软件过程/软件的生命周期

(一)软件定义期

1. **可行性研究阶段**

   ——《可行性研究报告》

   技术、人力、成本、时间、回报率、政策

2. **需求分析阶段**

   ——《软件需求说明书/规约》

   功能性需求： 完成的主要功能

   非功能性需求：可靠性、安全性、时效性、可维护性

(二)软件开发期

3. **概要设计阶段**

   ——《概要设计说明书》

   技术选型、子系统划分、模块功能描述、数据结构　架构师

4. **详细设计阶段**

   ——《详细设计说明书》

   函数、对象、方法、算法　　项目设计师

5. **编码实现阶段**

   ——《开发卷宗》

   UI设计师 => 前端工程师 => 后台工程师

6. **测试阶段**

   ——《测试报告》

   测试工程师

(三)软件维护期

7. **项目部署阶段**

   Develop Mode (PC) => Product Mode (Server)

   运维工程师 + 开发工程师

8. **项目维护阶段**

运维工程师 + 开发工程师

> Server：服务器，有两重含义
>
> 一种特别的电脑(硬件)，可以为其他电脑提供服务
> 一种特别的软件程序，可以运行起来为其他计算机提供服务

如何存储项目中的数据

- 内存中：访问速度快，但不是永久存储，容量有限
- 磁盘的文件中：存储容量大，永久存储，结构简单，但操作速度慢，存取效率低
- 云存储：可靠性高、费用低，但涉及到私有性问题
- 数据库服务器存储：磁盘+内存存储，数据保存时经过了特殊的优化，可以保证高效的存取操作

数据库服务器概述

(1)六十年代初：网状数据库

(2)六十年代末：树形/层次型数据库

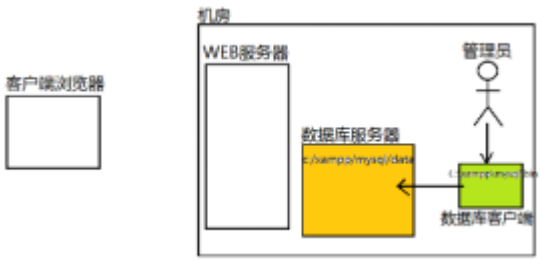
(3)七十年代至今：关系型数据库 —— 现在的主流

SQLite、MySQL、SQLServer、Oracle、DB2....

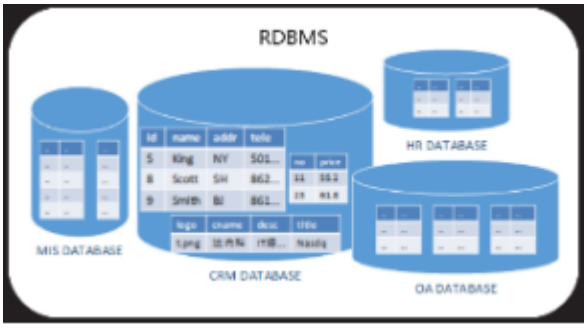
(4)最近几年：非关系型数据库/NoSQL —— 是关系型数据库的补充

MongoDB、Redis....

Oracle公司版本： www.mysql.com
MariaDB 基金会版本： www.mariadb.org



MySQL服务器中数据的逻辑结构



> Server => Database => Table => Row =>Column

MySQL基本命令

show databases;	显示所有数据库
quit	退出
use *db_name* ;	进入指定数据库
show tables;	查看当前库中的数据表
desc *tbl_name* ;	查看当前表中的列
drop database if exists *db_name* ;	丢弃指定数据库
create database *db_name* ;	创建新数据库
create table *tbl_name* (*col1_name col1_type,* *col2_name col2_type,* ...);	创建新数据表
insert into *tbl_name* values(*val1, val2, val3, ...*);	向指定表中插入一行数据
delete from *tbl_name* ;	删除所有行
delete from *tbl_name* where *where_definition* ;	删除满足特定条件的行
update *tbl_name* set *col1=val1, col2=val2, ...* where *where_definition* ;	更新 / 修改指定行
select * from *tbl_name* ;	查看指定表中所有的数据

