**[第一章 数据库简介](#_Toc23243)** [10](#_Toc23243)

**[一．文件存储](#_Toc6652)** [10](#_Toc6652)

**[1. 文件存储的发展历程：](#_Toc15497)** [10](#_Toc15497)

**[2. 文件存储方式保存数据的弊端：](#_Toc13428)** [10](#_Toc13428)

**[二． 数据库简介](#_Toc28205)** [10](#_Toc28205)

**[1. 数据库（DB）和数据库管理系统（DMG）](#_Toc23557)** [10](#_Toc23557)

**[2. 关系数据库](#_Toc801)** [10](#_Toc801)

**[3. 主流关系型数据库](#_Toc14039)** [10](#_Toc14039)

**[三． SQL(结构化查询语言)简介](#_Toc28613)** [10](#_Toc28613)

**[1. SQL简述](#_Toc6967)** [10](#_Toc6967)

**[2. SOL的划分](#_Toc24290)** [10](#_Toc24290)

[·数据定义语言(DDL):Data Definition Language 11](#_Toc10189)

[·数据操纵语言(DML):Data Manipulation Language 11](#_Toc8051)

[·事务控制语言(TCL):Transaction Control Language 11](#_Toc22205)

[·数据查询语言(DQL):Data Query Language 11](#_Toc21979)

[·数据控制语言(DCL):Data Control Language 11](#_Toc10878)

**[第二章 Qracle数据库](#_Toc13262)** [12](#_Toc13262)

**[一． Oracle数据类型](#_Toc19111)** [12](#_Toc19111)

**[1. NUMBER类型 ：](#_Toc21615)** [12](#_Toc21615)

**[2. CHAR 类型：](#_Toc22543)** [12](#_Toc22543)

**[3. VARCHAR2 类型：](#_Toc1651)** [12](#_Toc1651)

**[4. DATE 类型：](#_Toc15082)** [12](#_Toc15082)

**[二． 创建表，操作表](#_Toc23627)** [12](#_Toc23627)

**[1.数据定义语言(DDL)----创建表](#_Toc713)** [12](#_Toc713)

[1）创建一个employee表 12](#_Toc23198)

[2）查看表结构 12](#_Toc28584)

[3）给表中数据设置默认值 DEFFAULT语句 13](#_Toc8092)

[4） 非空约束条件 NOT NULL 13](#_Toc13431)

**[2.数据定义语言(DDL)----修改表](#_Toc29175)** [13](#_Toc29175)

[1） 修改表名 RENAME old\_name TO new\_name 13](#_Toc18098)

[2） 修改表结构 ALTER TABLE 语句 13](#_Toc23678)

[（1）向表中添加新字段（增加新的一列） 13](#_Toc27663)

[（2） 删除表中字段 13](#_Toc25518)

[（3） 修改表中的字段（修改列） 13](#_Toc32332)

**[3.数据操纵语言(DML)](#_Toc3255)** [14](#_Toc3255)

[1） INSERT语句 给数据表增加记录 14](#_Toc24213)

[（1）指定字段插入 14](#_Toc27081)

[（2）全列插入 14](#_Toc7472)

[2） commit（事务控制语句） 14](#_Toc15781)

[3） 插入日期字段 14](#_Toc16189)

[（1）插入默认日期格式 14](#_Toc6333)

[（2）插入自定义日期格式 to\_date() 14](#_Toc7125)

[4) UPDATE语句 ：更新表中的数据 14](#_Toc9333)

[--更新tttt表中名字为lisi的员工薪水更新为5999 14](#_Toc21200)

[5） DELETE 语句 删除表中记录 14](#_Toc13628)

[（1） 删除工作岗位为null的员工 14](#_Toc13040)

[6） DELETE和TRUNCATE的区别 14](#_Toc17046)

**[第三章 Oracle数据库基础](#_Toc21072)** [15](#_Toc21072)

[--创建职员表 emp 15](#_Toc7247)

[--创建部门表 dept 15](#_Toc29500)

[--补充：dual:oracle中的伪表，此表不存在 16](#_Toc11885)

**[一． Oacle字符串操作](#_Toc24799)** [16](#_Toc24799)

**[1. 字符串类型](#_Toc29092)** [16](#_Toc29092)

[1） CHAR 和 VARCHAR2 类型 16](#_Toc2678)

[2） LONG H和 CLOB 类型 16](#_Toc21579)

**[2. 字符串函数](#_Toc6365)** [16](#_Toc6365)

[1） 字符串链接函数concat和|| 16](#_Toc2089)

[2）length(char):查询的是字符串的字符个数 16](#_Toc22956)

[3）大小写转换函数 16](#_Toc17308)

[--eg：输入一个员工的名字，不管大小写，查询该员工的信息 17](#_Toc29777)

[4）截去字符串 TRIM ,LTRIM ,RTRIM 17](#_Toc14554)

[--eg：除空格 17](#_Toc16568)

[5）补位函数PAD：LPAD/RPAD 17](#_Toc23649)

[6）获取字符串的子串 SUBSTR(char,m,n) 17](#_Toc5258)

[7)查找字符串在另一个中的位置 instr（char1,char2,n,m） 17](#_Toc7126)

**[二． Oracle数值操作](#_Toc17160)** [18](#_Toc17160)

**[1. 数据值类型](#_Toc28896)** [18](#_Toc28896)

**[2. 数值函数](#_Toc11526)** [18](#_Toc11526)

[1）四舍五入 ROUND(n,m) 18](#_Toc22457)

[2） TRUNC(n,m) 截取数字 18](#_Toc27040)

[3）求余数 Mod(m,n) 18](#_Toc12185)

[4）ceil(n)和floor(n) 18](#_Toc5522)

**[三. Oracle日期操作](#_Toc31526)** [18](#_Toc31526)

**[1. 日期类型](#_Toc1633)** [18](#_Toc1633)

[1） DATE类型 18](#_Toc27005)

[2） TIMESTAMP类型 18](#_Toc18355)

**[2. 日期关键字](#_Toc5283)** [18](#_Toc5283)

[1）SYSDATE 18](#_Toc15499)

[2） SYSTIMESTAMP 时间戳 19](#_Toc2139)

[3） 日期类型可以比较大小 19](#_Toc19269)

[·eg：算1987-10-17到当前过了多少天 19](#_Toc31914)

**[3. 日期转换函数](#_Toc29313)** [19](#_Toc29313)

[1） to\_date() 把字符串转换成日期类型 19](#_Toc18843)

[·eg：查看1981年后入职的员工 19](#_Toc22114)

[·eg：查看每个员工到今天为止，入职了多少天 19](#_Toc1532)

[2） to\_char() 函数 19](#_Toc4300)

[·eg：将emp表中的每个员工的入职时间转换成yyyy年mm月dd天 19](#_Toc4398)

[·eg：将1998-02-19时间转换为1998年2月19日 19](#_Toc9349)

**[4. 日期常用函数](#_Toc2105)** [19](#_Toc2105)

[1）last\_day(date) 返回date所在月份的最后一天 19](#_Toc12899)

[2）add\_months(date,i) 返回日期date加上i个月后的日期值 19](#_Toc26821)

[·eg：计算周年庆日期 20](#_Toc18671)

[3）month\_between(date1,date2) 返回的是两个日期相差的月数 date1-date2 20](#_Toc30894)

[·eg：计算每个员工工做了几个月 20](#_Toc10793)

[4）next\_day(date,i) 返回的给定日期的最近的，还没过的周几的日期 20](#_Toc32577)

[5） least greatest比较函数 20](#_Toc74)

[6）extract()函数 可以提取一个日期中指定的时间分量对应的值 20](#_Toc1390)

**[四． 空值操作](#_Toc440)** [20](#_Toc440)

**[1. NULL的含义](#_Toc15467)** [20](#_Toc15467)

**[2. NULL的操作](#_Toc17805)** [20](#_Toc17805)

[1) 插入NULL值 20](#_Toc26276)

[2) 更新成NULL值 20](#_Toc21853)

[3） NULL 值条件查询 20](#_Toc15590)

[4） 非空约束 20](#_Toc11807)

**[3. 空值函数 NVL NVL2](#_Toc23079)** [21](#_Toc23079)

[1） NVL(a1，a2) 将一个null值转化为一个非null值 21](#_Toc12643)

[·eg：求员工工资 21](#_Toc19042)

[2） NVL2(a1，a2，a3) 将NULL转化为实际值 21](#_Toc20799)

[·eg：查看每个员工的奖金，如果有将建输出有奖金，如果没奖金，返回没奖金 21](#_Toc32079)

**[第四章 SQL(基础查询)](#_Toc21653)** [22](#_Toc21653)

**[一． 基本查询语句](#_Toc9233)** [22](#_Toc9233)

**[1. FROM字句](#_Toc16035)** [22](#_Toc16035)

**[2. 使用别名](#_Toc30984)** [22](#_Toc30984)

**[3. WHERE字句](#_Toc31742)** [22](#_Toc31742)

**[4. SELECT字句](#_Toc11103)** [22](#_Toc11103)

**[二． 查询条件](#_Toc18527)** [22](#_Toc18527)

**[1. 使用< , > , <= , >= , != , <> , =](#_Toc13348)** [22](#_Toc13348)

**[2. 使用AND ,OR关键字](#_Toc7155)** [22](#_Toc7155)

[--查看工资高于1000的CLERK于SALESMAN的员工信息 22](#_Toc18654)

**[3. 模糊查询：LIKE](#_Toc9225)** [22](#_Toc9225)

**[4.IN/NOT IN 判断等于列表中的任何一项即可](#_Toc32758)** [23](#_Toc32758)

**[5. BETWEEN...AND.. 用来查询符合某个范围的数据](#_Toc10836)** [23](#_Toc10836)

[·eg：查询工资在1500-3000之间的支援信息 23](#_Toc7551)

**[6. 使用IS NULL 和 IS NOT NULL](#_Toc10011)** [23](#_Toc10011)

**[7. 使用ANY 和 ALL](#_Toc29468)** [23](#_Toc29468)

**[8.使用表达式和函数](#_Toc24078)** [23](#_Toc24078)

**[9.使用distinct:除去重复行](#_Toc21512)** [23](#_Toc21512)

**[三． 排序](#_Toc8856)** [ORDER BY 字句 23](#_Toc8856)

**[四． 聚合函数](#_Toc26268)** [24](#_Toc26268)

[·聚合函数 又名分组函数，多行函数 24](#_Toc2020)

**[1.MAX()/MIN() 求最大值 最小值](#_Toc9922)** [24](#_Toc9922)

**[2.AVG()/SUM():求平均值，求和](#_Toc8691)** [24](#_Toc8691)

[--需求：求公司员工的平均奖金，奖金总和（涉及null值转换） 24](#_Toc18739)

**[3.COUNT() 计数](#_Toc26736)** [24](#_Toc26736)

[--需求：查询公司有多少人? 24](#_Toc28018)

**[五． 分组](#_Toc4396)**[GROUP BY 字句 24](#_Toc4396)

[·eg：列出同部门同职位的员工的最高于最低工资 24](#_Toc15593)

[·eg：查看部门的平均工资高于2000的部门的平均工资 24](#_Toc10084)

**[六． 查询语句执行顺序](#_Toc18315)** [24](#_Toc18315)

**[第五章 SQL关联查询](#_Toc3644)** [26](#_Toc3644)

**[一． 关联基础](#_Toc13033)** [26](#_Toc13033)

**[1. 关联的概念](#_Toc20502)** [26](#_Toc20502)

**[2. 笛卡儿积](#_Toc13986)** [26](#_Toc13986)

**[3. 等值连接](#_Toc27059)** [26](#_Toc27059)

**[二． 关联查询](#_Toc31120)** [26](#_Toc31120)

**[1. 内链接：返回所有满足连接条件的记录](#_Toc30524)** [26](#_Toc30524)

**[2. 外连接](#_Toc15650)** [26](#_Toc15650)

**[3. 全外链接](#_Toc6816)** [27](#_Toc6816)

**[4. 自连结](#_Toc28868)** [27](#_Toc28868)

[·eg：找king的下属？ 27](#_Toc28147)

[---按经理的名字排序 27](#_Toc21825)

[---查看jones的下属的平均工资 27](#_Toc10673)

[---每个经理下属的平均工资 27](#_Toc22663)

**[三． 子查询](#_Toc9735)** [28](#_Toc9735)

**[1. where子查询：子查询嵌套在where字句中](#_Toc21555)** [28](#_Toc21555)

[1）需求：查找和SCOTT同职位的员工信息 28](#_Toc22427)

[2）需求：和BLAKE同部门的员工 28](#_Toc28107)

[3）创建一个表emp10，保存所有10号部门的员工信息 29](#_Toc11118)

[4）将blake所在部门的所有员工的工资提高10% 29](#_Toc3485)

[·如果子查询返回多行，在主查询中要使用多行比较操作符 29](#_Toc700)

[5）查询MANAGER所在的部门 多行单列 29](#_Toc18824)

[6） 查找薪水比整个机构平均薪水高的员工 30](#_Toc22293)

[7） 查看 与salesman同部门的其他员工 30](#_Toc19276)

[8）查看比所有salesman和clerk工资都高的员工 30](#_Toc4095)

[9）exists 关键字 查看那些部门有员工？ 31](#_Toc13837)

**[2.having 子查询](#_Toc7904)** [31](#_Toc7904)

[1）查询列出最低薪水高于部门30的最低薪水的部门信息？ 31](#_Toc27943)

[2）查看平均薪水高于名叫clerk的平均薪水的职位 31](#_Toc23297)

**[3.FROM 子查询](#_Toc28308)** [31](#_Toc28308)

[1）查看那些员工比所在部门的平均薪水高 31](#_Toc9964)

**[4.SELECT 子查询](#_Toc17762)** [31](#_Toc17762)

**[四．分页查询](#_Toc29042)** [32](#_Toc29042)

[--截取6到10条数据 32](#_Toc32024)

[--分页查询范围的算法 32](#_Toc30743)

**[五． DECODE函数](#_Toc1269)** [33](#_Toc1269)

**[1.decode函数：可以实现类似于switch/case的功能](#_Toc7613)** [33](#_Toc7613)

[1）eg：查询emp表，按职位计算奖励金 33](#_Toc10859)

**[2. decode函数在分组查询中的应用](#_Toc32706)** [33](#_Toc32706)

[1）计算职位的人数，MANAGER，ANALYDT属于VIP，其他是OTHER 33](#_Toc19533)

[2） 自定义排序 33](#_Toc1259)

**[六．排序函数](#_Toc13443)** [33](#_Toc13443)

**[1.row\_number()函数：生成组内连续且唯一](#_Toc15573)** [33](#_Toc15573)

[·eg：需求：查看每个部门的工资排名 33](#_Toc13377)

**[2. rank函数：生成组内不连续也不唯一](#_Toc23492)** [34](#_Toc23492)

**[3. dense\_rank() 函数：生成组内连续但不唯一](#_Toc16148)** [34](#_Toc16148)

**[4. select语句的执行顺序](#_Toc15876)** [34](#_Toc15876)

**[第六章 数据库基础](#_Toc25415)** [35](#_Toc25415)

**[一． 视图](#_Toc30520)** [35](#_Toc30520)

**[1. 创建视图](#_Toc12901)** [35](#_Toc12901)

[1） 视图 35](#_Toc21841)

[·查询视图内容： 35](#_Toc744)

[·修改视图 35](#_Toc15358)

[·对视图进行DML操作 35](#_Toc31836)

[·简单视图： 35](#_Toc9426)

[2） 视图的作用 35](#_Toc10609)

[·简化复杂查询 35](#_Toc12667)

[·限制数据访问 35](#_Toc26586)

[--数据的污染 36](#_Toc26805)

**[2. 创建具有约束的视图](#_Toc3052)** [36](#_Toc3052)

[1）with check ption 36](#_Toc15333)

[2） WITH READ ONLY 36](#_Toc32335)

**[3. 通过查询user\_views获取相关信息](#_Toc8833)** [37](#_Toc8833)

**[4. 复杂视图](#_Toc3150)** [37](#_Toc3150)

[1） 创建一个视图V\_EMP\_SALARY 37](#_Toc16808)

[2）查看哪些员工比所在部门平均工资高 38](#_Toc10117)

[·使用子查询 38](#_Toc2462)

[·使用视图 38](#_Toc30706)

[3）查看那些员工的工资的一半都比30号部门的平均工资高 38](#_Toc16393)

**[5. 删除视图：不影响基表](#_Toc31680)** [38](#_Toc31680)

**[二． 序列](#_Toc25880)** [38](#_Toc25880)

**[1. 创建序列](#_Toc2630)** [38](#_Toc2630)

**[2.使用序列](#_Toc4873)** [38](#_Toc4873)

**[3. 删除序列](#_Toc31753)** [39](#_Toc31753)

**[三． 索引（了解）](#_Toc12960)** [39](#_Toc12960)

**[四． 约束（掌握）](#_Toc1989)** [39](#_Toc1989)

**[1. 约束的作用](#_Toc29836)** [39](#_Toc29836)

**[2. 约束的类型](#_Toc9415)** [39](#_Toc9415)

[1） 非空约束(NOT NULL) 39](#_Toc16084)

[（1） 建表时添加非空约束 39](#_Toc1317)

[（2） 修改表时添加非空约束 39](#_Toc30394)

[（3） 取消非空约束 39](#_Toc3727)

[2） 唯一性约束(Unique) 40](#_Toc4800)

[3） 主键约束(Primary Key) 40](#_Toc11985)

[（1）主键的意义 40](#_Toc32166)

[（2） 主键的选取原则 40](#_Toc4413)

[（3） 添加主键约束 40](#_Toc23204)

[4） 外键约束(Foreign Key) 41](#_Toc9369)

[（1） 添加外键约束 41](#_Toc31789)

[（2） 外键约束对一致性的维护 41](#_Toc7492)

[（3） 外键约束对性能的降低 41](#_Toc10743)

[5） 检查约束(Check) 41](#_Toc15784)

[第二部分plsql语法 43](#_Toc30754)

[PLSQL语法 44](#_Toc16194)

**[一．基本语法](#_Toc16557)** [44](#_Toc16557)

**[1.PLSQL的注释](#_Toc18615)** [44](#_Toc18615)

**[2.PLSQL程序的构成模块](#_Toc6568)** [44](#_Toc6568)

**[3.oracle中变量的标识符许满足以下规范：](#_Toc472)** [44](#_Toc472)

[1）必须以字母开头，不能有空格 44](#_Toc13750)

[2）长度不能超过30个字符 44](#_Toc7297)

[3）不能使用保留字 44](#_Toc18910)

[4）常量/变量名称不区分大小写，在字母后可带数字和特殊字符#（只能带#） 44](#_Toc8632)

**[4.基表数据类型：](#_Toc10951)** [45](#_Toc10951)

**[5. 常量](#_Toc3191)** [45](#_Toc3191)

**[6. 从键盘获取值](#_Toc9410)** [45](#_Toc9410)

[1）获取数值类型 45](#_Toc18431)

[2）获取字符串 45](#_Toc27996)

**[7.复合数据类型：](#_Toc5556)** [46](#_Toc5556)

[1.）%TYPE 列 46](#_Toc16975)

[2） %ROWTYPE 46](#_Toc16070)

[3 ）记录型 46](#_Toc10494)

**[8.表达式：](#_Toc22579)** [47](#_Toc22579)

[1）数值表达式: + - \* / \*\*(乘方) 47](#_Toc6889)

[2）字符运算符： || 47](#_Toc21491)

[3）关系表达式： < > <= >= != like in between 47](#_Toc3777)

[4）逻辑表达式： not and or 47](#_Toc12352)

[5）函数表达式： to\_date to\_char to\_num 47](#_Toc2811)

**[二．流程控制](#_Toc24149)** [47](#_Toc24149)

**[1.条件控制](#_Toc5650)** [47](#_Toc5650)

[·eg：输入值，判断如果大于100显示是三位数，如果小于100大于10显示两位数否则显示一位数 48](#_Toc1068)

[·eg输入员工编号，查看该员工是否有奖金，如果有输出有奖金，如果没有，输出没奖金 48](#_Toc14815)

**[2.循环](#_Toc6385)** [48](#_Toc6385)

[1）loop基础循环 48](#_Toc15023)

[·eg：循环输出10个数字 48](#_Toc6725)

[2)while循环 49](#_Toc26622)

[·eg：循环输出10个数字 49](#_Toc18582)

[·eg：向dept表中插入50，60，70，80，90，100部门编号 49](#_Toc23543)

[3)for循环 49](#_Toc2821)

[·eg：循环输出10个数字 49](#_Toc26680)

**[三． 异常处理](#_Toc24504)** [50](#_Toc24504)

**[1. 预定义异常](#_Toc8548)** [50](#_Toc8548)

[·eg：预定义异常案例 50](#_Toc9098)

[--输入一个员工编号，输出该员工的工资 50](#_Toc18566)

[--员工工资小于1500，加200，该员工以更新，大于的输出超过规定值 50](#_Toc5243)

**[2. 非预定义异常](#_Toc2134)** [51](#_Toc2134)

[·eg:输入一个员工编号，删除该部门的员工 51](#_Toc19526)

