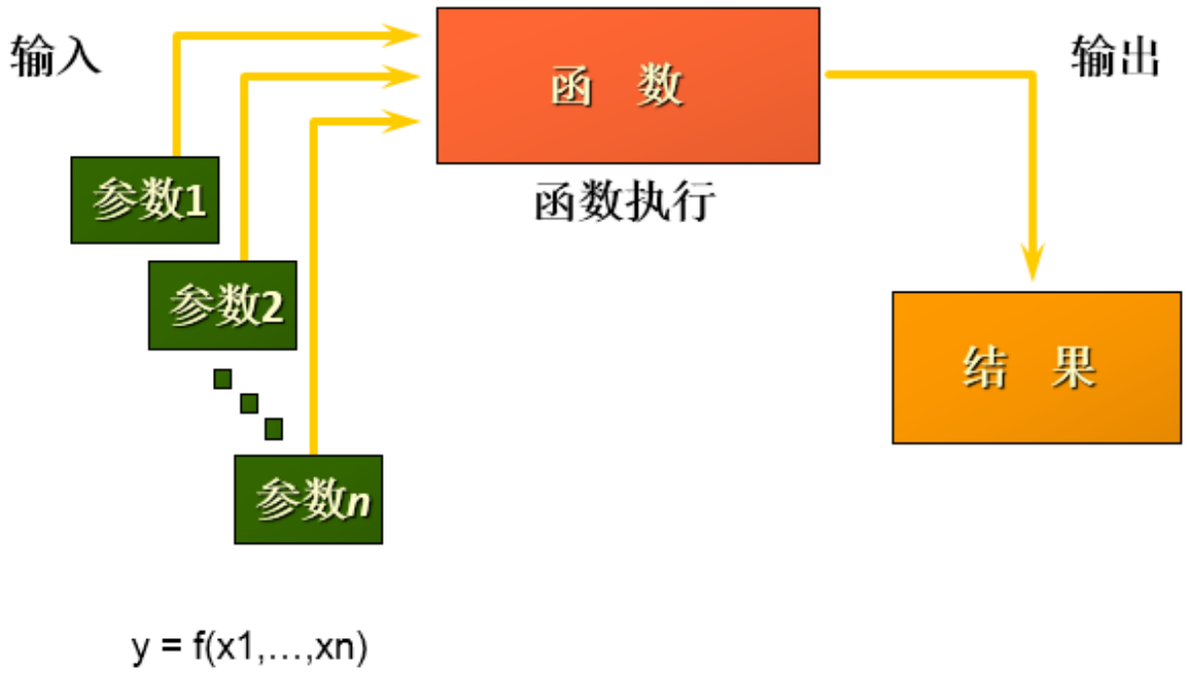
UNION ALL操作符



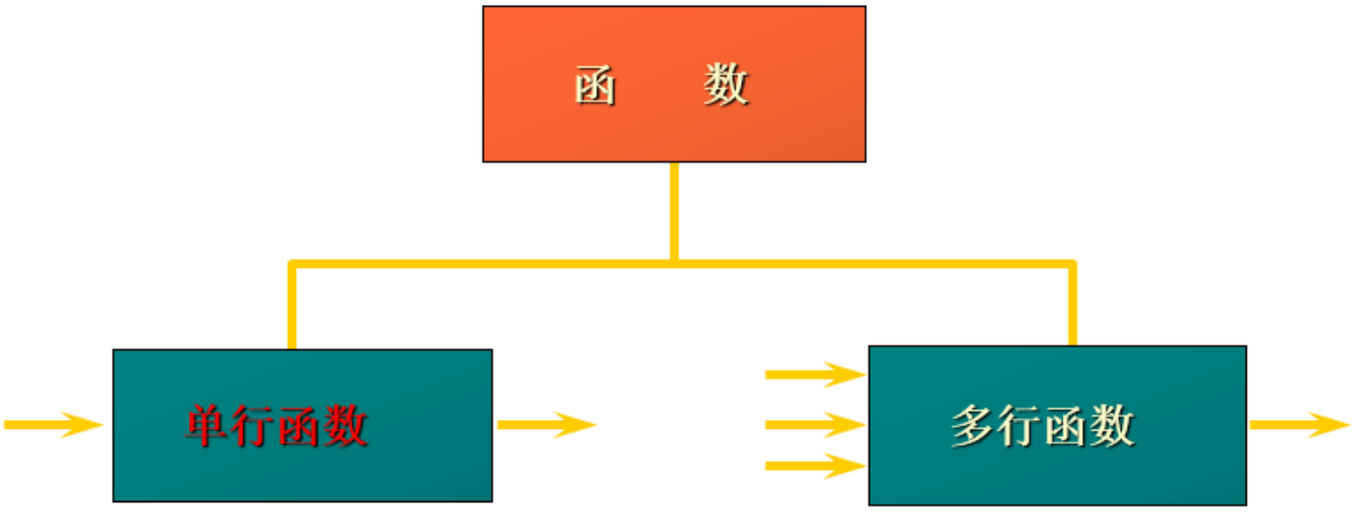
UNION ALL操作符返回两个查询的结果集的并集。对于两个结果集的重复部分，不去重。

2.5 单行函数

2.5.1 介绍



两种SQL函数



单行函数

- 操作数据对象
- 接受参数返回一个结果
- **只对一行进行变换**
- **每行返回一个结果**
- 可以嵌套
- 参数可以是一列或一个值

2.5.2 字符串函数

函数	用法
CONCAT(S1,S2,……,Sn)	连接S1,S2,……,Sn为一个字符串
CONCAT_WS(s, S1,S2,……,Sn)	同CONCAT(s1,s2,...)函数，但是每个字符串之间要加上s
CHAR_LENGTH(s)	返回字符串s的字符数
LENGTH(s)	返回字符串s的字节数，和字符集有关
INSERT(str, index , len, instr)	将字符串str从第index位置开始，len个字符长的子串替换为字符串instr
UPPER(s) 或 UCASE(s)	将字符串s的所有字母转成大写字母
LOWER(s) 或LCASE(s)	将字符串s的所有字母转成小写字母
LEFT(s,n)	返回字符串s最左边的n个字符
RIGHT(s,n)	返回字符串s最右边的n个字符
LPAD(str, len, pad)	用字符串pad对str最左边进行填充，直到str的长度为len个字符
RPAD(str ,len, pad)	用字符串pad对str最右边进行填充，直到str的长度为len个字符
LTRIM(s)	去掉字符串s左侧的空格
RTRIM(s)	去掉字符串s右侧的空格
TRIM(s)	去掉字符串s开始与结尾的空格
TRIM(【BOTH 】s1 FROM s)	去掉字符串s开始与结尾的s1
TRIM(LEADING s1 FROM s)	去掉字符串s开始处的s1
TRIM(TRAILING s1 FROM s)	去掉字符串s结尾处的s1
REPEAT(str, n)	返回str重复n次的结果
REPLACE（str, a, b）	用字符串b替换字符串str中所有出现的字符串a
STRCMP(s1,s2)	比较字符串s1,s2
SUBSTRING(s,index,len)	返回从字符串s的index位置其len个字符

- **举例1：大小写控制函数**

函数	结果
LOWER('SQL Course')	**sql course**
UPPER('SQL Course')	**SQL COURSE**

> 这类函数改变字符的大小写。

- **举例2：字符控制函数**

函数	结果
CONCAT('Hello','World')	**HelloWorld**
SUBSTR('HelloWorld',1,5)	**Hello**
LENGTH('HelloWorld')	**10**
INSTR('HelloWorld','W')	**6**
LPAD(salary,10,'*')	*******24000**
RPAD(salary,10, '*')	**24000*******
TRIM('H' FROM 'HelloWorld')	**elloWorld**
REPLACE('abcd','b','m')	**amcd**

2.5.3 数值函数

函数	用法
ABS(x)	返回x的绝对值
CEIL(x)	返回大于x的最小整数值
FLOOR(x)	返回小于x的最大整数值
MOD(x,y)	返回x/y的模
RAND()	返回0~1的随机值
ROUND(x,y)	返回参数x的四舍五入的有y位的小数的值
TRUNCATE(x,y)	返回数字x截断为y位小数的结果
SQRT(x)	返回x的平方根
POW(x,y)	返回x的y次方

- **举例1：ROUND:四舍五入**

```
ROUND(45.926, 2)     -->  45.93
```

- **举例2：TRUNCATE:截断**

```
TRUNCATE(45.926)      --> 45
```

- **举例3：MOD:求余**

```
MOD(1600, 300)   --> 100
```

2.5.4 日期函数

函数	用法
CURDATE() 或 CURRENT_DATE()	返回当前日期
CURTIME() 或 CURRENT_TIME()	返回当前时间
NOW() / SYSDATE() / CURRENT_TIMESTAMP() / LOCALTIME() / LOCALTIMESTAMP()	返回当前系统日期时间
YEAR(date) / MONTH(date) / DAY(date) / HOUR(time) / MINUTE(time) / SECOND(time)	返回具体的时间值
WEEK(date) / WEEKOFYEAR(date)	返回一年中的第几周
DAYOFWEEK(date)	返回周几，注意：周日是1，周一是2，。。。。周六是7
WEEKDAY(date)	返回周几，注意，周1是0，周2是1，。。。。周日是6
DAYNAME(date)	返回星期：MONDAY,TUESDAY.....SUNDAY
MONTHNAME(date)	返回月份：January，。。。。。。
DATEDIFF(date1,date2) / TIMEDIFF(time1, time2)	返回date1 - date2的日期间隔 / 返回time1 - time2的时间间隔
DATE_ADD(datetime, INTERVAL expr type)	返回与给定日期时间相差INTERVAL时间段的日期时间
DATE_FORMAT(datetime ,fmt)	按照字符串fmt格式化日期datetime值
STR_TO_DATE(str, fmt)	按照字符串fmt对str进行解析，解析为一个日期

其中：

（1）DATE_ADD(datetime,INTERVAL expr type)

表达式类型：

参数类型	参数类型
YEAR	YEAR_MONTH
MONTH	DAY_HOUR
DAY	DAY_MINUTE
HOUR	DAY_SECOND
MINUTE	HOUR_MINUTE
SECOND	HOUR_SECOND
	MINUTE_SECOND

举例：

```
SELECT DATE_ADD(NOW(), INTERVAL 1 YEAR);
SELECT DATE_ADD(NOW(), INTERVAL -1 YEAR);    #可以是负数
SELECT DATE_ADD(NOW(), INTERVAL '1_1' YEAR_MONTH);    #需要单引号
```

（2）DATE_FORMAT(datetime,fmt) 和 STR_TO_DATE(str, fmt)