> 注意：
>
> (1)SQL语言不区分大小写！习惯上，所有的SQL关键字都纯大写，非关键字都用小写

> (2)一条语句可以编写在多行中，但最后必须有一个英文分号！
>
> (3)SQL脚本中可以使用单行注释 #... 也可以使用多行注释 /*...*/

计算机如何保存字符

计算机中如何保存字符？

A-65	B-66	C-67
a-97	b-98	c-99

> 编码(Encode)：AABbCc => 656566986799
>
> 解码(Decode)：656566986799 => AABbCc

世界上常用的字符编码方案：

- ASCII编码：总共128个，对美国英语中常用的字符和标点都进行了编码
- GBK编码：总共3万+个，对中国大陆常用的简体字符和标点进行了编码，兼容ASCII 中-19978 国-26653
  BIG5编码：总共2万+个，对台湾常用繁体汉字和标点进行了编码，兼容ASCII
- Latin-1编码：总共256个，对西欧常用的字符和标点进行了编码，兼容ASCII
- Unicode字符集：总共6万+个，对世界上主流语言的主流符号都进行编码，兼容ASCII 中-37799 国-20055
  Unicode字符集在存储时采用多种特殊的方案，如UTF-8、UTF-16、UTF-32等

> 乱码问题的产生根源：字符串编码时和解码时所用的编码方案(字符集)不同。

小知识：为什么Windows下编写的网页名以及文件夹名不推荐使用中文？

Windows系统的默认字符集是GBK，
而服务器操作系统一般使用Linux/Unix，
它们的默认字符集是UTF-8！中文目录名和文件名上传到Linux都是乱码。

MySQL中的列类型

> `CREATE TABLE 表名( 列名 列类型, ....);`

- **数值类型 —— 数值数据可用引号也可不用**
  - TINYINT：微整数，占1字节，-128~127，如学生.年龄
  - SMALLINT：小整数，占2字节，-32768~32767，如部门.员工数量
  - INT：整数，占4字节，-2147483648~2147483647，如帖子.回复数
  - BIGINT：大整数，占8个字节，-9223372036854775808 ~ 9223372036854775807，用于科学计算领域
  - FLOAT(M, D)：单精度浮点型小数，底层存储为科学计数法形式，占4个字节，不超过3.4E38，表示范围大于INT，精度不如INT，可能产生四舍五入的精度丢失问题！

- DOUBLE(M, D)：双精度浮点型小数，底层存储为科学计数法形式，占8个字节, 不超过1.79769E308，表

  示范围大于BIGINT，精度不如BIGINT，可能产生四舍五入的精度丢失问题！
- DECIMAL(M, D)：定点小数，底层不存储为科学计数法，也不会产生精度丢失问题。 M表总的有效数，D表小数部分的有效位数，如笔记本.价格 DECIMAL(7, 2)； 如学生.高考成绩 DECIMAL(4, 1)
- BOOL：布尔类型，只能取值为 真(TRUE) 或 假(FALSE)，如用户.是否在线。提示：MySQL中没有真正的布尔数据，TRUE底层存储为1，FALSE底层存储为0. 注意：TRUE和FALSE是系统关键字，不能加引号！

- **日期时间类型 —— 日期时间数据必需用引号**
  - DATE：日期类型，如员工.生日 '1990-5-10'
  - TIME：时间类型，如学生.上课开始时间 '08:00:00'
  - DATETIME：日期时间类型，如帖子.发表时间 '2018-3-10 22:18:5'
- **字符串 —— 字符串数据必需用引号**
  - CHAR(M)：定长字符串，可能造成空间浪费，但存取速度快，适合于长度基本相当的数据，如：身份证号、书籍.ISBN号；M不能超过255
  - VARCHAR(M)：变长字符串，比较节省空间，但存储速度慢于定长字符串，适合于长度变化很大的数据，如员工.简历、帖子.内容；M不能超过65535
  - TEXT(M)：大型变长字符串，M不能超过2G

	CHAR(4)	**VARCHAR(4)**
'a'	a\0\0\0	a\0
'ab'	ab\0\0	ab\0
'abc'	abc\0	abc\0
'abcd'	abcd	abcd
'abcde'	abcd	abcd
'—'	—\0\0\0	—\0
'—二'	—二\0\0	—二\0
'—二三'	—二三\0	—二三\0
'—二三四'	—二三四	—二三四
'—二三四五'	—二三四	—二三四