**[3.用户自定义异常](#_Toc28125)** [51](#_Toc28125)

[1）根据员工编号，上调员工的工资，加200 51](#_Toc7437)

**[四． 函数和存储过程](#_Toc32530)** [52](#_Toc32530)

**[1. 函数](#_Toc18424)** [52](#_Toc18424)

[·eg：由员工的编号查找员工的姓名 53](#_Toc10070)

[·如何调用函数 53](#_Toc1593)

[·eg：由部门编号，输出这个部门的人数，返回部门的工资总和 53](#_Toc12554)

[1） 位置表示法传递参数 53](#_Toc24283)

[·案例 53](#_Toc31885)

[2） 名称表示法传递参数 54](#_Toc22585)

[·案例 54](#_Toc26719)

[3） 混合表示法传递参数 54](#_Toc18973)

[·案例 54](#_Toc15174)

[·参数默认值 54](#_Toc32204)

[·删除函数 使用drop 55](#_Toc10404)

**[2. 存储过程](#_Toc29586)** [55](#_Toc29586)

[·程序块 55](#_Toc2135)

[1）不带参数的存储过程 55](#_Toc8585)

[（1）调用过程的方法 55](#_Toc26341)

[·eg：输出员工7788的姓名 56](#_Toc23978)

[2） 带参数的存储过程 56](#_Toc30658)

[（1） in 向过程中传递参数 56](#_Toc30067)

[·输出指定员工编号的员工姓名 56](#_Toc743)

[·根据员工的编号删除该员工，使用自定义异常 56](#_Toc18337)

[（2） out 调用过程有返回值 57](#_Toc28705)

[·输入员工编号，返回员工姓名 57](#_Toc6349)

[（3） in out 既传入又传出 57](#_Toc26034)

[·输入数字然后输出 57](#_Toc25077)

[·需求：根据员工编号，上调工资50%，如果上调成功，返回1，输出成功，否则返回0，输出失败 58](#_Toc23879)

**[五． 游标](#_Toc4144)** [58](#_Toc4144)

[·官方定义： 58](#_Toc26957)

[·老师定义： 58](#_Toc10013)

**[1. 游标的定义及使用](#_Toc30244)** [58](#_Toc30244)

[1）声明游标 59](#_Toc986)

[2） 打开游标 59](#_Toc28105)

[3）提取数据 59](#_Toc11384)

[4）关闭游标 59](#_Toc6633)

[·案例 59](#_Toc14409)

[--eg：输出前10条姓名和工资 59](#_Toc16256)

[--eg：输出工资大于1500的员工的姓名和工资 60](#_Toc32644)

[--参数化游标 60](#_Toc2157)

**[2. 游标的属性](#_Toc29961)** [60](#_Toc29961)

[1） %isopen 61](#_Toc12959)

[2） %found 61](#_Toc25424)

[3) %notfound 61](#_Toc31183)

[4) %rowcount 61](#_Toc19914)

[5) sql%notfound 61](#_Toc6114)

[6) sql%rowcount 61](#_Toc564)

[·案例 61](#_Toc22878)

[--查询emp表中共有多少条记录，用游标 62](#_Toc437)

[--把emp表中的书放入myemp表中，删除myemp表中emp=7788的员工，如果找到输出，找到这个值删除他，并输出删除列几行如果找不到，输出为找到 62](#_Toc28319)

**[3. 游标的for循环](#_Toc26559)** [62](#_Toc26559)

[1）无参数的游标for循环 63](#_Toc10165)

[·eg：查询员工的编号，姓名，工资然后输出 63](#_Toc25856)

[·eg：查询部门号为20的员工的工资，工种 63](#_Toc30726)

[2）带默认参数的游标for循环 63](#_Toc22257)

[·eg：查询工资在2000(默认值)以上的员工的姓名，也可以指定其他值 64](#_Toc18474)

**[六． 包](#_Toc2240)** [64](#_Toc2240)

**[1.一个包是由两个分开的部分组成：](#_Toc4007)** [64](#_Toc4007)

[1）包定义的语法： 64](#_Toc20650)

[2）包体定义的部分： 64](#_Toc15722)

[·案例 65](#_Toc5375)

**[2.包的调用](#_Toc27957)** [66](#_Toc27957)

**[3.删除包](#_Toc24163)** [66](#_Toc24163)

[4](#_Toc6493)**[.数据字典](#_Toc6493)**[： 66](#_Toc6493)

[1）查询包 66](#_Toc2319)

[2）usre\_source 保存包的代码 66](#_Toc13193)

**[七． 触发器](#_Toc29980)** [66](#_Toc29980)

**[·创建触发器](#_Toc9296)** [66](#_Toc9296)

**[· 什么时候使用触发器：（记）](#_Toc22045)** [67](#_Toc22045)

**[·用触发器的优点](#_Toc28905)**[： 67](#_Toc28905)

[·案例： 67](#_Toc1173)

**[第三部分：Jdbc](#_Toc32594)** [69](#_Toc32594)

**[一． Jdbc的简介](#_Toc2057)** [70](#_Toc2057)

**[1.jdbc定义](#_Toc29835)** [70](#_Toc29835)

[1）jdbc中的常见接口： 70](#_Toc1729)

**[2. jdbc基础案例](#_Toc17886)** [70](#_Toc17886)

[1）java数据库的链接 70](#_Toc28790)

[2）查看每个员工的名字，职位，工资，部门名称，工作所在地 72](#_Toc7509)

[3）使用jdbc创建表 73](#_Toc9080)

[4）使用jdbc创建序列 74](#_Toc29371)

[5） 将序列作为一个值，向表中插入数据 75](#_Toc2077)

[6） 注册 75](#_Toc1511)

[7） 登陆 76](#_Toc15468)

[8） 使用jdbc进行分页查询 77](#_Toc3105)

[10）输入id删除用户 81](#_Toc31334)

[11） 转账操作 82](#_Toc12763)

**[3. 知识点总结](#_Toc30046)** [84](#_Toc30046)

[1） java链接数据库的步骤： 84](#_Toc23325)

[2）事物操纵语言控制 85](#_Toc3897)

**[二． jdbc方法封装](#_Toc8267)** [85](#_Toc8267)

**[1. 数据库连接，关闭的封装](#_Toc3218)** [85](#_Toc3218)

[1） 封装类 85](#_Toc12854)

[2） 测试类 86](#_Toc26385)

**[2. 从文件导入连接信息](#_Toc26516)** [87](#_Toc26516)

[1） 创建资源文件 87](#_Toc6016)

[2） 封装类 87](#_Toc1873)

**[3. 使用连接池使多个用户同时访问数据库](#_Toc15699)** [88](#_Toc15699)

[1）创建资源文件 88](#_Toc15962)

[2）封装类 88](#_Toc30668)

[3）预处理操作 90](#_Toc25810)

[（1） 使用预处理向userinfo表中插入100条信息 91](#_Toc9439)

[（2）通过ps查询用户余额，通过用户名和密码 92](#_Toc5581)

[（3）用预处理进行分页操作 93](#_Toc7288)

1. **数据库简介**

**一．文件存储**

1. **文件存储的发展历程：**

手动管理——>文件管理———>数据库

1. **文件存储方式保存数据的弊端：**
2. 缺乏对数据的整体管理，数据不便修改
3. 不利于数据分析和共享
4. 数据量急剧增长，大量数据不可能长期保存在文件中
5. **数据库简介**
6. **数据库（DB）和数据库管理系统（DMG）**
7. 数据库（Database简称DB）：是按照数据结构来组织,存储和管理数据的仓库
8. 数据库管理系统（Database Management System简称DBMS）：管理数据库的软件
9. **关系数据库**
10. 关系：·描述两个元素之间的关联或者对应关系

·使用关系模型把数据组织到二维数据表（Table）中

1. 表的概念

·一个关系数据库由多个数据表（Table）组成，数据表是关系数据库的基本结构

·表是二维的，由行和列组成

·表的行（Row）是横排数据，也称为记录（Record）

·表的列（Column）是纵向数据，也称为字段（Field）

1. **主流关系型数据库**

|  |  |  |
| --- | --- | --- |
| 编号 | 数据库 | 开发公司 |
| 1 | Oracle数据库 | Oracle（甲骨文）公司 |
| 2 | DB2数据库 | IBM公司 |
| 3 | Sybase数据库 | Sybase公司 |
| 4 | SQL Server数据库 | 微软 |
| 5 | MySQL | Oracle（甲骨文）公司 |

1. **SQL(结构化查询语言)简介**
2. **SQL简述**
3. SQL(Structured Query Language)是在关系型数据库上执行数据操作，检索及维护所使用的标准语言，可以用来查询数据，定义数据，控制数据
4. 所有数据库都使用相同或者相似的语言
5. SQL语句本身不区分大小写，但是为了增加可读性，
6. 通常会将关键字和非关键字用大小写区分开，
7. 通常关键字全大写，非关键字全小写
8. 但是需要注意：字符串的值是区分大小写的，
9. 数据库中，字符串的字面量使用单括号括起来
10. **SOL的划分**

·数据定义语言(DDL):Data Definition Language

·数据操纵语言(DML):Data Manipulation Language

·事务控制语言(TCL):Transaction Control Language

·数据查询语言(DQL):Data Query Language

·数据控制语言(DCL):Data Control Language

1. 数据定义语言 DDL：用于建立，修改和删除数据库对象；其包括：

·CREATE：创建表或其他对象的结构

·ALTER：修改表或者其他对象的结构

·DROP：删除表或者其他对象的结构

·TRUNCATE：删除表数据，保留表结构

1. 数据操纵语言DML：用于改变数据表中的数据；和事务相关，也就是说修改了表中的数据，需要通过提交语句才能将改变应用到数据库中，其包括：

·INSERT：将数据插入到数据表中

·UPDATE：更新数据表中已存在的数据

·DELETE：删除数据表中的数据

1. 事务控制语言TCL：用来维护数据一致性的语句；其包括：

·COMMIT：提交，确认已经进行的数据改变

·ROLLBACK：回滚，取消已经进行的数据改变。只能取消到上一次提交之后的数据状态

·SAVEPOINT：保存点，使当前的事务可以回退到指定的保存点，便于取消部分改变

1. 数据查询语言DQL：用来查询所需要的数据，SELECT语句
2. 数据控制语言DCL:用于执行权限的授予和回收操作；其包括：

·GRANT：授予，用于给用户或者角色授予权限

·REVOKE：用于收回用户或者角色已有的权限

·CREATE USER：创建用户

1. **Qracle数据库**
2. **Oracle数据类型**
3. **NUMBER类型 ：**

表示数字型；例如：

·ID NUMBER(4) 定义ID为数字类型，4表示最多4位数字，即最大四位整数

·salary NUMBER(6,2) 定义工资列为数字类型，6表示总共最多6位数字，2表示小数点后最多两位小数。即最大工资为9999.99

1. **CHAR 类型：**

表示固定长度的字符类型，最大长度是2000字节；例如：

·name CHAR(20) 定义名字为字符类型，name列最多可存20个字节的字符串，并且占用空间是固定的20个字节

1. **VARCHAR2 类型：**

表示变长的字符类型，最大长度是4000字节；例如：

·JOB VARCHAR2(40) 定义工作为varchar2类型，job列最多可以存长度为40个字节的字符串。若某一行job列存了30个字节的字符串，那么多余的10个字节的空间就会被释放

1. **DATE 类型：**

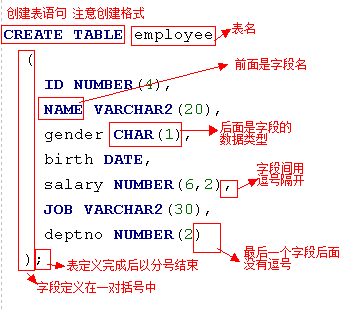
用于定义日期时间数据，长度是7个字节，默认格式是DD-MON-RR；例如：

·hiredate DATE 定义雇佣日期为date类型的数据

1. **创建表，操作表**

**1.数据定义语言(DDL)----创建表**

1）创建一个employee表

、

2）查看表结构

DESC employee; - -查看employee表



3）给表中数据设置默认值 DEFFAULT语句

·在数据库中，无论字段是什么数据类型，默认值都是null，在做插入语句时，只要该字段没有给定值，就会将null值插入该字段

·可以单独使用default语句为字段指定默认值，这样不给值的情况下则使用默认值而不是null

gender CHAR(1) default 'm' - -给gender字段设置默认值为m

1. 非空约束条件 NOT NULL

·NOT用于确保字段值不为空。在默认情况下，任何列都允许有空值，当某个字段被设置了非空约束条件，这个字段中必须存在有效值

·当执行插入数据操作时，必须提供这个列的数据

·当执行更新操作时，不能给这个的值设置为null

NAME VARCHAR2(20) not null - -NAME字段设置为 VARCHAR2类型，最多占20个字节，并且不能为空值

**2.数据定义语言(DDL)----修改表**

1. 修改表名 RENAME old\_name TO new\_name

RENAME employee1 TO tttt - -将employee1表名改为tttt

1. 修改表结构 ALTER TABLE 语句

（1）向表中添加新字段（增加新的一列）

·向表中添加的新字段会被自动追加到表中自动的末尾，而不会出现在现有字段的中间

ALTER TABLE ttttt - -向tttt表中添加hiredate字段，日期类型数据

ADD(

hiredate Date DEFAULT SYSDATE - - 默认为当前系统时间

);

1. 删除表中字段

·删除表中字段，最好是在表中没有数据的时候进行，若表中有数据，会将每一条记录对应的字段值都要删除，数据量大的时候会比较慢

ALTER TABLE ttttt - -删除tttt表中的hiredate字段

DROP (hiredate);

1. 修改表中的字段（修改列）

·一般表建成后，修改表中字段的类型，长度，默认值，字段是否有非空约束。

·一般不建议在表中有数据的时候对字段进行修改，修改的时候尽量不要减少长度，尽量不要修改字段类型

ALTER TABLE ttttt - - 修改tttt表中job的默认值为clerk

MODIFY (job varchar(60) DEFAULT 'clerk')

**3.数据操纵语言(DML)**

1. INSERT语句 给数据表增加记录

（1）指定字段插入

insert into ttttt(id,name,salary,job,deptno)

values(001,'刘洋',8000,'Manager',007);

（2）全列插入

·insert into 语句:可以不指定字段，若不指定，则是全列插入

insert into ttttt values(5,'王五','f','21-2月-17',9000,'teache',11);

1. commit（事务控制语句）

·提交完成后数据才真正插入到数据库中，不能回滚了

commit;

SELECT \* FROM ttttt; --查询表

1. 插入日期字段

（1）插入默认日期格式

insert into ttttt(id,name,birth)

values(3,'lisi','21-2月-17');

（2）插入自定义日期格式 to\_date()

insert into ttttt(id,name,birth)

values(4,'lddd',to\_date('21-02-17','YYYY-MM-DD'));

1. UPDATE语句 ：更新表中的数据

·update 表名 set 某些字段需要做出的改变

·update语句修改表中的数据时，通常要使用where语句来限制修改的记录，这样才能将只满足where条件的记录进行修改，若不添加where，则全表每一条记录都修改，很少这样做

update ttttt set salary = 5555 ; --不加条件将全部列修改

----加条件，只改满足条件的

--更新tttt表中名字为lisi的员工薪水更新为5999

update ttttt set salary = 5999 where name ='lisi';

update ttttt set salary = 5999,job = 'wwwww'

where name ='刘洋';

SELECT \* from ttttt;

commit;

1. DELETE 语句 删除表中记录

·DELETE FROM tttt WHERE id = 5；--删除表中id为5 的记录

·若无WHERE字句，则全表数据都会被删除

1. 删除工作岗位为null的员工

delete from ttttt where job is null;

1. DELETE和TRUNCATE的区别

·DELETE可以有条件的删除，TRUNCATE将表数据全部删除

·DELETE是DML 语句，可以回退；TRUNCATE是DDL语句，立即生效，无法回退

·如果是删除全部记录，且数据量较大，TRUNCATE语句效率较高

·删除全部记录 DELETE FROM tttt； 或者 TRUNCATE TABLE tttt；

1. **Oracle数据库基础**

--创建职员表 emp

CREATE TABLE emp(

empno NUMBER(4,0),--员工ID

ename VARCHAR2(10),--员工姓名

JOB VARCHAR2(9),--职位

mgr NUMBER(4,0),--管理者ID

hiredate DATE,--入职日期

sal NUMBER(7,2),--薪资

comm NUMBER(7,2),--绩效

deptno NUMBER(2,0)--员工所在部门ID

);

SELECT \* from emp;

INSERT INTO emp values(7369,'SMITH','CLERK',7902,TO\_DATE('1980-12-17','YYYY-MM-DD'),800.00,NULL,20);

INSERTINTO empvalues(7499,'ALLEN','SALESMAN',7698,TO\_DATE('1981-2-20','YYYY-MM-DD'),1600.00,300.00,30);

INSERT INTO emp values(7521,'WARD','SALESMAN',7698,TO\_DATE('1981-2-22','YYYY-MM-DD'),1250.00,500.00,30);

INSERT INTO emp values(7566,'JONES','MANAGER',7839,TO\_DATE('1981-4-2','YYYY-MM-DD'),2975.00,NULL,20);

INSERT INTO emp values(7654,'MARTIN','SALESMAN',7698,TO\_DATE('1981-9-28','YYYY-MM-DD'),1250.00,1400.00,30);

INSERT INTO emp values(7698,'BLAKE','MANAGER',7839,TO\_DATE('1981-5-1','YYYY-MM-DD'),2850.00,NULL,30);

INSERT INTO emp values(7782,'CLARK','MANAGER',7839,TO\_DATE('1981-6-9','YYYY-MM-DD'),2450.00,NULL,30);

INSERT INTO emp values(7788,'SCOTT','ANALYST',7566,TO\_DATE('1987-4-19','YYYY-MM-DD'),3000.00,NULL,20);

INSERT INTO emp values(7839,'KING','PRESIDENT',NULL,TO\_DATE('1981-11-17','YYYY-MM-DD'),5000.00,NULL,10);

INSERT INTO emp values(7844,'TURNER','SALESMAN',7698,TO\_DATE('1981-9-8','YYYY-MM-DD'),1500.00,0.00,30);

INSERT INTO emp values(7876,'ADAMS','CLERK',7788,TO\_DATE('1987-5-23','YYYY-MM-DD'),1100.00,NULL,20);

INSERT INTO emp values(7900,'JAMES','CLERK',7698,TO\_DATE('1981-12-3','YYYY-MM-DD'),950.00,NULL,30);

INSERT INTO emp values(7902,'FORD','ANALYST',7566,TO\_DATE('1981-12-3','YYYY-MM-DD'),3000.00,NULL,20);

INSERT INTO emp values(7934,'MILLER','CLERK',7782,TO\_DATE('1982-1-23','YYYY-MM-DD'),1300.00,NULL,10);

--创建部门表 dept

CREATE TABLE dept(

deptno NUMBER(2,0),--部门ID

chame VARCHAR2(14 BYTE),--部门名称

loc VARCHAR2(13 BYTE)--部门所在地

);

SELECT \* from dept;

INSERT INTO dept values(10,'ACCOUNTING','NEW YORK');

INSERT INTO dept values(20,'RESEARCH','DALLAS');

INSERT INTO dept values(30,'SALES','CHICAGO');

INSERT INTO dept values(40,'OPERATIONS','BOSTON');

--补充：dual:oracle中的伪表，此表不存在

--在使用SELECT语句查询内容时，语法要求后面必须跟一张表

--当查询查询的记录不来自任何一张表时，使用伪表

--伪表中查询出来的记录只有一条

1. **Oacle字符串操作**
2. **字符串类型**
3. CHAR 和 VARCHAR2 类型

·表示字符串数据类型，用来在表中存放字符串信息，如姓名，职业等

·CHAR存放定常字符，存不满补空格，最大取值为2000字节

·VARCHAR2存放变长字符，存多少占多少，最大取值为4000字节

·CHAR可以不指定长度，默认是1，VARCHAR2必须制定长度

1. LONG H和 CLOB 类型

·LONG 是VARCHAR2的加长版，存储变长字符串，最多2GB字符串数据，使用限制多

·CLOB 存储定常或者变长字符串，最多达4GB的字符串数据

·Oracle建议开发中使用CLOB代替LONG类型

1. **字符串函数**
2. 字符串链接函数concat和||

·链接 ename:sal

- -concat(char1,char2)

SELECT

concat(concat(ename,':'),sal) as sal -- as 起别名，可以省略

FROM emp;

- -||:此种拼接字符串更简单，常用

SELECT ename||':'||sal from emp;

2）length(char):查询的是字符串的字符个数

SELECT ename,length(ename) from emp;

3）大小写转换函数

--upper(char):把字符串转换成全大写

--lower(char):把字符串转换成全小写

--initcap(char):把字符串转换成首字母大写

--对于initcap说，单词间用空格隔开，这样才能保证每个单词首字母大写

SELECT

upper('hello world'),

lower('HELLOW WORD'),

initcap('hello world')

FROM dual;

--eg：输入一个员工的名字，不管大小写，查询该员工的信息

-- 先将这一列全部变成大写 将输入的变成大写 进行匹配

SELECT \* from emp where upper(ename) = upper('allen');

