

第03章_基本的SELECT语句

1. SQL概述

1.1 SQL背景知识

- 1946 年，世界上第一台电脑诞生，如今，借由这台电脑发展起来的互联网已经自成江湖。在这几十年里，无数的技术、产业在这片江湖里沉浮，有的方兴未艾，有的已经几幕兴衰。但在这片浩荡的波动里，有一门技术从未消失，甚至“老当益壮”，那就是 SQL。
 - 45 年前，也就是 1974 年，IBM 研究员发布了一篇揭开数据库技术的论文《SEQUEL：一门结构化的英语查询语言》，直到今天这门结构化的查询语言并没有太大的变化，相比于其他语言，SQL 的半衰期可以说是非常长了。
- 不论是前端工程师，还是后端算法工程师，都一定会和数据打交道，都需要了解如何又快又准确地提取自己想要的数据。更别提数据分析师了，他们的工作就是和数据打交道，整理不同的报告，以便指导业务决策

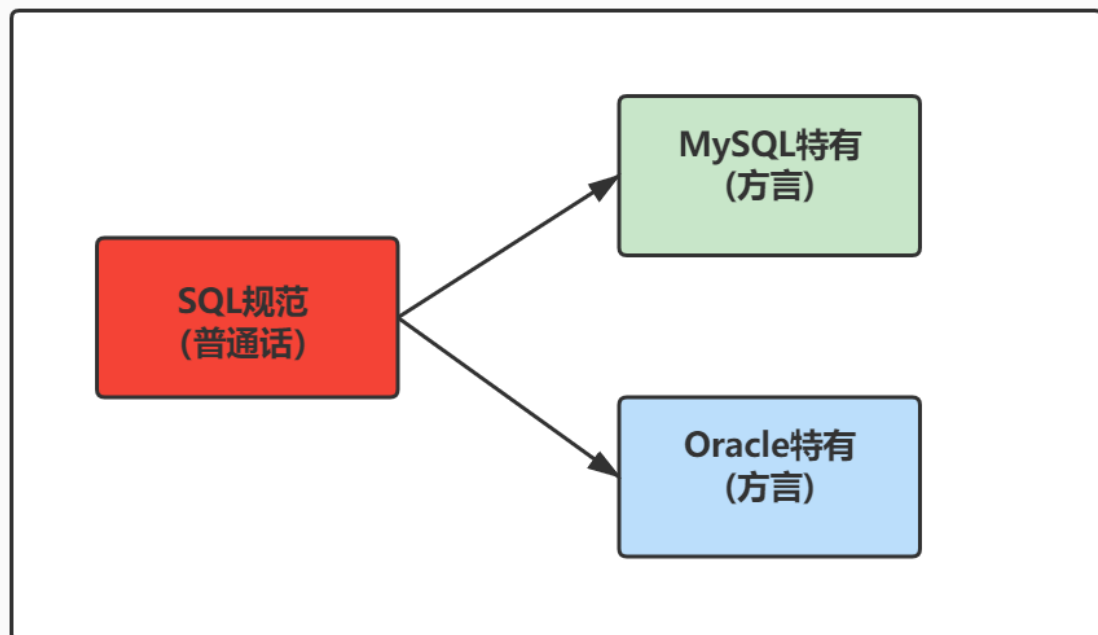
几个注意点：

1、SQL (Structured Query Language, 结构化查询语言) 是使用关系模型的数据库应用语言，与数据直接打交道，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

2、SQL 有两个重要的标准，分别是 SQL92 和 SQL99，它们分别代表了 92 年和 99 年颁布的 SQL 标准，我们今天使用的 SQL 语言依然遵循这些标准





3、不同的数据库生产厂商都支持SQL语句，但都有特有内容

SQL标准是由ANSI制定的；不同数据库管理系统对其实现有所不同；这些不同称为方言



1.2 SQL语言排行榜

自从 SQL 加入了 TIOBE 编程语言排行榜，就一直保持在 Top 10

| Oct 2021 | Oct 2020 | Change | Programming Language | | Ratings | Change |
|----------|----------|---|---|--------|---------|--------|
| 1 | 3 |  |  | Python | 11.27% | -0.00% |
| 2 | 1 |  |  | C | 11.16% | -5.70% |