行业小知识

二进制存储问题？
Bit位： 只能存储一个0或1
Byte字节：等于8个bit位

0000 0000 0
0000 0001 1
0000 0010 2
0000 0011 3
0000 0100 4 ....
0111 1111 127

1000 0000 -1
1000 0001 -2
....
1111 1111
-128
2个字节（=16bit）的表示范围：
0000 0000 0000 0000 0
0000 0000 0000 0001 1
0000 0000 0000 0010 2
0000 0000 0000 0011 3
....
0111 1111 1111 1111 32767

1000 0000 0000 0000 -1
1000 0000 0000 0001 -2
....
1111 1111 1111 1111 -32768

浮点小数：
1234.5678
123.4567*10^1 *123.45678E1*
*12.34567*10^2 12.345678E2
1.2345678*10^3 1.2345678E3
科学计数法表示
0.1234567*10^4 *0.12345678E4*
*12345.678*10-1 12345.678E-1
123456.78*10-2 123456.78E-2
....

列约束

> CREATE TABLE 表名(列名 列类型 列约束, ...)

> Constraint：约束，在插入数据时，要求必须满足特定的条件才能插入成功，否则就插入失败！如员工编号不能有重复值、性别只能取值为男或女、工资必须为正数、生日有范围检查、员工所在部门编号必须在部门

> 表中存在.....

SQL标准中定义了如下六种约束

(1) 唯一约束 —— UNIQUE

声明了唯一约束的列上不能出现重复的值。

(2) 非空约束 —— NOT NULL

声明为非空约束的列上不能出现NULL值。

(3) 主键约束 —— PRIMARY KEY

声明为主键的列上不能出现重复值，也不能出现NULL值

—— 表中的数据会自动安装主键列上的值由小到大排列，以提高查找效率

—— 一个表中至多只能有一个主键列！！

(4) 默认值约束 —— DEFAULT

声明了默认值约束的列可以指定一个默认的值。

在插入数据时，指定使用DEFAULT就可以自动采用前面指派的默认值。

(5) 检查约束 —— CHECK —— MySQL不支持

检查约束可用于检验数据的有效范围，如：

```
CREATE TABLE emp (
    age TINYINT  CHECK( age>=18  AND  age<=60)
);
```

(6)外键约束 —— foreign   key

声明了外键约束的列上，可以出现重复值或NULL值，

但只要出现了明确的值，该值必须在另一个表的主键列上曾经出现过

—— 即执行INSERT/UPDATE操作，都要参考一下另一个表的主键列上的已有值！

CREATE TABLE user(uname VARCHAR(32) PRIMARY KEY) 唯一 + 非空 + 排序； 一个表中最多只能有一个主键列
CREATE TABLE user(uname VARCHAR(32) NOT NULL UNIQUE) 唯一 + 非空； 一个表中可以有多个唯一且非空列

1. 小知识：项目中如何存储日期时间？

方式 1：使用VARCHAR来存储，如'2015-5-3'

不足：不会自动补零，不便于比较大小

方式 2：使用DATE/DATETIME来存储，如'2015-5-3'

好处：会自动补零，便于比较大小

不足：不便于实现国际化

方式 3：使用BIGINT数字来存储，如1513728000000

好处：便于实现国际化，使用后面学习的编程语言很容易实现格式的转换。

i18n：internationalization
国际化，一个实现了国际化的应用可以针对不同语言和风俗习惯的用户呈现不同的内容。 中国： 2015-10-25 美国： 10-25-2015 欧洲： 25/10/2015
计算机如何存储日期时间？——使用数字——表示距离计算机元年经过了多少毫秒 1970-1-1 0:0:0 0 1970-1-1 0:0:1 1000 1970-1-1 0:1:0 1000*60 1970-1-1 1:0:0 1000*3600 1970-1-2 0:0:0 1000*3600*24 1971-1-1 0:0:0 1000*3600*24*365 1969-1-1 0:0:0 -1000*3600*24*365 1971-1-1 0:0:0 1000*3600*24*365*48

2. 小知识：MySQL专用关键字—— **auto_increment**

可以为整形的主键列(INT PRIMARY KEY)上添加一个AUTO_INCREMENT关键字，实现“自动增长的列”。

```
CREATE TABLE user (
    uid INT  PRIMARY KEY  AUTO_INCREMENT
);
INSERT INTO user VALUES( 10 );   #自增列允许手工赋值
INSERT INTO user VALUES( NULL ); #在已有最大值基础上+1
```