SELECT \* from emp where lower(ename) = lower('BLAKE');

4）截去字符串 TRIM ,LTRIM ,RTRIM

--trim：截去字符串左右两边连续重复的指定字符

--ltrim/rttim：截去字符串左侧或者右侧连续的字符

SELECT

TRIM('e' FROM 'elite')as t1,

ltrim('elite','e')as t2,

RTRIM('elite','e')as t3

from dual;

SELECT

TRIM('e' FROM 'eeeleeiteee')as t4,

ltrim('eeeleeieeteee','e')as t5,--把左边的e全部截去，碰到其他字母停止

RTRIM('eleeieete','e')as t6

from dual;

--eg：除空格

SELECT

TRIM(' eeeleeiteee ')as t7,--去除两边空格

ltrim(' eeeleeieeteee')as t8,--去除左边空格

RTRIM('eleeieete ')as t9--去除右边空格

from dual;

5）补位函数PAD：LPAD/RPAD

--LPAD(char1,n,char2) n表示总位数

--显示char1必须是n位，不够用char2补位

--LPAD左补位可以实现右对齐，RPAD右补位可以实现左对齐

SELECT ename ,LPAD(sal,6,'$')from emp;

SELECT ename ,RPAD(sal,6,'$')from emp;

6）获取字符串的子串 SUBSTR(char,m,n)

--SUBSTR(char,m,n)，返回char中从m位开始取n个字符，

--m=o,从首字符开始，m为负数，则从尾部开始

--若没设置n，或者呢长度超过了char的长度，则取到末尾位置

SELECT

substr('thinking',7,2)

FROM dual;

commit;

7)查找字符串在另一个中的位置 instr（char1,char2,n,m）

--instr（char1,char2,n,m）

--char2在char1中的位置

--n:从第几个字符开始检索，省略默认值是1

--吗：第几次出现，省略默认是1

1. **Oracle数值操作**
2. **数据值类型**

·NUMBER(P) 表示整数

·NUMBER(P,S)表示浮点数

·制定了s但没制定p，则默认p是38

1. **数值函数**

1）四舍五入 ROUND(n,m)

--n是要被处理的数

--m是保留几位（缺省，默认是0），m>0,保留到小数点后几位

--m=0,保留整数，m<o,保留到小数点前几位

SELECT round(45.346,2)FROM dual; 结果：45.35

SELECT round(45.346,0)FROM dual; 结果：45

SELECT round(45.346,-2)FROM dual; 结果：0

1. TRUNC(n,m) 截取数字

--与round的区别在于不做四舍五入，只截取数字

--m省略，默认为0

SELECT trunc(45.346,2)FROM dual; 结果：45.34

SELECT trunc(45.346,9)FROM dual; 结果：45.346

SELECT trunc(45.346,-2)FROM dual; 结果：0

3）求余数 Mod(m,n)

--用m除以n，求余数，若n为0，则直接返回m

SELECT ename,sal,MOD(sal,1000)FROM emp;

4）ceil(n)和floor(n)

--其中n是数子，通常是小数

--CEIL：返回大于给定数字的在最小整数（向上取整）天花板

--FLOOR:返回小于给定数字的最大整数（向下取整）地板

SELECT ceil(45.678)from dual;----46

SELECT floor(45.678)from dual;----45

1. **Oracle日期操作**
2. **日期类型**
3. DATE类型

·Oracle中最常用的日期类型

·DATE类型在数据库库中的存储固定为7个字节，格式为：

第一字节：世纪+100 第二字节：年 第三字节：月 第四字节：天

第五字节：小时+1 第六字节：分+1 第七字节：秒+1

1. TIMESTAMP类型

·与DATE类型区别是精度可以达到纳秒（ns）

·数据库中用7或者11个字节存储，格式为

第1——7字节：和DATE相同

第8——11字节：纳秒，采用四个字节存储，内部运算类型为整型

1. **日期关键字**

1）SYSDATE

·其本质为一个Oracle内部函数，返回当前的系统时间，精确到秒

·默认显示格式是DD-MON-RR

SELECT SYSDATE FROM DUAL;--获取当前系统时间 不带时分秒

1. SYSTIMESTAMP 时间戳

·内部函数，返回当前系统日期和时间，精确到毫秒

SELECT SYSTIMESTAMP FROM dual; --获取当前系统时间 精确到毫秒

1. 日期类型可以比较大小

·时间越晚越大，越早越小

·时间之间可以做减法操作，差为两者之间相差的天数

·时间之间可以加减数字，等同于加减指定的天数

·eg：算1987-10-17到当前过了多少天

SELECT SYSDATE-to\_date('1987-10-17','YYYY-MM-DD') FROM dualL;

1. **日期转换函数**
2. to\_date() 把字符串转换成日期类型

·可以按照给定的日期格式解析给定的字符串，并转换成一个DATE类型的返回

·eg：查看1981年后入职的员工

SELECT ename,sal,hiredate

FROM emp

WHERE (hiredate>TO\_DATE('1981-01-01','YYYY-MM-DD'));

SELECT ename,sal,hiredate

FROM emp

WHERE (hiredate>TO\_DATE('1981年01月01','YYYY"年"MM"月"DD"日"'));

--日期转换格式问题：前后对应

·eg：查看每个员工到今天为止，入职了多少天

SELECT ename,trunc(SYSDATE-HIREDATE)

FROM emp; --trunc对天数取整

1. to\_char() 函数

·常用于将一个日期格式按照给定的格式转换成字符串输出

·eg：将emp表中的每个员工的入职时间转换成yyyy年mm月dd天

SELECT ename,hiredate,to\_char(hiredate,'YYYY"年"MM"月"DD"日"')

FROM emp;

SELECT ename,hiredate,to\_char(hiredate,'YYYY"/"MM"/"DD')

FROM emp;

·eg：将1998-02-19时间转换为1998年2月19日

SELECT to\_char(

to\_date(

'1998-02-19','YYYY-MM-DD'

),'YYYY"年"MM"月"DD"日"'

)

FROM dual;

--日期格式中，除字母与符号外的其他字符都需要使用双括号括起来

1. **日期常用函数**

1）last\_day(date) 返回date所在月份的最后一天

SELECT last\_day(SYSDATE) FROM dual;

SELECT last\_day('09-2月-09') FROM dual;

2）add\_months(date,i) 返回日期date加上i个月后的日期值

--i为正价，i为负，减

·eg：计算周年庆日期

SELECT ename,hiredate, add\_months(hiredate,20\*12) "20周年"

FROM emp; --起别名as可以省略

3）month\_between(date1,date2) 返回的是两个日期相差的月数 date1-date2

·eg：计算每个员工工做了几个月

SELECT ename,trunc(

months\_between(SYSDATE,hiredate)

) FROM emp;

4）next\_day(date,i) 返回的给定日期的最近的，还没过的周几的日期

--1到7表示周日到周六，周日为1

SELECT next\_day(SYSDATE,4)FROM dual;--返回下一个周三的日期

1. least greatest比较函数

--求多个数据中的最大最小数据，要求数据类型要相同

--least(exp1,exp2,exp3,...);返回参数中的最小值

--greatest(exp1,exp2,exp3,...);返回参数中的最大值

SELECT greatest(SYSDATE,'24-2月-17')FROM dual;--'24-2月-17'可以转化为date类型

--后面的数据可以转化为第一个数据类型（前提是可以转eg，若第一个是number类型，第二个为char类型，但是单引号里面必须是数字，这样才可以搬砖），就可以比较

SELECT greatest('你好急急急','24-2月-17')FROM dual;

SELECT least('你','我')FROM dual;

6）extract()函数 可以提取一个日期中指定的时间分量对应的值

SELECT **EXTRACT**(MONTH FROM SYSDATE)FROM dual;

SELECT ename,sal,hiredate,**EXTRACT**(year FROM hiredate) year from emp;

1. **空值操作**
2. **NULL的含义**

·数据库里的重要概念：NULL 空值

·有时表中某些字段值，数据未知或暂时不存在，取值NULL

·任何数据类型均可取值NULL

1. **NULL的操作**
2. 插入NULL值

·显示插入

·隐式插入

1. 更新成NULL值

·更新数据为null，前提是该列数据没有设置不为空

1. NULL 值条件查询

·where sal is null; 当判断条件为某字段是否为空

·用的是is null或 is not null，而不是 =

1. 非空约束

·NOT NULL (非空约束)用于确保字段值不为空

·当某个字段被设置了非空约束，这个字段中必须存在有效值，即当执行插入数据操作时，必须提供这个列的数据，当执行更新操作时，不能给这个列的值设置为NULL

·gender char not null

1. **空值函数 NVL NVL2**
2. NVL(a1，a2) 将一个null值转化为一个非null值

·eg：求员工工资

--直接写工资加绩效，绩效为null，则工资加绩效也为null

SELECT ename,sal,comm,sal+nvl(comm，0)

FROM emp;

1. NVL2(a1，a2，a3) 将NULL转化为实际值

--a1，a2，a3参数类型可以不一致，但建议a2，a3参数类型一致

--当a1不为null，返回a2；当a1 为null，返回a3

·eg：查看每个员工的奖金，如果有将建输出有奖金，如果没奖金，返回没奖金

SELECT ename,comm,nvl2(comm,'有奖金','没奖金')

FROM emp;

1. **SQL(基础查询)**
2. **基本查询语句**
3. **FROM字句**

SELECT ename,sal,comm,sal+nvl(comm，0) **FROM** emp;

SELECT \* FROM emp;

·SELECT用于制定要查询的列，要查询所有的列可以在SELECT后面使用\*，要查询指定的列，可以直接在SELECT后面制定列，列名，列名间用都好隔开

·FROM指定要从哪个表中查询

1. **使用别名**

·在SQL语句中可以通过使用列的别名改变标题的显示样式，或者表示计算结果的含义

·使用语法：列的别名跟在列之后，中间可以加也可以不加AS关键字

·如果希望别名中区分大小写字符，或者别名中包含字符或者空格，则必须使用双引号引起来

·SELECT ename,comm,nvl2(comm,'有奖金','没奖金') 奖金有无

FROM emp;

·SELECT ename,comm,nvl2(comm,'有奖金','没奖金') AS 奖金有无,

sal+nvl(comm，0) 总工资

FROM emp;

·SELECT ename,comm,nvl2(comm,'有奖金','没奖金') "奖 金 有 无"

FROM emp;

1. **WHERE字句**

·在SELECT字句中，可以在WHERE字句中使用比较操作符来限制查询结果

--查看10号部门下员工信息

SELECT \* from emp where deptno = 10;

--查看职位是”salesman“D的员工信息

SELECT \* from emp where JOB = 'SALESMAN'; --职位用引号括起来

--查看职位是CLERK并且工资大于1000的员工

SELECT \* FROM emp WHERE JOB = 'CLERK' AND sal > 1000;

1. **SELECT字句**
2. **查询条件**
3. **使用< , > , <= , >= , != , <> , =**
4. **使用AND ,OR关键字**

--查看工资高于1000的CLERK于SALESMAN的员工信息

--AND 并且 两个都为真才为真

--OR 或者 两个都为假才为加

--OR优先级低于AND

SELECT \*

FROM emp

WHERE sal > 1000 AND (JOB = 'CLERK' OR JOB = 'SALESMAN');

1. **模糊查询：LIKE**

--用于对字符串进行模糊匹配，支持两个通配符

--%表示多个字符

--\_表示单个字符

SELECT \*FROM emp WHERE ename LIKE '\_A%';

SELECT \*FROM emp WHERE ename LIKE '%A%';--中间有A

**4.IN/NOT IN 判断等于列表中的任何一项即可**

SELECT ename,JOB

FROM emp

WHERE JOB IN ('MANAGER','CLERK');

1. **BETWEEN...AND.. 用来查询符合某个范围的数据**

·eg：查询工资在1500-3000之间的支援信息

SELECT ename，sal

FROM emp

WHERE sal BETWEEN 1500 AND 3000；

1. **使用IS NULL 和 IS NOT NULL**

·NULL是一个特殊值，比较的时候不能使用=。得使用is

1. **使用ANY 和 ALL**

>ANY:大于最小 <ANY：小于最大

>ALL：大于最大 <ALL：小于最小

**8.使用表达式和函数**

**9.使用distinct:除去重复行**

--将结果集中的重复的字段值去除

--distinct关键字必须紧跟放在SELECT的后面

--distinct对多个字段去重时，不能保证结果集中某一个字段没有重复值

--作用是保证这几个字段值的组合没有重复值

SELECT \* FROM EMP;

SELECT DISTINCT deptno

FROM emp;

SELECT deptno,JOB

FROM emp;

SELECT DISTINCT deptno,JOB

FROM emp;

1. **排序** ORDER BY 字句

ORDER BY 字句 排序结果集，可以按照给定字段的值进行升序或降序操作

--排序结果集，可以按照给定字段的值进行升序或者降序排列

--ASC:升序，从小到大，默认就是升序，通常ASC省略

--DESC:降序，从大到小

SELECT ename,sal,deptno

FROM emp

order by deptno desc;

SELECT ename,sal,deptno

FROM emp

ORDER BY sal;

--多个字段排序，按部门编号升序排，按工资降序排

--先按照第一个字段排序，排序后得到结果集，再按照第二个字段排序，以此类推

SELECT ename,sal,deptno

FROM emp

ORDER BY deptno,sal DESC;

1. **聚合函数**

·聚合函数 又名分组函数，多行函数

--作用：进行统计工作，可以将多条记录的值进行统计，然后得到一个结果输出

**1.MAX()/MIN() 求最大值 最小值**

SELECT MAX(sal) 最大值,MIN(sal) 最小值

FROM emp;

**2.AVG()/SUM():求平均值，求和**

SELECT AVG(sal) 平均值,SUM(sal) 总和

FROM emp;

--需求：求公司员工的平均奖金，奖金总和（涉及null值转换）

SELECT AVG(nvl(comm,0)),SUM(comm)

FROM emp;

理解：当绩效为null时，无法做求平均运算，所以得将null转化为0

**3.COUNT() 计数**

--用来统计指定字段非null的记录总共有多少条，不关系具体取值

--需求：查询公司有多少人?

SELECT COUNT(ename)

FROM emp;

1. **分组**GROUP BY 字句

分组函数：GROUP BY 字句

--当SELECT语句中出现了聚合函数，那么凡是不再聚合函数中的字段，必须写在group by字句后面

SELECT MAX(sal),MIN(sal),deptno

FROM emp

GROUP BY deptno;

·eg：列出同部门同职位的员工的最高于最低工资

SELECT MAX(sal),MIN(sal),deptno ,JOB

FROM emp

GROUP BY deptno,JOB;

·eg：查看部门的平均工资高于2000的部门的平均工资

SELECT AVG(sal),deptno

FROM emp GROUP BY deptno

HAVING AVG(sal)>2000;

--当有GROUP BY 出现，后面需要条件，则需要用HAVING

--HAVING字句不能单独存在，若一个语句中有HAVING，必须要写GROUP BY字句

1. **查询语句执行顺序**

查询语句的执行顺序

--1，from字句：执行顺序为从往前，从右向左，数据量较少的表尽量放在后面

--2，where字句：执行顺序自下而上，从右到左，将能过滤掉最大数量记录的条件写在where字句的最右

--3.group by 执行顺序从左往右分组

--最好在group by前使用where将不需要的记录在group by之前过滤掉

--4.having字句：消耗资源

--尽量避免使用，having会在检索出所有记录后才对结果集进行过虑，需要排序等操作

--5.SELECT字句，尽量少使用\*，尽量取字段名

--6.oracle在解析的过程中通过查询数据字典将\*依次转换成所有的列名，消耗时间

--order by字句：执行顺序为从左到右排序，消耗资源

1. **SQL关联查询**
2. **关联基础**
3. **关联的概念**

·实际应用中所需要的数据，经常会需要查询多个表

·这种查询多个（两个及其以上）数据表或者视图的查询叫做连接查询

·连接查询通常建立在存在相互关系的父子表之间

1. **笛卡儿积**

·笛卡儿积是指做关联的每个表的每一行都和其他表的每一行做结合，假设两个表分别记录X和Y条数据，笛卡儿积将返回X\*Y条数据

·关联查询时不指定连接条件就会出现笛卡儿积，实际开发中要避免这种情况，通常笛卡儿积没有意义

1. **等值连接**

·链接查询中最常见的一种，通常在有主外键关联关系的表间建立，并将连接条件设定为有关系的列，使用= 连接相关的表

SELECT emp.deptno,dept.deptno

FROM emp,dept

WHERE emp.deptno=dept.deptno;

SELECT ename,emp.deptno 员工表,dept.deptno 部门表

FROM emp,dept

WHERE emp.deptno=dept.deptno;

1. **关联查询**
2. **内链接：返回所有满足连接条件的记录**

SELECT e.ename,d.chame

FROM emp e JOIN dept d --JOIN是链接两个表

ON e.deptno = d.deptno; --ON后面是链接条件

1. **外连接**

--不仅返回满足连接条件的记录，还将返回不满足连接条件的记录

--外联接：用于查询不满足链接条件的数据（包含满足连接条件的数据）

--外链接包括：左外连接，右外连接，全连接

--左外链接，以join为分界点，左边为主表（驱动表）

--将主表的数据依次和附表做匹配，主表数据全部显示，附表不满足条件的显示null。

SELECT e.ename,d.chame

FROM emp e left JOIN dept d

ON e.deptno = d.deptno;

--简单写法

SELECT e.ename,d.chame

FROM emp e , dept d

where e.deptno = d.deptno(+);

SELECT e.ename,d.chame

FROM emp e right outer JOIN dept d --右外链接，以join为分界点，右边为主表（驱动表）

ON e.deptno = d.deptno;

--简单写法

SELECT e.ename,d.chame

FROM emp e , dept d

where e.deptno (+)= d.deptno;

1. **全外链接**

--除了返回满足链接条件的记录，还会返回不满足连接条件的所有的其他行

--是左外连接和右外连接查询结果的总和

SELECT e.ename ,d.chame

FROM emp e **full OUTER JOIN** dept d

ON e.deptno = d.deptno;

1. **自连结**

--自连接：保存的数据是相同的，但数据间存在上下级关系，并且层级不确定

--数据来源是同一个表

SELECT e.ename 员工,m.ename 领导

FROM emp e,emp m

WHERE e.mgr = m.empno; --员工领导的编号=领导作为员工的编号

SELECT e.ename 员工,m.ename 领导

FROM emp e,emp m

WHERE e.mgr = m.empno(+);

·eg：找king的下属？

SELECT e.ename 员工,m.ename 领导,m.mgr

FROM emp e,emp m

WHERE (e.mgr = m.empno) AND (m.ename='KING');

---按经理的名字排序

SELECT e.ename,m.ename

FROM emp e,emp m

WHERE e.mgr = m.empno

ORDER BY m.ename;

---查看jones的下属的平均工资

SELECT m.ename ,avg(e.sal) jones的下属的平均工资

FROM emp e,emp m

WHERE e.mgr = m.empno AND m.ename = 'JONES'

GROUP by m.ename;

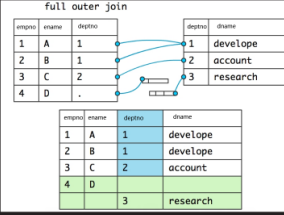
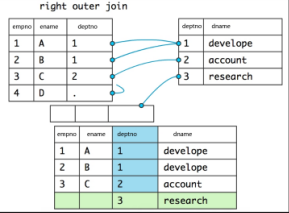
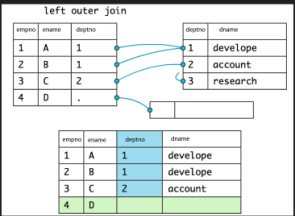
---每个经理下属的平均工资

**SELECT** m.ename ,avg(e.sal)

**FROM** emp e,emp m

**WHERE** e.mgr = m.empno

**GROUP by** m.ename;



1. **子查询**

--子查询：嵌入在其他SQL语句中的查询语句，通常是给外层SQL提供数据的

--子查询通常用于查询语句中，也可以用在DDL,DML.

