目录

[数据库基本概念 2](#_Toc501035002)

[Oracle数据库服务器 2](#_Toc501035003)

[SQL和SQLPLUS 3](#_Toc501035004)

[SQLPLUS常用命令 3](#_Toc501035005)

[DDL 3](#_Toc501035006)

[DROP, TRUNCATE和DELET 3](#_Toc501035007)

[回收站操作 4](#_Toc501035008)

[备份表结构 4](#_Toc501035009)

[DML 4](#_Toc501035010)

[TCL 4](#_Toc501035011)

[事务特性ACID 4](#_Toc501035012)

[事务相关术语 4](#_Toc501035013)

[MySQL事务 4](#_Toc501035014)

[DCL 4](#_Toc501035015)

[数据类型 ? 5](#_Toc501035016)

[字符串函数 5](#_Toc501035017)

[Oracle数值函数 5](#_Toc501035018)

[Oracle日期函数 5](#_Toc501035019)

[空值函数 6](#_Toc501035020)

[查询条件 6](#_Toc501035021)

[查询\_和插入' 6](#_Toc501035022)

[聚合函数 6](#_Toc501035023)

[查询语句执行顺序 6](#_Toc501035024)

[关联查询 6](#_Toc501035025)

[子查询 7](#_Toc501035026)

[分页查询 7](#_Toc501035027)

[DECODE函数 7](#_Toc501035028)

[排序函数 7](#_Toc501035029)

[集合操作 7](#_Toc501035030)

[高级分组函数 7](#_Toc501035031)

[视图 8](#_Toc501035032)

[Oracle序列 8](#_Toc501035033)

[索引 8](#_Toc501035034)

[ROWID 8](#_Toc501035035)

[约束 8](#_Toc501035036)

[MySQL存储引擎 ? 9](#_Toc501035037)

[PL/SQL 9](#_Toc501035038)

[PL/SQL标识符 9](#_Toc501035039)

[PL/SQL程序单元 9](#_Toc501035040)

[PL/SQL数据类型和变量 9](#_Toc501035041)

[PL/SQL DBMS输出 9](#_Toc501035042)

[PL/SQL 条件控制 9](#_Toc501035043)

[PL/SQL循环语句 10](#_Toc501035044)

[PL/SQL EXIT语句 10](#_Toc501035045)

[PL/SQL存储过程 10](#_Toc501035046)

[PL/SQL函数 10](#_Toc501035047)

[存储过程、函数 10](#_Toc501035048)

[PL/SQL游标 11](#_Toc501035049)

[PL/SQL异常 11](#_Toc501035050)

[JDBC 11](#_Toc501035051)

[JDBC连接数据库步骤 12](#_Toc501035052)

[连接池 12](#_Toc501035053)

[DBUtil 12](#_Toc501035054)

[Statement和PreparedStatement 12](#_Toc501035055)

# 数据库基本概念

|  |  |
| --- | --- |
| 数据库 | 按照数据结构来组织、存储和管理数据的仓库 |
| 关系数据库 | 关系：描述两个元素间的关联或对应关系  使用关联模型把数据组织到二维数据表中  一个关系数据库由多个数据表组成 |
| 表 | 数据表是关系数据库的基本存储结构  表的行(Row)是横排数据，也被称作记录(Record)  表的列(Column)是纵列数据，也被称作字段(Field) |

# Oracle数据库服务器

|  |  |
| --- | --- |
| 实例 | 位于内存里，它由一个共享的内存池和多个进程所组成，共享的内存池可以被所有进程访问 |
| 数据库 | 指硬盘上实际存放数据的文件，必须要与内存里实例合作，才能对外提供数据管理服务 |

# SQL和SQLPLUS

|  |  |
| --- | --- |
| SQL | SQLPLUS |
| 是操作所有关系型数据库的规则，是第四代语言  交互性强，非过程化，无需关注如何实现 | Oracle自带的一款工具所指定的命令 |
| 关键字不能简写 | 关键字可以简写 |
| 大小写不敏感，提倡大写 | 大小写不敏感，提倡大写 |
| 必须以分号结束 | 可以不用分号结束 |
|  | 提供了&进行动态操作 |