格式符	说明	格式符	说明
%Y	4位数字表示年份	%y	表示两位数字表示年份
%M	月名表示月份（January,....）	%m	两位数字表示月份（01,02,03。。。）
%b	缩写的月名（Jan.，Feb.，....）	%c	数字表示月份（1,2,3,...）
%D	英文后缀表示月中的天数（1st,2nd,3rd,...）	%d	两位数字表示月中的天数(01,02...)
%e	数字形式表示月中的天数（1,2,3,4,5.....）		
%H	两位数字表示小数，24小时制（01,02..）	%h和%I	两位数字表示小时，12小时制（01,02..）
%k	数字形式的小时，24小时制(1,2,3)	%l	数字形式表示小时，12小时制（1,2,3,4....）
%i	两位数字表示分钟（00,01,02）	%S和%s	两位数字表示秒(00,01,02...)
%W	一周中的星期名称（Sunday...）	%a	一周中的星期缩写（Sun.，Mon.,Tues.，..）
%w	以数字表示周中的天数(0=Sunday,1=Monday....)		
%j	以3位数字表示年中的天数(001,002...)	%U	以数字表示年中的第几周，（1,2,3。。）其中Sunday为周中第一天
%u	以数字表示年中的第几周，（1,2,3。。）其中Monday为周中第一天		
%T	24小时制	%r	12小时制
%p	AM或PM	%%	表示%

举例：

```
SELECT STR_TO_DATE('09/01/2009','%m/%d/%Y')
FROM DUAL;

SELECT STR_TO_DATE('20140422154706','%Y%m%d%H%i%s')
FROM DUAL;

SELECT STR_TO_DATE('2014-04-22 15:47:06','%Y-%m-%d %H:%i:%s')
FROM DUAL;
```

2.5.5 流程函数

函数	用法
IF(value,t ,f)	如果value是真，返回t，否则返回f
IFNULL(value1, value2)	如果value1不为空，返回value1，否则返回value2
CASE WHEN 条件1 THEN result1 WHEN 条件2 THEN result2 .... [ELSE resultn] END	相当于Java的if...else if...else...
CASE expr WHEN 常量值1 THEN 值1 WHEN 常量值1 THEN 值1 .... [ELSE 值n] END	相当于Java的switch...case...

```
SELECT employee_id,salary, CASE WHEN salary>=15000 THEN '高薪'
                  WHEN salary>=10000 THEN '潜力股'
                  WHEN salary>=8000 THEN '屌丝'
                  ELSE '草根' END  "描述"
FROM employees;
```

```
SELECT oid,`status`, CASE `status` WHEN 1 THEN '未付款'
                                   WHEN 2 THEN '已付款'
                                   WHEN 3 THEN '已发货'
                                   WHEN 4 THEN '确认收货'
                                   ELSE '无效订单' END
FROM t_order;
```

- 举例1：

```
SELECT employee_id,12 * salary * (1 + IFNULL(commission_pct,0))
FROM employees;
```

- 举例2：

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE      salary END     "REVISED_SALARY"
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

练习：查询部门号为 10,20, 30 的员工信息, 若部门号为 10, 则打印其工资的 1.1 倍, 20 号部门, 则打印其工资的 1.2 倍, 30 号部门打印其工资的 1.3 倍数。

2.5.6 其他函数

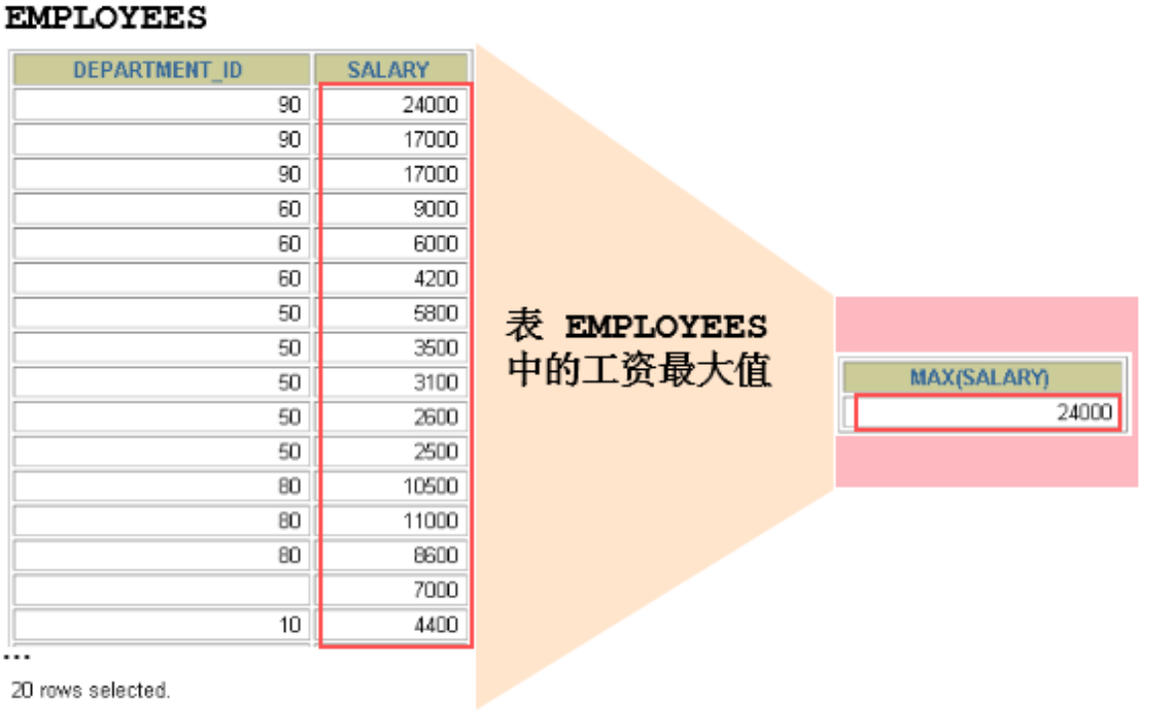
函数	用法
database()	返回当前数据库名
version()	返回当前数据库版本
user()	返回当前登录用户名
password(str)	返回字符串str的加密版本，41位长的字符串
md5(str)	返回字符串str的md5值，也是一种加密方式

2.6 分组函数

2.6.1 组函数介绍

- **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。



- **组函数类型**
  - **AVG()**
  - **SUM()**
  - **MAX()**
  - **MIN()**
  - **COUNT()**

- 组函数语法

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

2.6.2 AVG和SUM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

2.6.3 MIN和MAX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

2.6.4 COUNT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(expr) 返回**expr不为空**的记录总数。

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

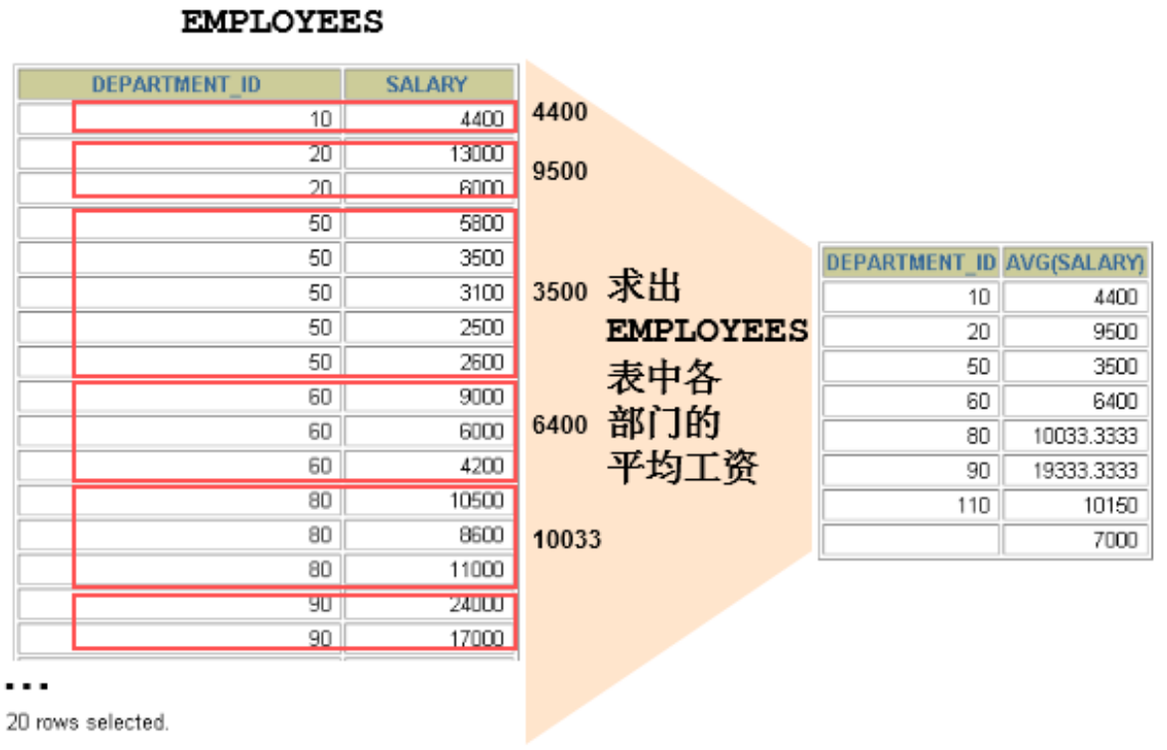
COUNT(COMMISSION_PCT)
3

- **问题：用count(*)，count(1)谁好呢?**

  其实，对于myisam引擎的表是没有区别的。这种引擎内部有一计数器在维护着行数。

  Innodb引擎的表用count(*)直接读行数，效率很低，因为innodb真的要去数一遍。

2.6.5 GROUP BY



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

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

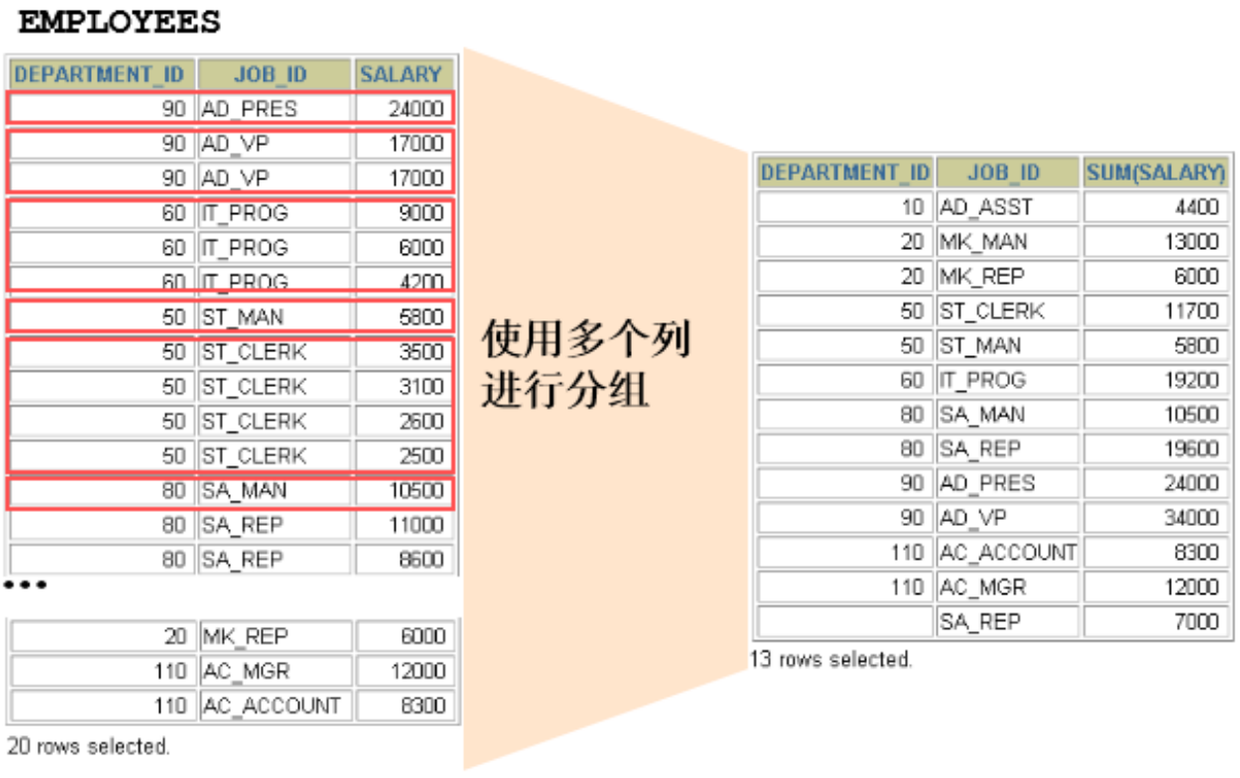
8 rows selected.