--根据返回结果的不同，子查询分为：单行子查询，多行子查询，多列子查询

1. **where子查询：子查询嵌套在where字句中**

--子查询中，如果子查询得到的是单行单列，一般作为过滤条件放在where字句中

1）需求：查找和SCOTT同职位的员工信息

----第一步：查询SCOTT的职位

SELECT JOB FROM emp WHERE ename = 'SCOTT' ;

----第二步：将查询SCOTT职位的查询语句作为查找和SCOTT同职位员工信息的条件语句

SELECT \*

FROM emp

WHERE JOB = (SELECT JOB

FROM emp

WHERE ename = 'SCOTT'

);

2）需求：和BLAKE同部门的员工

------第一步：查找BLAKE的部门

SELECT deptno FROM emp WHERE ename = 'BLAKE';

------第二步：查找和他部门相同的员工

SELECT \*

FROM emp

WHERE deptno = (SELECT deptno

FROM emp

WHERE ename = 'BLAKE'

);

--多行多列

3）创建一个表emp10，保存所有10号部门的员工信息

--第一步：查询10号部门员工

SELECT \* FROM emp WHERE deptno = 10;

--第二部：创建表

**CREATE TABLE** emp10

**AS**

SELECT \*

FROM emp

WHERE deptno = 10;

SELECT \* FROM emp10;

--单行单列

4）将blake所在部门的所有员工的工资提高10%

--第一步查询blake所在的部门

SELECT deptno FROM emp WHERE ename = 'BLAKE';

--第二部：更新工资

UPDATE emp

SET sal = sal\*1.1

WHERE deptno = (

SELECT deptno

FROM emp

WHERE ename = 'BLAKE'

);

--第三部：查询

SELECT sal

FROM emp

WHERE deptno = (SELECT deptno

FROM emp

WHERE ename = 'BLAKE'

);

SELECT \*

FROM emp

WHERE deptno = (SELECT deptno

FROM emp

WHERE ename = 'BLAKE'

);

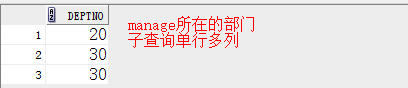
·如果子查询返回多行，在主查询中要使用多行比较操作符

·多行单列:多行比较符 in all any ，一般情况下不单独使用，通常和> < >= <= 一起使用

5）查询MANAGER所在的部门 多行单列

SELECT deptno FROM emp WHERE JOB = 'MANAGER';

--查询MANAGER同部门的员工

**SELECT** \*

**FROM** emp

**WHERE** deptno **IN** (

**SELECT** deptno

**FROM** emp

**WHERE** JOB = 'MANAGER'

);

1. 查找薪水比整个机构平均薪水高的员工

**SELECT** avg(sal) **FROM** emp;

**SELECT** ename ,sal

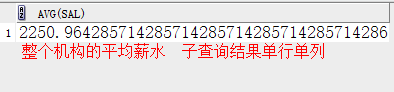
**FROM** emp

**WHERE** sal **>** (

**SELECT** avg(sal)

**FROM** emp

);



1. 查看 与salesman同部门的其他员工

SELECT deptno FROM emp WHERE JOB = 'SALESMAN';

**SELECT** \* **FROM** emp

**WHERE** deptno **IN**(

**SELECT** deptno

**FROM** emp

**WHERE** JOB = 'SALESMAN'

)

**AND** JOB != 'SALESMAN';

理解：·从emp表中查找数据条件是职部门在子查询所得到的部门中，并且工作不是salesman

·子查询是查找salesman所在的部门

8）查看比所有salesman和clerk工资都高的员工

SELECT sal FROM emp WHERE JOB = 'SALESMAN'OR JOB = 'CLERK';

SELECT sal FROM emp WHERE JOB in( 'SALESMAN', 'CLERK');

**SELECT** ename , sal

**FROM** emp

**WHERE** sal > **ALL** (

**SELECT** sal

**FROM** emp

**WHERE** JOB **IN** ('SALESMAN', 'CLERK')

);

9）exists 关键字 查看那些部门有员工？

--exists关键字后面跟一个子查询，只要该查询能查出至少一条记录，就返回true

--查看那些部门有员工？

**SELECT** deptno ,chame

**FROM** dept

**WHERE EXISTS**(

**SELECT** \* **FROM** emp,dept **WHERE** dept.deptno = emp.deptno

);

**2.having 子查询**

1）查询列出最低薪水高于部门30的最低薪水的部门信息？

**SELECT** MIN(sal),deptno

**FROM** emp

**GROUP BY** deptno

**HAVING** MIN(sal) >(

**SELECT** MIN(sal)

**FROM** emp

**WHERE** deptno = 30

);

2）查看平均薪水高于名叫clerk的平均薪水的职位

**SELECT** JOB ,avg(sal)

**FROM** emp

**GROUP BY** JOB

**HAVING** avg( sal)>(

**SELECT** avg(sal)

**FROM** emp

**WHERE** JOB = 'CLERK'

);

**3.FROM 子查询**

1）查看那些员工比所在部门的平均薪水高

**SELECT** e.ename,e.sal,e.deptno

**FROM** emp e,(

**SELECT** avg(sal) avg\_sal,deptno

**FROM** emp

**GROUP BY** deptno

) t

**WHERE** e.deptno = t.deptno

and e.sal >t.avg\_sal;

**4.SELECT 子查询**

--子查询出现在SELECT子句中，通常是外链接效果

--从emp表中查找员工名，及所在部门编号和部门的名字。

**SELECT** e.ename,e.deptno,(**SELECT** d.chame **FROM** dept d **WHERE** d.deptno = e.deptno)chame

**FROM** emp e;

**四．分页查询**

--当一条语句执行后，数据量过于庞大，通常会使用分页措施

--一次查询值显示一部分数据，分段显示

--提高响应速度，提高用户体验度，减小内部开销

--分页的sql语句不是固定的，不同的数据库有不同的分页sql

--oracle中使用rownum策略，为结果集编号，然后分段获取

--rownum称为伪列，用于返回表示行的数据顺序的数字

--只能从1计数，不能直接从结果集中直接截取

**SELECT** ename,sal,deptno **FROM** emp; --查看字段

**SELECT ROWNUM** ,ename,sal,deptno **FROM** emp; --查看字段及伪列

**SELECT ROWNUM** ,ename,sal,deptno **FROM** emp

**WHERE** rownum <6; --查看<6号的数据

--rownum：伪列，不世故表中一个真实的字段，可以应用于任何一张表中，一行一个行号i

--行号在生成时，从1开始

--在查询时，查询出来一条数据就为其编号为1

--在第一次查询数据进行编号的过程中不要使用rownum进行大于1以上数字作为过滤条件

--否则查不出任何数据

--截取6到10条数据

**SELECT** \*

**FROM** (**SELECT** **ROWNUM** rn ,ename,sal,deptno **FROM** emp order by sal desc) t

**WHERE** t.rn **BETWEEN** 6 **AND** 10;--半成品

--先排好再截取:

**SELECT** \***FROM** --最后拿出伪列的6~10条数据

(

**SELECT ROWNUM** rn ,t.\* --再给排序后的查询结果添加伪列

**FROM**(

**SELECT** ename,sal,deptno

**FROM** emp

**ORDER BY** sal **DESC** --先把工资按降序排列

) t

)

**WHERE** rn **BETWEEN** 6 AND 10;

--分页查询范围的算法

--pagesSize:每页的条目数

--page:第几页

--MIN:(page-1)\*pageSize+1

--max:page\*pageSize

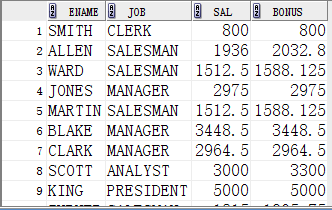
1. **DECODE函数**

**1.decode函数：可以实现类似于switch/case的功能**

1）eg：查询emp表，按职位计算奖励金

当职位分别是’MANAGE‘,'ANALYST','SALESMAN'时，奖励金分别是薪水的1.2倍，1.5倍，1.05倍，其他职位奖励金取薪水值

**SELECT** ename,JOB,sal,

 **decode**(JOB,

'MANAGE',sal\*1.2,

'ANALYST',sal\*1.1,

'SALESMAN',sal\*1.05,

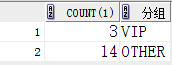
sal

)bonus

**FROM** emp;

1. **decode函数在分组查询中的应用**

1）计算职位的人数，MANAGER，ANALYDT属于VIP，其他是OTHER

 **SELECT** count(1), **decode**(JOB,

'MANAGER','VIP',

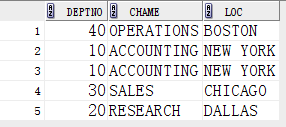
'ANALYDT','VIP',

'OTHER') 分组

**FROM** emp

**GROUP BY** **decode** (JOB,

'MANAGER','VIP',

 'ANALYDT','VIP',

'OTHER'

) ;

1. 自定义排序

定制排序：OPERATIONS第一行

ACCOUNTING第二行

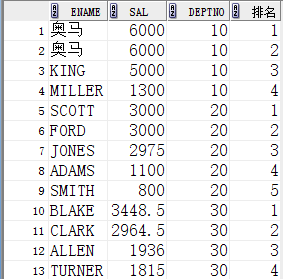
SALES 第三行

**SELECT** deptno,chame,loc

**FROM** dept

**ORDER BY**

**decode**(chame,'OPERATIONS','1', --加1表示第1行显示

 'ACCOUNTING','2', --加2表示第2行显示

'SALES','3'); --加3表示第3行显示

**六．排序函数**

**1.row\_number()函数：生成组内连续且唯一**

·eg：需求：查看每个部门的工资排名

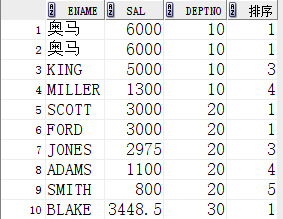
--PARTITION BY 后面加分组字段

--ORDER BY 后面加依据哪个字段排序

**SELECT** ename,sal,deptno,row\_number() OVER(

**PARTITION BY** deptno **ORDER BY** sal **DESC**

) 排名

**FROM** emp;

1. **rank函数：生成组内不连续也不唯一**

**SELECT** ename,sal,deptno,**rank() OVER**(

**PARTITION BY** deptno **ORDER BY** sal **DESC**

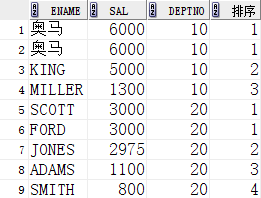
) 排序

**FROM** emp;

1. **dense\_rank() 函数：生成组内连续但不唯一**

**SELECT** ename,sal,deptno,**DENSE\_RANK() OVER**(

**PARTITION BY** deptno **ORDER BY** sal DESC

 ) 排序

**FROM** emp;

1. **select语句的执行顺序**

--1.from --2.where --3.group by

--4.having --5.select--6.order by

--除了子查询外，一般不再where，group by，having字句中出现别名

--order by 字句中可以出现别名

1. **数据库基础**
2. **视图**
3. **创建视图**

**CREATE** **VIEW** v\_emp\_10

**AS**

**SELECT** empno,ename,sal,deptno

**FROM** emp

**WHERE** deptno = 10;

理解：创建的这个视图只能看到基表中deptno=10的empno,ename,sal,deptno这些字段，其他的看不到，这样就限制了视图的访问权限

1. 视图

·视图也称作虚表，是一组数据的逻辑表示

·视图对应于一条select语句，结果集被赋予一个名字，即视图名字

·视图本身不含有任何数据，它只包含映射到基表的一个查询语句，当基表数据发生变化，视图数据也随之变化

·查询视图内容：

--数据库会先将该视图对应的select语句执行，然后得到结果集，

---再将该结果集当作一张表，然后去执行我们的查询语句

--查看视图结构

**DESC** v\_emp\_10;

--视图本身没有结构，而是将其子查询查询的基表对应的字段类型列出来

·修改视图

--由于视图没有结构，只是对应一条子查询

--所以修改视图就是替换原有子查询

--creatr or replace 创建修改都可以，没有则创建，有则被替换

·对视图进行DML操作

--只能对简单视图进行操作，对视图进行DML操作，就是对该视图对应基表的操作

·简单视图：

---数据从单一的一张表中获取，并且没有对表中的数据做过任何的加工（字段不含有表达式，函数，不含有分组等操作）

**INSERT INTO** v\_emp\_10

**VALUES**(1001,'哈喽',6000,10);

**SELECT** \* **FROM** v\_emp\_10;

**SELECT** \* **FROM** emp;

--向视图中插入一条数据就是将该数据插入到基表中，并且该条记录，只有视图看得到字段会有值，其他的字段默认值都为null（单独制定的字段除外）

--所以，若基表中看不到的字段定义为not null，那么插入失败

1. 视图的作用

·简化复杂查询

--如果需要经常执行某项复杂查询，可以基于这个复杂查询建立视图，此后查询此视图即可

·限制数据访问

--数据的污染

**INSERT INTO** v\_emp\_10

**VALUES**(1001,'奥巴马',6000,20);

**SELECT** \* **FROM** v\_emp\_10;

--插入的奥巴马视图看不到，因为视图看的是10号部门的数据，插入的奥巴马是20号部门

**SELECT** \* **FROM** emp;

--有时，我们通过视图插入的数据，视图本身看不到，但是却进入列基表，那么这样的数据就会对基表产生污染

--同样修改视图也会产生污染

**UPDATE** v\_emp\_10

**SET salary** = salary\*1.5;

**SET salary** = salary\*1.5;

--污染效果

**UPDATE** v\_emp\_10

**SET** deptno = 20;

--这样将元表中部门为10 全部改为20，所以视图中没数据了

ROLLBACK;

--删除数据

--因为删除数据只能删除视图看得到的数据，所以不会产生污染

**DELETE FROM** v\_emp\_10

**WHERE** name = '哈喽';

1. **创建具有约束的视图**

1）with check ption

--为视图添加with check ption选项，当一个视图添加列检查选项后，那么对视图进行增改操作后，若视图对数据不可见，则不允许操作

**CREATE OR REPLACE VIEW** v\_emp\_10

**AS**

**SELECT** empno id,ename name ,sal salary,deptno

**FROM** emp

**WHERE** deptno = 10

**WITH CHECK OPTION**;

**DESC** v\_emp\_10;

**SELECT** \* **FROM** v\_emp\_10;

**INSERT INTO** v\_emp\_10

**VALUES**(1001,'奥马',6000,20);

--SQL 错误: ORA-01402: 视图 WITH CHECK OPTION where 子句违规

注意：在创建视图时，将基表中的字段统一做别名处理，那么select \*从视图中得到的字段就不是基表中的字段，这样可以保证基表中的字段的安全性

**INSERT INTO** v\_emp\_10

**VALUES**(1001,'奥马',6000,10);

--同理修改也一样

1. WITH READ ONLY

--如果只将视图设置为只读，那么无法对只读视图执行 DML 操作

**CREATE OR REPLACE** VIEW v\_emp\_10

**AS**

**SELECT** empno id,ename name ,sal salary,deptno

**FROM** emp

**WHERE** deptno = 10

**WITH READ ONLY**;

**INSERT INTO** v\_emp\_10

**VALUES**(1001,'奥马',6000,10);

--SQL 错误: ORA-42399: 无法对只读视图执行 DML 操作

1. **通过查询user\_views获取相关信息**

·和视图相关的数据字典

--user\_objects

--user\_views

--user\_updae\_columns

**SELECT** object\_name **FROM** user\_objects

**WHERE** object\_type = 'TABLE';

**SELECT** object\_name **FROM** user\_objects

**WHERE** object\_type = 'VIEW';

**SELECT** \*FROM user\_views

**WHERE** view\_name = 'v\_emp\_10';

1. **复杂视图**

--不允许DML 操作

1. 创建一个视图V\_EMP\_SALARY

--把职员的数据按部门名字和部门分组

--获得每一个部门的平均薪水，薪水总和，最高薪水和最低薪水

--聚合函数，函数，表达式必须使用别名

**CREATE OR REPLACE VIEW** V\_EMP\_SALARY

**AS**

**SELECT** d.chame,e.deptno ,avg(sal) 平均薪资，sum(sal) 薪资总和，max(sal) 最高薪水，min(sal) 最低薪水

**FROM** emp e **FULL OUTER JOIN** dept d

**ON** (e.deptno=d.deptno)

**GROUP BY** e.deptno,d.chame;

**SELECT** 薪资总和 **FROM** V\_EMP\_SALARY;

**CREATE VIEW** V\_EMP\_SALARY2

**AS**

**SELECT** d.chame,d.deptno,avg(sal) 平均薪资,sum(sal) 薪资总和,MAX(sal) 最高薪水,MIN(sal) 最低薪水

**FROM** emp e,dept d

**WHERE** e.deptno = d.deptno

**GROUP BY** d.chame,d.deptno;

**SELECT** \***FROM** V\_EMP\_SALARY2;

2）查看哪些员工比所在部门平均工资高

·使用子查询

**SELECT** e.ename,e.sal,e.deptno

**FROM** emp e,

(**SELECT** deptno，avg(sal) avg

**FROM** emp

**GROUP BY** deptno) t

**WHERE** e.deptno=t.deptno

**AND** e.sal >t.avg;

·使用视图

**SELECT** e.ename,e.sal,e.deptno

**FROM** emp e,V\_EMP\_SALARY2 s

**WHERE** e.deptno = s.deptno

**AND** e.sal>s.平均薪资;

3）查看那些员工的工资的一半都比30号部门的平均工资高

**SELECT** e.ename,e.sal,e.deptno

**FROM** emp e

**WHERE** e.sal/2>(**select** avg\_sal **from** v\_emp\_salary **where** deptno=30);

1. **删除视图：不影响基表**

**SELECT** \***FROM** V\_EMP\_SALARY2;

**DROP VIEW** V\_EMP\_SALARY2;

--删除视图中的数据：影响基表（不建议使用 忽略）

1. **序列**

·用于生成一组数字的数据库对象，通常是为某表的主键字段提供值

1. **创建序列**

**CREATE SEQUENCE** emp\_seq

**START WITH** 100 --序列从多少开始

**INCREMENT BY** 1; --序列的步距

**SELECT \* FROM** emp;

**2.使用序列**

--序列提供两个伪列

--nextval：获取序列的下一个值

--currval：获取序列的当前值

--nextval在新创建的序列中使用，会首先获取stare with指定的值。以后则是用最后一次生成的数字加上步长得到，序列不可以后退，所以当调用nextval后就无法再得到之前生成的数字

**SELECT** emp\_seq.nextval **FROM** dual;

--currval可以获得最后生成的数字，无论调用多少次值都一样，除非调用nextval生成下一个数字

--注意：刚创建的序列必须先执行一次nextval后才可以使用currval

**SELECT** emp\_seq.currval FROM dual;

**INSERT INTO** emp(empno,ename,sal)

**VALUES**(emp\_seq.nextval,'张无忌',8000);

1. **删除序列**

·序列和数据表都是独立存在的。删除数据表后。得删除表使用的序列，要不一会有序列残留

**DROP** **sequence** emp\_seq;

1. **索引（了解）**

--用于提高查询效率，排序效率

--索引可以建立在一张表中的单列也可以多列上

--索引的算法是数据库自动维护的。我们无需关注

--当进行查询或者排序时，数据库自动使用可用的索引

--简单索引（单列索引）

--在emp表中的ename列上建立索引

CREATE INDEX id\_emp\_ename ON emp(ename);

--复杂索引，

--也叫多列索引，是基于把多个列的索引

CREATE INDEX idx\_emp\_job\_sal ON emp(JOB,sal);

SELECT ename,JOB,sal,deptno FROM emp

ORDER BY JOB,sal;

--基于函数的索引

CREATE INDEX emp\_ename\_upper\_idx ON emp(upper(ename));

--删除索引

DROP INDEX id\_emp\_ename;

1. **约束（掌握）**
2. **约束的作用**

·约束（constraint）的全称是约束条件，也称作完整性约束条件

·约束是在数据表上强制执行的一些数据校验规则，当执行DML操作时，数据必须符合这些规则，若不符合，则无法执行

·约束条件可以保证表中数据的完整性，保证数据间的商业逻辑

1. **约束的类型**
2. 非空约束(NOT NULL)

·建表时添加非空约束

·非空约束用于确保字段值不为空

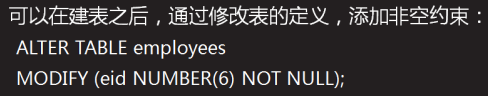
·默认情况下，任何列都允许有空值，但业务逻辑可能会要求某些列不能取空值

·当某个字段被设置列非空约束条件，这个字段中必须存在有效值，即

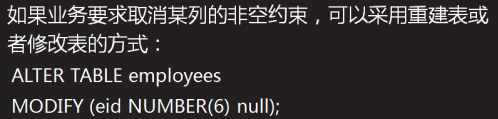
--当执行insert操作时，必须提供这个列的数据

--当执行update操作时，不能给列的值设置为null

1. 建表时添加非空约束
2. 修改表时添加非空约束



1. 取消非空约束

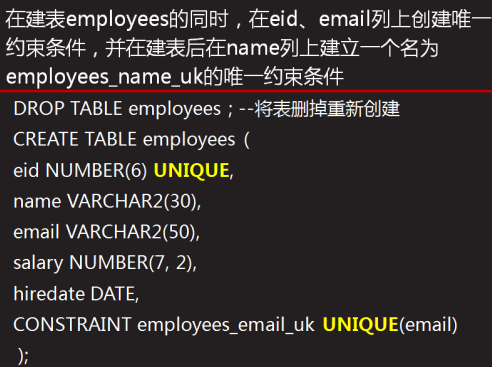


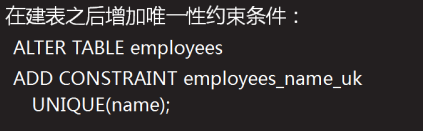
1. 唯一性约束(Unique)

·唯一性约束条件用于保证字段或者字段的组合不出现重复值

·当给表的某个列定义列唯一约束条件，该列的值不允许重，但允许是null值

·唯一性约束条件可以在建表同时建立，也可以在建表以后再建立





1. 主键约束(Primary Key)