1.3 SQL 分类

SQL语言在功能上主要分为如下3大类：

- **DDL (Data Definition Languages、数据定义语言)**
 - 这些语句定义了不同的数据库、表、视图、索引等数据库对象，还可以用来创建、删除、修改数据库和数据表的结构。
 - 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)**，用于定义数据库、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

1、DDL:数据定义语言:

CREATE 、 ALTER 、 DROP 、 RENAME 、 TRUNCATE

2、DML:数据操作语言:

INSERT 、 DELETE UPDATE 、 SELECT （重中之重）

3、DCL:数据控制语言:

COMMIT 、 ROLLBACK 、 SAVEPOINT 、 GRANT 、 REVOKE

提交回滚涉及事务;

savepoint设置保存点; 可以回滚到具体的保存点上

数据控制语言; 用于控制CRUD操作;

权限的赋予grant

权限的回收revoke

因为查询语句使用的非常的频繁，所以很多人把查询语句单拎出来一类：DQL（数据查询语言）

还有单独将 COMMIT、ROLLBACK 取出来称为TCL（Transaction Control Language，事务控制语言）

2. SQL语言的规则与规范

2.1 基本规则

- 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
- 每条命令以;或 \g 或 \G 结束

单独一行语句分号可加可不加; 建议加上

\g 或\G需要在命令行窗口中使用

```
mysql> select * from m_comment\g                                #这里\g
+----+-----+-----+-----+-----+-----+
| id | content                                | parent_id | blog_id | user_id | status |
created                                |
+----+-----+-----+-----+-----+-----+
| 15 | 风格差异---》测试评论                | -1 | 2 | 1 | 0 |
2022-06-12 09:50:45 |
| 17 | 17665661294----->测试回复          | 15 | 2 | 8 | 0 |
2022-06-12 16:19:31 |
```

```

+-----+-----+-----+-----+-----+
-----+
4 rows in set (0.00 sec)

mysql> select * from m_comment\G
***** 1. row *****
      id: 15
    content: 风格差异---》测试评论
parent_id: -1
   blog_id: 2
   user_id: 1
    status: 0
   created: 2022-06-12 09:50:45
***** 2. row *****
      id: 17
    content: 17665661294----->测试回复
parent_id: 15
   blog_id: 2
   user_id: 8
    status: 0
   created: 2022-06-12 16:19:31

4 rows in set (0.00 sec)

```

在sqlyoung中不能使用\g

查询: select * from m_commentg LIMIT 0, 1000

错误代码: 1146

Table 'blog.m_commentg' doesn't exist

执行耗时 : 0 sec

传送时间 : 0 sec

总耗时 : 0 sec

- 关键字不能被缩写也不能分行
- 关于标点符号
 - 必须保证所有的()、单引号、双引号是成对结束的
 - 必须使用英文状态下的半角输入方式
 - 字符串型和日期时间类型的数据可以使用单引号 (') 表示

2.2 SQL大小写规范（建议遵守）

- MySQL 在 Windows 环境下是大小写不敏感的



- **MySQL 在 Linux 环境下是大小写敏感的**
 - 数据库名、表名、表的别名、变量名是严格区分大小写的
 - 关键字、函数名、列名(或字段名)、列的别名(字段的别名) 是忽略大小写的
- **推荐采用统一的书写规范：**
 - 数据库名、表名、表别名、字段名、字段别名等都小写
 - SQL 关键字、函数名、绑定变量等都大写

2.3 注释

可以使用如下格式的注释结构

单行注释：#注释文字(MySQL特有的方式)
单行注释：-- 注释文字(--后面必须包含一个空格。)
多行注释：/* 注释文字 */

2.4 命名规则（暂时了解）

- 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
- 数据库名、表名、字段名等对象名中间不要包含空格
- 同一个MySQL软件中，数据库不能同名；同一个库中，表不能重名；同一个表中，字段不能重名
- 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。如果坚持使用，请在SQL语句中使用`（着重号）引起来
- 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