# SQLPLUS常用命令

|  |  |
| --- | --- |
| 以超级管理员角色登录 | SQLPLUS / AS sysdba |
| 以普通用户角色登录 | SQLPLUS username/password ? |
| 再连接上某个用户 | CONN username/passowrd |
| 设置显示的列宽 | COLUMN col\_name FORMAT a4 （字符类型和日期类型，这里表示显示4位） |
| COLUMN col\_name FORMAT 9999 （数字类型，这里表示显示4位） |
| 设置一页显示多少条记录 | SET PAGESIZE pagesize |
| 执行最近一次的SQL语句 | / |
|  | set timing on |
| 清屏 | HOST CLS |
| 保存SQL语句到硬盘 | SPOOL filepath; SPOOL OFF; |
| 读取SQL文件并执行SQL语句 | @ filepath; |

# DDL

|  |  |
| --- | --- |
| 创建数据库 | CREATE DATABASE db\_name; Oracle创建数据库??? |
| 列出所有数据库 | SHOW DATABASES db\_name; |
| 开开数据库 | USE db\_name; |
| 删除… | DROP DATABASE/VIEW/INDEX/SEQUENCE/SYNONYM/PROCEDURE/FUNCTION …name; |
| 创建表 | CREATE TABLE table\_name (…) |
| 修改表名 | RENAME old\_name TO new\_name; |
| 修改表 | ALTER TABLE table\_name ADD column\_name datatype |
| ALTER TABLE table\_name DROP COLUMN column\_name |
| ALTER TABLE table\_name MODIFY column\_name datatpye |
| 删除表 | DROP TABLE table\_name |
| TRUNCATE TABLE table\_name |
| 创建或修改视图 | CREATE OR REPLACE VIEW view\_name AS SELECT column\_name(s)  FROM table\_name WHERE condition |
| 创建序列 | CREATE SEQUENCE seq\_name START WITH i INCREMENT BY j [MAXVALUE m] [MINVALUE n] |
| 创建索引 | CREATE INDEX index\_name ON table\_name(column1 …) |
| 重建索引 | ALTER INDEX index\_name REBUILD |
| 创建同义词 | CREATE SYNONYM synonym\_name FOR original\_name |

# DROP, TRUNCATE和DELET

|  |  |  |
| --- | --- | --- |
| DROP | TRUNCATE | DELETE |
| DDL，隐式提交 | | DML，可回滚 |
| 删除数据和表结构 | 只删除数据 | |
|  | MySQL中自增计数制初始化 | MySQL中保留自增计数制 |
| 占用空间全部释放 | 表和索引所占用空间恢复到初始大小 | 不会减少表和索引所占用空间 |
|  |  | 慢 |
|  | 不能在有fk的表上使用，因为不记录删除log，不触发trigger ??? |  |

# 回收站操作

|  |  |
| --- | --- |
| 删除表 | DROP TABLE table\_name; |
| 从回收站将表闪回 | FLASHBACK TABLE table\_name TO BEFORE DROPp; |
| 查看回收站 | SHOW RECYCLEBIN; |
| 清空回收站 | PURGE RECYCLEBIN; |
| 彻底删除表 | DROP TABLE table\_name PURGE; |

# 备份表结构

|  |  |
| --- | --- |
| MySQL | Oracle |
| create table demo\_bak like demo; | create table demo\_bak as select \* from where 1=2; |

# DML

|  |  |
| --- | --- |
| 插入数据 | INSERT INTO table\_name [(column1, …)] VALUES (value1,value2 …) |
| 修改数据 | UPDATE table\_name SET column1=value1,… [WHERE condition] |
| 删除数据 | DELETE [FROM] table\_name [WHERE condition] |

# TCL

|  |  |
| --- | --- |
| 提交 | COMMIT |
| 回滚 | ROLLBACK |
| 保存点 | SAVEPOINT savepoint\_name |

# 事务特性ACID

|  |  |
| --- | --- |
| 原子性 Atomicity | 事务必须是原子工作单元；对于其数据修改，要么全部执行，要么全都不执行 |
| 一致性 Consistency | 事务在完成时，必须使所有的数据都保持一致状态 |
| 隔离性 Isolation | 由并发事务所作的修改必须与任何其它并发事务所作的修改隔离 |
| Oracle支持read committed和serializable (set transaction isolation level …) |
| 持久性 Durabiltiy | 事务完成后之后，它对于系统的影响是永久性的 |
| conn.setAutoCommit(fasle) conn.commit() conn.rollback() | |