（1）主键的意义

·主键约束条件从功能上看相当于非空(NOT NULL)且唯一(Unique)

的组合

·主键字段可以是单个字段或者多个字段，即在主键约束下的单字段或者多字段组合上不允许有空值，也不允许有重复值

·主键可以用来在表中唯一的确定一行数据

·一个表上只允许建立一个主键，而其他约束条件则没有明确个数限制

1. 主键的选取原则

·主键是对系统无意义的数据

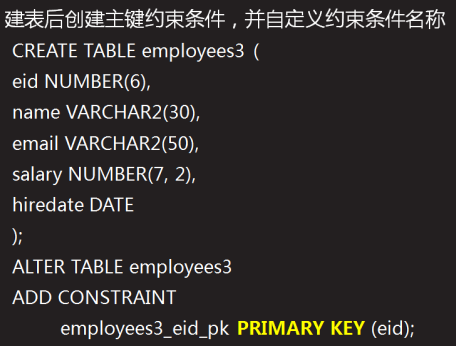
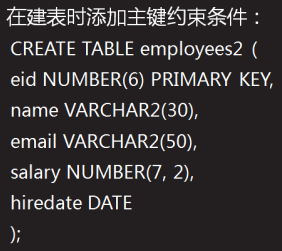
·永远不要更新主键，让主键除了唯一标识一行之外，再无其他用途

·主键不应包含动态变化的数据，如时间戳

·主键应自动生成，不要人为干预，以免是它带有除了唯一标识一行以外的意义

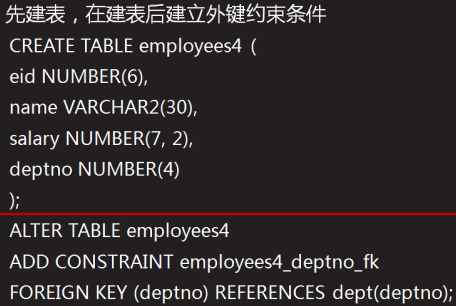
·主键尽量建立在单列上

1. 添加主键约束



1. 外键约束(Foreign Key)
2. 添加外键约束

·先建表，再添加外键约束



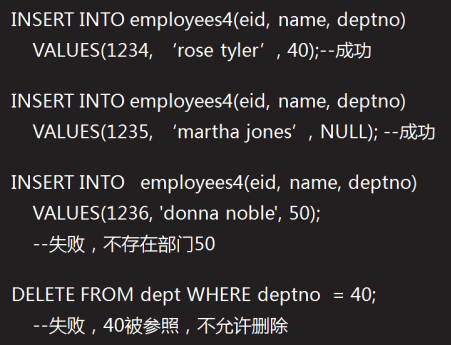
1. 外键约束对一致性的维护

·外键约束条件包含两个方面的数据约束：

--从表上定义的外键值的列值必须从主表被参照的列值中选取

--当主表参照列的值被从表参照时，主表的该行记录不允许被删除

即：外键和主键有主从表关系 主表的外键指向从表的主键



1. 外键约束对性能的降低

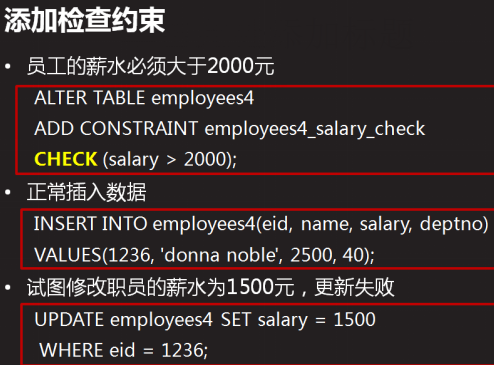
·如果在一个频繁DML操作的表上建立外键，每次DML操作，都会将导致数据库自动对外键所关联的对应表做检查，产生开销，如果已在程序中控制逻辑，这些判断将增加额外负担，可以省区

·外键确定了主从表的先后生成关系，有时会影响业务逻辑

1. 检查约束(Check)

·检查约束条件用来强制在字段上的每一个值都要满足Check中定义的条件

·当定义列Check约束的列新增或修改数据时，必须复合Check约束中定义的条件



第二部分plsql语法

PLSQL语法

**一．基本语法**

**1.PLSQL的注释**

· 两种 -- /\* \*/

**2.PLSQL程序的构成模块**

**declare:**

定义声明部分，以declare为标识，在该部分常包含常量，变量，游标，异常

plsql用到的所有定义必须在此集中定义，不允许在执行过程中定义

**begin**

执行语句必须从begin开始，以end结束

该部分是每个plaql必备的，包含列队数据库的操作和流程控制语句

**EXCEPTION**（可省略）

异常处理部分，该部分属于执行部分，对程序执行产生的异常进行处理

**end**

end必须以分号结尾“；”

·eg：演示

**SET** serveroutput **ON** --**提交语句：**每次新打开程序，提交plsql代码块时运行一次

**DECLARE**

i INT **:=** 10; --赋值语句 ：=

**BEGIN**

dbms\_output.put\_line(i); --**输出语句**

**END;**

**DECLARE**

i NUMBER **:=** 5;

j NUMBER; --先声明

**BEGIN**

j **:=** 6; --再初始化

dbms\_output.put\_line('id的变量值为'||i); --输出语句

dbms\_output.put\_line('j的变量值为'||j);

**END;**

**3.oracle中变量的标识符许满足以下规范：**

1）必须以字母开头，不能有空格

2）长度不能超过30个字符

3）不能使用保留字

4）常量/变量名称不区分大小写，在字母后可带数字和特殊字符#（只能带#）

·eg：求和

**DECLARE**

v\_a **NUMBER** **:=** 5;

v\_b **NUMBER** **:=** 6;

v\_sum **NUMBER**;

**BEGIN**

/\*

v\_sum **:=** v\_a + v\_b ;

dbms\_output.put\_line('v\_sum的变量值为' **||** v\_sum);

\*/

dbms\_output.put\_line('v\_sum =' **||** (v\_a + v\_b));

**END;**

**4.基表数据类型：**

数字类型 number(n) number(p,s)

数值类型 int long clob

字符类型 char varchare2

日期类型 date

布尔类型 true false null

注意：oracle中没有布尔类型，布尔类型是plsql中卫方便编程添加的数据类型

1. **常量**

·常量必须定义时初始化，且在执行过程中不可改变

·语法：常量名 constant 类型标识符 := 值

·演示常量的定义

**DECLARE**

i constant INT **:=** 20;

**BEGIN**

-- i **:=** 30; --PLS-00363: 表达式 'I' 不能用作赋值目标

--常量值不能被改变

dbms\_output.put\_line(i);

**END**;

1. **从键盘获取值**

1）获取数值类型

**DECLARE**

v\_i NUMBER;

**BEGIN**

v\_i **:=** &n; --从键盘获取值 &后面跟任意字母

dbms\_output.put\_line(v\_i);

**END**;

2）获取字符串

·输入时该字符串必须带单引号

**DECLARE**

v\_name varchar2(8);

**BEGIN**

v\_name **:=** &n; --从键盘获取值 &后面跟任意字母

dbms\_output.put\_line(v\_name);

**END;**

**7.复合数据类型：**

1.）%TYPE 列

·%type：动态获取表中的列的数据类型

·该变量的数据类型与指定表中的列的数据类型一致，

· 一旦表中的类型变化列，他会自动获得新的数据类型

·语法：变量名 用户名.表名.列名%TYPE

**DECLARE**

v\_name zhangyi.EMP.ename%TYPE;

v\_sal zhangyi.EMP.sal%TYPE;

**BEGIN**

**SELECT** ename,sal **INTO** v\_name ,v\_sal

**FROM** emp

**WHERE** empno = 7788;

dbms\_output.put\_line('name='**||**v\_name**||**'sal='**||**v\_sal);

**END;**

理解：zhangyi.EMP.ename%TYPE 数据库的用户名.表名.字段名%TYPE

当定义变量用来接收表格中的某个字段的值，须将该变量的数据类型设置成和将要获取字段的数据类型一致，此时就需要动态的获取表格中某个字段的数据类型。这种定义方式很类似于java中导入类时的全限定名

2） %ROWTYPE

·%ROWTYPE:动态获取表中一条数据类型的所有数据类型

·语法：变量名 用户.表名.%ROWTYPE

·使用时：变量名.列名

**DECLARE**

v\_row zhangyi.emp%ROWTYPE;

**BEGIN**

**SELECT** \* **INTO** v\_row --into 将查找到的东西赋给变量

**FROM** emp

**WHERE** empno **=** 7788;

dbms\_output.put\_line('name='**||**v\_row.ename**||**'sal='**||**v\_row.sal);

**END;**

3 ）记录型

·&记录型变量 部分列

·记录型是一个新的数据类型，用户自定义的，与java中相似

·用法：声明记录类型

·语法： TYPE 记录类型名 is RECORD

(变量名 类型，变量名 类型)

**DECLARE**

**TYPE** REC\_record **IS** record

(

v\_name **VARCHAR2**(20),

v\_hire **DATE**,

v\_sal **NUMBER**(20)

);

--声明记录类型的变量

V\_RE REC\_record;

**BEGIN**

--使用该变量

**SELECT** ename,hiredate,sal **INTO** V\_RE

**FROM** emp

**WHERE** empno = 7788;

dbms\_output.put\_line('name=' **||** V\_RE.v\_name);

**END;**

**8.表达式：**

1）数值表达式: + - \* / \*\*(乘方)

2）字符运算符： ||

3）关系表达式： < > <= >= != like in between

4）逻辑表达式： not and or

5）函数表达式： to\_date to\_char to\_num

**DECLARE**

v\_s NUMBER;

**BEGIN**

v\_s **:=** 4\*\*3;

dbms\_output.put\_line(v\_s);

**END;**

**二．流程控制**

**1.条件控制**

·语法：if 条件1 then 满足条件1执行语句

elsif 条件2 then 满足条件2执行语句

else 条件都不满足执行语句

end if；

if。。。。end if；匹配

·eg：输入值，如果大于10，显示大数，

**DECLARE**

i INT;

**BEGIN**

i **:=** &n;

**IF** i > 10 THEN

dbms\_output.put\_line('大数');

**ELSE**

dbms\_output.put\_line('小数');

**END IF**;

**END;**

·eg：输入值，判断如果大于100显示是三位数，如果小于100大于10显示两位数否则显示一位数

**DECLARE**

i INT;

**BEGIN**

i **:=** &n;

**IF** i > 100 THEN

dbms\_output.put\_line('三位数');

**ELSIF** i>10 and i<100 THEN

dbms\_output.put\_line('两位数');

**ELSE**

dbms\_output.put\_line('一位数');

**END IF**;

**END**;

·eg输入员工编号，查看该员工是否有奖金，如果有输出有奖金，如果没有，输出没奖金

**DECLARE**

v\_row zhangyi.emp%ROWTYPE;

v\_e number(20);

**BEGIN**

v\_e := &n;

**SELECT** \* **INTO** v\_row --into 将查找到的东西赋给变量

**FROM** emp

**WHERE** empno = v\_e;

**IF** v\_row.comm **>** 0 then

dbms\_output.put\_line('comm='**||**v\_row.comm);

**ELSE**

dbms\_output.put\_line('无奖金');

**END IF**;

**END;**

**2.循环**

1）loop基础循环

语法是：

loop ----循环开始

循环语句

exit when；------退出条件

end loop； ----循环结束

·eg：循环输出10个数字

**DECLARE**

i NUMBER := 0;

**BEGIN**

**loop**

dbms\_output.put\_line(i);

i := i + 1;

exit WHEN i = 10;

**END loop**;

**END**;

2)while循环

语法：

while 满足条件

loop

循环语句

end loop；

·eg：循环输出10个数字

**DECLARE**

i NUMBER := 1;

**BEGIN**

**while** i < 10

**loop**

dbms\_output.put\_line(i);

i := i + 1;

**END loop**;

**END;**

·eg：向dept表中插入50，60，70，80，90，100部门编号

**DECLARE**

v\_n NUMBER;

**BEGIN**

v\_n := 50;

**while** v\_n < 100

**loop**

INSERT INTO dept (deptno) VALUES (v\_n);

v\_n := v\_n + 10;

**END loop**;

**END;**

3)for循环

语法：

for 循环变量 in 循环下界 .. 循环上界

loop

循环语句

end loop；

·eg：循环输出10个数字

**DECLARE**

v\_n NUMBER := -10;

**BEGIN**

**FOR** v\_n **IN** -10..0

**loop**

dbms\_output.put\_line(v\_n);

END loop;

**END**;

1. **异常处理**

·异常处理（exception）是用来处理正常执行过程中未预料的事件

·程序的异常处理预定义的错误和自定义错误

·plsql程序块一旦产生异常而没有指出如何处理时，程序就会终止运行

·异常分3种：

1）oracle中自己定义24种异常，这种异常无需在程序中定义，由oracle自动将其引用

2）非预定义异常：即其他标准的oracle错误，对这种异常的处理，需要用户在程序中定义，然后由oracle将其引用

3）用户定义异常：程序执行过程中，出现编程人员认为的非正常情况，对这种异常处理，需要用户在程序中定义，然后显示在程序中将其引用

1. **预定义异常**

·异常处理部分一般放在plsql程序体后半部分，结构为

EXCEPTION

when first\_exception then .....

when second\_exception then ......

when others then ......

END;

注意：when others then 必须放在最后面

·eg：预定义异常案例

--输入一个员工编号，输出该员工的工资

**DECLAR**E

v\_empno zhangyi.emp.empno%TYPE;

v\_sal zhangyi.emp.sal%TYPE;

**BEGIN**

v\_empno := &n;

**SELECT** sal **INTO** v\_sal FROM emp WHERE empno = v\_empno;

dbms\_output.put\_line(v\_sal);

**exception**

WHEN no\_data\_found THEN

dbms\_output.put\_line('该员工不存在,请重新输入：');

**END**;

--员工工资小于1500，加200，该员工以更新，大于的输出超过规定值

**DECLARE**

v\_sal zhangyi.emp.sal%TYPE;

v\_empno zhangyi.emp.empno%TYPE;

**BEGIN**

v\_empno := &n;

**SELEC**T sal **INT**O v\_sal FROM emp WHERE empno = v\_empno;

**IF** v\_sal<1500 THEN

update emp set sal = sal +200 WHERE empno = v\_empno;

dbms\_output.put\_line('该员工以更新');

**ELSE**

dbms\_output.put\_line('该员工工资已超过规定值');

**END IF**;

exception

**WHEN** no\_data\_found **THEN**

dbms\_output.put\_line('该员工不存在,请重新输入：');

**WHEN** others **THEN**

dbms\_output.put\_line('程序有错，请调试');

**END**;

1. **非预定义异常**

·针对此异常处理，必须先对异常进行定义

语法：

异常情况 exception；

将定义好的异常情况与标准的oracle错误联系起来，使用exception\_init

pragma exception\_init （异常情况，异常错误）

·eg:输入一个员工编号，删除该部门的员工

**DECLARE**

v\_deptno emp.deptno%TYPE **:=** &n;

--声明异常

deptno\_remaining exception;

--于oracle链接

pragma exception\_init(deptno\_remaining,-2292);

-- -2292违反完整约束条件

**BEGIN**

**DELETE FROM** dept **WHERE** deptno **=** v\_deptno;

exception

**WHEN** deptno\_remaining **THEN** --用自己定义的异常处理异常

dbms\_output.put\_line('违反完整约束条件');

**WHEN** others **THEN**

dbms\_output.put\_line('程序有错，请调试');

**END;**

**3.用户自定义异常**

·当与一个异常错误相关的错误出现时，就会隐含触发该异常错误用户自定义异常是通过显示使用RAISE语句来触发，当引发一个异常错误时，控制就转向到exception块部分，执行错误处理

·异常语法：

在plsql块声明部分定义异常

异常情况 exception

RAISE 异常情况

在plsql块的异常情况处理部分对异常情况做出相应的处理

1）根据员工编号，上调员工的工资，加200

**DECLARE**

v\_empno emp.empn%TYPE **:=** &n;

**BEGIN**

**UPDATE** emp SET sal = sal +200 **WHERE** empno = v\_empno;

**END**;

·当输入123.程序运行结果 匿名块已完成，因为没123这个编号，

·应该报错，但是没报错，就需要自定义异常

**DECLARE**

v\_empno emp.empno%TYPE := &n;

no\_result exception; --声明异常

**BEGIN**

**UPDATE** emp **SET** sal = sal +200 **WHERE** empno = v\_empno;

**IF** SQL%notfound **THEN**

raise no\_result; --手动触发异常

**END IF**;

exception

**WHEN** no\_result **THEN**

dbms\_output.put\_line('更新失败');

**WHEN** others **THEN**

dbms\_output.put\_line(sqlcode);

**END**;

1. **函数和存储过程**

·oracle提供列可以把plsql程序存储在数据库中，并且可以在任何地方来运行，这样就叫做**存储过程或函数**。过程和函数统称为plsql子程序，被命名为plsql程序块，均存储在数据库中，并通过输入输出参数与调用者之间交换信息，过程和函数唯一的区别是函数总返回数据，过程不需要返回数据

1. **函数**

·带名的plsql块，是oracle数据库的一个对象，有返回值，必须放在表达式中调用

·语法：

create or replace function 函数的名字

[（参数1 in/out /in out 类型，参数2 in/out 类型，参数3 in/out 类型，。。。）]

--[]表示可有可无，参数默认是in

return 返回值类型

is/as

返回的变量及类型和类型的描述

begin

。。。

end；

·eg：求和

CREATE OR REPLACE FUNCTION sum\_fun

(v\_a IN NUMBER,v\_b IN NUMBER)

RETURN NUMBER

AS

v\_sum NUMBER(10);

BEGIN

v\_sum := v\_a+v\_b;

RETURN v\_sum;

END;

SELECT sum\_fun(1,2)FROM dual; --调用函数

·eg：由员工的编号查找员工的姓名

CREATE OR REPLACE FUNCTION fun\_01

(v\_empno IN NUMBER) --参数类型为in型

RETURN VARCHAR2

AS

v\_name emp.ename%TYPE;

BEGIN

SELECT ename INTO v\_name FROM emp WHERE empno = v\_empno;

RETURN v\_name;

END;

DECLARE --调用函数

v\_n emp.ename%TYPE;

BEGIN

v\_n := fun\_01(7788);

dbms\_output.put\_line(v\_n);

END;

·如何调用函数

函数声明时所定义的参数成为形式参数，应用程序所传递的 参数成为实际参数应用程序在调用函数时，传参有以下三种方法：

·eg：由部门编号，输出这个部门的人数，返回部门的工资总和

CREATE OR REPLACE FUNCTION get\_salary

(dept\_no in NUMBER,emp\_count out NUMBER)

RETURN NUMBER

AS

v\_sum NUMBER;

BEGIN

SELECT sum(sal),count(\*) INTO v\_sum,emp\_count FROM emp

WHERE deptno = dept\_no;

RETURN v\_sum;

exception

WHEN no\_data\_found THEN

dbms\_output.put\_line('查询数据不存在');

WHEN others THEN

dbms\_output.put\_line('其他错误');

END;

1. 位置表示法传递参数

·语法：

[参数1，参数2]

·案例

DECLARE

v\_sum NUMBER;

v\_num NUMBER;

BEGIN

v\_sum := get\_salary(30,v\_num);

dbms\_output.put\_line('30号部门工资总和'||v\_sum||'部门人数'||v\_num);

END;

1. 名称表示法传递参数

·语法：

形参1=>实参，[...]