举例：

```
#以下两句是一样的，不区分大小写
show databases;
SHOW DATABASES;

#创建表格
#create table student info(...); #表名错误，因为表名有空格
create table student_info(...);

#其中order使用``飘号，因为order和系统关键字或系统函数名等预定义标识符重名了
CREATE TABLE `order`(
    id INT,
    lname VARCHAR(20)
);

select id as "编号", `name` as "姓名" from t_stu; #起别名时，as都可以省略
select id as 编号, `name` as 姓名 from t_stu; #如果字段别名中没有空格，那么可以省略""
select id as 编 号, `name` as 姓 名 from t_stu; #错误，如果字段别名中有空格，那么不能省略""
```

2.5 数据导入指令

1、在命令行客户端登录mysql，使用source指令导入

```
mysql> source d:\mysql\db.sql
```

```
mysql> desc employees;
```

| Field | Type | Null | Key | Default | Extra |
|----------------|-------------|------|-----|---------|-------|
| employee_id | int(6) | NO | PRI | 0 | |
| first_name | varchar(20) | YES | | NULL | |
| last_name | varchar(25) | NO | | NULL | |
| email | varchar(25) | NO | UNI | NULL | |
| phone_number | varchar(20) | YES | | NULL | |
| hire_date | date | NO | | NULL | |
| job_id | varchar(10) | NO | MUL | NULL | |
| salary | double(8,2) | YES | | NULL | |
| commission_pct | double(2,2) | YES | | NULL | |
| manager_id | int(6) | YES | MUL | NULL | |
| department_id | int(4) | YES | MUL | NULL | |

11 rows in set (0.00 sec)

2、使用图形化界面导入

3. 基本的SELECT语句

3.1 SELECT ... FROM...

- 语法:

SELECT 标识选择哪些列
FROM 标识从哪个表中选择

SELECT 1; #没有任何子句
SELECT 9/2; #没有任何子句

-- 简单的计算

SELECT 1 + 1, 3 * 2;

```
+-----+-----+
| 1 + 1 | 3 * 2 |
+-----+-----+
|      2 |      6 |
+-----+-----+
```

SELECT 1 + 1, 3 * 2
FROM DUAL; #dual: 伪表

```
+-----+-----+
| 1 + 1 | 3 * 2 |
+-----+-----+
|      2 |      6 |
+-----+-----+
```

-- 查询所有

-- *:表中的所有的字段（或列）

SELECT employee_id, last_name, salary
FROM employees;

```
+-----+-----+-----+
| employee_id | last_name | salary |
+-----+-----+-----+
|          100 | King      | 24000.00 |
|          101 | Kochhar   | 17000.00 |
|          205 | Higgins   | 12000.00 |
|          206 | Gietz     |  8300.00 |
+-----+-----+-----+
```

#查出来的是结果集ResultSet

一般情况下，除非需要使用表中所有的字段数据，最好不要使用通配符'*'。使用通配符虽然可以节省输入查询语句的时间，但是获取不需要的列数据通常会降低查询和所使用的应用程序的效率。通配符的优势是，当不知道所需要的列的名称时，可以通过它获取它们。

在生产环境下，不推荐你直接使用 `SELECT *` 进行查询

- 选择特定的列：

```
mysql> select employee_id,last_name,salary from employees;
```

| employee_id | last_name | salary |
|-------------|-----------|----------|
| 100 | King | 24000.00 |
| 101 | Kochhar | 17000.00 |
| 102 | De Haan | 17000.00 |
| 103 | Hunold | 9000.00 |
| 104 | Ernst | 6000.00 |
| 105 | Austin | 4800.00 |
| 106 | Pataballa | 4800.00 |
| 107 | Lorentz | 4200.00 |
| 108 | Greenberg | 12000.00 |
| 109 | Faviet | 9000.00 |
| 110 | Chen | 8200.00 |
| 111 | Sciarra | 7700.00 |
| 112 | Urman | 7800.00 |

#查出来的是结果集ResultSet

MySQL中的SQL语句是不区分大小写的，因此SELECT和select的作用是相同的，但是，许多开发人员习惯将关键字大写、数据列和表名小写，读者也应该养成一个良好的编程习惯，这样写出来的代码更容易阅读和维护