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




```sql
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.

2.6.6 HAVING

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

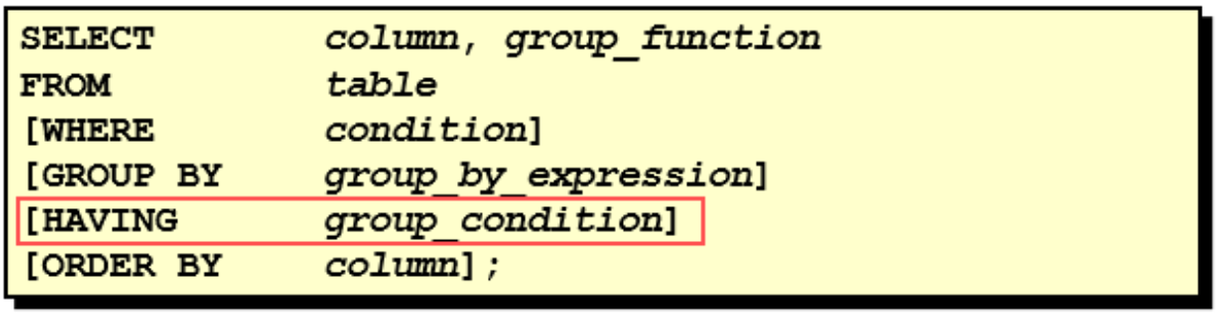
- **非法使用组函数**
  - **不能在 WHERE 子句中使用组函数。**
  - **可以在 HAVING 子句中使用组函数。**

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

```
WHERE  AVG(salary) > 8000
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

过滤分组：HAVING子句

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。




```sql
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

2.7 小结

- 查询的结构

```sql

SELECT ...,...,...
FROM ...,...,...
WHERE 多表的连接条件
AND 不包含组函数的过滤条件
GROUP BY ...,...
HAVING 包含组函数的过滤条件
ORDER BY ... ASC/DESC
LIMIT ...,...
或
SELECT ...,...,...
FROM ... JOIN ...
ON 多表的连接条件
JOIN ...
ON ...
WHERE 不包含组函数的过滤条件
AND/OR 不包含组函数的过滤条件
GROUP BY ...,...
HAVING 包含组函数的过滤条件
ORDER BY ... ASC/DESC
LIMIT ...,...

```

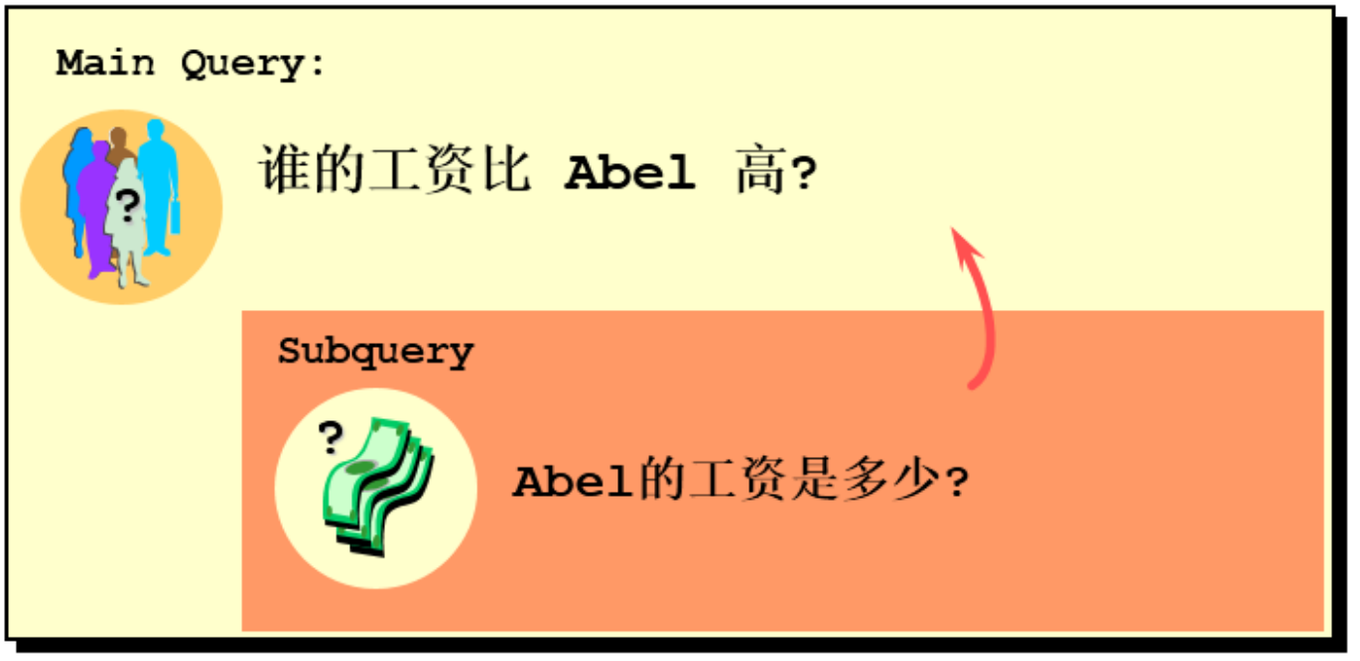
- 执行的顺序

```
FROM <left_table>
ON <join_condition>
<join_type> JOIN <right_table>
WHERE <where_condition>
GROUP BY <group_by_list>
HAVING <having_condition>
SELECT
DISTINCT <select_list>
ORDER BY <order_by_condition>
LIMIT <limit_number>
```

第3章：子查询

3.1 基本介绍

实际问题：



```
SELECT    select_list
FROM      table
WHERE     expr operator
                      (SELECT       select_list
                       FROM         table);
```

- **子查询 (内查询) 在主查询之前一次执行完成。**
- **子查询的结果被主查询(外查询)使用 。**
- **子查询的类型：**
  - 单行子查询

  - 多行子查询

- **注意事项**
  - **子查询要包含在括号内**
  - **将子查询放在比较条件的右侧**
  - 单行操作符对应单行子查询，多行操作符对应多行子查询

3.2 单行子查询

3.2.1 单行比较操作符

操作符	含义
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
<>	not equal to

```
SELECT last_name
FROM   employees
WHERE  salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Abel');
```

LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

题目：返回job_id与141号员工相同，salary比143号员工多的员工姓名，job_id和工资

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141)
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  employee_id = 143);
```

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

题目：返回公司工资最少的员工的last_name,job_id和salary

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

3.2.2 子查询中的 HAVING 子句

- 首先执行子查询。

- 向主查询中的HAVING 子句返回结果。

题目：查询最低工资大于50号部门最低工资的部门id和其最低工资

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                        (SELECT MIN(salary)
                         FROM   employees
                         WHERE  department_id = 50);
```

3.2.3 子查询中的空值问题

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');
```

```
no rows selected
```

> 子查询不返回任何行

3.2.4 非法使用子查询

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                (SELECT   MIN(salary)
                 FROM     employees
                 GROUP BY department_id);
```

```
错误代码：  1242
Subquery returns more than 1 row
```

> 多行子查询使用单行比较符

3.2 多行子查询

- 返回多行。

- 使用多行比较操作符。

操作符	含义
IN	等于列表中的**任意一个**
ANY	和子查询返回的**某一个**值比较
ALL	和子查询返回的**所有**值比较

> 体会any和all的区别

3.2.1 使用ANY或ALL操作符

题目：返回其它job_id中比job_id为'IT_PROG'部门任一工资低的员工的员工号、姓名、job_id 以及salary

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000,4800, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

10 rows selected.

题目：返回其它job_id中比job_id为'IT_PROG'部门所有工资都低的员工的员工号、姓名、job_id以及salary

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4800,4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

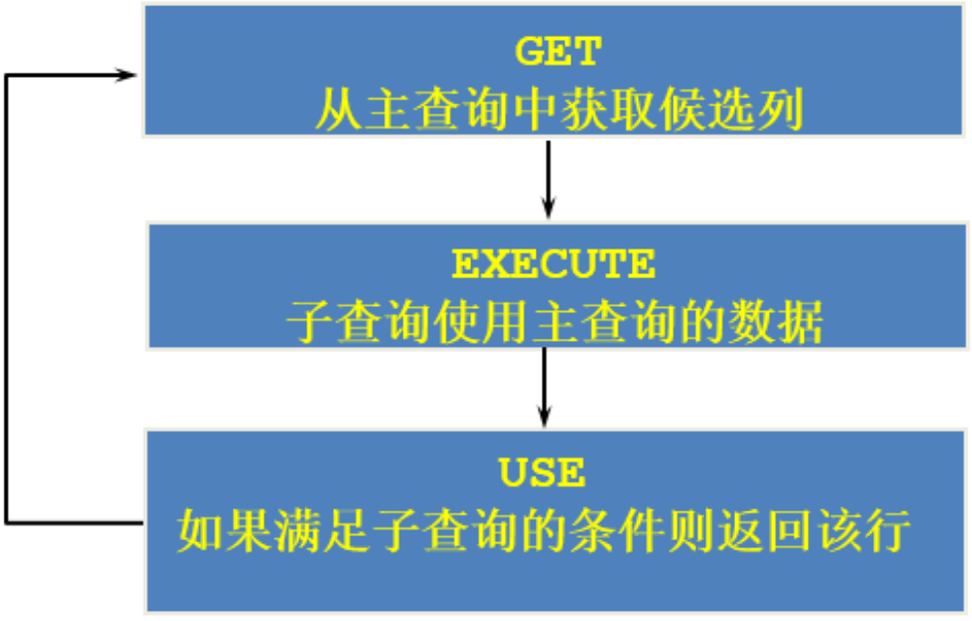
3.2.2 空值问题

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                              (SELECT mgr.manager_id
                               FROM   employees mgr);

no rows selected
```