·案例

DECLARE

v\_num NUMBER;

v\_sumsal NUMBER;

BEGIN

v\_sumsal := get\_salary(dept\_no => 30,emp\_count => v\_num);

dbms\_output.put\_line('30号部门工资总和'||v\_sumsal||'部门人数'||v\_num);

END;

1. 混合表示法传递参数

·即在调用ige函数，同时使用位置表示法和名称表示法为函数传递参数，采用这种参数传递，使用位置表示法所传递的参数必须放在名称表示法所传递的参数前面，也就是说，只要有一个参数使用名称表示法，其后面所有的参数都必须使用名称表示法

·案例

DECLARE

v\_num NUMBER;

v\_sumsal NUMBER;

BEGIN

v\_sumsal := get\_salary(30,emp\_count => v\_num);

dbms\_output.put\_line('30号部门工资总和'||v\_sumsal||'部门人数'||v\_num);

END;

·参数默认值

在create or replace function 语句中声明函数的参数，可以使用default关键字为输入参数指定默认值

CREATE OR REPLACE FUNCTION demo\_fun

(NAME VARCHAR2,age NUMBER,sex VARCHAR2 DEFAULT '男')

RETURN VARCHAR2

AS

v\_var VARCHAR2(32);

BEGIN

v\_var := NAME||','||age||'岁'||sex;

RETURN v\_var;

END;

DECLARE

var VARCHAR2(32);

BEGIN

var :=demo\_fun('掌握',age =>12);

--var :=demo\_fun('掌握',age =>12,sex =>'女');

dbms\_output.put\_line(var);

END;

·删除函数 使用drop

DROP FUNCTION demo\_fun;

1. **存储过程**

·将代码库存储到‘过程’对象中，方便使用

·语法:

create or replace procedure [用户名.p\_]存储过程名

[(参数1 in/out/in out 数据类型，...)]

is/as

类型.变量说明

begin

执行部分

[exception]

end；

·程序块

DECLARE

v\_name emp.ename%TYPE;

BEGIN

SELECT ename INTO v\_name FROM emp WHERE empno=8888;

exception

WHEN no\_data\_found THEN

dbms\_output.put\_line('编号不存在');

END;

1）不带参数的存储过程

CREATE OR REPLACE PROCEDURE zhangyi.p\_emp

AS

v\_name emp.ename%TYPE;

BEGIN

SELECT ename INTO v\_name FROM emp

WHERE empno = 8888;

exception

WHEN no\_data\_found THEN

dbms\_output.put\_line('编号不存在');

END;

（1）调用过程的方法

·用关键字execute在plsql中调用过程：

execute 过程名 [(参数1，。。。)]

·begin

过程名 [(参数1，。。。)];

end;

·eg：

execute zhangyi.p\_emp;

BEGIN

zhangyi.p\_emp;

END;

·eg：输出员工7788的姓名

CREATE OR REPLACE PROCEDURE zhangyi.p\_name

AS

v\_name emp.ename%TYPE;

BEGIN

SELECT ename INTO v\_name FROM emp

WHERE empno = 7788;

dbms\_output.put\_line(v\_name);

exception

WHEN no\_data\_found THEN

dbms\_output.put\_line('编号不存在');

END;

execute zhangyi.p\_name;

1. 带参数的存储过程
2. in 向过程中传递参数

·输出指定员工编号的员工姓名

CREATE OR REPLACE PROCEDURE zhangyi.p\_n

(v\_num emp.empno%TYPE)

AS --AS后面加返回值类型，下面将返回值类型输出列

v\_name emp.ename%TYPE;

BEGIN

SELECT ename INTO v\_name FROM emp

WHERE empno = v\_num;

dbms\_output.put\_line(v\_name);

exception

WHEN no\_data\_found THEN

dbms\_output.put\_line('编号不存在');

END;

·调用过程

--execute zhangyi.p\_n('7788');--传递指定参数

EXECUTE zhangyi.p\_n(&n);--从键盘获取参数

·根据员工的编号删除该员工，使用自定义异常

CREATE OR REPLACE PROCEDURE delemp

(v\_empno IN emp.empno%TYPE)

AS --在创建过程的时候，若要使用用户自定义异常，将异常定义在as后面

no\_result exception;

BEGIN

DELETE FROM emp

WHERE empno = v\_empno;

IF SQL%notfound THEN

raise no\_result;

END IF;

dbms\_output.put\_line('删除成功');

exception

WHEN no\_result THEN

dbms\_output.put\_line('员工编号不存在');

WHEN others THEN

dbms\_output.put\_line(sqlcode);

END;

execute delemp(&n);

1. out 调用过程有返回值

·输入员工编号，返回员工姓名

CREATE OR REPLACE PROCEDURE getname

(v\_empno NUMBER,v\_name out VARCHAR2)

IS -- is/as后面加返回值类型，如果没有，就无需定义

BEGIN

SELECT ename INTO v\_name FROM emp

WHERE empno = v\_empno;

dbms\_output.put\_line('员工的名字'||v\_name);

exception

WHEN no\_data\_found THEN

dbms\_output.put\_line('该员工不存在');

END;

--out类型参数在调用过程的时候，一般用declare。。。begin。。。end；而不用execute

--调用过程

DECLARE

v\_n emp.ename%TYPE;

BEGIN

getname(7369,v\_n);

END;

--注意：在调用带有out参数的存储过程，要用一个兼容的变量去接受返回值

--不用execute getname（&n）此方法报错

1. in out 既传入又传出

·输入数字然后输出

CREATE OR REPLACE PROCEDURE p\_01

(v\_empno IN out NUMBER)

AS

BEGIN

dbms\_output.put\_line('v\_empno:'||v\_empno);

v\_empno:=31;

END;

DECLARE

v\_n NUMBER:=&n;

BEGIN

p\_01(v\_n);

dbms\_output.put\_line(v\_n);

END;

·需求：根据员工编号，上调工资50%，如果上调成功，返回1，输出成功，否则返回0，输出失败

CREATE OR REPLACE PROCEDURE p\_02

(v\_empno IN out NUMBER)

AS

no\_result exception;

BEGIN

UPDATE emp SET sal = sal\*1.5 WHERE empno = v\_empno;

IF SQL%notfound THEN

raise no\_result;

END IF;

v\_empno := 1;

exception

WHEN no\_result THEN

v\_empno := 0;

END;

--调用过程

**DECLARE**

v\_num NUMBER := &n;

**BEGIN**

p\_02(v\_num);

**IF** v\_num = 1 THEN

dbms\_output.put\_line('成功');

**ELSE**

dbms\_output.put\_line('失败');

**END IF**;

**END**;

1. **游标**

·官方定义：

为了处理sql语句，oracle必须分配一片叫上下文（context）的区域来处理所必须的信息，其中包括处理行的数目，一个指向语句被分析以后的表现形式的指针以及查询的活动集（active set）。游标是一个指向上下文的句柄（handle）或指针。通过游标，plsql可以控制上下文和处理语句时上下文会发生的哪些事

·老师定义：

游标：就是从表中提取出来的数据，以临时表的形式存放在（上下文，工作区）内存中，在游标中有一个数据指针，在初始状态下，指针指向的是首位 ，利用fetch语句可以移动指针，从而对游标中的数据进行操作，然后将操作后的结果写回表中。

1. **游标的定义及使用**

·游标是一种数据类型，在declare中声明，一般放在声明的最后一条语句，

必须先定义游标变量，才可以使用。

1）声明游标

·语法：

**cursor 游标名 is select语句**

游标的使用必须放在begin中

1. 打开游标

·语法：

**open 游标名；**

打开游标实际就是执行游标对应的select语句，将其查询的结果放入工作区，并且指针指向工作区的首部

（1）将符合条件的记录存入内存；（2）将指针指向第一条数据

3）提取数据

提取数据：就是检索结果集中的数据行，放入指定的输出变量中。

**fetch 游标名 into 变量名**

·fetch：提取数据，移动指针

4）关闭游标

关闭游标：当提取和处理完游标结果集数据后，应及时关闭，已释放游标所占的内存资源，并使游标的工作区变无效，不能再使用fetch语句读取其中的数据。关闭后的游标可以使用open语句重新打开

**close 游标名**

--注：关闭游标后再打开游标，指针还是指向首部（第一行数据）

·案例

**SELECT** \* **FROM** emp;

**SET** serveroutput **ON**;

**DECLARE**

v\_emp zhangyi.emp%rowtype;--声明变量去接受fetch提取出来的数据

--声明游标

CURSOR c\_emp IS SELECT \* FROM emp;

**BEGIN**

--打开游标

**OPEN** c\_emp;

--提取数据

**fetch** c\_emp **INTO** v\_emp;

--fetch语句从第一行开始一行一行提取数据。fetch一次，提取一条数据

--fetch的功能：提取数据，移动指针

dbms\_output.put\_line('v\_emp:'||v\_emp.ename);

--关闭游标

**CLOSE** c\_emp;

**END**;

--eg：输出前10条姓名和工资

DECLARE

v\_ename emp.ename%TYPE;

v\_sal emp.sal%TYPE;

CURSOR c\_cursor IS SELECT ename,sal FROM emp WHERE ROWNUM <11;

BEGIN

OPEN c\_cursor;

fetch c\_cursor INTO v\_ename,v\_sal;

WHILE c\_cursor %found loop --%found表示还有数据为true

dbms\_output.put\_line(v\_ename||'...............'||v\_sal);

fetch c\_cursor INTO v\_ename,v\_sal;--游标下移

END loop;

CLOSE c\_cursor;

END;



--eg：输出工资大于1500的员工的姓名和工资

**DECLARE**

v\_ename emp.ename%TYPE;

v\_sal emp.sal%TYPE;

CURSOR c\_cursor IS SELECT ename,sal FROM emp WHERE sal >1500;

**BEGIN**

OPEN c\_cursor;

fetch c\_cursor INTO v\_ename,v\_sal;

WHile c\_cursor %found loop

dbms\_output.put\_line(v\_ename||'...............'||v\_sal);

fetch c\_cursor INTO v\_ename,v\_sal;--游标下移

END loop;

CLOSE c\_cursor;

**END**;

--参数化游标

**DECLARE**

v\_ename emp.ename%TYPE;

v\_sal emp.sal%TYPE;

**CURSOR** c\_cursor(v\_s emp.sal%TYPE) **IS SELECT** ename,sal **FROM** emp **WHERE** sal >v\_s;

**BEGIN**

**OPEN** c\_cursor(&n);--传参

**fetch** c\_cursor INTO v\_ename,v\_sal;

**WHile** c\_cursor %found loop

dbms\_output.put\_line(v\_ename||'...............'||v\_sal);

**fetch** c\_cursor INTO v\_ename,v\_sal;--游标下移

**END loop**;

**CLOSE** c\_cursor;

**END**;

1. **游标的属性**

·语法： **游标名 [属性]**

1. %isopen

布尔型属性，当游标打开，返回true，否则返回false

1. %found

布尔类型属性，当最近一次读取记录时成功返回，返回true，否则返回false

1. %notfound

布尔类型属性，鱼%found相反

1. %rowcount

数字型属性，返回已从游标中读取的记录数

该属性用于返回游标的数据行数，若返回0，表名游标已经打开，但没有提取出数据

1. sql%notfound

sql语句执行没有找到数据

1. sql%rowcount

sql语句执行的数据数据行

·案例

**DECLARE**

v\_ename zhangyi.emp.ename%TYPE;

v\_sal emp.sal%TYPE;

CURSOR c\_emp IS SELECT ename,sal FROM emp;

**BEGIN**

OPEN c\_emp;

IF c\_emp %isopen THEN

dbms\_output.put\_line('游标、一打开');

ELSE

dbms\_output.put\_line('游标没打开');

END IF;

close c\_emp;

**END**;

**DECLARE**

v\_ename zhangyi.emp.ename%TYPE;

v\_sal emp.sal%TYPE;

CURSOR c\_emp IS SELECT ename,sal FROM emp;

**BEGIN**

OPEN c\_emp;

fetch c\_emp INTO v\_ename,v\_sal;

IF c\_emp %found THEN

dbms\_output.put\_line('成功执行');

dbms\_output.put\_line('共查询'|| c\_emp%rowcount||'条数据');

ELSE

dbms\_output.put\_line('没有数据');

END IF;

close c\_emp;

**END**;

--查询emp表中共有多少条记录，用游标

**DECLARE**

v\_ename zhangyi.emp.ename%TYPE;

v\_sal emp.sal%TYPE;

CURSOR c\_emp IS SELECT ename,sal FROM emp;

**BEGIN**

OPEN c\_emp;

fetch c\_emp INTO v\_ename,v\_sal;

WHile c\_emp %found

**loop**

dbms\_output.put\_line(v\_ename||'...............'||v\_sal);

fetch c\_emp INTO v\_ename,v\_sal;--游标下移

**END loop**;

dbms\_output.put\_line('共查询'|| c\_emp%rowcount||'条数据');

**END**;

--把emp表中的书放入myemp表中，删除myemp表中emp=7788的员工，如果找到输出，找到这个值删除他，并输出删除列几行如果找不到，输出为找到

CREATE TABLE myemp AS SELECT \* FROM emp;

SELECT \* FROM myemp;

**DECLARE**

BEGIN

DELETE FROM myemp WHERE empno = 7934;

--DELETE FROM myemp WHERE empno in (7934,7876);--删除多行

IF **SQL%notfound** THEN

dbms\_output.put\_line('没有数据');

ELSE

dbms\_output.put\_line('找到值，删除');

dbms\_output.put\_line('删除了'||sql%rowcount||'行');

END IF;

**END**;

1. **游标的for循环**

·plsql语句提供列游标的for循环语句，自动执行游标的open，fetch，close语句和循环语句的功能。

·当进入循环时，游标for循环自动打开open游标；并提取第一行数据，当程序处理完当前当前所有提取的数据而进入下一次循环时，游标for循环自动提取下一行数据，直到提取完为止。

·语法：

for rowtype变量 in 游标名|查询语句

loop

循环体

end loop；

注意：不需要打开游标，自动打开，字段关闭

自动提取数据：自动fetch数据到rowtype变量

1）无参数的游标for循环

·eg：查询员工的编号，姓名，工资然后输出

**DECLARE**

CURSOR c\_sal IS SELECT empno,ename,sal FROM emp;

**BEGIN**

--隐含打开游标

FOR v\_sal IN c\_sal

loop

--隐含执行fetch语句

dbms\_output.put\_line(v\_sal.empno||','||v\_sal.ename||','||v\_sal.sal);

--隐含检测c\_sal%not found

END loop;

--隐含关闭游标

**END**;

--游标for循环

**DECLARE**

**BEGIN**

**FOR** v\_sal IN ( SELECT empno,ename,sal FROM emp)

**loop**

dbms\_output.put\_line(v\_sal.empno||','||v\_sal.ename||','||v\_sal.sal);

**END loop**;

**END**;

·eg：查询部门号为20的员工的工资，工种

**DECLARE**

**BEGIN**

FOR v\_sal IN ( SELECT sal,JOB FROM emp WHERE deptno = 20 ) loop

dbms\_output.put\_line(v\_sal.sal||','||v\_sal.JOB);

END loop;

**END**;

2）带默认参数的游标for循环

**DECLARE**

**CURSOR** c\_cursor(dept\_no number default 10)

IS SELECT chame,loc FROM dept WHERE deptno=dept\_no;

**BEGIN**

dbms\_output.put\_line('dept\_no参数为30');

**FOR** c1\_rec **IN** c\_cursor(30) loop --传递参数为30

dbms\_output.put\_line(c1\_rec.chame||','||c1\_rec.loc);

END loop;

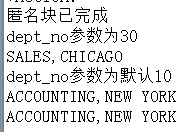
dbms\_output.put\_line('dept\_no参数为默认10');

**FOR** c1\_rec **IN** c\_cursor loop --使用默认参数

dbms\_output.put\_line(c1\_rec.chame||','||c1\_rec.loc);

END loop;

**END**;



SELECT \* FROM dept;

·eg：查询工资在2000(默认值)以上的员工的姓名，也可以指定其他值

**DECLARE**

CURSOR c\_sal(v\_sal emp.sal%TYPE DEFAULT 2000)

IS SELECT ename,sal FROM emp WHERE sal>v\_sal;

**BEGIN**

**FOR** v\_rec **IN** c\_sal loop --使用默认值

dbms\_output.put\_line(v\_rec.ename||' , '||v\_rec.sal);

END loop;

dbms\_output.put\_line('----------------');

**FOR** v\_rec **IN** c\_sal(&n) loop --键盘录入值

dbms\_output.put\_line(v\_rec.ename||' , '||v\_rec.sal);

END loop;

**END**;

1. **包**

**1.一个包是由两个分开的部分组成：**

**包定义(package)：**包定义部分声明数据类型，变量，常量，游标，

子程序和异常处理等元素，这些元素为包的公有元素。

**包体定义(package body)：**包体定义部分主要是包定义部分的具体实现，

它定义了包定义部分所声明的游标和子程序，

在包体中还可以声明包的私有元素。

包定义和包体定义是分开编译的，并作为两部分分开的对象存放在数据库字典中。

1）包定义的语法：

create or replace package package\_name

is/as

公有的元素

end[package\_name];

2）包体定义的部分：

create or replace package body package\_name

is/as

私有元素；

公有元素的具体实现；

begin

plsql语句

end[package\_name];

注意：在包体中定义公有元素的具体实现时，

必须与包定义中所声明的子程序格式完全一致。

·包{

变量

常量

游标

子程序(函数和过程)

异常

}

包名.元素

包由包定义和包体定义两部分组成：

包定义：声明部分，所有公有的元素；

包体定义：将包中的声明全部实现，另外可以添加自己私有的元素。

包：是一组变量，常量，函数，过程，游标等plsql程序设计元素的组合。

·案例

·包定义部分

CREATE OR REPLACE PACKAGE pack\_01

IS

--声明变量

v\_a NUMBER;

--声明常量

v\_pi constant NUMBER (5,4) := 3.1415;

--声明函数

FUNCTION fun\_01 (ID INT) RETURN NUMBER;

--声明过程

PROCEDURE p\_01;

end pack\_01;

--包体定义部分

CREATE OR REPLACE PACKAGE BODY pack\_01

IS

--函数的实现

FUNCTION fun\_01(ID INT)

RETURN NUMBER

IS

BEGIN

dbms\_output.put\_line(ID);

RETURN v\_pi;

END fun\_01;

--过程的实现

PROCEDURE p\_01

IS

BEGIN

dbms\_output.put\_line('过程被调用');

END p\_01;

END pack\_01;

**2.包的调用**

DECLARE

v\_num NUMBER(5,4);

BEGIN

--函数调用

v\_num := pack\_01.fun\_01(9);

dbms\_output.put\_line(v\_num);

END;

--过程调用

BEGIN

pack\_01.p\_01;

END;

**3.删除包**

DROP PACKAGE [BODY] zhangyi.pack\_o1;

用户.包名

4**.数据字典**：

1）查询包

·user\_objects 保存包的名字，状态，类型，创建时间

SELECT \* FROM user\_objects;

2）usre\_source 保存包的代码

SELECT \* FROM user\_source where name = 'PACK\_01';

1. **触发器**

·触发器在数据库中以独立的对象存储，它与存储过程不同的是，存储过程通过

其他程序来启动运行，或者直接启动运行。而触发器是由一个事件来启动运行，

即触发器是当某个事件发生时自动隐身运行。并且触发器不能接受参数，所以运

行触发器就叫做点火或触发。

·oracle事件中指的是对数据库的表进行insert，delete，update操作或者对视图进行类似的操作。

·oracle将触发器的功能扩展触发oracle，像数据库的启动或关闭等

**·创建触发器**

create or replace trigger tigger\_name

before/after

insert/delete/update

for each row --行级触发，针对每一行去触发

[when contition]

tigger\_body;

·before/after ：指触发器触发的时间顺序分别为触发前和后触发方式。

前触发是在执行触发事件之前触发当前所创建的触发器

后触发是在执行触发事件之后触发当前所创建的触发器

·for each row 选项说明触发器为行触发器

行触发器要求当DML语句操作影响数据库中的多行数据时，对其中的每个数据行，

只要他们满足约束条件，均激活一次触发器

如果省略行触发器（for each row），before/after触发器均为语句触发器

·when子句是触发的约束条件，contition为一个逻辑表达式，其中必须包含相关的名称，

不能包含查询语句，也不能包含plsql语句

·when字句指定的触发约束条件，只能用在before/after行触发器中，不能用于其他触发器

**· 什么时候使用触发器：（记）**

当一个基表被修改的时候（insert/delete/update）时要执行行的存储过程，执行时

根据其所依附的基表的改动而自动触发，因此与应用程序无关

**·用触发器的优点**：

可以保证数据的一致性和完整性（记）

·案例：

--创建一个学生表

CREATE TABLE student(

ID NUMBER(4) PRIMARY KEY,

NAME VARCHAR2(20),

sal NUMBER(5,1)

);

--创建一个序列

CREATE SEQUENCE student\_seq

START WITH 1--初始值

MINVALUE 1--最小值

MAXVALUE 9999--最大值

INCREMENT BY 1;

--插入一条数据

INSERT INTO student(ID,NAME,sal) VALUES(student\_seq.nextval,'张无忌',5000.0);

SELECT \* FROM student;

--创建触发器

CREATE OR REPLACE TRIGGER student\_t

--在对student表做插入之前，触发触发器

BEFORE INSERT ON student

--是一个行触发器

FOR EACH ROW

--开始

BEGIN

SELECT student\_seq.nextval INTO :NEW.ID FROM dual;

--从伪表中查询一个id赋给一行新的数据

END;

INSERT INTO student(NAME,sal) VALUES('张三丰',6700.0);

SELECT \* FROM student;