SQL简单查询

查询特定的列

示例：查询出所有员工的姓名和工号

```
SELECT  ename, eid  FROM  emp ;
```

练习：查询所有部门的名称及其所在地

```
SELECT  dname, location  FROM  dept ;
```

练习：查询所有员工的姓名、生日、工资

```
SELECT  ename,  birthday,  salary
```

```
FROM  emp;
```

练习：查询所有员工的所在部门编号、性别、姓名、性别

```
SELECT  deptId,  sex,  ename,  sex
FROM  emp ;
```

示例：查询员工的编号、姓名、生日、工资、性别、所在部门编号

```
SELECT  eid, ename, birthday, salary, sex, deptId
FROM  emp ;
SELECT  *  FROM  emp ;  #SQL中的*指代所有列
```

给列取别名

示例：查询所有员工的编号和姓名，列名用中文显示

```
SELECT  eid  AS  编号,  ename  AS  姓名
FROM  emp ;   #as翻译为"看作是...."
```

练习：查询出所有员工的姓名、工资、生日、所在部门编号，列名用中文显示

```
SELECT  ename 姓名, salary 工资, birthday 生日, deptId 部门编号
FROM  emp;        #AS关键字可以省略
```

练习：查询出所有员工的姓名、工资、生日、所在部门编号，每个列名都用一个英文字符显示

```
SELECT  ename  n, salary  s, birthday  b, deptId  i
FROM  emp;
```

只显示不同的值/相同的值只显示一次

示例：查询出有员工的部门的编号

```
SELECT  DISTINCT  deptId
FROM  emp;   #distinct：不同的
```

练习：查询出当前公司有哪些性别的员工

```
SELECT  DISTINCT  sex  FROM  emp;
```

查询时执行计算

示例：计算一个算式的结果

SELECT (1+2)*3/4 ;

示例：查询出所有员工的姓名、月薪、年薪

SELECT ename, salary AS 月薪, salary*12 AS 年薪

FROM emp;

练习：假设每个员工了基本月薪外，每月还有500补助，每年底还有一次性的年终奖20000，查询出每个员工每月的实际平均工资

SELECT ename, ((salary+500)*12+20000)/12 实际月薪

FROM emp;

练习：假设每人每月缴纳20%的个税，计算所有员工的姓名以及实际到手的工资

SELECT ename, salary*(1-0.2)

FROM emp;

单条件查询 —— 小难点

提示：SQL中可用的比较运算符：

= < <= = != (不等于)

示例：查询出编号为7788的员工所有信息

```
SELECT * FROM emp
WHERE eid=7788 ;  #where：满足...条件
```

练习：查询出20号部门的部门名称

```
SELECT dname FROM dept
WHERE did=20 ;
```

练习：查询出TOM的工资和生日

```
SELECT salary, birthday FROM emp
WHERE ename='TOM' ;
```

练习：查询出工资在6000及以上的员工所有信息

```
SELECT * FROM emp
WHERE salary >= 6000;
```

练习：查询出所有女员工的姓名和生日

```
SELECT ename, birthday FROM emp
WHERE sex=0;
```

练习：查询出20号部门的所有员工姓名

```
SELECT ename FROM emp
WHERE deptId=20;
```

练习：查询出不在20号部门的所有员工信息

```
SELECT * FROM emp
WHERE deptId != 20;
```

练习：查询出不在任何一个部门的员工所有信息

```
SELECT * FROM emp
WHERE deptId = NULL;  #查不到任何记录!!! NULL就不能使用=或!=运算！
```

注意：计算机中的NULL指具有“不确定的值”；

不与任何值相等，也不与任何值不等，甚至不与自己相等或不等！

```
SELECT * FROM emp
WHERE deptId IS NULL;          #正确
SELECT * FROM emp
WHERE deptId IS NOT NULL;      #正确
```

多条件查询

> 多条件并列出现时需要使用如下两个关键字之一拼接：
>
> 条件1 AND(并且) 条件2 —— 两个条件必须都满足
>
> 条件1 OR(或者) 条件2 —— 两个条件满足一个即可

