## 数据库

## 01、进入mysql数据库：

--->显示密码：mysql> mysql -uroot -p123456

--->隐藏密码：mysql> mysql -uroot -p

password:\*\*\*\*\*\*

## 02、退出数据库

mysql> exit

## 03、显示数据库（所有）

mysql> show databases；

## 04、创建数据库

mysql> create database mysql;//创建了mysql名的数据库

## 05、删除数据库

mysql> drop database mysql；

## 06、选择数据库

mysql> use mysql;

## 07、显示数据库的表名

mysql> show tables；

## 08、删除数据库的表

mysql> drop table users；

## 09、显示数据库的表的详细信息

mysql> describe users；//也可以写成desc users；

## 10、创建表

<--

not null：不为空

primary key:主键

default：默认值（不写的话默认null）

-->

mysql> create table t\_user4(

-> id int not null primary key default '';

-> );

<--创建表的过程中插入另一个同一结构的表的信息 -->

create table user5 as select \* from user;

## 11、查询表的信息

mysql> select \* from t\_user;

## 12、重命名表

mysql> rename table t\_user4 to user;

## 13、复制表

mysql> create table user2 like user;

## 14、模糊查询

<--查询姓叶的用户信息 -->

select \* from t\_user3 where user\_username like '叶%';

<--查询名字中带叶的用户信息 -->

select \* from t\_user3 where user\_username like '%叶%';

## 15、插入信息

mysql> insert into t\_user3 values ('8','mimi','112233');

mysql> insert into t\_user3(user\_id,user\_username,user\_password) values('8','mimi','112233');

mysql> insert into user1 select \* from user;

## 16、更新信息

mysql> update t\_user3 set user\_username = '李四' where user\_id = 8;

## 17、删除信息

mysql> delete from t\_user3 where user\_id = 8;

--删除表数据 (知识删除表中的数据，表结构不会变)

delete from new\_dept;

truncate table new\_dept;

## 18、增加一个字段

mysql> alter table user add pwd char(20) not null default '';

在某一字段后面增加一个新的字段

mysql> alter table user add info char(20) not null default '' after id;

在表开始增加一个字段

mysql> alter table user add info2 char(20) not null default '' first;

## 19、删除一个字段

mysql> alter table user drop info2;

## 20、修改一个字段

数据库中修改表中列的（字段）方法

alter table <表名> modify <字段名> 类型...

<-- 将user表中phone的字段类型由char(20)变为了char(30) -->

eg:alter table user modify phone char(30) not null default '';

alter table <表名> modify <旧字段名> <新字段名> 类型...

< --将原来数据库表的mobile字段修改为phone，还可以修改字段的类型 -- >

eg: alter table user change mobile phone char(20) not null default '';

区别：modify仅可以修改字段的类型，不可以修改字段的名称

而change则不仅可以修改字段名，还可以修改字段的类型的值。

## 21、关于存放日期时间

date：表示日期（yyyy-mm-dd）

<--增加字段 -->

mysql> alter table user add year date not null ;

<--将信息插入表中-->

mysql> insert into user values (5,'aaa','kzw','123456','1999-12-12');

mysql> insert into user values (6,'aaa','kzw','123456','19991212');

mysql> insert into user values (7,'aaa','kzw','123456','991212');

time: 表示时间（HH:mm:ss）

<--增加时间time的字段 -->

mysql> alter table user add time time not null ;

mysql> insert into user values (8,'aaa','kzw','123456','19991024','222222');

datetime:表示date+time

mysql> alter table user add datetime datetime not null ;

mysql> insert into user values (9,'sss','abc','654321','20001024','111111','2000-10-20 10:20:32');

## 22、多表查询（内连接 外连接）

--SQL1992连接查询语法

--内连接：查询仅返回满足连接条件的记录

--外连接：不仅返回满足连接条件的记录，不满足连接条件的也返回

--左连接（左外连接）是以左表为基准，将两表的数据进行连接，然后将左表没有的对应项显示，右表的列为NULL

--右连接（右外连接）是以右表为基准，将两表的数据进行连接，然后将右表没有的对应项显示，左表的列为NULL

--全连接：完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之 间有匹配行，则整个结果集行包含基表的数据值。

--交叉连接：交叉联接返回左表中的所有行，左表中的每一行与右表中的所有行组合。交叉联接也称作笛卡尔积。

--查询员工编号，姓名，部门名称（笛卡尔集）

SELECT employee\_id,

last\_name,

department\_name

FROM employees,

departments;