3.2 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。
- AS 可以省略
- 建议别名简短，见名知意
- 举例


```
# as:全称: alias(别名),可以省略
# 列的别名可以使用一对""引起来,不要使用''
-- 字符串和日期用单引号''

SELECT employee_id emp_id,last_name AS lname,department_id "部门id",salary * 12
AS "annual sal"
FROM employees;

+-----+-----+-----+-----+
| emp_id | lname      | 部门id | annual sal |
+-----+-----+-----+-----+
| 100    | King       | 90     | 288000.00  |
| 206    | Gietz      | 110    | 99600.00   |
+-----+-----+-----+-----+
```

必须要使用 "" 的场景: 别名中间有空格隔开

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

```
+-----+-----+
| Name      | Annual Salary |
+-----+-----+
| King      | 288000.00     |
| Kochhar   | 204000.00     |
| Higgins   | 144000.00     |
| Gietz     | 99600.00      |
+-----+-----+
```

3.3 去除重复行

在SELECT语句中使用关键字DISTINCT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;
```

```
#查询员工表中一共有哪些部门id呢?
#错误的:没有去重的情况
SELECT department_id
FROM employees;

+-----+
| department_id |
+-----+
| NULL          |
| 10             |
| 20             |
| 20             |
```

```

|          30 |
|          30 |
|         110 |
|         110 |
+-----+

```

#正确的：去重的情况

```

SELECT DISTINCT department_id
FROM employees;

```

```

+-----+
| department_id |
+-----+
|          NULL |
|           10 |
|           20 |
|           30 |
|           40 |
|           50 |
|           60 |
|           70 |
|           80 |
|           90 |
|          100 |
|          110 |
+-----+

```

错误的去重场景：

```

SELECT salary,DISTINCT department_id
FROM employees;

```

/*

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DISTINCT department_id

*/

#仅仅是没有报错，但是没有实际意义。【总体作为去重条件，只要不是两个都一样，就不会去重】

```

SELECT DISTINCT department_id,salary
FROM employees;

```

```

+-----+
| department_id | salary |
+-----+
|           90 | 24000.00 |
|           90 | 17000.00 |
|           60 | 9000.00 |
|           60 | 6000.00 |
|           60 | 4800.00 |
|           60 | 4200.00 |

```

```
|          110 | 12000.00 |
|          110 | 8300.00  |
+-----+-----+
```

这里有两点需要注意：

1. DISTINCT 需要放到所有列名的前面，如果写成 `SELECT salary, DISTINCT department_id FROM employees` 会报错。
2. DISTINCT 其实是对后面所有列名的组合进行去重，你能看到最后的结果是 74 条，因为这 74 个部门id不同，都有 salary 这个属性值。如果你想要看都有哪些不同的部门（department_id），只需要写 `DISTINCT department_id` 即可，后面不需要再加其他的列名了

3.4 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;
```

这里你一定要注意，在 MySQL 里面，空值不等于空字符串'' 一个空字符串的长度是 0，而一个空值的长度是空

而且，在 MySQL 里面，空值是占用空间的。

空值：null

null不等同于0, "", 'null'

-- 空值参与运算：结果一定也为空

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

```
+-----+-----+-----+-----+
| employee_id | 月工资    | 年工资    | commission_pct |
+-----+-----+-----+-----+
|          100 | 24000.00 | NULL     | NULL           |
|          101 | 17000.00 | NULL     | NULL           |
|          102 | 17000.00 | NULL     | NULL           |
|          175 | 8800.00  | 132000.00 | 0.25           |
|          176 | 8600.00  | 123840.00 | 0.20           |
|          177 | 8400.00  | 120960.00 | 0.20           |
|          178 | 7000.00  | 96600.00  | 0.15           |
|          206 | 8300.00  | NULL     | NULL           |
+-----+-----+-----+-----+
```

-- commission_pct有一部分值为null

引入单行函数IFNULL()

#实际问题的解决方案：引入IFNULL 不空，取前面，空取后面

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

| employee_id | 月工资 | 年工资 | commission_pct |
|-------------|----------|-----------|----------------|
| 100 | 24000.00 | 288000.00 | NULL |
| 101 | 17000.00 | 204000.00 | NULL |
| 102 | 17000.00 | 204000.00 | NULL |
| 103 | 9000.00 | 108000.00 | NULL |
| 104 | 6000.00 | 72000.00 | NULL |
| 105 | 4800.00 | 57600.00 | NULL |
| 205 | 12000.00 | 144000.00 | NULL |
| 206 | 8300.00 | 99600.00 | NULL |