--当序列和触发器都被创建之后，就可以通过触发器来自动生成表中数据的主键

**第三部分：Jdbc**

1. **Jdbc的简介**

**1.jdbc定义**

·JDBC (java database connectivity):java 访问数据库的解决方案。用java代码去访问数据库。

·Jdbc：是一套标准的接口，即访问数据库的通用API，不同的数据库厂商根据各自不同的特点去实现这些接口

1）jdbc中的常见接口：

DriverManager :驱动管理

Connection :链接接口

Statement :语句对象接口

ResultSet： 结果集接口

1. **jdbc基础案例**

1）java数据库的链接

**/\***

**\* JDBC:java数据库的链接**

**\* 1.加载驱动**

**\* 2.与数据库建立连接**

**\* 3.执行sql语句**

**\* 4.处理结果集**

**\* 5.关闭连接**

**\* \*/**

**public class JDBCDemo01 {**

**public static void main(String[] args) throws SQLException {**

**Connection conn = null;**

**//1.加载驱动**

**try {**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**/\***

**\* ClassNotFoundException:异常出现原因**

**\* 1.Class.forName中字符拼写不正确**

**\* 2.jar包没导入环境**

**\*/**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**/\***

**\* 通过DriverManager类.静态方法getConnection建立连接**

**\* 三个参数：**

**\* 参数1:数据库的地址及端口（不同数据库参数值不同）**

**\* 参数2：数据库的用户名**

**\* 参数3：数据库密码**

**\*/**

**conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**/\***

**\* Statement针对不同的sql语句，提供了不同的执行方法：**

**\* ResultSet executeQuery(sql)**

**\* 该方专门用来执行DQL语句，返回值为结果集**

**\***

**\* int executeUpdate(String sql)**

**\* 该方法专门用来执行DML语句，返回数字，表示执行**

**\* 该语句后影响了表中多少条数据**

**\***

**\* boolean execute(String sql) 默认为false**

**\* 该方法理论上什么语言都可以执行，但是DQL,DML都有专门的方法执行，**

**\* 所以该方法通常用来执行DDL。返回值表示执行该方法后是否有结果集，如果**

**\* 查询有结果集则返回true，否则返回false**

**\***

**\* 注意：若sql语句写错，会抛sqlException异常**

**\***

**\*/**

**Statement state = conn.createStatement();**

**String sql = "select ename,sal,job,deptno from emp";**

**System.*out*.println(sql);**

**System.*out*.println("正在查询。。。");**

**ResultSet rs = state.executeQuery(sql);**

**System.*out*.println("查询完毕");**

**//4.遍历结果集**

**/\***

**\* ResultSet提供了用于遍历结果集的方法--next（）**

**\* boolean next（）**

**\* 该方法的作用：**

**\* 1.首先当我们查询出结果集后rs的指针指向第一条数据之上的位置，**

**\* 所以我们需要先调用一次next()使其指针指向第一条数据。**

**\* 2.看返回值：**

**\* 看返回值，若指针向下移动，若发现下面没有数据列，会返回false，**

**\* 若有返回true，所以我们只有在该方法返回true的情况下，获取**

**\* 当前记录的字段的值**

**\* rs还提供列若干个getXxx(String fielName)**

**\*/**

**while (rs.next()) {**

**//获取员工的名字**

**String ename = rs.getString("ename");**

**//获取工资**

**int sal = rs.getInt("sal");**

**//获取工种**

**String job = rs.getString("job");**

**//获取部门号**

**int deptno = rs.getInt("deptno");**

**System.*out*.println("姓名：" + ename + ",工资：" + sal + ",工作：" + job + ",部门号：" + deptno);**

**}**

**//5.关闭连接**

**rs.close();//先关闭小流，再关闭总流**

**} catch (Exception e) {**

**e.printStackTrace();**

**} finally {**

**if (conn != null) {**

**conn.close();//关闭总流**

**}**

**}**

**}**

**}**

2）查看每个员工的名字，职位，工资，部门名称，工作所在地

**/\***

**\* 需求：查看每个员工的名字，职位，工资，部门名称，工作所在地**

**\*/**

**public class JDBCDemo02 {**

**public static void main(String[] args) throws ClassNotFoundException, SQLException {**

**//1.加载驱动 Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe",**

**"zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**Statement state = conn.createStatement();**

**String sql = "SELECT e.ename,e.JOB,e.sal,e.deptno,d.loc "**

**+ "FROM emp e JOIN dept d "**

**+ "ON e.deptno = d.deptno";**

**System.*out*.println(sql);**

**System.*out*.println("正在查询。。。");**

**ResultSet rs = state.executeQuery(sql);**

**System.*out*.println("查询完毕");**

**//4.遍历结果集**

**while (rs.next()) {**

**//获取员工的名字**

**String ename = rs.getString("ename");**

**//获取工资**

**int sal = rs.getInt("sal");**

**//获取工种**

**String job = rs.getString("job");**

**//获取部门号**

**int deptno = rs.getInt("deptno");**

**//获取工作所在地**

**String loc = rs.getString("loc");**

**System.*out*.println("姓名：" + ename + ",工资：" + sal + ",工作：" + job +**

**",部门号：" + deptno+",工作所在地"+loc);**

**}**

**//5.关闭连接**

**rs.close();//先关闭小流，再关闭总流**

**conn.close();//关闭总流**

**}**

**}**

3）使用jdbc创建表

**/\***

**\* 创建一个表，表名userinfo**

**\* id number(6),**

**\* username varchar(36),**

**\* password varchar2(50),**

**\* email varchar2(50),**

**\* accont number(12,2)**

**\*/**

**public class JDBCDemo03 {**

**public static void main(String[] args) throws ClassNotFoundException, SQLException {**

**//1.加载驱动**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**Statement state = conn.createStatement();**

**String sql = "create table userinfo("**

**+ "id number(6),"**

**+ "username varchar(36),"**

**+ "password varchar2(50),"**

**+ "email varchar2(50),"**

**+ "accont number(12,2)"**

**+ ")";**

**if(!state.execute(sql)){**

**System.*out*.println("表创建完毕");**

**}else{**

**System.*out*.println("表创建失败");**

**}**

**conn.close();//关闭总流**

**}**

**}**

4）使用jdbc创建序列

**/\***

**\* 创建一个序列**

**\* sep\_userinfo\_id**

**\* 起始1**

**\* 步距1**

**\*/**

**public class JDBCDemo04 {**

**public static void main(String[] args) throws ClassNotFoundException, SQLException {**

**//1.加载驱动**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**Statement state = conn.createStatement();**

**String sql = "CREATE SEQUENCE sep\_userinfo\_id "**

**+ "START WITH 1 "**

**+ "INCREMENT BY 1 ";**

**if(!state.execute(sql)){**

**System.*out*.println("序列建完毕");**

**}else{**

**System.*out*.println("序列建失败");**

**}**

**conn.close();//关闭总流**

**}**

**}**

1. 将序列作为一个值，向表中插入数据

**/\***

**\*向表中插入一条数据**

**\*/**

**public class JDBCDemo05 {**

**public static void main(String[] args) throws ClassNotFoundException, SQLException {**

**//1.加载驱动**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe",**

**"zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**Statement state = conn.createStatement();**

**String sql = " insert into userinfo values(sep\_userinfo\_id.nextval,'张三','123456','@136',555.33)";**

**if((state.executeUpdate(sql))>0){**

**System.*out*.println("数据已插入");**

**}else{**

**System.*out*.println("数据已插入失败");**

**}**

**conn.close();**

**}**

**}**

1. 注册

**/\***

**\* 程序运行后：控制台先输出：欢迎注册**

**\* 提示：分别要求用户输入：用户名，密码，邮箱，余额**

**\* 保存成功后告知用户：注册成功**

**\*/**

**public class JDBCDemo06 {**

**public static void main(String[] args) throws ClassNotFoundException, SQLException {**

**System.*out*.println("欢迎注册");**

**System.*out*.println("请输入用户名");**

**String name = JDBCDemo06.*get*();**

**System.*out*.println("请输入密码");**

**String pasword = JDBCDemo06.*get*();**

**System.*out*.println("请输入邮箱");**

**String emile = JDBCDemo06.*get*();**

**System.*out*.println("请输入余额");**

**double num = JDBCDemo06.*get1*();**

**//String num = JDBCDemo06.get();**

**//1.加载驱动**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe",**

**"zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**Statement state = conn.createStatement();**

**String sql = "insert into userinfo values(sep\_userinfo\_id.nextval,'"+name+"','"+pasword+"','"+emile+"',"+num+")";**

**if((state.executeUpdate(sql))>0){**

**System.*out*.println("数据已插入");**

**}else{**

**System.*out*.println("数据已插入失败");**

**}**

**conn.close();**

**}**

**private static String get() {**

**Scanner s = new Scanner(System.*in*);**

**String t = s.next();**

**return t;**

**}**

**private static double get1() {**

**Scanner s = new Scanner(System.*in*);**

**double t = s.nextDouble();**

**return t;**

**}**

**}**

1. 登陆

/\*

\*登陆

\*要求：程序运行后。输出欢迎登陆

\*请输入用户名

\*请输入密码

\*用户输入完毕后根据用户的名字和查询密码查询该用户的信息，

\*若都对，则显示登陆成功，输出余额

\*若输入错误，则显示用户名或密码错误

\*/

**public class JDBCDemo07 {**

**public static void main(String[] args) {**

**while (true) {**

**Connection conn = null;**

**try {**

**System.*out*.println("欢迎登录！");**

**Scanner scan = new Scanner(System.*in*);**

**System.*out*.println("请输入用户名：");**

**String username = scan.nextLine();**

**System.*out*.println("请输入密码：");**

**String password = scan.nextLine();**

**String sql = "select id,username,password,email,accont from userinfo where username='" + username**

**+ "' and password='" + password + "'";**

**System.*out*.println(sql);**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "zhangyi", "12346");**

**Statement state = conn.createStatement();**

**ResultSet rs = state.executeQuery(sql);**

**if (rs.next()) {**

**double account = rs.getDouble("accont");**

**System.*out*.println("登陆成功！你的余额为" + account);**

**} else {**

**System.*out*.println("用户名或密码错误！请重新输入");**

**}**

**} catch (Exception e) {**

**e.printStackTrace();**

**} finally {**

**if (conn != null) {**

**try {**

**conn.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**}**

1. 使用jdbc进行分页查询

**/\***

**\* 显示emp表中的员工信息**

**\* 按照工资降序排序**

**\* 用户可以输入每页显示的条目数**

**\* 并显示用户输入的页数和对应的数据**

**\*/**

**public class JDBCDemo08 {**

**public static void main(String[] args) throws ClassNotFoundException, SQLException {**

**//用户自定义每页的条数和需要查看的页数**

**System.*out*.println("请输入查看的页数");**

**int i = (new Scanner(System.*in*)).nextInt();**

**System.*out*.println("请输入每页显示的条数");**

**int j = (new Scanner(System.*in*)).nextInt();**

**/\*--分页查询范围的算法**

**--pagesSize:每页的条目数**

**--page:第几页**

**--MIN:(page-1)\*pageSize+1**

**--max:page\*pageSize\*/**

**int min = (i-1)\*j+1;**

**int max = i\*j;**

**//1.加载驱动**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe",**

**"zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**Statement state = conn.createStatement();**

**String sql = "SELECT \*FROM (SELECT ROWNUM rn ,t.\* "**

**+ "FROM(SELECT ename,sal,job,deptno FROM emp ORDER BY sal DESC ) t)"**

**+ "WHERE rn BETWEEN "+min+" AND "+max;**

**System.*out*.println(sql);**

**System.*out*.println("正在查询。。。");**

**ResultSet rs = state.executeQuery(sql);**

**System.*out*.println("查询完毕");**

**//4.遍历结果集**

**System.*out*.println("您查看的是第"+ i +"页，每页显示"+ j +"条数据");**

**while (rs.next()) {**

**//获取员工的名字**

**String ename = rs.getString("ename");**

**//获取工资**

**int sal = rs.getInt("sal");**

**//获取工种**

**String job = rs.getString("job");**

**//获取部门号**

**int deptno = rs.getInt("deptno");**

**//获取工作所在地**

**System.*out*.println("姓名：" + ename + ",工资：" + sal + ",工作：" + job +**

**",部门号：" + deptno);**

**}**

**//5.关闭连接**

**rs.close();//先关闭小流，再关闭总流**

**conn.close();//关闭总流**

**}**

**}**

1. 输入id修改信息

/\*

\* 要求用户输入id号，以及对于的各项信息的新值

\* 然后可以进行修改该用户信息的修改操作

\*/

**public class JDBCDemo09 {**

**public static void main(String[] args) throws ClassNotFoundException, SQLException {**

**//获取用户输入的id**

**System.*out*.println("请输入id");**

**int i = (new Scanner(System.*in*)).nextInt();**

**//1.加载驱动**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**Statement state = conn.createStatement();**

**//查询用户数据**

**JDBCDemo09.*show*(state, i);**

**//修改用户嘻嘻**

**System.*out*.println("请选择需要更新的信息，1-姓名；2-密码；3-邮箱；4-余额");**

**JDBCDemo09.*change*((new Scanner(System.*in*)).nextInt(), i, state);**

**conn.close();**

**}**

**//定义方法去修改用户信息**

**private static void change(int nextInt, int i, Statement state) throws SQLException {**

**switch (nextInt) {**

**case 1:**

**System.*out*.println("请输入您想修改的名字");**

**String sql = "UPDATE userinfo SET username = '" + ((new Scanner(System.*in*)).next()) + "' WHERE ID = " + i;**

**if ((state.executeUpdate(sql)) > 0) {**

**System.*out*.println("数据已更新");**

**} else {**

**System.*out*.println("数据更新失败");**

**}**

**break;**

**case 2:**

**System.*out*.println("请输入您想修改的密码");**

**String s = "UPDATE userinfo SET password = '" + ((new Scanner(System.*in*)).next()) + "' WHERE ID = " + i;**

**if ((state.executeUpdate(s)) > 0) {**

**System.*out*.println("数据已更新");**

**} else {**

**System.*out*.println("数据更新失败");**

**}**

**break;**

**case 3:**

**System.*out*.println("请输入您想修改的邮箱");**

**String sq = "UPDATE userinfo SET email = '" + ((new Scanner(System.*in*)).next()) + "' WHERE ID = " + i;**

**if ((state.executeUpdate(sq)) > 0) {**

**System.*out*.println("数据已更新");**

**} else {**

**System.*out*.println("数据更新失败");**

**}**

**break;**

**case 4:**

**System.*out*.println("请输入您想修改的余额");**

**String sql1 = "UPDATE userinfo SET accont = '" + ((new Scanner(System.*in*)).nextInt()) + "' WHERE ID = " + i;**

**if ((state.executeUpdate(sql1)) > 0) {**

**System.*out*.println("数据已更新");**

**} else {**

**System.*out*.println("数据更新失败");**

**}**

**break;**

**default:**

**System.*out*.println("请输入正确的编号");**

**}**

**}**

**//定义方法查询用户信息**

**private static void show(Statement state, int i) throws SQLException {**

**String sq = "SELECT \* FROM userinfo WHERE ID = " + i;**

**System.*out*.println(sq);**

**System.*out*.println("正在查询。。。");**

**ResultSet rs = state.executeQuery(sq);**

**System.*out*.println("查询完毕");**

**while (rs.next()) {**

**//获取员工的名字**

**String username = rs.getString("username");**

**//获取工资**

**String password = rs.getString("password");**

**//获取部门号**

**String email = rs.getString("email");**

**//获取工作所在地**

**int accont = rs.getInt("accont");**

**System.*out*.println("姓名：" + username + ",密码：" + password + ",邮箱：" + email + ",余额：" + accont);**

**}**

**}**

**}**

10）输入id删除用户

**/\***

**\* 输入一个id，删除用户的信息**

**\*/**

**public class JDBCDemo10 {**

**public static void main(String[] args) throws ClassNotFoundException, SQLException {**

**//第一步链接数据库**

**//1.加载驱动**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**System.*out*.println("驱动加载完毕");**

**//2.建立连接**

**Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "zhangyi", "12346");**

**System.*out*.println("数据库已连接");**

**//3.执行sql语句**

**Statement state = conn.createStatement();**

**System.*out*.println("请输入需要删除用户的id");**

**String sql = "delete userinfo where id = "+(new Scanner(System.*in*)).nextInt();**

**if ((state.executeUpdate(sql)) > 0) {**

**System.*out*.println("数据已删除");**

**} else {**

**System.*out*.println("数据删除失败");**

**}**

**conn.close();**

**}**

**}**

1. 转账操作

/\*

\*要求用户输入一个转账金额的账户

\*然后再输入一个转出金额的账户

\*然后输入要转出的金额

\*完成转账工作

\*/

**public class JDBCDemo11 {**

**public static void main(String[] args) {**

**Connection conn = null;**

**try {**

**Scanner scan = new Scanner(System.*in*);**

**System.*out*.println("请输入转出账号的用户名：");**

**String outName = scan.nextLine();**

**System.*out*.println("请输入转入账号的用户名：");**

**String inName = scan.nextLine();**

**System.*out*.println("请输入转出金额：");**

**int money = scan.nextInt();**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**conn = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "zhangyi", "12346");**

**/\***

**\* 事务操作是在connection中的**

**\* 并且默认是自动提交的**

**\* 也就是说每当执行一条DML语句，事务就会自动提交**

**\* 所以，要想控制事务，需要将自动提交关闭**

**\* conn.setAutoCommit(false);**

**\*/**

**conn.setAutoCommit(false);//关闭自动提交**

**Statement state = conn.createStatement();**

**//查看转出账号的余额是否足够？**

**String sql = "select accont from userinfo where username='" + outName + "'";**

**System.*out*.println(sql);**

**ResultSet rs = state.executeQuery(sql);**

**if (rs.next()) {**

**int account = rs.getInt("accont");**

**if (account < money) {**

**System.*out*.println("转出余额不足" + money + "元");**

**return;**

**}**

**} else {**

**System.*out*.println("转出账号不存在");**

**return;**

**}**

**/\***

**\* 这个操作应当同时满足才算成功，其中任何一个出问题，转账环节就失败需要回滚。**

**\*/**

**//1.转出账号余额减去转出的金额**

**sql = "update userinfo " + "set accont=account-" + money + " " + "where username='" + outName + "' ";**

**System.*out*.println(sql);**

**int n = state.executeUpdate(sql);**

**if (n > 0) {**

**System.*out*.println("转出金额成功！");**

**} else {**

**System.*out*.println("转出金额失败！");**

**return;**

**}**

**//2.转入账号余额加上转出的金额**

**sql = "update userinfo " + "set account=accont+" + money + " " + "where username='" + inName + "' ";**

**System.*out*.println(sql);**

**int n1 = state.executeUpdate(sql);**

**if (n1 > 0) {**

**System.*out*.println("转入金额成功！");**

**conn.commit();**

**} else {**

**System.*out*.println("转入金额失败");**

**conn.rollback();**

**}**

**state.close();**

**} catch (Exception e) {**

**e.printStackTrace();**

**} finally {**

**if (conn != null) {**

**try {**

**conn.rollback();**

**conn.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

1. **知识点总结**
2. java链接数据库的步骤：

·第一步：加载驱动 固定格式

Class.forName("oracle.jdbc.driver.OracleDriver");

/\*  
\* ClassNotFoundException:异常出现原因  
\* 1.Class.forName 中字符拼写不正确  
\* 2.jar 包没导入环境  
\*/

·第二步：与数据库建立链接 固定格式  
/\*  
\* 通过 DriverManager 类.静态方法 getConnection 建立连接  
\* 三个参数：  
\* 参数 1:数据库的地址及端口（ 不同数据库参数值不同）  
\* 参数 2： 数据库的用户名  
\* 参数 3： 数据库密码  
\*/

Connection conn =  
DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:  
xe", "zhangyi", "12346");

·第三步：执行 sql 语句

//通过连接对象的方法获得Statement（语句对象接口）的对象

Statement state = conn.createStatement();

//定义sql操作语句

String sql = "select ename,sal,job,deptno from emp";

//通过Statement的对象的方法将sql语句传递给数据库执行操作，并得到反//馈

ResultSet rs = state.executeQuery(sql);

a.若sql语句为DQL操作（数据查询语言 select），则使用Statement类中的ResultSet executeQuery(sql) 方法，该方法返回一个ResultSet类型的结果集，该类中有多种getXxx（）方法可以获得结果集中各个数据，通过循环就可以将结果集中的数据一一提取出来，循环判断条件ResultSet类中的boolean next（ ） 方法。

b.若sql语句执行的是DML操作（数据操纵语言 insert，update，delete），则使用语句对象接口的对象的int executeUpdate(String sql) 方法，该方法返回值是int类型，可以判断返回值是否大于0来确定插入/删除/更新数据是否成功，在操作是否成功的基础上执行后续代码（if语句来选择判断）

c.若sql语句执行的是DDL（数据定义语言create，drop等）操作，则使用语句对象接口的对象的boolean execute(String sql) 默认为 false 方法，该方法返回值为boolean类型，默认返回false即操作不成功。可依据返回值来判断操作是否成功执行以及后续代码操作

第四步：关闭连接

若执行DQL操作，获得结果集对象，先关闭结果集流：结果集对象.close();

连接对象.close();//再关闭总流

2）事物操纵语言控制

事物操纵语言在jdbc中是在链接数据库过程中执行的，并且默认是自动提交的。即建立链接后，执行一条DML语言，事物操纵语言就会自动提交一次。所以想控制事物操纵语言，需要将其自动提交关闭。使用的是connection 类中的Void setAutoCommit(boolean b)方法，将参数设置为false关闭其自动提交功能。

在建立连接后关闭其自动提交功能，当执行完所有sql语句后，判断所有sql操作是否成功，若成功进行手动提交，使数据库发生改变，若sql语句没按要求执行完毕，进行回滚，撤销已经进行的sql语言。

如转账案例中，建立连接后，需要进行两个DML操作，一个账户余额减少，一个账户余额增加。若不关闭自动提交功能，则在第一个DML操作后，一个账户余额减少，并且自动提交。这样数据库中这个账户的金额已经发生改变。若输入另一个账户有误，第二个转入账户进行余额增加的DML操作失败，余额不会增加，没提交，数据库中余额不会发生改变，这样就造成数据库中一个账户余额减少，另一个账户余额没增加，出现问题。所以当链接建立后，关闭自动提交功能，当所以sql语句正确执行再手动提交。有一个sql语句为正确执行就回滚并且结束方法，或者执行其他代码操作。