3.3 相关子查询

相关子查询按照一行接一行的顺序执行，主查询的每一行都执行一次子查询。

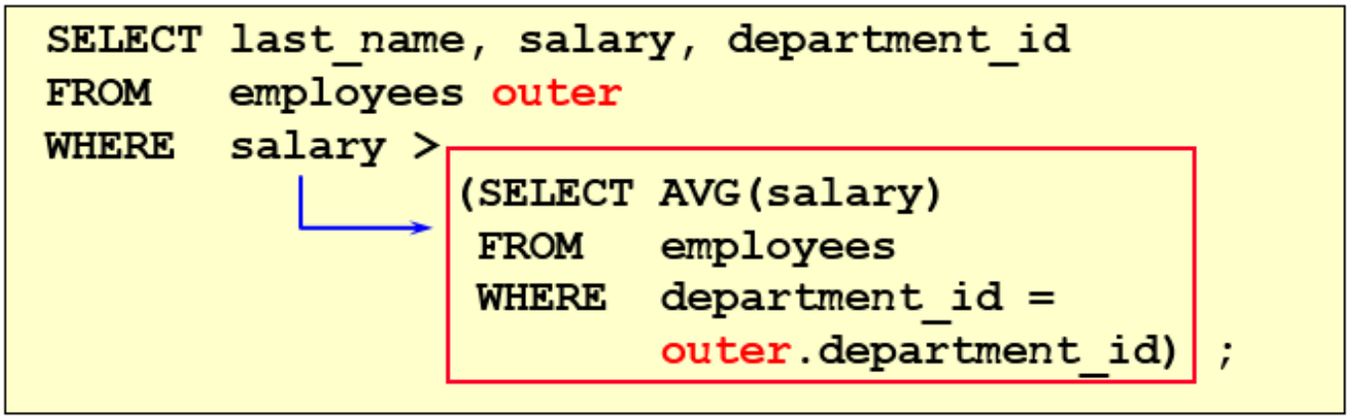



```
SELECT column1, column2, ...
FROM   table1  outer
WHERE  column1 operator
                      (SELECT  colum1, column2
                       FROM    table2
                       WHERE   expr1 =
                                  outer.expr2);
```

说明：**子查询中使用主查询中的列**

题目：查询员工中工资大于本部门平均工资的员工的last_name,salary和其department_id

方式一：相关子查询



方式二：在from中使用子查询

```
SELECT last_name,salary,e1.department_id
FROM employees e1,(SELECT department_id,AVG(salary) dept_avg_sal FROM employees GROUP
BY department_id) e2
WHERE e1.`department_id` = e2.department_id
AND e2.dept_avg_sal < e1.`salary`;
```

题目：查询员工的id,salary,按照department_name 排序

```
SELECT employee_id,salary
FROM employees e
ORDER BY (
      SELECT department_name
      FROM departments d
      WHERE e.`department_id` = d.`department_id`
    );
```

3.4 EXISTS操作符

- EXISTS 操作符检查在子查询中是否存在满足条件的行
- **如果在子查询中存在满足条件的行：**
  - 不在子查询中继续查找
  - 条件返回 TRUE
- **如果在子查询中不存在满足条件的行：**
  - 条件返回 FALSE
  - 继续在子查询中查找

题目：查询公司管理者的employee_id,last_name,job_id,department_id信息

方式一：

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees e1
WHERE  EXISTS ( SELECT 'X'
                FROM   employees e2
                WHERE  e2.manager_id =
                       e1.employee_id);
```

方式二：

```
SELECT DISTINCT e1.employee_id, e1.last_name, e1.job_id, e1.department_id
FROM   employees e1 JOIN employees e2
WHERE e1.employee_id = e2.manager_id;
```

方式三：

```
SELECT employee_id,last_name,job_id,department_id
FROM employees
WHERE employee_id IN (
             SELECT DISTINCT manager_id
             FROM employees

             );
```

题目：查询departments表中，不存在于employees表中的部门的department_id和department_name

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM   employees
                  WHERE  department_id = d.department_id);
```

DEPARTMENT_ID	DEPARTMENT_NAME
190	Contracting

3.5 相关更新

```
UPDATE table1 alias1
SET    column = (SELECT expression
                 FROM   table2 alias2
                 WHERE  alias1.column = alias2.column);
```

使用相关子查询依据一个表中的数据更新另一个表的数据。

练习：

```
1)
ALTER TABLE employees
ADD(department_name VARCHAR2(14));
2)
UPDATE employees e
SET    department_name =  (SELECT department_name
                           FROM   departments d
                           WHERE  e.department_id = d.department_id);

```

3.6 相关删除

```
 DELETE FROM table1 alias1
 WHERE column operator (SELECT expression
                        FROM   table2 alias2
                        WHERE  alias1.column = alias2.column);
```

使用相关子查询依据一个表中的数据删除另一个表的数据

题目：删除表employees中，其与emp_history表皆有的数据

```
DELETE FROM employees e
WHERE employee_id in
            (SELECT employee_id
             FROM   emp_history
             WHERE  employee_id = e.employee_id);
```

第4章：创建和管理表 ---DDL

4.1 创建和删除数据库

- 创建一个保存员工信息的数据库

```
create database employees;
```

> 注意：database不能改名。一些可视化工具可以改名，它是建新库,把所有表复制到新库,再删旧库完成的。

- 查看当前所有的数据库

```
show databases;
```

- 使用指定的某个数据库

```
use 数据库名;
```

- 查看指定库下所有的表

```
show tables from 数据库名;
```

- 删除指定的数据库

```
drop database 数据库名;
```

- 查看当前正在使用的数据库

```
SELECT DATABASE() from dual;
```

> 注意：要操作表格和数据之前必须先说明是对哪个数据库进行操作，否则就要对所有对象加上“数据库名.”。

4.2 标识符命名规则

- 数据库名不得超过30个字符，变量名限制为29个
- 必须只能包含 A–Z, a–z, 0–9, _共63个字符
- 不能在对象名的字符间留空格
- 必须不能和用户定义的其他对象重名
- 必须保证你的字段没有和保留字、数据库系统或常用方法冲突
- 保持字段名和类型的一致性,在命名字段并为其指定数据类型的时候一定要保证一致性。假如数据类型在一个表里是整数,那在另一个表里就不要变成字符型了

4.3 常用的数据类型

4.3.1 MySQL中的数据类型

- **整型（xxxint）**
- 位类型(bit)
- **浮点型**（float和double、real）
- 定点数（decimal,numeric）
- **日期时间类型**（date,time,datetime,year）
- **字符串**（char,varchar,xxxtext）
- 二进制数据（xxxBlob、xxbinary）
- 枚举（enum）
- 集合（set）

4.3.2 常用数据类型及其范围

数据类型	描述
INT	从-2^31到2^31-1的整型数据。存储大小为 4个字节
CHAR(*size*)	定长字符数据。若未指定，默认为1个字符，最大长度255
VARCHAR(*size*)	可变长字符数据，根据字符串实际长度保存，**必须指定长度**
FLOAT(M,D)	单精度，M=整数位+小数位，D=小数位。 D<=M<=255,0<=D<=30，默认M+D<=6
DOUBLE(M,D)	双精度。D<=M<=255,0<=D<=30，默认M+D<=15
DATE	日期型数据，格式'YYYY-MM-DD'
BLOB	二进制形式的长文本数据，最大可达4G
TEXT	长文本数据，最大可达4G

4.4 创建表

4.4.1 创建方式一

- **必须具备：**
  - CREATE TABLE权限
  - 存储空间

```
CREATE TABLE [schema.]table
(column datatype [DEFAULT expr][, ...]);
```

- **必须指定：**
  - 表名
  - 列名, 数据类型, **尺寸**

```
CREATE TABLE dept(
deptno INT(2),
dname VARCHAR(14),
loc VARCHAR(13));
```

```
DESCRIBE dept;
```

Field	Type	Null	Key	Default	Extra
id	int(2) 6B	YES		(NULL) 0K	
name	var... 11B	YES		(NULL) 0K	

```sql
--创建表
CREATE TABLE emp (
  --int类型，自增
  emp_id INT AUTO_INCREMENT,
  --最多保存20个中英文字符
  emp_name CHAR (20),
  --总位数不超过15位
  salary DOUBLE,
  --日期类型
  birthday DATE,
  --主键
  PRIMARY KEY (emp_id)
) ;
```

Field	Type		Null	Key	Default		Extra
emp_id	int(11)	7B	NO	PRI	(NULL)	0K	auto_increment
emp_name	char(20)	8B	YES		(NULL)	0K	
salary	double	6B	YES		(NULL)	0K	
birthday	date	4B	YES		(NULL)	0K	

- 查看表的定义：

```sql
SHOW CREATE TABLE 表名;
```

4.4.2 创建方式二：基于现有的表

- 使用 AS subquery 选项，**将创建表和插入数据结合起来**

```
CREATE TABLE table
        [(column, column...)]
AS subquery;
```

- 指定的列和子查询中的列要一一对应
- 通过列名和默认值定义列

```sql
create table emp1 as select * from employees;
create table emp2 as select * from employees where 1=2; --创建的emp2是空表
```

```sql
CREATE TABLE dept80
AS
SELECT  employee_id, last_name, salary*12 ANNSAL, hire_date
FROM    employees
WHERE   department_id = 80;