# 事务相关术语

|  |  |
| --- | --- |
| 脏读 | 一个事务读取到另一事务未提交的更新数据 |
| 不可重复读 | 同一事务中，多次读取同一数据，后续读取可以读到另一事务已提交的更新数据 |
| 幻读 | 一个事务读到另一个事务已提交的insert数据 ? |

# MySQL事务

|  |  |
| --- | --- |
| 设置事务开始点 | START TRANSACTION; |
| 设置保存点 | SAVEPOINT point\_name; |
| 回滚 | ROLLBACK TO SAVEPOINT point\_name; |

# DCL

|  |  |  |
| --- | --- | --- |
| 授予权限 | GRANT | GRANT create view TO ljh; |
| 收回权限 | REVOKE | REVOKE select any table FROM ljh; |
| 创建用户 | CREATE USER | |

|  |  |
| --- | --- |
| 给普通用户解锁 | ALTER USER username ACCOUNT UNLOCK; |
| 给普通用户修改密码 | ALTER USER username IDENTIFIED BY password; |
| 修改自己的密码 | PASSWORD; |
| 查看当前用户是谁 | SHOW USER; |
| 查询自己的权限 | select \* from user\_sys\_privs; |

# 数据类型 ?

|  |  |
| --- | --- |
| Oracle | MySQL |
| NUMBER(precision, scale) | int/decimal |
| VARCHAR2(N) 变长字符，最大4000字节 | varchar(N) |
| DATE 长度7字节，默认格式：DD-MON-RR | datetime |
| EXCEPTION | SQLEXCEPTION |
| sequence | auto\_increment |

# 字符串函数

|  |  |  |
| --- | --- | --- |
| CONTACT  "||" | select CONCAT(CONCAT('hello ', 'world'), '!') from dual;  select 'hello ' || 'world' || '!' from dual; | hello world!  hello world! |
| LENGTH  LENGTHB | select LENGTH('你好') from dual;  select LENGTHB('你好') from dual; | 2  6 |
| INITCAP | select INITCAP('hello world') from dual; | Hello World! |
| TRIM  LTRIM | select TRIM('e' FROM 'eeEEEee') from dual;  select LTRIM('eeEEEee', 'e') from dual; | EEE  EEEee |
| REPLACE | select REPLACE('hello', 'h', 'H') from dual; | Hello |
| LPAD | select LPAD('hello', 8, '\_') from dual; | \_\_\_hello |
| SUBSTR | select SUBSTR('hello world', 1, 5) from dual; | hello |
| INSTR | select INSTR('hello world', 'l', 4, 2) from dual; | 10 |

# Oracle数值函数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ROUND | select ROUND(45.678, 2), ROUND(45.678, -1) from dual;  select ROUND(sysdate , 'month') from dual; | |  |  | | --- | --- | | 45.68 | 50 |   01-12月-17 |
| TRUNC | select TRUNC(45.678, 2), ROUND(45.678, -1) from dual;  select TRUNC(sysdate , 'year') from dual; | |  |  | | --- | --- | | 45.67 | 50 |   01-1月 -17 |
| MOD | select MOD(3333, 1000) from dual; | 333 |
| CEIL  FLOOR | select CEIL(45.678), FLOOR(45.678) from dual; | |  |  | | --- | --- | | 46 | 45 | |