示例：查询出10号部门的女员工所有信息

```
SELECT * FROM emp
WHERE deptId=10 AND sex=0;
```

练习：查询出工资小于6000的男员工所有信息

```
SELECT * FROM emp
WHERE salary<6000 AND sex=1;
```

练习：查询出出生晚于1990-1-1的20号部门员工信息

```
SELECT * FROM emp
WHERE birthday>'1990-1-1' AND deptId=20;
```

练习：查询出10号和30号部门的员工所有信息

```
SELECT * FROM emp
WHERE deptId=10 OR deptId=30;
```

练习：查询出工资在6000~8000之间的员工所有信息

```
SELECT * FROM emp
WHERE 6000<salary<8000; #错误
WHERE salary>6000 AND salary<8000;
```

练习：查询出工资在6000以下，以及8000以上的员工所有信息

```
SELECT * FROM emp
WHERE salary<6000 OR salary>8000;
```

练习：查询出在1990年出生的员工所有信息

```
SELECT * FROM emp
WHERE birthday>='1990-1-1' AND birthday<='1990-12-31';
```

模糊条件查询

> SQL中使用 % 表示“任意多个任意字符”；
>
> SQL中使用 _ 表示“任意一个任意字符”；
>
> 模糊条件查询不能用 = 判定！必须使用 LIKE 代替（看起来像）。

示例：查询出姓名中包含字符E的员工所有信息

```
SELECT * FROM emp WHERE ename='E'; #错误
SELECT * FROM emp WHERE ename='%E%'; #错误
SELECT * FROM emp WHERE ename LIKE '%E%';
```

练习：查询出姓名以E开头的所有员工信息

```
SELECT * FROM emp WHERE ename LIKE 'E%';
```

练习：查询出姓名以E结尾的所有员工信息

```
SELECT * FROM emp WHERE ename LIKE '%E';
```

练习：查询出姓名倒数第二个字符为E的所有员工信息

```
SELECT * FROM emp WHERE ename LIKE '%E_';
```

查询结果的排序

示例：查询出所有的员工信息，要求按工资由小到大排列

```
SELECT * FROM emp
ORDER BY salary; #order：排序 by：按照
```

```
SELECT * FROM emp
ORDER BY salary ASC ;  #ascendant：升序
SELECT * FROM emp
ORDER BY salary DESC ;  #descendant：降序
```

练习：查询出 10 号部门所有的员工信息，按照姓名由小到大排列

```
SELECT * FROM emp
WHERE deptId = 10
ORDER BY ename ASC ;
```

练习：查询出所有的员工信息，按照姓名由大到小排列

```
SELECT * FROM emp ORDER BY ename DESC ;
```

练习：查询出所有的员工信息，按照年龄由大到小排列

```
SELECT * FROM emp ORDER BY birthday ASC;
```

练习：查询出所有的员工信息，按照工资由大到小排列，工资相同的员工再按姓名由小到大排序

```
SELECT * FROM emp
ORDER BY salary DESC , ename ASC ;
```

> 常用的SQL语句：
>
> ```
> set names utf 8;
> drop database if exists 库名;
> create database 库名charset=utf8;
> use 库名;
> create table 表名(列名列类型 列约束, ...);
> insert into 表名 values(值, ...);
> delete from 表名where 条件;
> update 表名 set 列=值, 列=值 where 条件;
> select * from 表名;
> ```
>
> 列类型：
>
> 数值类型：
>
> tinyint / smallint / int / bigint
>
> float / double / decimal
>
> bool
>
> 日期时间类型：

> date / time / datetime
>
> 字符串类型：
>
> char / varchar / text
>
> 列约束：
>
> unique / not null / primary key / default/ check / foreign key (列) references 表(列)
>
> 简单查询：
>
> 查询特定的列： select ename,salaryfrom emp;
>
> 给列取别名：select ename as n,salary月薪 from emp;
>
> 只显示不同的值： select distinct sex from emp;
>
> 执行运算： select salary*12 fromemp;
>
> 单条件查询： select ename from empwhere salary=7000;
>
> 多条件查询： select * from empwhere deptid=10 and sex=0;
>
> 模糊条件查询： select * from empwhere ename like '__E%';
>
> 结果排序： select * from emp orderby salary desc, ename asc;

分页查询

> 当可显示的数据行数非常多时，无需一次性全部显示，
>
> 可以每一次显示N条(称为页面大小)，逐页显示出来
>
> ——第1页、第2页....