```

```
DESCRIBE dept80;
```

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

34 rows selected

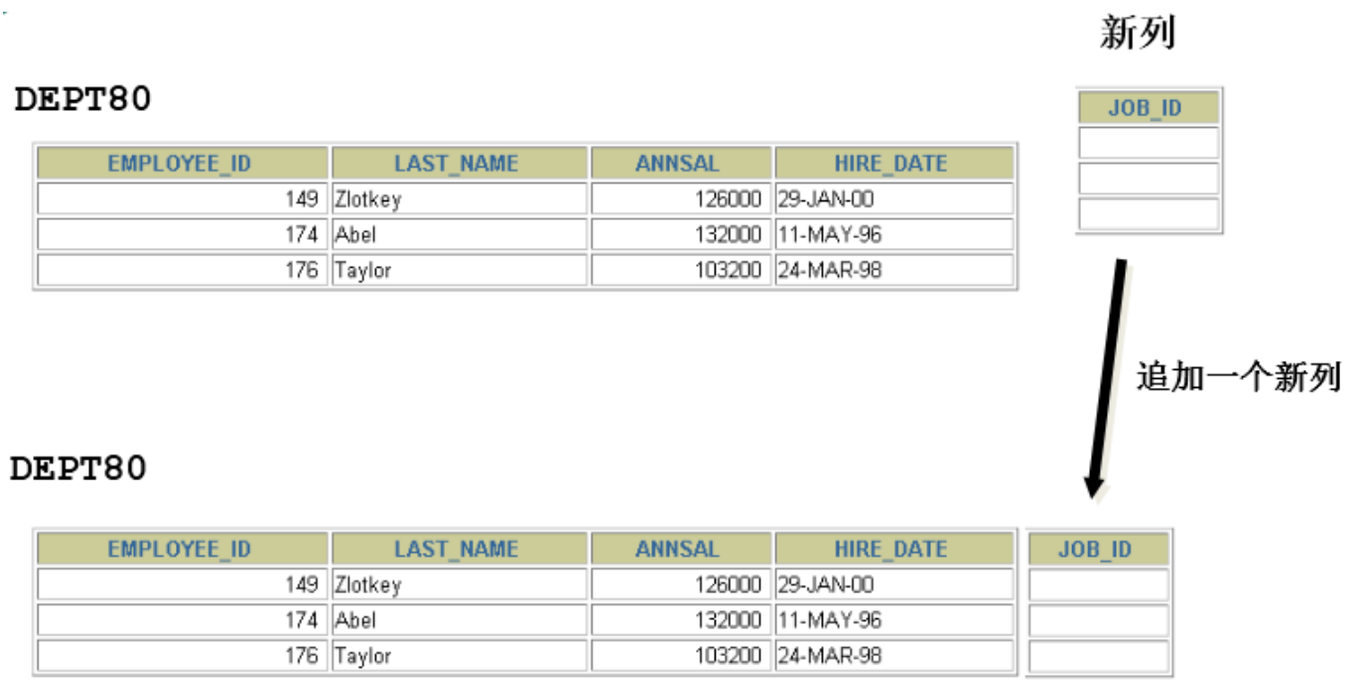
4.5 修改表

使用 ALTER TABLE 语句可以实现：

- 向已有的表中添加列
- 修改现有表中的列
- 删除现有表中的列
- 重命名现有表中的列

4.5.1 追加一个列

```
ALTER TABLE dept80
ADD job_id varchar(15);
```



4.5.2 修改一个列

- 可以修改列的数据类型, 尺寸和默认值

```
ALTER TABLE dept80
MODIFY (last_name VARCHAR(30));
```

```
ALTER TABLE dept80
MODIFY      (salary double(9,2) default 1000);
```

- 对默认值的修改只影响今后对表的修改

4.5.3 重命名一个列

使用 CHANGE old_column new_column dataType子句重命名列。

```
ALTER TABLE  dept80
CHANGE department_name dept_name varchar(15);
```

4.5.4 删除一个列

使用 DROP COLUMN 子句删除不再需要的列。

```
ALTER TABLE  dept80
DROP COLUMN  job_id;
```

4.6 重命名表

- 执行RENAME语句改变表, 视图的名称
  - 方式一：

```
RENAME TABLE emp
TO myemp;
```

  - 方式二：

```
ALTER table dept
RENAME [TO] detail_dept;  -- [TO]可以省略
```

- 必须是对象的拥有者

4.7 删除表

- 数据和结构都被删除
- 所有正在运行的相关事务被提交
- 所有相关索引被删除
- DROP TABLE 语句不能回滚

```
DROP TABLE dept80;
```

4.8 清空表

- TRUNCATE TABLE语句:
  - 删除表中所有的数据
  - 释放表的存储空间

```
TRUNCATE TABLE detail_dept;
```

- TRUNCATE语句**不能回滚**
- 可以使用 DELETE 语句删除数据,可以回滚
- 对比：

```
  set autocommit = false;

  delete from emp2;

  select * from emp2;

  rollback;

  select * from emp2;
```

第5章：数据库事务

5.1 什么是事务

- **事务：**一组逻辑操作单元,使数据从一种状态变换到另一种状态。
- 数据库事务由以下的部分组成:
  - **一个或多个DML 语句**
  - 一个 DDL(Data Definition Language – 数据定义语言) 语句
  - 一个 DCL(Data Control Language – 数据控制语言) 语句

5.2 如何处理事务

- **事务处理**（事务操作）：**保证所有事务都作为一个工作单元来执行，即使出现了故障，都不能改变这种执行方式。当在一个事务中执行多个操作时，要么所有的事务都被提交(commit)，那么这些修改就永久地保存下来；要么数据库管理系统将放弃所作的所有修改，整个事务回滚(rollback)到最初状态。**
- 为确保数据库中数据的**一致性**,数据的操纵应当是离散的成组的逻辑单元:当它全部完成时,数据的一致性可以保持,而当这个单元中的一部分操作失败,整个事务应全部视为错误,所有从起始点以后的操作应全部回退到开始状态。
- 当一个连接对象被创建时，默认情况下是自动提交事务：每次执行一个SQL 语句时，如果执行成功，就会向数据库自动提交，而不能回滚。
- **具体过程：**
  1. **设置提交状态：SET AUTOCOMMIT = FALSE;**
  2. 以第一个 DML 语句的执行作为开始
  3. 以下面的其中之一作为结束:
     - **COMMIT或ROLLBACK语句**
     - DDL 语句（**自动提交**）
     - 用户会话正常结束
     - 系统异常终止

- **COMMIT和ROLLBACK语句的优点**
  - **确保数据完整性。**
  - 数据改变被提交之前预览。
  - 将逻辑上相关的操作分组。

> **数据完整性**：
>
> 存储在数据库中的所有数据值均处于正确的状态。如果数据库中存储有不正确的数据值，则该数据库称为已丧失数据完整性。
>
> **数据库采用多种方法来保证数据完整性，包括外键、束约、规则和触发器。**

- **提交或回滚前的数据状态**
  - 改变前的数据状态是可以恢复的
  - 执行 DML 操作的用户可以通过 SELECT 语句查询提交或回滚之前的修正
  - 其他用户不能看到当前用户所做的改变，直到当前用户结束事务。
  - DML语句所涉及到的行被锁定， 其他用户不能操作。
- **提交后的数据状态**
  - 数据的改变已经被保存到数据库中。
  - 改变前的数据已经丢失。
  - 所有用户可以看到结果。
  - 锁被释放， 其他用户可以操作涉及到的数据。
- **数据回滚后的状态**

  使用 ROLLBACK 语句可使数据变化失效：
  - 数据改变被取消。
  - 修改前的数据状态可以被恢复。

5.3 事务的ACID属性

1. **原子性（Atomicity）**原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
2. **一致性（Consistency）**事务必须使数据库从一个一致性状态变换到另外一个一致性状态。
3. **隔离性（Isolation）**事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
4. **持久性（Durability）**持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响

5.4 数据库的隔离级别

- 对于同时运行的多个事务, 当这些事务访问数据库中相同的数据时, 如果没有采取必要的隔离机制, 就会导致各种并发问题:
  - **脏读**: 对于两个事务 T1, T2, T1 读取了已经被 T2 更新但还**没有被提交**的字段。 之后, 若 T2 回滚, T1读取的内容就是临时且无效的。
  - **不可重复读**: 对于两个事务T1, T2, T1 读取了一个字段, 然后 T2 **更新**了该字段。 之后, T1再次读取同一个字段, 值就不同了。

- **幻读**: 对于两个事务T1, T2, T1 从一个表中读取了一个字段, 然后 T2 在该表中**插入**了一些新的行。 之后, 如果 T1 再次读取同一个表, 就会多出几行。

- **数据库事务的隔离性**：数据库系统必须具有隔离并发运行各个事务的能力, 使它们不会相互影响, 避免各种并发问题。
- **一个事务与其他事务隔离的程度称为隔离级别。**数据库规定了多种事务隔离级别, 不同隔离级别对应不同的干扰程度, 隔离级别越高, 数据一致性就越好, 但并发性越弱。
- **数据库提供的 4 种事务隔离级别：**

隔离级别	描述
READ UNCOMMITTED (读未提交数据)	允许事务读取未被其他事物提交的变更. 脏读, 不可重复读和幻读的问题都会出现
READ COMMITED (读已提交数据)	只允许事务读取已经被其它事务提交的变更. 可以避免脏读, 但不可重复读和幻读问题仍然可能出现
REPEATABLE READ (可重复读)	确保事务可以多次从一个字段中读取相同的值. 在这个事务持续期间, 禁止其他事物对这个字段进行更新. 可以避免脏读和不可重复读, 但幻读的问题仍然存在.
SERIALIZABLE(串行化)	确保事务可以从一个表中读取相同的行. 在这个事务持续期间, 禁止其他事务对该表执行插入, 更新和删除操作. 所有并发问题都可以避免, 但性能十分低下.

- Oracle 支持的 2 种事务隔离级别：**READ COMMITED**, SERIALIZABLE。 Oracle 默认的事务隔离级别为: READ COMMITED 。
- Mysql 支持 4 种事务隔离级别。 Mysql 默认的事务隔离级别为: **REPEATABLE READ**

- **在MySql中设置隔离级别**
  - 每启动一个 mysql 程序, 就会获得一个单独的数据库连接。每个数据库连接都有一个全局变量 @@tx_isolation, 表示当前的事务隔离级别.
  - 查看当前的隔离级别: **SELECT @@tx_isolation;**
  - 查看全局的隔离级别: **SELECT @@global.tx_isolation;**
  - 设置当前 mySQL 连接的隔离级别:

```
set tx_isolation ='repeatable-read';
```

  - 设置数据库系统的全局的隔离级别:

```
set global tx_isolation ='read-committed';
```

> 注意：这里的隔离级别中间是减号，不是下划线。

第6章：数据处理之增删改

6.1 插入数据

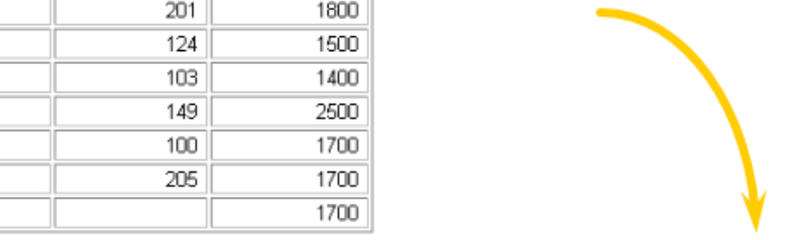
70	Public Relations	100	1700

新行

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

向 DEPARTMENTS 表中插入新的记录



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
70	Public Relations	100	1700

- 使用 INSERT 语句向表中插入数据。

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

- 使用这种语法一次只能向表中插入**一条**数据。
- 为每一列添加一个新值。
- 按列的默认顺序列出各个列的值。
- 在 INSERT 子句中随意列出列名和他们的值。
- VALUES也可以写成VALUE，但是VALUES是标准写法。
- **字符和日期型数据应包含在单引号中。**

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES      (70, 'Pub', 100, 1700);
```

- **向表中插入空值**
  - 隐式方式：在列名表中省略该列的值。

```
INSERT INTO departments (department_id,department_name )
VALUES (30, 'Purchasing');

```

  - 显示方式：在VALUES 子句中指定空值。

```
INSERT INTO departments
VALUES      (100, 'Finance', NULL, NULL);
```

- **同时插入多行数据**