# Oracle日期函数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SYSTIMSTAMP | | select SYSTIMESTAMP from dual; | 10-12月-17 08.52.39.416000000 下午 +08:00 | |
| TO\_DATE | select TO\_DATE('2017-11月-11', 'YYYY-MON-DD') from dual; | | | 11-11月-17 |
| TO\_CHAR | select TO\_CHAR(SYSDATE, 'DY HH24:MI:SS') from dual; | | | 星期日 21:05:24 |
| LAST\_DAY | select LAST\_DAY(to\_date('2017-11-11', 'YY-MM-DD')) from dual; | | | 30-11月-17 |
| ADD\_MONTHS | select ADD\_MONTHS('11-11月-17', 2) from dual; | | | 11-1月 -18 |
| MONTHS\_ BETWEEN | select MONTHS\_BETWEEN ('11-11月-17', '11-1月 -18') from dual; | | | -2 |
| NEXT\_DAY | select NEXT\_DAY('11-11月-17', 1) from dual; | | | 12-11月-17 |
| LAST\_DAY | select LAST\_DAY('11-11月-17') from dual; | | | 30-11月-17 |
| GREATEST  LEAST | select GREATEST('12-11月-17', '11-1月 -18') from dual; | | | 12-11月-17 |
| EXTRACT | select EXTRACT(YEAR FROM SYSDATE) from dual; | | | 2017 |

# 空值函数

|  |  |  |
| --- | --- | --- |
| NVL | select NVL(null, 0) from dual; | 0 |
| NVL2 | select NVL2(null, 0, 1) from dual; | 1 |

# 查询条件

|  |  |  |
| --- | --- | --- |
| LIKE | select ename from emp where ename LIKE '\_A%'; | JACK |
| IN和NOT IN | select ename, job from emp where deptno NOT IN (10, 20, 30); | |
| BETWEEN … AND … | select ename, sal from emp where sal BETWEEN 1500 AND 3000; | |
| IS NOT NULL | select ename, sal, comm from emp where comm IS NOT NULL; | |
| ANY和ALL | select ename, job, sal from emp where sal > ANY(3000, 4000); | |
| DISTINCT | select DISTINCT deptno, job from emp; | |

# 查询\_和插入'

|  |  |
| --- | --- |
| 查询名字中有'\_'的员工 | select \* from test where name like '%\\_%' ESCAPE '\'; （可以用其它字符代替\） |
| 插入一个姓名叫'的员工 | insert into test(id, name, sal) values('11', '''', 6000); |

# 聚合函数

|  |
| --- |
| 需要统计的数据并不能在表中直观列出，需要根据现有的数据计算得到结果，如MAX, MIN, SUM, AVG, COUNT  因为是多行数据参与运算返回一行结果，也称作分组函数、多行函数、集合函数  注：聚合函数忽略NULL值，可以这样处理AVG(NVL(comm, 0)) |

# 查询语句执行顺序

|  |  |  |  |
| --- | --- | --- | --- |
| 1. from | ← | 从后往前，从右到左 | 数据量较少的表尽量放在后面 |
| 1. where | ← | 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 |
| 1. group by | ← | 从左往右 | 最好在group by前使用where将不需要的记录过滤掉 |
| 1. having |  | 尽量避免使用（having会在检索出所有记录之后才对结果集进行过滤，需要排序等操作） | |
| 1. select |  | 少用\*号，尽量取字段名称 （Oracle在解析的过程中，通过查询数据字典将\*号依次转换成所有的列名，消耗时间） | |
| 1. order by | → | 从左到右 | 消耗资源（可以order by 列号） |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 1 | 2 | 3 | 4 | 6 |
| select | from | where | group by | having | order by |

# 关联查询

|  |  |
| --- | --- |
| 内连接 | emp e JOIN dept d ON(e.deptno = d.deptno) |
| 左外连接 | emp e LEFT OUTER JOIN dept d ON e.deptno = d.deptno  emp e, dept d where e.deptno = d.deptno(+) |
| 右外连接 | emp e RIGHT OUTER JOIN dept d ON e.deptno = d.deptno  emp e, dept d where e.deptno(+) = d.deptno |
| 全外连接 | emp e FULL OUTER JOIN dept d ON e.deptno = d.deptno |
| 自连接 | emp e JOIN emp e2 ON e.mgr = e2.empnop |

# 子查询

|  |  |
| --- | --- |
| 查询条件未知的事物 | |
| 相关子查询 | 在主查询中，每查询一条记录，需要重新做一次子查询  子查询的条件依赖于主查询的某个属性值时：EXISTS |
| 非相关子查询 | 在主查询中，子查询只需要执行一次，子查询结果不再变化，供主查询使用  单行单列：=, <, > 多行单列：IN, ALL, ANY |