--查询员工编号，姓名，部门名称

SELECT employee\_id,

last\_name,

department\_name

FROM employees,

departments

WHERE employees.department\_id = departments.department\_id;

--查询员工编号，姓名，部门编号,部门名称（表别名）

SELECT e.employee\_id,

e.last\_name,

d.department\_id,

d.department\_name

FROM employees e,

departments d

WHERE e.department\_id = d.department\_id;

--查询员工编号，姓名，部门编号,部门名称，职务编号，职务名称

SELECT e.employee\_id,

e.last\_name,

d.department\_id,

d.department\_name,

j.job\_id,

j.job\_title

FROM employees e,

departments d,

jobs j

WHERE e.department\_id = d.department\_id

AND e.job\_id = j.job\_id

AND e.salary >= 8000

ORDER BY e.salary DESC;

--查询每个员工编号，姓名，工资，工资级别（非等值连接）

SELECT e.employee\_id,e.last\_name,e.salary,g.grade\_level

FROM employees e,job\_grades g

WHERE e.salary BETWEEN g.lowest\_sal AND g.highest\_sal

ORDER BY e.employee\_id;

--左连接

左连接：

以左表为基础,显示左表中的所有列,不管是否与关联条件相匹配,而右表中的数据只显示与关联条件相匹配的列,不匹配的列以NULL字符填充.

select \* from t\_A a left join t\_B b on a.id = b.id;

或

select \* from t\_A a left outer join t\_B b on a.id = b.id;

LEFT JOIN是以左表的记录为基础的,示例中t\_A可以看成左表,t\_B可以看成右表,它的结果集是t\_A表中的全部数据，

再加上t\_A表和t\_B表匹配后的数据。换句话说,左表(t\_A)的记录将会全部表示出来,而右表(t\_B)只会显示符合搜索

条件的记录。t\_B表记录不足的地方均为NULL。

用（+）来实现：

Select \* from t\_A a,t\_B b where a.id=b.id(+);

这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。如果加号写在右表，

左表就是全部显示，所以是左连接。

--全外连接

全外连接（FULL OUTER JOIN/FULL JOIN）：

左表就是全部显示，所以是左连接。

左表和右表都不做限制，所有的记录都显示，两表不足的地方均为NULL。 全外连接不支持（+）写法。

select \* from t\_A a full join t\_B b on a.id = b.id;

或

select \* from t\_A a full outer join t\_B b on a.id = b.id;

## 23、数据库中的事务的特性（ACID）

--原子性：任何事务都是一个整体，不可分割，要么一起成功执行，要么一起失败回退

--一致性：必须从一个正确的状态直接切换到另一个正确的状态

--隔离性：一个会话看不到另一个会话未提交的事务操作，一个事务执行不能被其他事务干扰

--持久性：提交事务以后，事务操作会被持久化到数据库中，不可回退，别的会话可以看到事务操作的结果， 一个事务一旦提交，他对数据库中数据的改变就是永久性的

并发控制的方式：

乐观锁：乐观锁认为一个用户读数据的时候，别人不会去写自己所读的数据；

悲观锁：悲观锁就刚好相反，觉得自己读数据库的时候，别人可能刚好在写自己刚读的数据，其实就是持一种比较保守的态度；

时间戳：时间戳就是不加锁，通过时间戳来控制并发出现的问题。

《三种方式读取数据时详情》

**悲观锁**就是在读取数据的时候，为了不让别人修改自己读取的数据，就会先对自己读取的数据加锁，只有自己把数据读完了，才允许别人修改那部分数据，或者反过来说，就是自己修改某条数据的时候，不允许别人读取该数据，只有等自己的整个事务提交了，才释放自己加上的锁，才允许其他用户访问那部分数据。

**乐观锁**就比较简单了，就是不做控制，这只是一部分人对于并发所持有的一种态度而已。

**时间戳**就是在数据库表中单独加一列时间戳，比如“TimeStamp”，每次读出来的时候，把该字段也读出来，当写回去的时候，把该字段加1，提交之前 ，跟数据库的该字段比较一次，如果比数据库的值大的话，就允许保存，否则不允许保存，这种处理方法虽然不使用数据库系统提供的锁机制，但是这种方法可以大大提高数据库处理的并发量，因为这种方法可以避免了长事务中的数据库加锁开销。