> 提示：不同的数据库分页查询语句各不相同，MySQL的分页查询最简单。

语法：

SELECT ....

FROM ....

WHERE ....

ORDER BY ....

LIMIT start, count ;

start和count都是一个数字。

start表示从哪一行开始读取数据，count表示一次最多可以读取多少行。

假设一页最多显示5行记录，那么：

第1页： ... LIMIT 0, 5; #0/1/2/3/4

第2页： ... LIMIT 5, 5; #5/6/7/8/9

第3页： ... LIMIT 10, 5; #10/11/12/13/14

第4页： ... LIMIT 15, 5; #15/16/17/18/19

.....

第N页： **... LIMIT (N-1)*5, 5**;

练习：假设页面大小为6，查询所有员工的信息，分页显示第1页；

```
SELECT *  FROM  emp LIMIT  0, 6;
```

练习：假设页面大小为6，查询所有员工的信息，分页显示第2页；

```
SELECT *  FROM  emp LIMIT  6, 6;
```

练习：假设页面大小为6，查询所有员工的信息，分页显示第3页；

```
SELECT *  FROM  emp LIMIT  12, 6;
```

练习：假设页面大小为6，查询所有员工的信息，分页显示第4页；

```
SELECT *  FROM  emp LIMIT  18, 6;
```

练习：假设页面大小为6，查询10号部门员工的所有信息，按工资由大到小排序，分页显示第1页；

```
SELECT *
FROM emp
WHERE deptId=10
ORDER BY  salary  DESC
LIMIT 0, 6;
```

练习：假设页面大小为6，查询10号部门员工的所有信息，按工资由大到小排序，分页显示第2页；

```
SELECT *
FROM emp
WHERE deptId=10
ORDER BY  salary  DESC
LIMIT 6, 6;
```

MySQL复杂查询

聚合查询

> 函数：Function，功能体，**接收**若干数据，加以**处理**，**返回**运算的结果——饺子机。
>
> SQL标准中提供了5个聚合函数(把若干数据放在一起加以处理)：
>
> COUNT(数据)、SUM(数据)、AVG(数据)、MAX(数据)、MIN(数据)

示例：查询出所有员工的数量 —— 数一数数据的行数

```
SELECT COUNT(*)  FROM emp;
```

练习：查询出公司中部门的数量，列名取为“部门数量”

```
SELECT COUNT(*) AS  部门数量  FROM dept;
```

示例：查询出所有员工工资的总和 —— 14个工资求和

```
SELECT SUM(salary)  FROM emp;
```

练习：查询出所有员工的平均工资（用两种写法实现）

```
SELECT  SUM(salary) / COUNT(*)  FROM emp;
SELECT AVG(salary)  FROM emp;
```

练习：查询出所有员工的工资最大值和最小值

```
SELECT  MAX(salary),MIN(salary)  FROM  emp;
```

练习：查询出所有员工的生日最大值和最小值

```
SELECT  MAX(birthday), MIN(birthday)  FROM emp;
```

示例：每个部门中的部门编号及员工数量——先分组再聚合运算

```
SELECT deptId, COUNT(*)
FROM emp
GROUP  BY deptId;      #group：小组、分组
```

练习：查询出每个部门的编号以及该部门员工工资最大值、最小值和平均值

```
SELECT deptId, MAX(salary), MIN(salary), AVG(salary)
FROM emp
GROUP  BY deptId ;
```

练习：查询出男性和女性员工的生日最大值和最小值

```
SELECT sex, MAX(birthday), MIN(birthday)
FROM emp
GROUP  BY  sex;
```

子查询

示例：查询出工资比SCOTT高的员工所有信息

步骤1：查询出SCOTT的工资（假设为X） —— 子查询

```
SELECT salary FROM emp WHERE ename='SCOTT';
```

步骤2：查询出工资高于X的员工所有信息 —— 父查询

```
SELECT * FROM emp WHERE salary>6000;
```

综合上述两步： 步骤2依赖于步骤1

```
SELECT * FROM emp WHERE salary>(
    SELECT salaryFROM emp WHERE ename='SCOTT'
); #在一个父查询中嵌套一个子查询
```