# 分页查询

|  |  |
| --- | --- |
| MySQL | Oracle |
| SELECT \* FROM LIMIT begin, pagesize; | select \* from  (select ROWNUM rn, t.\* from  (select ename,sal from emp order by sal desc)t  )  where rn between 8 and 10; |

# DECODE函数

|  |  |
| --- | --- |
| DECODE | CASE |
| select ename, job, sal,  DECODE(job,  'MANAGER', sal\*1.2,  'ANALYST', sal\*1.1,  sal  )bonus from emp; | select ename, job, sal,  CASE job WHEN 'MANAGER' THEN sal\*1.2  WHEN 'ANALYST' THEN sal\*1.1  ELSE sal END  bonus from emp; |

|  |
| --- |
| 查询工资高于2000，和工资高于3000，分别有多少人？ |
| select sum(case when sal>2000 then 1 else 0 end),  sum(case when sal>3000 then 1 else 0 end) from emp; |

# 排序函数

|  |  |  |
| --- | --- | --- |
| ROW\_NUMBER() | select deptno, name, sal,  dense\_rank() OVER(PARTITION BY deptno order by sal desc)  from demo; | 1 2 3 4 5 |
| RANK() | 1 2 2 4 5 |
| DENSE\_RANK() | 1 2 2 3 4 |

# 集合操作

|  |  |  |  |
| --- | --- | --- | --- |
| UNION | UNION ALL | INTERSECT | MINUS |
|  |  |  |  |

# 高级分组函数

|  |  |  |
| --- | --- | --- |
| GROUP BY | | |
| ROLLUP(a,b,c) | CUBE(a,b,c) | GROUPING SETS((…),(…)…) |
| (a,b,c) + (a,b) + (a) | (a,b,c) + (a,b) + (a,c) + (a) + (b,c) + (b) + (c) + () | (…) + (…) |

# 视图

|  |  |  |  |
| --- | --- | --- | --- |
| 视图也被称作虚表，对应于一条SELECT语句；  视图本身并不包含任何数据，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  视图必须写别名； | | | |
| 简单视图 | | | 复杂视图 |
| 基于单表 | | | 基于多个表 |
| 不包含函数运算、表达式或分组函数 | | | 包含了单行函数、表达式、分组函数 |
|  | | | 必须定义列名 |
| DML操作 | 1. 简单视图能够进行DML操作（简单视图没有包含基表中非空列时，不能执行INSERT操作） 2. DELETE操作时，只能删除现有视图里能查到的记录 3. 视图定义中包含了函数、表达式、分组、DISTINCT或ROWNUM，不允许执行DML操作 | | |
| WITH CHECK OPTION | | 通过视图所做的修改，必须在视图的可见范围内 | |
| READ ONLY | | 只读 | |

# Oracle序列

|  |  |
| --- | --- |
| 一种用来生成唯一数字值的数据库对象，可以为一个表或多个表提供主键值 | |
| seq.NEXTVAL | 获取序列的下个值 |
| seq.CURRVAL | 获取序列的当前值，序列创建后必须先执行一次NEXTVAL，才能使用CURRVAL |

# 索引

|  |  |
| --- | --- |
| 一种树形结构对象，在查找时应用二分查找算法，提高查询效率 ?  一旦被建立就被Oracle系统自动维护，查询语句中不用指定使用哪个索引  在定义primary key或unique约束后，系统自动在相应的列上创建索引  创建多列索引时，如果查询的位置只出现右边的字段，不使用索引（如emp(ename, job)，只出现job，不使用索引） | |
| 1. 通过指针加速Oracle服务器的查询速度 2. 通过rowid快速定位数据的方法，减少磁盘I/O | |
| 基于函数的索引 | CREATE INDEX … ON table\_name(UPPER(name)) |
| 合理使用 | 1. 经常出现在WHERE子句中的列 2. 经常出现在ORDER BY、DISTINCT后面的字段；顺序要一致 3. 经常作为表的连接条件 4. 不要在经常做DML操作的表建索引 5. 不要在小表上建立索引 6. 索引不是越多越好 7. 删除很少被使用的、不合理的索引 |

# ROWID

|  |
| --- |
| 1. 18位大小写加数字，唯一表示该条记录在DBF文件中的位置 2. 可以参与=和like比较，用'单引号将rowid的值包起来 |