并发操作还会造成活锁和死锁。

活锁指的是T1封锁了数据R，T2同时也请求封锁数据R，T3也请求封锁数据R，当T1释放了锁之后，T3会锁住R，T4也请求封锁R，则T2就会一直等待下去，这种处理方法就是采用“先来先服务”策略；

死锁就是我等你，你又等我，双方就会一直等待下去，比如：T1封锁了数据R1，正请求对R2封锁，而T2封住了R2,正请求封锁R1，这样就会导致死锁，死锁这种没有完全解决的方法，只能尽量预防，

**死锁预防的方法**有：

①一次封锁发，指的是一次性把所需要的数据全部封锁住，但是这样会扩大了封锁的范围，降低系统的并发度；

②顺序封锁发，指的是事先对数据对象指定一个封锁顺序，要对数据进行封锁，只能按照规定的顺序来封锁，但是这个一般不大可能的。

另外，系统如何判断出现死锁呢，毕竟出现死锁不能一直干等下去，要及时发现死锁同时尽快解决出现的死锁，诊断和判断死锁有两种方法，一是超时法，二是等待图法。超时法就是如果某个事物的等待时间超过指定时限，则判定为出现死锁；等待图法指的是如果事务等待图中出现了回路，则判断出现了死锁。对于解决死锁的方法，只能是撤销一个处理死锁代价最小的事务，释放此事务持有的所有锁，同时对撤销的事务所执行的数据修改操作必须加以恢复。

## 24、组函数

所有组函数都是忽略空值的。例如在进行count统计时，如果值为null，则不进行统计。

SELECT SUM(salary),AVG(salary),MAX(salary),MIN(salary),COUNT(salary)

FROM employees

WHERE department\_id=50;

消除重复项：

搜索department\_id排除重复

select DISTINCT department\_id from EMPLOYEES;

统计department\_id的数量

SELECT COUNT(DISTINCT department\_id) FROM employees;

## 25、分组查询

单个分组查询：

--查询每个部门的ID，员工工资总和，最高工资

SELECT department\_id,SUM(salary) sal\_sum,MAX(salary)

FROM employees

WHERE department\_id IS NOT NULL

GROUP BY department\_id

ORDER BY sal\_sum DESC;

多个分组查询：

--多个分组表达式

SELECT department\_id,job\_id,SUM(SALARY)

FROM employees

GROUP BY department\_id,job\_id;

## 26、to\_char和to\_date的区别

to\_date：把实际数据转化为数据库保存的时间格式的数据去进行比较

SELECT EMPLOYEE\_ID,LAST\_NAME,HIRE\_DATE from EMPLOYEES where HIRE\_DATE > to\_date('1997-11-11','yyyy-mm-dd');

to\_char：把数据库的信息转化为指定的格式，然后去进行比较

SELECT EMPLOYEE\_ID,LAST\_NAME,HIRE\_DATE from EMPLOYEES where to\_char(HIRE\_DATE,'yyyy-mm-dd') > '1997-11-11';

注意：oracle的数据库不区分大小写，所以月份和分钟会显示一样 为了区分月份和分钟 月份（mm）分钟（mi）

<--mi显示的分钟 2018-11-05 14:18:56（正确时间）-->

select to\_char(sysdate,'yyyy-MM-dd HH24:mi:ss') from dual;

<--mm显示的月份 2018-11-05 14:11:02-->

select to\_char(sysdate,'yyyy-MM-dd HH24:mm:ss') from dual;

## 27、having和where的区别

having：过滤分组之后的数据

where：过滤分组之前的数据

--查询每个部门的ID，员工工资总和

SELECT department\_id ,SUM(salary) sal\_sum

FROM employees

WHERE department\_id IS NOT NULL

GROUP BY department\_id

HAVING SUM(salary)>=30000

ORDER BY department\_id

## 28、创建表空间

create tablespace userspace

datafile 'D:\test.dbf'

size 100m

autoextend on next 20m

maxsize 1024m

userspace:表空间名

datafile:为表空间创建数据文件，后面写地址。

size ：表空间的大小

autoextend:是否允许自动扩展数据文件，off：代表禁止 on：代表允许 next：指定当需要更多盘区时分配给数据文件的磁盘空间。

maxsize：允许分配给数据文件的最大磁盘空间。

## 29、删除表空间及其数据文件

drop tablespace userspace including contents and datafiles cascade constraint

## 30、用户操作

--创建用户

create user kzw <用户名>

identified by kzw <密码>

default tablespace userspace <默认表空间>

--创建新用户

CREATE USER kzwill IDENTIFIED BY kzwill;