3.5 着重号

- 错误的

排序的关键字是ORDER BY

form 后面要先跟表名，然后才能是ORDER BY

```
mysql> SELECT * FROM ORDER;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the right syntax to use near
'ORDER' at line 1
```

- 正确的

```
SELECT * FROM `order`;
+-----+-----+
| order_id | order_name |
+-----+-----+
| 1 | shkstart |
| 2 | tomcat |
| 3 | dubbo |
+-----+-----+
-- 当表名或列名为mysql关键字时，要使用着重号标识
```

- 结论

我们需要保证表中的**字段**、**表名**等没有和保留字、数据库系统或常用方法冲突。如果真的相同，请在SQL语句中使用一对``（着重号）引起来

即：当表名，字段名与数据库中关键字和保留字相同时；需要使用``

3.6 5、查询常数

SELECT 查询还可以对常数进行查询。对的，就是在 SELECT 查询结果中**增加一列固定的常数列**。这列的取值是我们指定的，而不是从数据表中动态取出的。

你可能会问为什么我们还要对常数进行查询呢？

SQL 中的 SELECT 语法的确提供了这个功能，一般来说我们只从一个表中查询数据，通常不需要增加一个固定的常数列，但如果我们想整合不同的数据源，用常数列作为这个表的标记，就需要查询常数。

比如说，我们想对 employees 数据表中的员工姓名进行查询，同时增加一列字段 corporation，这个字段固定值为“尚硅谷”，可以这样写：

```
SELECT '尚硅谷' as corporation, last_name FROM employees;
```

| corporation | last_name |
|-------------|-----------|
| 尚硅谷 | King |
| 尚硅谷 | Kochhar |
| 尚硅谷 | De Haan |
| 尚硅谷 | Higgins |
| 尚硅谷 | Gietz |

4. 显示表结构

使用DESCRIBE 或 DESC 命令，表示表结构

```
DESCRIBE employees; #显示了表中字段的详细信息
DESC employees;
```

| Field | Type | Null | Key | Default | Extra |
|----------------|-------------|------|-----|---------|-------|
| employee_id | int(6) | NO | PRI | 0 | |
| first_name | varchar(20) | YES | | NULL | |
| last_name | varchar(25) | NO | | NULL | |
| email | varchar(25) | NO | UNI | NULL | |
| phone_number | varchar(20) | YES | | NULL | |
| hire_date | date | NO | | NULL | |
| job_id | varchar(10) | NO | MUL | NULL | |
| salary | double(8,2) | YES | | NULL | |
| commission_pct | double(2,2) | YES | | NULL | |
| manager_id | int(6) | YES | MUL | NULL | |

```

| department_id | int(4) | YES | MUL | NULL | |
+-----+-----+-----+-----+-----+
DESC departments;
+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| department_id  | int(4)        | NO   | PRI | 0        |       |
| department_name | varchar(30)    | NO   |     | NULL     |       |
| manager_id     | int(6)        | YES  | MUL | NULL     |       |
| location_id    | int(4)        | YES  | MUL | NULL     |       |
+-----+-----+-----+-----+-----+

```

其中，各个字段的含义分别解释如下：

- Field：表示字段名称。
- Type：表示字段类型，这里 barcode、goodsname 是文本型的，price 是整数类型的。
- Null：表示该列是否可以存储NULL值。
- Key：表示该列是否已编制索引。PRI表示该列是表主键的一部分；UNI表示该列是UNIQUE索引的一部分；MUL表示在列中某个给定值允许出现多次。
- Default：表示该列是否有默认值，如果有，那么值是多少。
- Extra：表示可以获取的与给定列有关的附加信息，例如AUTO_INCREMENT等。

5. 过滤数据

#练习：查询90号部门的员工信息

```

SELECT employee_id,first_name,phone_number,salary
FROM employees
#过滤条件,声明在FROM结构的后面
WHERE department_id = 90;

```

```

+-----+-----+-----+-----+
| employee_id | first_name | phone_number | salary |
+-----+-----+-----+-----+
| 100 | Steven | 515.123.4567 | 24000.00 |
| 101 | Neena | 515.123.4568 | 17000.00 |
| 102 | Lex | 515.123.4569 | 17000.00 |
+-----+-----+-----+-----+