# 约束

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NOT NULL | | UNIQUE | PRIMARY KEY | FOREIGN KEY | CHECK |
| NN | | UK | PK | FK | *CK* |
| 允许是NULL值 | 一个表一个 |
| id NUMBER(6) CONSTRAINT emp\_id\_uk UNIQUE | | | |  | |
| id NUMBER(6), CONSTRAINT emp\_id\_uk UNIQUE(id) | | | |
| 主键选取 | 1. 主键应是对系统无意义的数据 2. 永远不要更新主键，让主键除了唯一标识一行之外，再无其他的用途 3. 主键不应包含动态变化的数据，如时间戳 4. 主键应自动生成，不要人为干预，以免使它带有除了唯一标识一行以外的意义 5. 主键尽量建立在单列上 | | | | |

# MySQL存储引擎 ?

|  |  |
| --- | --- |
| InnoDB | MyISAM |
| 支持事务处理 | 不支持 |
| 支持外键 | 不支持 |
| 支持行级锁和表级锁 | 支持行级锁 |
| 适合DML | 适合DQL |

# PL/SQL

|  |  |
| --- | --- |
| Oracle数据库服务器对SQL99的规则做了符合自身的特点的扩展，叫做PL/SQL(过程化SQL)，强调过程 | |
| 1. 因为SQL是第四代命令式语言，无法显式处理过程化的业务，所以得用一个过程化程序设计语言弥补SQL的不足 2. 提高性能 | |
| 匿名PL/SQL块 | [declare] 声明（变量，游标，子程序，和其他元素）  begin 可执行命令(DML/TCL)  [exception] 异常处理  end; |

# PL/SQL标识符

|  |  |  |  |
| --- | --- | --- | --- |
| % | 属性索引 | := | 赋值运算符 |
| . | 组件选择 |  |  |

# PL/SQL程序单元

|  |
| --- |
| 1)PL/SQL块 2)函数 3)包 4)包体 5)过程 6)触发器 7)类型 8)类型体 |

# PL/SQL数据类型和变量

|  |  |  |  |
| --- | --- | --- | --- |
| emp.ename%TYPE | 与表的某字段类型相同 | emp%ROWTYPE | 与表或游标的所有字段类型相同 |
| SELECT name INTO c\_name from test where id = c\_id; | | 使用SELECT INTO语句指定SQL查询结果为PL/SQL变量 | |

# PL/SQL DBMS输出

|  |  |
| --- | --- |
| DBMS\_OUTPUT是一个内置的软件包，能够显示输出显示调试信息，并从PL/ SQL块，子程序，包和触发器发送消息 | |
| dbms\_output.put\_line(…); | 用于输出字符串自动换行 |
| set serveroutput on/off; | 打开/关闭显示PL/SQL程序的执行结果 |

# PL/SQL 条件控制

|  |  |  |
| --- | --- | --- |
| IF condition THEN  S;  END IF; | IF condition THEN  S1;  ELSE  S2;  END IF; | IF condition THEN  S1;  ELSIF condition THEN (可以多个)  S2;  ELSE  S3;  END IF; |
| CASE语句参考SQL99 | | |

# PL/SQL循环语句

|  |  |  |
| --- | --- | --- |
| LOOP  Sequence of statements;  END LOOP; | WHILE condition LOOP  sequence\_of\_statements;  END LOOP; | FOR counter IN first .. last LOOP  sequence\_of\_statements;  END LOOP; |

# PL/SQL EXIT语句

|  |  |
| --- | --- |
| EXIT; | EXIT WHEN cemp%notfound; |

# PL/SQL存储过程

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 一组为了完成特定功能的SQL语句集，经过第一次编译后再次调用不需要再次编译，用户通过存储过程的名字来执行 | | | | | |
| CREATE [OR REPLACE] PROCEDURE procedure\_name  [(parameter\_name [IN | OUT | IN OUT] type [, ...])]  {IS | AS}  BEGIN  < procedure\_body >  END procedure\_name; | | | 参数  模式 | IN | 表示该值将被从外部传递 |
| OUT | 表示该参数将被用于从过程返回一个值到外面 |
| 执行存储过程 | EXECUTE procedure\_name | | | | |
| 调用存储过程 | BEGIN procedure\_name END;/ | | | | |
| 方法传递参数  ??? | 位置表示法 | procedure\_name(a, b, c, d); | | | |
| 命名表示法 | procedure\_name(x=>a, y=>b, z=>c, m=>d); | | | |
| 混合表示法 | procedure\_name(a, b, c, m=>d); （位置表示法要先写） | | | |