--DCL授权语句

--GRANT 权限1，权限2，…… TO 用户;

--DCL收权语句

--REVOKE 权限1，权限2，……FROM 用户;

--授予创建会话的权限给kzwill

GRANT CREATE SESSION TO kzwill;

--回收权限

REVOKE CREATE SESSION FROM kzwill;

--授予开发人员权限

GRANT CONNECT,RESOURCE TO kzwill;

GRANT CREATE VIEW TO kzwill;

--修改密码

ALTER USER kzwill IDENTIFIED BY abcd;

--解锁用户

ALTER USER kzwill ACCOUNT UNLOCK;

--授予对象访问权限

GRANT SELECT ON scott.employees TO kzwill;

GRANT ALL ON scott.employees TO kzwill;

--回收对象访问权限

REVOKE ALL ON scott.employees FROM kzwill;

--删除用户

DROP USER kzwill CASCADE;

## 31、授予权限

--授予用户权限（connect用户基本权限 resource开发人员角色）

grant connect,resource to kzw

## 32、撤回权限

--撤销用户权限

revoke connect,resource from kzw

1. 数据库三大范式和五大约束

第一范式（1NF）：数据表中的每一列（每个字段）必须是不可拆分的最小单元，也就是确保每一列的原子性；

第二范式（2NF）：满足1NF后，要求表中的所有列，都必须依赖于主键，而不能有任何一列与主键没有关系，也就是说一个表只描述一件事情；

第三范式（3NF）：必须先满足第二范式（2NF），要求：表中的每一列只与主键直接相关而不是间接相关，（表中的每一列只能依赖于主键）；

【如何更好的区分三大范式】

第 一范式和第二范式在于有没有分出两张表，第二范式是说一张表中包含了所种不同的实体属性，那么要必须分成多张表， 第三范式是要求已经分成了多张表，那么一张表中只能有另一张表中的id（主键），而不能有其他的任何信息（其他的信息一律用主键在另一表查询）。

【数据库五大约束】

1.primary KEY：设置主键约束；

2.UNIQUE：设置唯一性约束，不能有重复值；

3.DEFAULT：默认值约束，height DOUBLE(3,2)DEFAULT 1.2 height不输入是默认为1,2

4.NOT NULL：设置非空约束，该字段不能为空；

5.FOREIGN key ：设置外键约束。

【主键】

1.主键的注意事项？

主键默认非空，默认唯一性约束，只有主键才能设置自动增长，自动增长一定是主键，主键不一定自动增长；

2.设置主键的方式？

在定义列时设置：ID INT PRIMARY KEY

在列定义完之后设置：primary KEY（id）

【外键】

1.设置外键的注意事项：

只有INNODB的数据库引擎支持外键，修改my.ini文件设置default-storage-engine=INNODB 外键必须与参照列的数据类型必须相同（数值型要求长度和无符号都相同，字符串要求类型相同，长度可以不同）。

2设置外键的语法：

CONSTRAINT 外键名 FOREIGN KEY (外键字段)REFERENCES 参照表 (参照字段) ON DELETE SET NULL ON UPDATE CASCADE 设置参照完整性

3.外键约束的参照操作？

参照表的完整性操作：当对参照表的参照字段进行删除或更新时，外键表中的外键如何应对；

参照操作可选值：

RESTRICT拒绝参照表删除或更新参照字段；

RESTRICT和NO ACTION相同，但这个指令只在mysql生效;

CASCADE删除或更新参照表的参照字段时，外键表的记录同步删除更新；

SET NULL 删除或更新参照表的参照字段时，外键表的外键设为NULL;

34、over函数连续求和和排名

--over函数连续求和

SELECT employee\_id,

salary,

department\_id,

SUM(salary) over(ORDER BY employee\_id) over1,

SUM(salary) over() over2

FROM employees;

--按照部门编号做排名

SELECT row\_number() over(ORDER BY department\_id) row\_number,

dense\_rank() over(ORDER BY department\_id) dense\_rank,

rank() over(ORDER BY department\_id) rank,

department\_id,

employee\_id,

last\_name

FROM employees;