```

#练习：查询last_name为'King'的员工信息

```
SELECT employee_id,first_name,phone_number,salary
FROM EMPLOYEES
WHERE LAST_NAME = 'King';
```

| employee_id | first_name | phone_number | salary |
|-------------|------------|--------------------|----------|
| 100 | Steven | 515.123.4567 | 24000.00 |
| 156 | Janette | 011.44.1345.429268 | 10000.00 |

SQL在Window中不区分大小写；但是在字符串是区分大小写的；此处可以的效果是因为MySQL对ANSI的SQL标准实现上偷工减料，‘字符串’大小写不敏感

```
SELECT employee_id,first_name,phone_number,salary
FROM EMPLOYEES
WHERE LAST_NAME = 'kInG'; # MySQL对ANSI的SQL标准实现上偷工减料，‘字符串’大小写不敏感
```

| employee_id | first_name | phone_number | salary |
|-------------|------------|--------------------|----------|
| 100 | Steven | 515.123.4567 | 24000.00 |
| 156 | Janette | 011.44.1345.429268 | 10000.00 |

关于MySQL偷工减料小结：

- 1、字符串中的字母不区分大小写
- 2、字符串使用双引号，不报错#

课后练习

1、查询员工12个月的工资总和，并起别名为ANNUAL SALARY

#理解1：计算12月的基本工资

```
SELECT employee_id,last_name,salary * 12 "ANNUAL SALARY"
FROM employees;
```

| employee_id | last_name | ANNUAL SALARY |
|-------------|-----------|---------------|
| 100 | King | 288000.00 |
| 101 | Kochhar | 204000.00 |
| 206 | Gietz | 99600.00 |

#理解2: 计算12月的基本工资和奖金

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

| employee_id | last_name | ANNUAL SALARY |
|-------------|-----------|---------------|
| 100 | King | 288000.00 |
| 101 | Kochhar | 204000.00 |
| 206 | Gietz | 99600.00 |

2、查询employees表中去除重复的job_id以后的数据

```
SELECT DISTINCT job_id
FROM employees;
```

| job_id |
|------------|
| AC_ACCOUNT |
| AC_MGR |
| AD_ASST |
| AD_PRES |
| AD_VP |
| FI_ACCOUNT |
| FI_MGR |
| HR_REP |
| IT_PROG |
| MK_MAN |
| MK_REP |
| PR_REP |
| PU_CLERK |
| PU_MAN |
| SA_MAN |
| SA_REP |
| SH_CLERK |
| ST_CLERK |
| ST_MAN |

3、查询工资大于12000的员工姓名和工资

```
SELECT last_name,salary
FROM employees
WHERE salary > 12000;
```

| last_name | salary |
|-----------|--------|
|-----------|--------|


```

+-----+-----+
| King      | 24000.00 |
| Kochhar   | 17000.00 |
| De Haan   | 17000.00 |
| Russell   | 14000.00 |
| Partners  | 13500.00 |
| Hartstein | 13000.00 |
+-----+-----+

```

4. 查询员工号为176的员工的姓名和部门号

```

SELECT last_name, department_id
FROM employees
WHERE employee_id = 176;

```

```

+-----+-----+
| last_name | department_id |
+-----+-----+
| Taylor    | 80            |
+-----+-----+

```

5、显示表 departments 的结构，并查询其中的全部数据

```

DESCRIBE departments;

```

```

+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| department_id  | int(4)        | NO   | PRI | 0        |       |
| department_name | varchar(30)   | NO   |     | NULL     |       |
| manager_id     | int(6)        | YES  | MUL | NULL     |       |
| location_id    | int(4)        | YES  | MUL | NULL     |       |
+-----+-----+-----+-----+-----+-----+

```

```

SELECT * FROM departments;

```

```

+-----+-----+-----+-----+
| department_id | department_name | manager_id | location_id |
+-----+-----+-----+-----+
| 10            | Administration  | 200        | 1700        |
| 20            | Marketing       | 201        | 1800        |
| 270           | Payroll         | NULL       | 1700        |
+-----+-----+-----+-----+

```