```
INSERT INTO emp(NAME)
VALUES('Tom'),('Jerry'),('Rose');
```

- **从其它表中拷贝数据**
  - 在 INSERT 语句中加入子查询。
  - **不必书写 VALUES 子句。**
  - 子查询中的值列表应与 INSERT 子句中的列名对应。

```
INSERT INTO emp2
SELECT *
FROM employees
WHERE department_id = 90;
```

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM   employees
WHERE  job_id LIKE '%REP%';
```

6.2 更新数据

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_F
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

更新 EMPLOYEES 表

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSIO
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	30	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	30	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	30	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

- 使用 UPDATE 语句更新数据。

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- 可以一次更新**多条**数据。
- 如果需要回滚数据，需要保证在DML前，进行设置：**SET AUTOCOMMIT = FALSE;**

- 使用 **WHERE** 子句指定需要更新的数据。

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
```

- 如果省略 WHERE 子句，则表中的所有数据都将被更新。

```
UPDATE  copy_emp
SET     department_id = 110;
```

- **更新中的数据完整性错误**

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

```
错误代码： 1452
Cannot add or update a child row: a foreign key
constraint fails (`myemployees`.`employees`,
CONSTRAINT `dept_id_fk` FOREIGN KEY (`department_id`)
REFERENCES `departments` (`department_id`))
```

> 说明：不存在 55 号部门

6.3 删除数据

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
100	Finance		
50	Shipping	124	1500
60	IT	103	1400

从表DEPARTMENTS 中删除一条记录。

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400

- 使用 DELETE 语句从表中删除数据

```
DELETE FROM      table
[WHERE           condition];
```

- 使用 WHERE 子句删除指定的记录。

```
1 DELETE FROM departments
2 WHERE  department_name = 'Finance';
```

- 如果省略 WHERE 子句，则表中的全部数据将被删除

```
1 DELETE FROM  copy_emp;
```

- **删除中的数据完整性错误**

```
1 DELETE FROM departments
2 WHERE       department_id = 60;
```

```
错误代码： 1451
Cannot delete or update a parent row: a foreign key
constraint fails (`myemployees`.`employees`,
CONSTRAINT `dept_id_fk` FOREIGN KEY (`department_id`)
REFERENCES `departments` (`department_id`))
```

> 说明：You cannot delete a row that contains a primary key that is used as a foreign key in another table.

第7章：约束(constraint)

7.1 什么是约束

- 为了保证数据的一致性和完整性，SQL规范以约束的方式对**表数据进行额外的条件限制。**
  - 实体完整性（Entity Integrity）：例如，同一个表中，不能存在两条完全相同无法区分的记录
  - 域完整性（Domain Integrity）：例如：年龄范围0-120，性别范围"男/女"
  - 引用完整性（Referential Integrity）：例如：员工所在部门，在部门表中要能找到这个部门
  - 用户自定义完整性（User-defined Integrity）：例如：用户名唯一、密码不能为空等，本部门经理的工资不得高于本部门职工的平均工资的5倍。
- 约束是表级的强制规定
- 可以在**创建表时规定约束（通过 CREATE TABLE 语句）**，或者在**表创建之后也可以（通过 ALTER TABLE 语句）**
- **根据约束数据列的限制，**约束可分为：
  - **单列约束**：每个约束只约束一列
  - **多列约束**：每个约束可约束多列数据
- **根据约束的作用范围**，约束可分为：
  - **列级约束**：只能作用在一个列上，跟在列的定义后面
  - **表级约束**：可以作用在多个列上，不与列一起，而是单独定义
- **根据约束起的作用**，约束可分为：
  - **NOT NULL 非空约束，规定某个字段不能为空**
  - **UNIQUE 唯一约束，规定某个字段在整个表中是唯一的**
  - **PRIMARY KEY 主键(非空且唯一)约束**
  - **FOREIGN KEY 外键约束**
  - **CHECK 检查约束**
  - **DEFAULT 默认值约束**

> 注意： MySQL不支持check约束，但可以使用check约束，而没有任何效果

- 查看某个表已有的约束

```
SELECT * FROM information_schema.table_constraints
WHERE table_name = 'employees';
```

7.2 NOT NULL约束

- not null：**保证列值不能为空**

EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	90
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	60
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	60
178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	
200	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	10

...

- 非空约束用于确保当前列的值不为空值，非空约束只能出现在表对象的列上。
- Null类型特征：
  - 所有的类型的值都可以是null，包括int、float等数据类型
  - 空字符串''不等于null，0也不等于null
- **创建 not null 约束**

```
CREATE TABLE emp(
id INT(10) NOT NULL,
NAME VARCHAR(20) NOT NULL DEFAULT 'abc',
sex CHAR NULL
);
```

- **增加 not null 约束**

```
ALTER TABLE emp
MODIFY sex VARCHAR(30) NOT NULL;
```

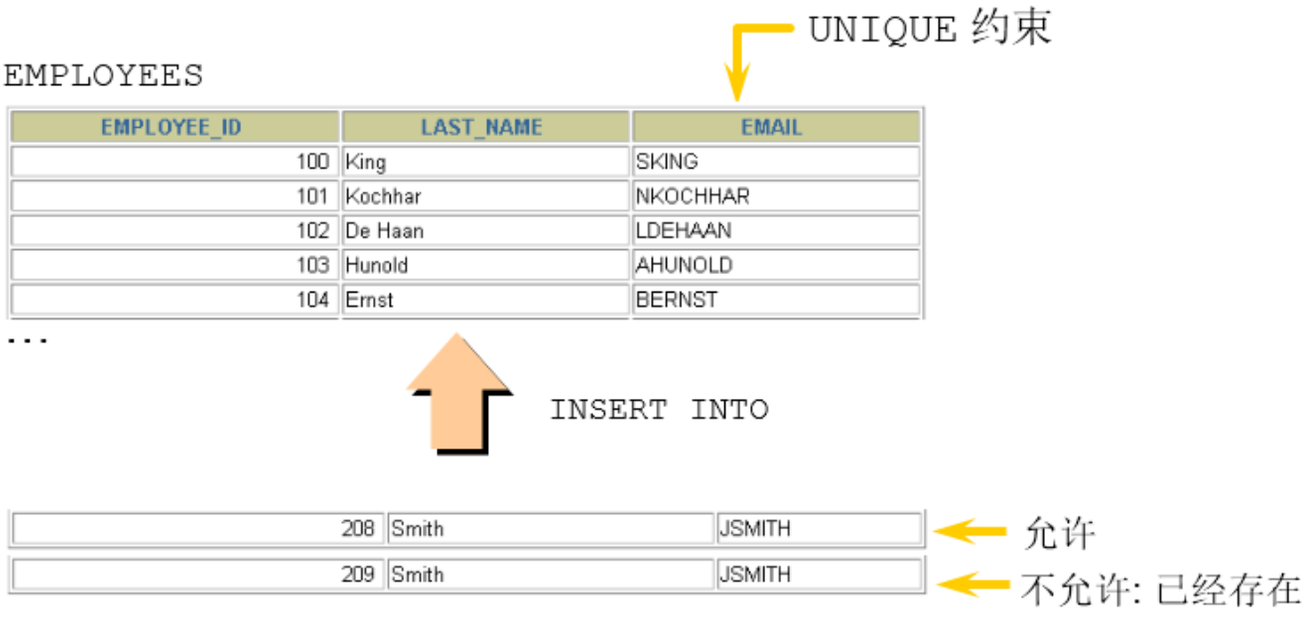
- 取消 not null 约束

```
ALTER TABLE emp
MODIFY sex VARCHAR(30) NULL;
```

- 取消 not null 约束，增加默认值

```
ALTER TABLE emp
MODIFY NAME VARCHAR(15) DEFAULT 'abc' NULL;
```

7.3 UNIQUE约束



唯一约束，允许出现多个空值：NULL。

- 同一个表可以有多个唯一约束。
- 唯一约束可以是某一个列的值唯一，也可以多个列组合的值唯一。
- 在创建唯一约束的时候，如果不给唯一约束名称，就默认和列名相同。
- **MySQL会给唯一约束的列上默认创建一个唯一索引。**

```
CREATE TABLE t_course(
    cid INT PRIMARY KEY,
    cname VARCHAR(100) UNIQUE,
    description VARCHAR(200)
);

```

```
CREATE TABLE USER(
 id INT NOT NULL,
 NAME VARCHAR(25),
 PASSWORD VARCHAR(16),
 --使用表级约束语法
 CONSTRAINT uk_name_pwd UNIQUE(NAME,PASSWORD)
);
```

> 表示用户名和密码组合不能重复

- 添加唯一约束

```
ALTER TABLE USER
ADD UNIQUE(NAME,PASSWORD);
```

```sql
ALTER TABLE USER
ADD CONSTRAINT uk_name_pwd UNIQUE(NAME,PASSWORD);
```

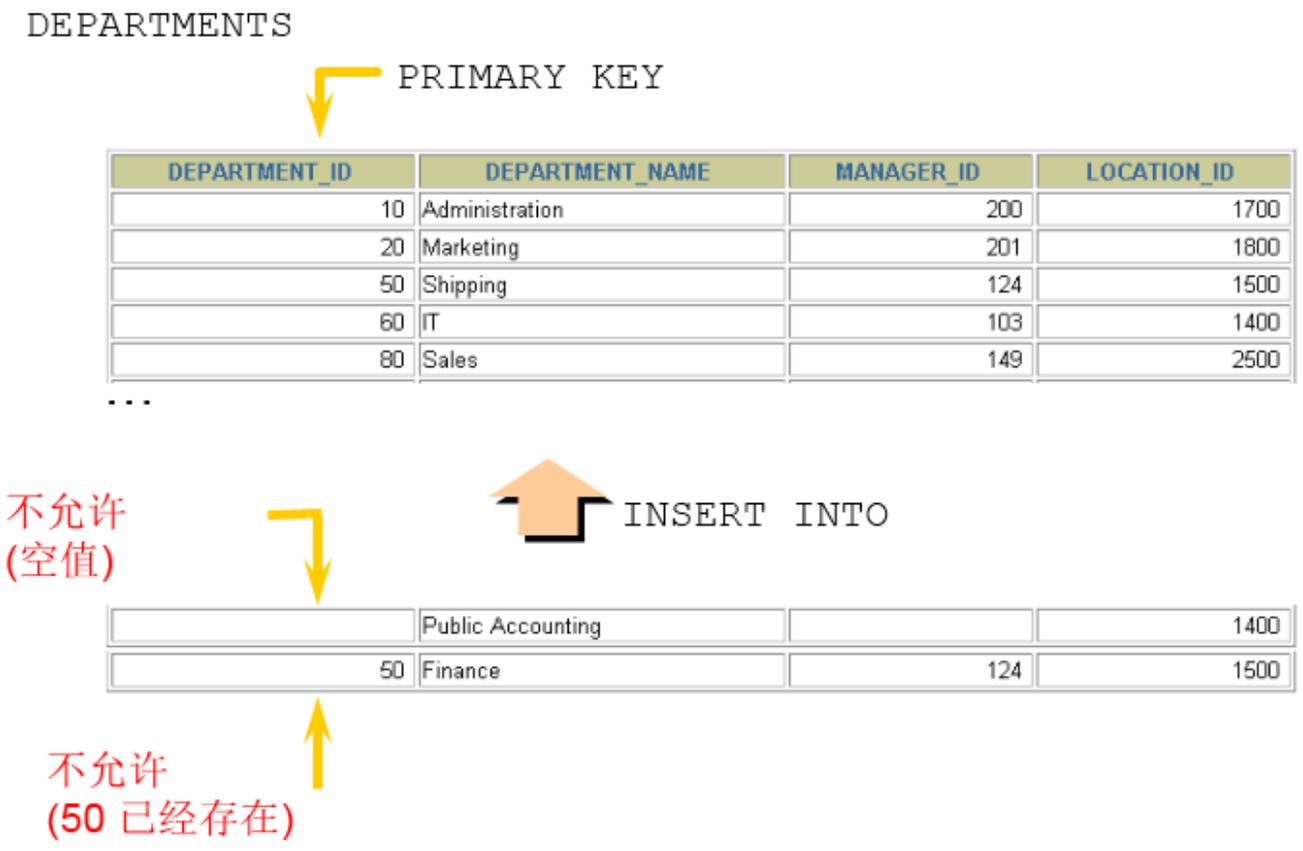
```sql
ALTER TABLE USER
MODIFY NAME VARCHAR(20) UNIQUE;
```

- 删除约束
  - 删除唯一约束只能通过删除唯一索引的方式删除。
  - 删除时需要指定唯一索引名，唯一索引名就和唯一约束名一样。
  - 如果创建唯一约束时未指定名称，如果是单列，就默认和列名相同，如果是组合列，那么默认和()中排在第一个的列名相同。也可以自定义唯一性约束名。