# PL/SQL函数

|  |  |
| --- | --- |
| 数据库已定义的方法，它接受参数并返回某种类型的值 | |
| CREATE [OR REPLACE] FUNCTION function\_name  [(parameter\_name [IN | OUT | IN OUT] type [, ...])]  RETURN return\_datatype  {IS | AS}  BEGIN  < function\_body >  END [function\_name]; | |
| 调用函数 | function\_name() |

# 存储过程、函数

|  |  |
| --- | --- |
| 存储过程 procedure | 函数 function |
| 涉及特定表或其它对象的任务 | 不涉及特定用户表 |
| 不需要描述返回类型 | 要描述返回类型 |
| 一般作为独立的部分来执行 | 必须作为表达式的一个部分来调用 |
| 可以返回参数 | 只能返回值或者表对象 |
| 可以通过out/in out返回0个或多个值 | 通过return返回一个值。改值要与声明部分一致，也可以是通过out类型参数带出变量 |

# PL/SQL游标

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 定义 | 一种控制结构，可以帮助我们处理多条记录。不是一种数据类型 | | | | |
| 隐式游标 | Oracle在一个SQL语句的执行，当没有显式游标的语句隐式游标时自动创建。每当发出一个DML语句，隐式游标与此语句关联。 sql%notfound | | | | |
| 显式游标 | 步骤 | 声明游标 | 无参光标 | CURSOR c\_emp IS SELECT ename, sal from emp; | |
| 有参光标 | CURSOR c\_emp(c\_deptno emp.deptno&type) IS SELECT ename, sal from emp where deptno = c\_deptno; | |
| 打开游标 | 无参光标 | OPEN c\_emp; | |
| 有参光标 | OPEN c\_emp(deptno); | |
| 获取游标 | FETCH c\_emp INTO c\_ename, c\_sal; | | |
| 关闭游标 | CLOSE c\_emp; | | |
| 游标属性 | 1)%FOUND | | | | 2)%NOTFOUND |
| 3)ISOPEN 执行完SQL语句之后，隐式会自动关闭 | | | | 4)%ROWCOUNT 返回DML和DQL语句影响的行数 |

# PL/SQL异常

|  |  |  |
| --- | --- | --- |
| 语法 | [EXCEPTION]  WHEN exception\_name THEN  statement; | |
| 常见内置异常 | NO\_DATA\_FOUND | 没有找到数据 |
| TOO\_MANY\_ROWS | SELECT INTO 语句匹配多行 |
| ZERO\_DIVIDE | 被0除 |
| VALUE\_ERROR | 算术或转换错误 |
| TIMEOUT\_ON\_RESOURCE | 在等待资源时发生超时 |
| 用户自定义异常 | [DECLARE]  my-exception EXCEPTION; | |

**PL/SQL触发器**

|  |  |  |
| --- | --- | --- |
| 存储在数据库中的块，这些块一旦被构造后，就可以多次执行，当触发它的事件发生时，调用触发器。 | | |
| CREATE [OR REPLACE ] TRIGGER trigger\_name  {BEFORE | AFTER | INSTEAD OF }  {INSERT [OR] | UPDATE [OR] | DELETE}  [OF col\_name]  ON table\_name  [REFERENCING OLD AS o NEW AS n]  [FOR EACH ROW]  WHEN (condition)  …匿名PL/SQL块 | INSTEAD OF | 用于在视图中创建触发器 |
| OF col\_name | 指定被更新的列名 |
| REFERENCING OLD AS o NEW AS n | ??? |
| FOR EACH ROW | 行级触发器  :new 为一个引用最新的列值  :old 为一个引用以前的列值 |
| WHEN (condition) | 仅适用于行级触发器 |