练习：查询出年龄比SCOTT大的员工所有信息

步骤1：查询出SCOTT的生日（假设为X）

```
SELECT birthday FROM emp WHEREename='SCOTT';
```

步骤2：查询出生日比X小的员工所有信息

```
SELECT * FROM emp WHEREbirthday<'1990-1-1';
```

综合上述两步：

```
SELECT * FROM emp WHERE birthday<(
    SELECTbirthday FROM emp WHERE ename='SCOTT'
);
```

练习：查询出“研发部”所有员工的工号和姓名

步骤1：到部门表查询出研发部的部门编号（假设为X）

```
SELECT did  FROM  dept WHERE  dname='研发部';
```

步骤2：到员工表查询部门编号为X的员工工号和姓名

```
SELECT eid,ename FROM emp WHEREdeptId=10;
```

综合：

```
SELECT eid,ename FROM emp WHERE deptId=(
    SELECT did FROMdept WHERE dname='研发部'
);
```

练习：查询出工资比公司平均工资高的员工的数量

步骤1：查询出所有员工的平均工资(假设为X)

```
SELECT AVG(salary) FROM emp;
```

步骤2：查询出工资大于X的员工的数量

```
SELECT COUNT(*) FROM emp WHERESalary>6842.78;
```

综合：

```
SELECT COUNT(*) FROM emp WHERE salary>(
    SELECTAVG(salary) FROM emp
);
```

练习：查询出工资比TOM高，年龄比MARTEN小的员工工号和姓名

```
SELECT eid,ename
FROM emp
WHERE salary>( )  AND  birthday > ( ) ;
```

跨表查询 / 多表查询

示例：查询出每个员工的姓名及其所在部门的名称

```
SELECT ename,  dname
FROM emp,  dept;    #错误！！产生了笛卡尔积
SELECT ename, dname
FROM emp,  dept
WHERE  deptId=did ;   #跨表查询—— 必须防止笛卡尔积！
```

> 上述语法是SQL-92标准中的语法，有一定的不足
>
> ——无法显示出没有部门的员工（如KING），
>
> 也无法显示出没有员工的部门（如40号测试部）。
>
> SQL-99标准中改进了此问题，将跨表查询分为四种类型：

(1)内连接查询

——INNER JOIN...ON——效果与SQL-92一样

```
SELECT  ename,  dname
FROM  emp INNER  JOIN  dept
ON  deptId = did ;
```

(2)左外连接查询

——LEFT OUTER JOIN...ON——会把左侧表中的所有数据全部显示出来，即使在右侧表中它没有对应的项

```
SELECT ename, dname
FROM emp LEFT  OUTER  JOIN dept
ON  deptId = did ;
```

(3)右外连接查询

——RIGHT OUTER JOIN...ON——会把右侧表中的所有数据全部显示出来，即使在左侧表中它没有对应的项

```
SELECT ename, dname
FROM emp  RIGHT  OUTER JOIN  dept
ON  deptId = did ;
```

(4)全连接查询——FULL JOIN

——会把左侧表和右侧表中所有的记录全部显示一遍，即使对方表中没有对应的记录 13 + 1 + 1

```
SELECT ename, dname
FROM emp  FULL  JOIN dept
ON  deptId = did ;   #MySQL不支持全连接
```

练习：查询出员工姓名及其所在部门的名称，必须每个部门都至少显示一次（用两种语法）

```
SELECT ename, dname
FROM emp  RIGHT OUTER JOIN dept
ON deptId=did ;
SELECT ename, dname
FROM dept  LEFT OUTER JOIN  emp
ON deptId=did ;
```

补充小知识：

> MySQL如何解决不支持全连接的问题？
>
> 结果集的合并：union，把多个结果集合并为一个结果集。

示例：请查询出所有的美国员工和中国员工的姓名和工资

```
(SELECT ename,salary  FROM  emp_us)
UNION                    #合并两个集合中的相同行
(SELECT ename,salary  FROM  emp_cn) ;
```

```
(SELECT ename,salary  FROM  emp_us)
UNION  ALL   #若两个集合中有相同行则全部显示，不合并
(SELECT ename,salary  FROM  emp_cn) ;
```

> 如何使用结果集的合并实现“全连接”的效果？

```
( SELECT ename,dname FROM emp LEFT JOINdept ON deptId=did )
UNION
( SELECT ename,dname FROM emp RIGHT JOINdept ON deptId=did ) ;
```