```sql
ALTER TABLE USER
DROP INDEX uk_name_pwd;
```

> 注意：如果忘记名称，可以通过“`show index from 表名称;`”查看。

7.4 PRIMARY KEY 约束



- 主键约束相当于**唯一约束+非空约束的组合**，主键约束列不允许重复，也不允许出现空值
- 如果是多列组合的主键约束，那么这些列都不允许为空值，并且组合的值不允许重复。
- **每个表最多只允许一个主键**，建立主键约束可以在列级别创建，也可以在表级别上创建。
- **MySQL的主键名总是PRIMARY**，就算自己命名了主键约束名也没用。

- 当创建主键约束时，系统默认会在所在的列和列组合上建立对应的**主键索引**。删除主键时，也会直接删除主键索引。
    - 列级模式

```
CREATE TABLE emp4(
id INT PRIMARY KEY AUTO_INCREMENT ,
NAME VARCHAR(20)
);
```

    - 表级模式

```
CREATE TABLE emp5(
id INT NOT NULL AUTO_INCREMENT,
NAME VARCHAR(20),
pwd VARCHAR(15),
CONSTRAINT emp5_id_pk PRIMARY KEY(id)
);
```

    - 组合模式

```
CREATE TABLE emp6(
id INT NOT NULL,
NAME VARCHAR(20),
pwd VARCHAR(15),
CONSTRAINT emp7_pk PRIMARY KEY(NAME,pwd)
);
```

- 删除主键约束

```
ALTER TABLE emp5
DROP PRIMARY KEY;
```

> 说明：删除主键约束，不需要指定主键名，因为一个表只有一个主键，删除主键约束后，非空还存在。

- 添加主键约束

```
ALTER TABLE emp5
ADD PRIMARY KEY(NAME,pwd);
```

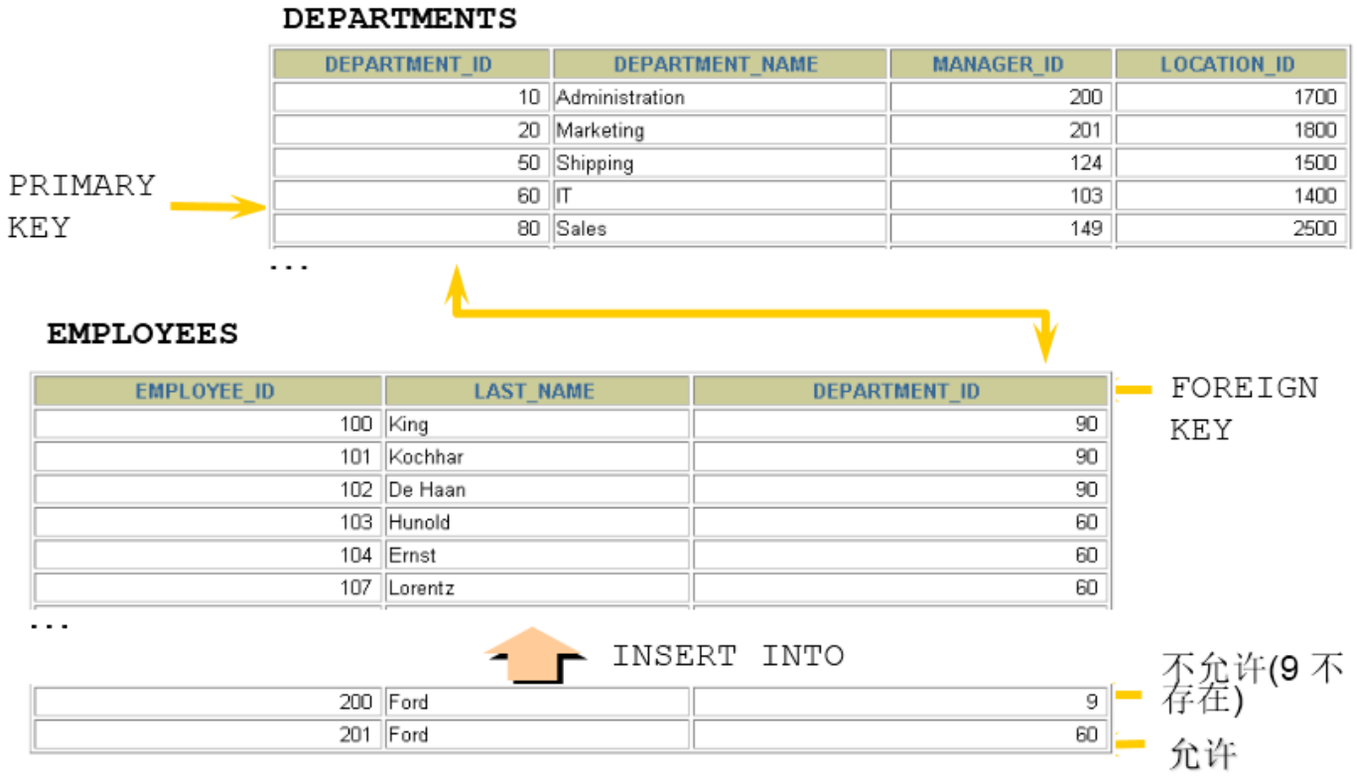
- 修改主键约束

```
ALTER TABLE emp5
MODIFY id INT PRIMARY KEY;
```

- **补充：关于自增长auto_increment的使用**
    - 整数类型的字段才可以设置自增长
    - 当需要产生唯一标识符或顺序值时，可设置自增长
    - **一个表最多只能有一个自增长列**

- 自增长列必须非空（已经隐式满足了）
- **自增长列必须声明有主键约束或唯一性约束**
- InnoDB表的自动增长列可以手动插入，但是插入的值如果是空或者0，则实际插入的将是自动增长后的值。

7.5 FOREIGN KEY 约束



- **基本理解：**
  - 外键约束是保证一个或两个表之间的参照完整性，外键是构建于一个表的两个字段或是两个表的两个字段之间的参照关系
  - 同一个表可以有多个外键约束
  - 从表的外键值必须在主表中能找到或者为空。当主表的记录被从表参照时，主表的记录将不允许删除，如果要删除数据，需要先删除从表中依赖该记录的数据，然后才可以删除主表的数据
  - **注意：外键约束的参照列，在主表中引用的只能是主键或唯一键约束的列**
  - 从表的外键列与主表被参照的列名字可以不相同，但是数据类型必须一样
- **注意点：**
  - 在创建外键约束时，如果不给外键约束名称，**默认名不是列名，而是自动产生一个外键名**（例如student_ibfk_1;），也可以指定外键约束名。
  - **当创建外键约束时，系统默认会在所在的列上建立对应的普通索引。**但是索引名是列名，不是外键的约束名。
  - **删除外键时，关于外键列上的普通索引需要单独删除。**
  - 还有一种就是级联删除子表数据。

- **创建外键约束**
  - 主表

```sql
CREATE TABLE dept(
dept_id INT AUTO_INCREMENT PRIMARY KEY,
dept_name VARCHAR(20)
);
```

- 从表

```sql
CREATE TABLE emp(
emp_id INT AUTO_INCREMENT PRIMARY KEY,
last_name VARCHAR(15),
dept_id INT,
CONSTRAINT emp_dept_id_fk FOREIGN KEY(dept_id)
REFERENCES dept(dept_id)
);
```

> – FOREIGN KEY: 在表级指定子表中的列
>
> – REFERENCES: 标示在父表中的列

- **创建多列外键组合，必须使用表级约束**
  - 主表

```sql
CREATE TABLE classes(
id INT,
NAME VARCHAR(20),
number INT,
PRIMARY KEY(NAME,number)
);
```

  - 从表

```sql
CREATE TABLE student(
id INT AUTO_INCREMENT PRIMARY KEY,
classes_name VARCHAR(20),
classes_number INT,
FOREIGN KEY(classes_name,classes_number)
REFERENCES classes(NAME,number)
);
```

- 删除外键约束

```sql
ALTER TABLE emp
DROP FOREIGN KEY emp_dept_id_fk;
```

> 说明：外键列上的索引，需要单独删除。比如：
>
> ```sql
> ALTER TABLE 表名称 DROP INDEX 外键列索引名;
> ALTER TABLE t_emp DROP INDEX dept_id; -- 举例
> ```
>
> 查看索引名：

```
show index from 表名称;
```

- 增加外键约束

```
ALTER TABLE emp
ADD [CONSTRAINT emp_dept_id_fk] FOREIGN KEY(dept_id)
REFERENCES dept(dept_id);
```

- 关于删除操作：级联删除与级联置空
  - **ON DELETE CASCADE(级联删除)**: 当父表中的列被删除时，子表中相对应的列也被删除
  - **ON DELETE SET NULL(级联置空)**: 当父表中的列被删除时，子表中相应的列的值置空

```
CREATE TABLE student(
id INT AUTO_INCREMENT PRIMARY KEY,
NAME VARCHAR(20),
classes_name VARCHAR(20),
classes_number INT,
/*表级别联合外键*/
FOREIGN KEY(classes_name, classes_number)
REFERENCES classes(NAME, number) ON DELETE CASCADE);
```

- 关于修改操作：级联修改与级联置空
  - **ON UPDATE CASCADE(级联修改)：**当父表中的列值由a修改为b时，子表中相对应的列值为a的改为值b
  - **ON UPDATE SET NULL(级联置空)：**子表中相应的列的值置空