Oracle中row\_number()、rank()、dense\_rank() 的区别

row\_number的用途非常广泛，排序最好用它，它会为查询出来的每一行记录生成一个序号，依次排序且不会重复，注意使用row\_number函数时必须要用over子句选择对某一列进行排序才能生成序号。（1 2 3 4 5 6）

rank函数用于返回结果集的分区内每行的排名，行的排名是相关行之前的排名数加一。简单来说rank函数就是对查询出来的记录进行排名，与row\_number函数不同的是，rank函数考虑到了over子句中排序字段值相同的情况，如果使用rank函数来生成序号，over子句中排序字段值相同的序号是一样的，后面字段值不相同的序号将跳过相同的排名号排下一个，也就是相关行之前的排名数加一，可以理解为根据当前的记录数生成序号，后面的记录依此类推。（1 2 2 4 4 4）

dense\_rank函数的功能与rank函数类似，dense\_rank函数在生成序号时是连续的，而rank函数生成的序号有可能不连续。dense\_rank函数出现相同排名时，将不跳过相同排名号，rank值紧接上一次的rank值。在各个分组内，rank()是跳跃排序，有两个第一名时接下来就是第三名，dense\_rank()是连续排序，有两个第一名时仍然跟着第二名。（1 2 2 3 3 3）

35、视图

--视图：虚拟表，作用是保存查询结果，仅保存查询语句，而不保存真实数据（物化视图例外）

--创建视图语法（如果查询语句中有衍生列，必须起别名）

CREATE [OR REPLACE] VIEW 视图名称

AS

查询语句

--创建视图

CREATE (OR REPLACE) VIEW v\_emp

AS

SELECT employee\_id,last\_name,salary

FROM employees

WHERE department\_id=50

WITH READ ONLY;<--只读 -->

--查询视图

SELECT \* FROM v\_emp;

--删除视图

DROP VIEW v\_emp;

36、序列

--创建序列（使用默认值）

CREATE SEQUENCE stu\_seq;

--创建序列

CREATE SEQUENCE stu\_seq

START WITH 100 --起始值，默认是1

INCREMENT BY 10 --递增值，默认是1

NOMAXVALUE --最大值，默认无最大值

NOCYCLE --假如设置了最大值，到达最大值之后，是否从头开始，默认NOCYCLE

CACHE 10 --缓存数量，默认20

;

--删除序列

DROP SEQUENCE stu\_seq;

--使用序列

SELECT stu\_seq.nextval FROM dual;

SELECT stu\_seq.currval FROM dual;

nextval的值是每调用一次就增加一次；

currval的值调用时，还是和自身会话的当前值一样

--在插入语句中使用序列

INSERT INTO user VALUES (stu\_seq.nextval,'tom');

37、子查询

--查询和149号员工同一个部门的其他员工

--employee\_id <> 149代表id不等于149 即排除自身

SELECT \*

FROM employees

WHERE department\_id = (SELECT department\_id

FROM employees

WHERE employee\_id = 149)

AND employee\_id <> 149;

--查询工资最高的前5名的员工（临时视图，内联视图）

（思路：现将工资查询出来进行排序，然后截取前五个名额）

SELECT e.\*

FROM (SELECT \*

FROM employees

ORDER BY salary DESC) e

WHERE ROWNUM <= 5;

--查询员工表中第6到第12条数据

SELECT \*

FROM (SELECT ROWNUM rnum,

e.\*

FROM employees e)

WHERE rnum BETWEEN 6 AND 12;

--推荐以下写法，效率较高

SELECT \*

FROM (SELECT ROWNUM rnum,

e.\*

FROM employees e

WHERE ROWNUM <= 12)

WHERE rnum >= 6;

--查询所有是部门经理的员工（多行子查询）

SELECT \*

FROM employees

WHERE employee\_id IN (SELECT manager\_id FROM departments);

<--查询出满足in 后面的条件的所有数据 -->

in和 = 的区别

1.IN操作符一般比OR操作符清单执行更快.