1. **jdbc方法封装**
2. **数据库连接，关闭的封装**
3. 封装类

**/\***

**\* 负责管理数据库的链接**

**\*/**

**public class DBUtil01 {**

**/\***

**\* 获取数据库的链接**

**\* \*/**

**public static Connection getConnection() throws ClassNotFoundException, SQLException{**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**return DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe",**

**"zhangyi", "12346");**

**}**

**//关闭流**

**public static void claseConnetion(Connection conn){**

**try {**

**if(conn != null){**

**conn.close();}**

**} catch (SQLException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**}**

1. 测试类

/\*

\* 向userinfo表中插入100条数据

\*/

**public** **class** JDBCDemo01 {

**public** **static** **void** main(String[] args) {

Connection conn = **null**;

Statement state = **null**;

**try** {

conn = DBUtil01.*getConnection*();

state = conn.createStatement();

String sql = "";

**for** (**int** i = 100; i < 200; i++) {

sql = "INSERT INTO userinfo (ID,username,PASSWORD,email,accont) "

+ "values(sep\_userinfo\_id.nextval,'zhangasan " + i + " ','123456','qq.com',5000)";

System.***out***.println(sql);

//state.executeUpdate(sql);

/\*

\* 批量插入：

\* 当批量插入时，最好不要单句执行，因为会提高网络访问量，增加内存资源消耗

\* 我们需要通过减少执行次数来提高效率

\* \*/

//此语句功能是将sql语句添加到批中，等待添加完毕后一起发送

**state.addBatch(sql);**

**}**

/\*

\* 执行批量处理：

\* 该方法返回的是一个int[]数组

\* 数组中每一个数字表示批量处理中的一条语句影响了数据库中的多少条数据

\* 批量处理通常用来执行DML操作

\* \*/

**int[] num = state.executeBatch();**

**for (int i : num) {**

**System.*out*.println(i);**

**}**

**} catch (ClassNotFoundException | SQLException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**} finally {**

**try {**

**if (state != null) {**

**state.close();**

**}**

**} catch (SQLException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**DBUtil01.*claseConnetion*(conn);**

**}**

**}**

1. **从文件导入连接信息**
2. 创建资源文件

driverName=oracle.jdbc.driver.OracleDriver

url=jdbc:oracle:thin:@localhost:1521:xe

username=zhangyi

password=12346

1. 封装类

**public class DBUtil02 {**

**//属性**

**private static String *driverName*;**

**private static String *url*;**

**private static String *username*;**

**private static String *password*;**

**static {**

**/\***

**\* 静态块由于是类的信息，在第一次加载的时候执行，并且只加载一次**

**\*/**

**try {**

**Properties prop = new Properties();//创建对象**

**prop.load(new FileInputStream("config.properties"));//加载文件**

**//给属性赋初值**

***driverName* = prop.getProperty("driverName");**

***url* = prop.getProperty("url");**

***username* = prop.getProperty("username");**

***password* = prop.getProperty("password");**

**} catch (Exception e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**/\***

**\* 获取数据连接**

**\*/**

**public static Connection getConnection() throws Exception {**

**Class.*forName*(*driverName*);**

**return DriverManager.*getConnection*(*url*, *username*, *password*);**

**}**

**/\***

**\* 关闭数据连接**

**\*/**

**public static void closeConnection(Connection conn) {**

**try {**

**if (conn != null) {**

**conn.close();**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**//测试**

**public static void main(String[] args) throws Exception {**

**Connection conn = DBUtil02.*getConnection*();**

**DBUtil02.*closeConnection*(conn);**

**}**

**}**

1. **使用连接池使多个用户同时访问数据库**

1）创建资源文件

**driverName=oracle.jdbc.driver.OracleDriver**

**url=jdbc:oracle:thin:@localhost:1521:xe**

**username=zhangyi**

**password=12346**

**maxActive=500**

**maxWait=1000**

2）封装类

**package com.chinasofti.etc.day02;**

**import java.io.FileInputStream;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**import java.util.Properties;**

**import org.apache.commons.dbcp.BasicDataSource;**

**public class DBUtil {**

**//属性**

**private static BasicDataSource *cp*;//连接池**

**private static String *driverName*;**

**private static String *url*;**

**private static String *username*;**

**private static String *password*;**

**private static int *maxActive*;//最大连接数**

**private static int *maxWait*;//最大等待时间**

**static {**

**/\***

**\* 静态块由于是类的信息，在第一次加载的时候执行，并且只加载一次**

**\*/**

**try {**

**Properties prop = new Properties();//创建对象**

**prop.load(new FileInputStream("config.properties"));//加载文件**

**//给属性赋初值**

***driverName* = prop.getProperty("driverName");**

***url* = prop.getProperty("url");**

***username* = prop.getProperty("username");**

***password* = prop.getProperty("password");**

***maxActive* = Integer.*parseInt*(prop.getProperty("maxActive"));**

***maxWait* = Integer.*parseInt*(prop.getProperty("maxWait"));**

**//初始化连接池对象**

***cp* = new BasicDataSource();**

**//将数据加入到连接池中**

***cp*.setDriverClassName(*driverName*);**

***cp*.setUrl(*url*);**

***cp*.setUsername(*username*);**

***cp*.setPassword(*password*);**

***cp*.setMaxActive(*maxActive*);**

***cp*.setMaxWait(*maxWait*);**

**} catch (Exception e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**/\***

**\* 获取数据连接**

**\*/**

**public static Connection getConnection() throws Exception {**

**Class.*forName*(*driverName*);**

**return DriverManager.*getConnection*(*url*, *username*, *password*);**

**}**

**/\***

**\* 关闭数据连接**

**\*/**

**public static void closeConnection(Connection conn) {**

**try {**

**if (conn != null) {**

**conn.close();**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**//测试**

**public static void main(String[] args) throws Exception {**

**Connection conn = DBUtil02.*getConnection*();**

**DBUtil02.*closeConnection*(conn);**

**}**

**}**

3）预处理操作

·Statement存在不足：

\* 1.若sql语句中有变化的内容，我们需要拼接sql语句，这样

\* 会导致可读性差，编码的复杂度高

\* 2.拼接的时候可能涉及到敏感信息，会导致安全漏洞

\* 3.当多次执行语义相同（插入100条数据），但是内容不同（100条数据的值完全不一样）

\* Statement执行语句效率特别差

· 为了解决以上三个不足，可以使用PreparedStatement类：预执行

\* PreparedStatement是Statement的子类

1. 使用预处理向userinfo表中插入100条信息

**package com.chinasofti.etc.day02;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

/\*

\* Statement存在不足：

\* 1.若sql语句中有变化的内容，我们需要拼接sql语句，这样

\* 会导致可读性差，编码的复杂度高

\* 2.拼接的时候可能涉及到敏感信息，会导致安全漏洞

\* 3.当多次执行语义相同（插入100条数据），但是内容不同（100条数据的值完全不一样）

\* Statement执行语句效率特别差

\*

\* 为了解决以上三个不足，可以使用PreparedStatement类：预执行

\* PreparedStatement是Statement的子类

\*

\*/

**public class JDBCDemo02 {**

**/\***

**\* 向userinfo表中插入100条信息**

**\***

**\* \*/**

**public static void main(String[] args) {**

**// TODO Auto-generated method stub**

**Connection conn = null;**

**try {**

**conn = DBUtil.*getConnection*();**

**String sql = "INSERT INTO userinfo (ID,username,PASSWORD,email,accont) "**

**+ "values(sep\_userinfo\_id.nextval,?,?,?,?)";**

**/\***

**\* Statement：一般用于静态sql**

**\* PreparedStatement：常用于动态sql**

**\*/**

**PreparedStatement ps = conn.prepareStatement(sql);**

**for (int i = 0; i < 100; i++) {**

**ps.setString(1, "张三丰" + i);**

**ps.setString(2, "123456");**

**ps.setString(3, "张三丰" + i + "@163.com");**

**ps.setInt(4, 8000);**

**}**

**ps.executeBatch();**

**System.*out*.println("插入完毕");**

**} catch (Exception e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**} finally {**

**DBUtil.*closeConnection*(conn);**

**}**

**}**

**}**

（2）通过ps查询用户余额，通过用户名和密码

**package com.chinasofti.etc.day02;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.util.Scanner;**

**/\***

**\* 通过ps查询用户余额，通过用户名和密码**

**\***

**\* \*/**

**public class JDBCDemo03 {**

**public static void main(String[] args) throws Exception {**

**System.*out*.println("欢迎进行账户余额查询");**

**// 1.连接数据库**

**Connection conn = DBUtil.*getConnection*();**

**//2.编写sql语句**

**String sql = "select accont from userinfo where username = ? and PASSWORD = ?";**

**PreparedStatement ps = conn.prepareStatement(sql);**

**System.*out*.println("请输入用户名");**

**String username = new Scanner(System.*in*).next();**

**System.*out*.println("请输入密码");**

**String password = new Scanner(System.*in*).next();**

**ps.setString(1, username);**

**ps.setString(2, password);**

**ResultSet rs = ps.executeQuery();**

**if (rs.next()) {**

**double account = rs.getDouble("accont");**

**System.*out*.println("登陆成功！你的余额为" + account);**

**} else {**

**System.*out*.println("用户名或者密码错误");**

**}**

**ps.close();**

**rs.close();**

**DBUtil.*closeConnection*(conn);**

**}**

**}**

（3）用预处理进行分页操作

**package com.chinasofti.etc.day02;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.util.Scanner;**

**/\***

**\* 用预处理进行分页操作**

**\* \*/**

**public class JDBCDemo04 {**

**public static void main(String[] args) throws Exception {**

**System.*out*.println("请输入查看的页数");**

**int i = (new Scanner(System.*in*)).nextInt();**

**System.*out*.println("请输入每页显示的条数");**

**int j = (new Scanner(System.*in*)).nextInt();**

**/\*--分页查询范围的算法**

**--pagesSize:每页的条目数**

**--page:第几页**

**--MIN:(page-1)\*pageSize+1**

**--max:page\*pageSize\*/**

**int min = (i-1)\*j+1;**

**int max = i\*j;**

**// 1.连接数据库**

**Connection conn = DBUtil.*getConnection*();**

**//2.执行sql语句**

**String sql = "SELECT \*FROM (SELECT ROWNUM rn ,t.\* FROM"**

**+ "(SELECT ename,sal,job,deptno FROM emp ORDER BY sal DESC ) t) "**

**+ "WHERE rn BETWEEN ? AND ?";**

**PreparedStatement ps = conn.prepareStatement(sql);**

**System.*out*.println(sql);**

**ps.setInt(1, min);**

**ps.setInt(2, max);**

**ResultSet rs = ps.executeQuery();**

**System.*out*.println("查询完毕");**

**System.*out*.println("您查看的是第"+ i +"页，每页显示"+ j +"条数据");**

**while (rs.next()) {**

**//获取员工的名字**

**String ename = rs.getString("ename");**

**//获取工资**

**int sal = rs.getInt("sal");**

**//获取工种**

**String job = rs.getString("job");**

**//获取部门号**

**int deptno = rs.getInt("deptno");**

**//获取工作所在地**

**System.*out*.println("姓名：" + ename + ",工资：" + sal + ",工作：" + job + ",部门号：" + deptno);**

**}**

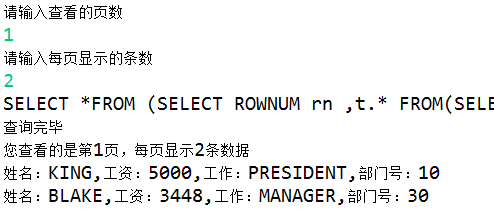
**ps.close();**

**rs.close();**

**DBUtil.*closeConnection*(conn);**

**}**

**}**



**【存储过程的概念】**

存储过程（**Stored Procedure**）是一组**为了完成特定功能的SQL语句集**。经编译后存储在**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/yang3wei/article/details/_blank)**中。

存储过程是数据库中的一个重要对象，**用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它**。

存储过程是由 **流控制** 和 **SQL语句**书写的过程，**这个过程经编译和优化后存储在数据库服务器中**。

存储过程 **可由应用程序通过一个调用来执行，而且允许用户声明变量。**

同时，存储过程**可以接收和输出参数、返回执行存储过程的状态值，也可以嵌套调用**。

**【存储过程的功能】**

这类语言主要提供以下功能，让用户可以设计出符合引用需求的程序：

① 变量说明

② ANSI兼容的SQL命令(如Select,Update….)

③ 一般流程控制命令(if…else…、while….)

④ 内部函数

**【存储过程的**优点**】**

① 存储过程的能力**大大增强了SQL语言的功能和灵活性**。

存储过程可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的 运算。

② 可**保证数据的安全性和完整性**。

理由1：通过存储过程可以使没有权限的用户在控制之下间接地存取数据库，从而保证数据的安全。

理由2：通过存储过程可以使相关的动作在一起发生，从而可以维护数据库的完整性。

③ 在运行存储过程前，数据库已对其进行了语法和句法分析，并给出了优化执行方案。

这种已经编译好的过程**可极大地改善SQL语句的性能**。

**由于执行SQL语句的大部分工作已经完成，所以存储过程能以极快的速度执行**。

④ 可以降低网络的通信量。

理由：**客户端调用存储过程只需要传存储过程名和相关参数即可**，与传输SQL语句相比自然数据量少了很多。

⑤ 使体现企业规则的运算程序放入数据库服务器中，以便：

1). 集中控制。

2). 当企业规则发生变化时在服务器中改变存储过程即可，无须修改任何应用程序。

**企业规则的特点是要经常变化，如果把体现企业规则的运算程序放入应用程序中，则当企业规则发生变化时，就需要修改应用程序工作量非常之大（修改、发行和安装应用程序）。**

如果把体现企业规则的运算放入存储过程中，则当企业规则发生变化时，只要修改存储过程就可以了，应用程序无须任何变化。

**【存储过程优点概括】**

① **存储过程只在创造时进行编译，以后每次执行存储过程都不需再重新编译，而一般SQL语句每执行一次就编译一次,所以使用存储过程可提高数据库执行速度。**

② 当对数据库进行复杂操作时(如对多个表进行Update,Insert,Query,Delete时)，**可将此复杂操作用存储过程封装起来与数据库提供的事务处理结合一起使用**。

③ 存储过程**可以重复使用**,可减少数据库开发人员的工作量。

④ 安全性高,**可设定只有某些用户才具有对指定存储过程的使用权**

**【存储过程缺点】**

① 调试麻烦，但是用 PL/SQL Developer 调试很方便！弥补这个缺点。

② 移植问题，数据库端代码当然是与数据库相关的。但是如果是做工程型项目，基本不存在移植问题。

③ 重新编译问题，因为后端代码是运行前编译的，如果带有引用关系的对象发生改变时，受影响的存储过程、包将需要重新编译（不过也可以设置成运行时刻自动编译）。

④ 如果在一个程序系统中大量的使用存储过程，到程序交付使用的时候随着用户需求的增加会导致**[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/yang3wei/article/details/_blank)**的变化，接着就是系统的相关问题了，最后如果用户想维护该系统可以说是很难很难、而且代价是空前的，维护起来更麻烦。

**【存储过程的**种类**】**

① 系统存储过程：以sp\_开头,用来进行系统的各项设定.取得信息.相关管理工作。

② 本地存储过程：用户创建的存储过程是由用户创建并完成某一特定功能的存储过程，事实上一般所说的存储过程就是指本地存储过程。

③ 临时存储过程：分为两种存储过程：

一是本地临时存储过程，以井字号(#)作为其名称的第一个字符，则该存储过程将成为一个存放在tempdb数据库中的本地临时存储过程，且只有创建它的用户才能执行它;

二是全局临时存储过程，以两个井字号(##)号开始，则该存储过程将成为一个存储在tempdb数据库中的全局临时存储过程，全局临时存储过程一旦创建，以后连接到服务器的任意用户都可以执行它，而且不需要特定的权限。

④ 远程存储过程：在SQL Server2005中，远程存储过程(Remote Stored Procedures)是位于远程服务器上的存储过程，通常可以使用分布式查询和EXECUTE命令执行一个远程存储过程。

⑤ 扩展存储过程：扩展存储过程(Extended Stored Procedures)是用户可以使用外部程序语言编写的存储过程，而且扩展存储过程的名称通常以xp\_开头。

**【[Oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "http://blog.csdn.net/yang3wei/article/details/_blank)** **数据库的一个分页存储过程示例】**

**create or replace procedure fenye(**

tableName in varchar2,--表名  
pageSize in number,--1页显示的记录数  
pageNow in number,--当前为第几页  
myrows out number, --总记录数(一定要小心代码中不能有**[oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "http://blog.csdn.net/yang3wei/article/details/_blank)**的保留字，保留字是以青色显示的)  
myPageCount out number,--总页数  
p\_cursor out testpackage.test\_cursor--返回的记录集  
**) is**--定义部分  
--定义一个sql语句字符串  
v\_sql varchar2(1000);  
--定义两个整数  
v\_begin number**:=**(pageNow-1)\*pageSize+1;  
v\_end number**:=**pageNow\*pageSize;  
**begin**--执行部分(排序的话在最内层加条件)  
v\_sql**:=**'select \* from (select t1.\*,rownum rn from (select \* from '|| tableName  
   ||' order by sal) t1 where rownum<='|| v\_end ||') where rn>='|| v\_begin;  
--把游标和sql关联  
**open** p\_cursor **for** v\_sql;  
--要计算myrows和myPageCount  
--组织一个sql  
v\_sql**:=**'select count(\*) from '|| tableName;  
--执行sql，并把返回的值，赋给myrows  
**execute immediate** v\_sql **into** myrows**;**--计算myPageCount  
**if** mod(myrows,pageSize)=0 **then**  myPageCount**:=**myrows/pageSize;  
**else**  myPageCount**:=**myrows/pageSize+1;  
**end if;**--关闭游标(关了的话运行**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/yang3wei/article/details/_blank)**程序竟然报游标已经关闭的错误！)  
--close p\_cursor;  
**end;**

**【java调用存储过程的示例】**

****[java]**** [view plain](http://blog.csdn.net/yang3wei/article/details/6281255" \o "view plain" \t "http://blog.csdn.net/yang3wei/article/details/_blank) [copy](http://blog.csdn.net/yang3wei/article/details/6281255" \o "copy" \t "http://blog.csdn.net/yang3wei/article/details/_blank)

1. **package org.bruce.test.oracle.procedure;**
3. **import java.sql.CallableStatement;**
4. **import java.sql.Connection;**
5. **import java.sql.DriverManager;**
6. **import java.sql.ResultSet;**
7. **import java.sql.SQLException;**
9. **public class TestOracleProc6 {**
10. **public static void main(String[] args) {**
11. **Connection ct = null;**
12. **CallableStatement cs = null;**
13. **int pageNow =**1**;**
14. **try {**
15. **Class.forName(**"oracle.jdbc.driver.OracleDriver"**);**// 1.加载驱动
16. **ct = DriverManager.getConnection(**"jdbc:oracle:thin:@127.0.0.1:1521:MYORA1"**,**"SCOTT"**,**"m123"**);**
17. **cs = ct.prepareCall(**"{call my\_fenye(?,?,?,?,?,?)}"**);**
19. **cs.setString(**1**,**"emp"**);**// 取emp表
20. **cs.setInt(**2**,**5**);**// 每页取 5 条记录
21. **cs.setInt(**3**, pageNow);**// 当前页为第 pageNow 页
22. **cs.registerOutParameter(**4**, oracle.jdbc.OracleTypes.INTEGER);**// 注册总记录数
23. **cs.registerOutParameter(**5**, oracle.jdbc.OracleTypes.INTEGER);**// 注册总页数
24. **cs.registerOutParameter(**6**, oracle.jdbc.OracleTypes.CURSOR);**// 注册返回的结果集
26. **cs.execute();**
28. **int rowNum = cs.getInt(**4**);**// 取出总记录数(getInt(4)中4，是由该参数的位置决定的
29. **int pageCount = cs.getInt(**5**);**
30. **ResultSet rs = (ResultSet)cs.getObject(**6**);**
32. **System.out.println(**"总共有多少行记录："**+ rowNum);**
33. **System.out.println(**"总共有多少页："**+ pageCount);**
34. **System.out.println(**"当前为多少页："**+ pageNow);**
35. **while(rs.next()) {**
36. **System.out.println(rs.getInt(**1**)+**" "**+rs.getString(**2**)+**" "**+rs.getFloat(**6**));**
37. **}**
38. **} catch(Exception e) {**
39. **e.printStackTrace();**
40. **} finally {**
41. **try {**// 关闭各个打开的资源...
42. **if(cs != null) {**
43. **cs.close();**
44. **}**
45. **if(ct != null) {**
46. **ct.close();**
47. **}**
48. **} catch (SQLException e) {**
49. **e.printStackTrace();**
50. **}**
51. **}**
52. **}**
53. **}**