```
CREATE TABLE student(
id INT AUTO_INCREMENT PRIMARY KEY,
NAME VARCHAR(20),
classes_name VARCHAR(20),
classes_number INT,
/*表级别联合外键*/
FOREIGN KEY(classes_name, classes_number)
REFERENCES classes(NAME, number) ON UPDATE CASCADE);
```

7.6 CHECK 约束

- MySQL可以使用check约束，但check约束对数据验证没有任何作用,添加数据时，没有任何错误或警告

```
CREATE TABLE temp(
id INT AUTO_INCREMENT,
NAME VARCHAR(20),
age INT CHECK(age > 20),
PRIMARY KEY(id)
);
```

7.7 DEFAULT约束

- 一旦设置默认值，在插入数据时，如果此字段没有显式赋值，则赋值为默认值。

```
CREATE TABLE myemp(
id INT AUTO_INCREMENT PRIMARY KEY,
NAME VARCHAR(15),
salary DOUBLE(10,2) DEFAULT 2000
);
```

第8章：其它数据库对象

8.1 常见的数据库对象

对象	描述
表(TABLE)	**基本的数据存储集合，由行和列组成**
视图(VIEW)	**从表中抽出的逻辑上相关的数据集合**
序列(SEQUENCE)	**提供有规律的数值。**
索引(INDEX)	**提高查询的效率**
同义词(SYNONYM)	**给对象起别名**

8.2 视图（VIEW）

表EMPLOYEES：

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	26
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	25
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	105
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	110
176	Jonathan	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	86
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	70
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	44
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	130
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	60
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	120
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	83

EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

20 rows selected.

- 视图是一种虚表
- **视图建立在已有表的基础上**, 视图赖以建立的这些表称为**基表**
- 向视图提供数据内容的语句为 SELECT 语句, 可以将视图理解为**存储起来的 SELECT 语句**
- 视图向用户提供基表数据的另一种表现形式
- 为什么使用视图？
    - 控制数据访问
    - **简化查询**
    - 避免重复访问相同的数据
- 简单视图和复杂视图

特性	简单视图	复杂视图
表的数量	一个	一个或多个
函数	没有	有
分组	没有	有
DML操作	可以	有时可以

- 创建视图
    - **在 CREATE VIEW 语句中嵌入子查询**

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

```sql
CREATE VIEW     empvu80
AS
SELECT  employee_id, last_name, salary
FROM    employees
WHERE   department_id = 80;
```

- 子查询可以是复杂的 SELECT 语句

```sql
create or replace view empview
as
select employee_id emp_id,last_name name,department_name
from employees e,departments d
where e.department_id = d.department_id;
```

- 创建视图时在子查询中给列定义**别名**

```sql
CREATE VIEW salvu50
AS
SELECT  employee_id ID_NUMBER, last_name NAME,salary*12 ANN_SALARY
FROM    employees
WHERE   department_id = 50;
```

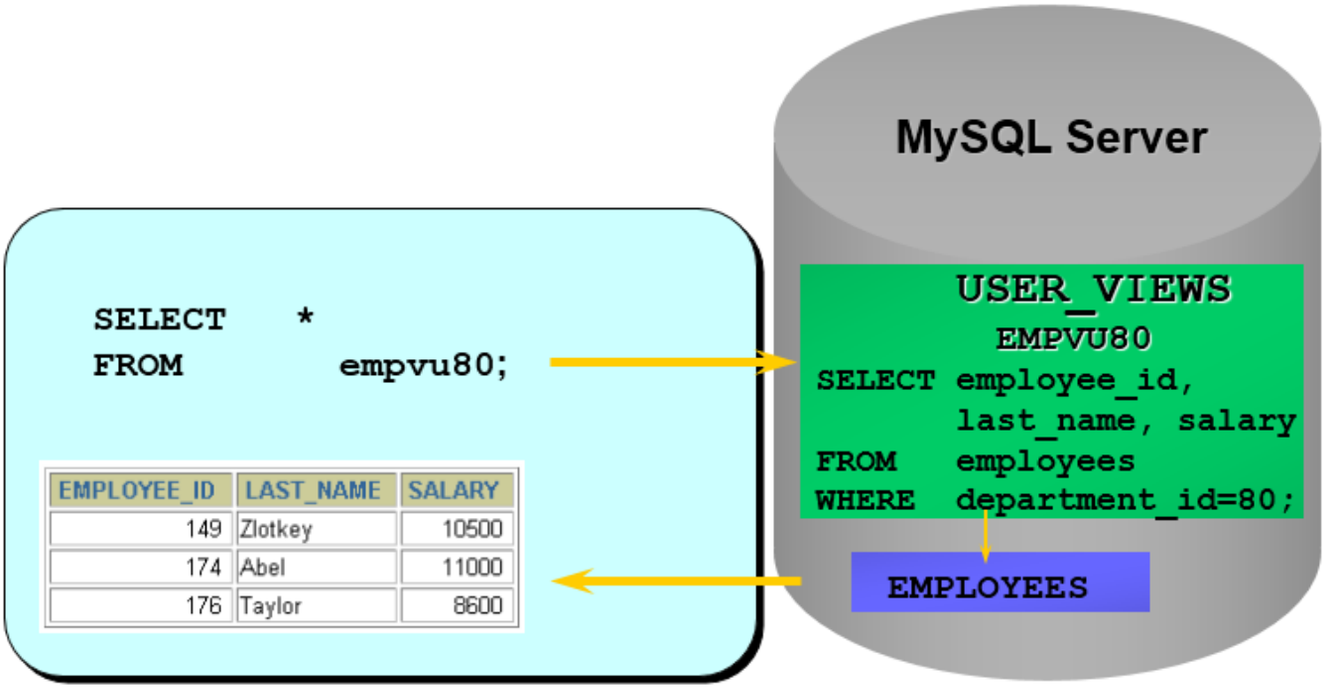
- 描述视图结构

```sql
DESCRIBE empvu80;
```

> 说明：在选择视图中的列时应使用别名

- 查询视图

```sql
SELECT *
FROM    salvu80;
```



- **修改视图**
  - 使用CREATE **OR REPLACE** VIEW 子句**修改视图**

```
CREATE OR REPLACE VIEW empvu80
(id_number, name, sal, department_id)
AS
SELECT  employee_id, first_name || ' ' || last_name, salary, department_id
FROM employees
WHERE department_id = 80;
```

> 说明：CREATE VIEW 子句中各列的别名应和子查询中各列相对应。

- **创建复杂视图**

```
CREATE VIEW dept_sum_vu
(name, minsal, maxsal, avgsal)
AS
SELECT d.department_name, MIN(e.salary), MAX(e.salary),AVG(e.salary)
FROM employees e, departments d
WHERE e.department_id = d.department_id
GROUP BY  d.department_name;
```

- **视图中使用DML的规定**
  - 可以在简单视图中使用DML操作
  - 当视图定义中包含以下元素之一时不能使用delete:
    - 组函数
    - GROUP BY 子句
    - DISTINCT 关键字
    - ROWNUM 伪列
  - 当视图定义中包含以下元素之一时不能使用update:

        - 组函数
        - GROUP BY子句
        - DISTINCT 关键字
        - ROWNUM 伪列
        - 列的定义为表达式
    - 当视图定义中包含以下元素之一时不能使用insert:
        - 组函数
        - GROUP BY 子句
        - DISTINCT 关键字
        - ROWNUM 伪列
        - 列的定义为表达式
        - 表中非空的列在视图定义中未包括
- **删除视图**

  删除视图只是删除视图的定义，并不会删除基表的数据。

  ```
  DROP VIEW view;
  ```

  ```
  1 DROP VIEW empvu80;
  ```

8.3 索引（INDEX）

8.3.1 基本介绍

- 一种独立于表的模式对象, 可以存储在与表不同的磁盘或表空间中，**是一个单独的、物理的数据库结构**
- 索引被删除或损坏, 不会对表产生影响, 其影响的只是**查询的速度**
- 索引一旦建立, 数据库管理系统会对其进行自动维护, 而且由数据库管理系统决定何时使用索引。用户不用在查询语句中指定使用哪个索引
- 在删除一个表时,所有基于该表的索引会自动被删除
- 通过指针加速数据库服务器的查询速度
- **通过快速定位数据的方法，减少磁盘 I/O**

8.3.2 索引的介绍

例如：一本字典，如何快速找到某个字，可以给字典加目录，对数据库来说,索引的作用即是给"数据"加目录。

设有N条随机记录,不用索引,平均查找N/2次,那么用了索引之后呢。如果是btree(二叉树)索引， 则是logN。 如果是hash(哈希)索引，时间复杂度是1。

- MySQL提供多种索引类型供选择：
    - 普通索引
    - 唯一性索引
    - 主键索引：只有一个主键索引
    - 全文索引：MySQL5.X版本只有MyISAM存储引擎支持FULLTEXT，并且只限于CHAR、VARCHAR和TEXT类型的列上创建。
- **MySQL的索引方法：**
    - HASH
    - BTREE（MySQL中多数索引都以BTREE的形式保存。）

8.3.3 创建索引

- **自动创建**: 在定义 PRIMARY KEY 或 UNIQUE 约束后系统自动在相应的列上创建**唯一性**索引
- **手动创建**: 用户可以在其它列上创建非唯一的索引，以加速查询
- 在一个或多个列上创建索引

```
CREATE INDEX index
ON table (column[, column]...);
```

- 在表 EMPLOYEES的列 LAST_NAME 上创建索引

```
CREATE INDEX    emp_last_name_idx
ON      employees(last_name);
```

- **索引的优缺点**
  - 索引的优点：加快了查询速度(select)
  - 索引的缺点：降低了增,删,改的速度(update/delete/insert)，增大了表的文件大小(索引文件甚至可能比数据文件还大)
- **什么时候创建索引**

  以下情况可以创建索引:
  - 列中数据值分布范围很广
  - 列经常在 WHERE 子句或连接条件中出现
  - 表经常被访问而且数据量很大 ，访问的数据大概占数据总量的2%到4%

- **什么时候不要创建索引**

  下列情况不要创建索引:
  - 表很小
  - 列不经常作为连接条件或出现在WHERE子句中
  - 查询的数据大于2%到4%
  - 表经常更新
  - 索引散列值,过于集中的值不要索引，例如:给性别"男","女"加索引,意义不大

> 索引不需要用，只是说我们在用name进行查询的时候，速度会更快。当然查的速度快了，插入的速度就会慢。因为插入数据的同时，还需要维护一个索引。

8.3.4 删除索引

- 使用DROP INDEX 命令删除索引

```
DROP INDEX 索引名 ON 表名;
```

```
DROP INDEX upper_last_name_idx
ON uppers;
```

- 只有索引的拥有者或拥有DROP ANY INDEX 权限的用户才可以删除索引
- 删除操作是不可回滚的