2.IN最大的优点是可以包含其他SELECT语句。使得能够动态的建立WHERE语句

--查询大于60号部门任何一个员工工资的员工

SELECT \*

FROM employees

WHERE salary > ANY (SELECT salary

FROM employees

WHERE department\_id = 60);

SELECT \*

FROM employees

WHERE salary > ALL (SELECT salary

FROM employees

WHERE department\_id = 60);

ALL与ANY的比较：

salary > ALL(10,20,30) 表示：salary要比ALL中所有的都要大,也就是说要大于ALL(10,20,30)中最大的

salary > ANY(10,20,30) 表示：salary只要满足比ANY中其中一个值要大就可以,也就是说要大于ALL(10,20,30)中最小的

exists:

exists (sql 返回结果集为真)

not exists (sql 不返回结果集为真)

exists用于测试子查询的结果是否为空表，若子查询结果不为空，则exists返回true，否则，返回false。

--查询所有是部门经理的员工（代替了IN语法，效率比较高）

（思路：先查询所有员工的信息，再从中筛选是部门经理的员工）

SELECT \*

FROM employees e

WHERE EXISTS (

SELECT \*

FROM departments d

WHERE e.employee\_id = d.manager\_id

);

38、分级查询

详细解释：https://www.cnblogs.com/benbenduo/p/4588612.html

--查询206号员工所有的上级管理者(从下向上)

SELECT level,employee\_id,last\_name,manager\_id

FROM employees

START WITH employee\_id=206

CONNECT BY PRIOR manager\_id=employee\_id;

--查询101号员工所有的下级员工(从上向下)

SELECT level,employee\_id,last\_name,manager\_id

FROM employees

START WITH employee\_id=101

CONNECT BY PRIOR employee\_id=manager\_id;

39、索引

--索引：作用加快查询速度，但是会降低DML速度

--索引的分类：

--B-TREE索引：默认，适合列值基数比较高，冗余数据很少的时候

--位图索引： 适合列值基数比较低，冗余数据比较多的时候

--创建唯一性索引

ALTER TABLE new\_emp ADD CONSTRAINTS newemp\_empid\_pk PRIMARY KEY (employee\_id);

--创建普通索引

CREATE INDEX newemp\_lname\_idx ON new\_emp (last\_name);

--删除索引

DROP INDEX newemp\_lname\_idx;

-创建基于函数的索引

CREATE INDEX newemp\_lname\_idx ON new\_emp (LOWER(last\_name));

--创建位图索引

CREATE BITMAP INDEX newemp\_jobid\_idx ON new\_emp (job\_id);

适合创建B-TREE索引的时机：

1.列值基数比较高（重复率比较低）

2.不以空值做条件

3.列经常出现在条件子句中

4.表很大

5.经常做查询操作，而不是DML操作

6.满足条件的数据不超过总记录数的15%

7.模糊查询不走索引，如果模糊查询想使用索引，必须用全文检索技术

关于索引：

数据表在创建时，主键的创建必须依赖于索引，默认创建的是聚集索引，主键的创建默认创建了聚集性索引，

而一个表中聚集性索引只可以创建一个。

40、游标

--游标：指向查询结果集的指针，指针指向哪一行，提取的就是哪一行的数据

--PLSQL的游标默认指向结果集的第1行

/\*

使用游标的四个基本步骤：

1.声明游标：cursor 游标变量 is 查询语句

2.打开游标：open 游标变量（不能重复打开游标）

3.提取数据：fetch 游标变量 into 变量1，变量2，……

4.关闭游标：close 游标变量

游标的四个属性

游标变量%found：当最近一次读记录时成功返回,则值为TRUE

游标变量%notfound：同上，求反

游标变量%isopen：判断游标是否已经打开

游标变量%rowcount：返回已从游标中读取的记录数

\*/

--查询输出所有员工的编号，姓名，工资

DECLARE

v\_empid employees.employee\_id%TYPE;

v\_name employees.last\_name%TYPE;

v\_sal employees.salary%TYPE;

--声明游标

cursor cur\_emp is

select employee\_id,last\_name,salary

from employees;

BEGIN

--打开游标

open cur\_emp;

--提取数据（select .... into .....）

fetch cur\_emp into v\_empid,v\_name,v\_sal;

--fetch cur\_emp into v\_empid,v\_name,v\_sal; ....

--输出

dbms\_output.put\_line('编号：'||v\_empid||',姓名：'||v\_name||',工资：'||v\_sal);

--关闭游标

close cur\_emp;

END;

41、存储过程

--创建存储过程add\_dept，传入部门编号，部门名称，部门地址增加部门

CREATE OR REPLACE PROCEDURE add\_dept

(p\_deptno dept.deptno%TYPE,p\_dname dept.dname%TYPE,p\_loc dept.loc%TYPE)

AS

BEGIN

INSERT INTO dept VALUES (p\_deptno,p\_dname,p\_loc);

COMMIT;

END;

--测试add\_dept

BEGIN

add\_dept(50,'NEC','北京');

END;