**Oracle常用数据字典**

|  |  |
| --- | --- |
| dba\_ | 数据库实例的所有对象信息 dba\_users |
| user\_ | 用户的对象信息 user\_segments user\_part\_tables ? |
| all\_ | 用户被授权访问的对象信息 |
| v$\_ | 当前实例的动态视图 |
| gv\_ | 分布式环境下所有实例的动态视图 |

# JDBC

|  |
| --- |
| JDBC是JAVA官方提供的一套接口，用于连接并操作数据库。不同的数据库厂商都提供了一套JDBC的实现类，并以jar包形式发布，用于使我们的程序可以通过这套实现类来操作其数据库，这套jar成为"驱动"。我们无需记住这些数据库提供商提供的JDBC实现类的名字，只要使用JDBC接口看待它们即可，以多态形式调用它们来操作数据库。 |
| [JDBC-ODBC](https://baike.baidu.com/item/JDBC-ODBC" \t "_blank)桥 ??? |

# JDBC连接数据库步骤

|  |  |  |
| --- | --- | --- |
| 将数据库驱动加载到项目中 | Oracle: ojdbc6.jar / ojdbc14.jar | |
| MySQL: mysql-connector-java-5.0.4-bin.jar | |
| 加载类库（驱动jar包） | Class.forName("oracle.jdbc.OracleDriver") | |
| Class.froName("com.mysql.jdbc.Driver") | |
| 获取连接 | DriverManager.getConnection(url,username,password) | |
| 创建statement对象 | conn.createStatement(), conn.prepareStatement(sql) | |
| 执行sql语句 | boolean flag = state.execute(sql); | 一般用来执行DDL语句 |
| ResultSet rs = state.executeQuery(sql);   1. boolean rs.next() 移动到下一行 2. xxx rs.getXxx() 取得字段的内容 | 执行select |
| int i = executeUpdate(sql); | 执行insert, update, delete |
| 处理结果集 |  | |
| 关闭连接 |  | |

# 连接池

|  |  |
| --- | --- |
| 建立与数据库的连接是一项较耗资源的工作，频繁的进行数据库连接建立操作会产生较大的系统开销，为了解决上述问题，可以采用数据库连接池技术 | |
| 原理 | 系统初始运行时，主动创建若干的连接，组成一个池。之后应用程序请求数据库连接时，从池中取出已有连接，使用完后，不再关闭，而是归还 |

# DBUtil

|  |  |  |
| --- | --- | --- |
| 导包 | commons-dbcp | |
| 数据库连接池 | private static BasicDataSource ds; | |
| 静态块初始化 | 加载db文件 | Properties prop = new Properties();  prop.load(new FileInputStream("config.properties")); |
| 初始化连接池 | ds = new BasicDataSource(); |
| 加载驱动 | ds.setDriverClassName(prop.getProperty("drivername")); |
| 设置url | ds.setUrl(prop.getProperty("url"); |
| 设置用户名 | ds.setUsername(prop.getProperty("username")); |
| 设置密码 | ds.setPassword(prop.getProperty("password")); |
| 获取连接 | return ds.getConnection(); | |

# Statement和PreparedStatement

|  |  |
| --- | --- |
| Statement | PreparedStatement |
| PreparedStatement继承Statement | |
| createStatement() | preparedStatement(sql) ps.setXxx() |
| 不支持预编译，每次执行sql语句都要数据库执行SQL编译 | 支持预编译，创建对象的时候就预先进行编译在数据库系统中，对于批量处理可以大大提高效率 |
| 不支持 | 支持动态设置参数 |
| 不可以，如"where username = '" + username + "'"; | 可避免单引号的编码麻烦，如"where username = ?"; |
| 不可以防止SQL注入，"1' OR '1'='1" | 可防止SQL注入 |
| 单条语句执行性能更好 | 单条语句执行开销更大 |

ResultSetMetaData

可滚动结果集

批量更新

|  |
| --- |
| for (int i = 0; i < 1000; i++) {  ps.setString(1, "Twins" + i);  ps.addBatch(); // 添加到批操作缓存  if (i%500 == 0) {  ps.executeBath(); // 及时处理，发送至数据库执行  ps.clearBath(); // 清空列表  }  ps.executeBatch(); |

返回自动主键

JDBC分页查询