# oracle数据库笔记

## 1. 常用增删查改等操作

|  |  |
| --- | --- |
| **类型** | **方式** |
| 数据库 | **创建数据库：**  create database database\_name  **删除数据库：**  drop database database\_name |
| 表 | **创建新表：**  create table tabname(col1 type1 [not null] [primary key],col2 type2 [not null],..)  **根据已有表创建新表：**  Create table table\_new like table\_old;  Create table table\_new as select col1, col2 ... from table\_old definition only;  注：definition only表示只定义无数据  **删除表：**  drop table table\_name; |
| 列 | **增加列：**  alter table table\_name add column col type;  注：列增加后将不能删除。DB2中列加上后数据类型也不能改变，唯一能改变的是增加varchar类型的长度。  **改变列属性：**  alter table table\_name modify col type; |
| 主键 | **添加主键：**  Alter table table\_name add primary key(col);  **删除主键：**  Alter table table\_name drop primary key(col); |
| 索引 | **创建索引：**  Create [unique] index index\_name on table\_name[col1, col2 ...];  **删除索引：**  Drop index index\_name;  注：  索引的优点：加快数据检索速度；“唯一性索引”可保证数据记录唯一性；加快表的连接速度。  索引的缺点：本身占用空间；增删改数据时需要动态维护索引。 |
| 视图 | 创建视图：  Create view view\_name as select statement;  删除视图：  Drop view view\_name; |
| 基本sql | **选择：**  select \* from table1 where 范围  **插入：**  insert into table1(field1,field2) values(value1,value2)  **删除：**  delete from table1 where 范围  **更新：**  update table1 set field1=value1 where 范围  **查找：**  select \* from table1 where field1 like ’%value1%’--like的语法很精妙，查资料!  **排序：**  select \* from table1 order by field1,field2 [desc]  **总数：**  select count(field1) as totalcount from table1  **求和：**  select sum(field1) as sumvalue from table1  **平均：**  select avg(field1) as avgvalue from table1  **最大：**  select max(field1) as maxvalue from table1  **最小：**  select min(field1) as minvalue from table1 |
| 外连接 | **Left (outer) join:**  Select table\_a.\*, table\_b.\* from table\_a left join table\_b on table\_a.col=table\_b.col;  **Right (outer) join:**  Select table\_a.\*, table\_b.\* from table\_a right join table\_b on table\_a.col=table\_b.col;  **Full (outer) join:**  Select table\_a.\*, table\_b.\* from table\_a full join table\_b on table\_a.col=table\_b.col;  **Inner join:**  Select table\_a.\*, table\_b.\* from table\_a inner join table\_b on table\_a.col=table\_b.col;  注：  1. 外连接是相对内连接，其outer可以省略；  2. 内连接的inner可以省略用逗号表示，这时就是普通的双表联合查询； |
| 几个高级查询运算词 | **UNION:**  Select col1, col2 ... from table1 union [all] Select col1, col2 ... from table2;  **INTERSECT:**  Select col1, col2 ... from table1 intersect Select col1, col2 ... from table2;  **EXCEPT: (oracle中用minus)**  Select col1, col2 ... from table1 except [all] Select col1, col2 ... from table2;  共同要求：  连接的两个结果集必须有相同的列数和列顺序，同时列的数据类型必须兼容。 |
| Group by | SELECT column\_name, aggregate\_function(column\_name)  FROM table\_name  WHERE column\_name operator value  GROUP BY column\_name |
| Between | Select \* from table1 where col [not] between value1 and value2; |
| In | Select \* from table1 where col [not] in (value1, value2 ...); |
| 分页查询 | SELECT \* FROM  (  SELECT A.\*, ROWNUM RN  FROM (SELECT \* FROM TABLE\_NAME) A  WHERE ROWNUM <= 40  )  WHERE RN >= 21  注意，oracle的select语句中不能设置ROWNUM下限，即>=，因为oracle查询是从第一条开始根据结果递增，如果下限大于1则无法继续 |

## 2. oracle对误删表的恢复

1、从FLASHBACK TABLE里查询被删除的表

SELECT \* FROM RECYCLEBIN ORDER BY DROPTIME DESC;

2.执行表的恢复

FLASHBACK TABLE TABLE\_NAME TO BEFORE DROP;

或者FLASHBACK TABLE SCOTT.TEST TO TIMESTAMP TO\_TIMESTAMP('2009-12-11 20:47:30','yyyy-mm-dd hh24:mi:ss');

看已删除表的历史情况：

SELECT \* FROM FLASHBACK\_TRANSACTION\_QUERY WHERE TABLE\_NAME='TEST';

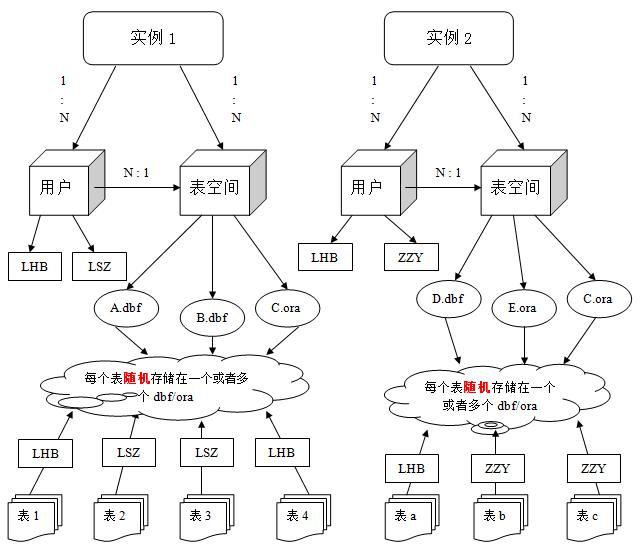
看是否有数据：

SELECT \* FROM SCOTT.TEST AS OF TIMESTAMP TO\_TIMESTAMP('2009-12-11 20:53:57','yyyy-mm-dd hh24:mi:ss');

## 3. oracle死锁

|  |  |  |
| --- | --- | --- |
| 类型 | 方法 | 说明 |
| 发现死锁 | select username,lockwait,status,machine,program from v$session where sid in (select session\_id from v$locked\_object) | 如果有输出的结果，则说明有死锁，且能看到死锁的机器是哪一台。字段说明：  Username：死锁语句所用的数据库用户；  Lockwait：死锁的状态，如果有内容表示被死锁。  Status： 状态，active表示被死锁  Machine： 死锁语句所在的机器。  Program： 产生死锁的语句主要来自哪个应用程序。 |
| 查看死锁 | select sql\_text from v$sql where hash\_value in (select sql\_hash\_value from v$session where sid in (select session\_id from v$locked\_object)) |  |
| 查找死锁的进程 | SELECT s.username,l.OBJECT\_ID,l.SESSION\_ID,s.SERIAL#, l.ORACLE\_USERNAME,l.OS\_USER\_NAME,l.PROCESS FROM V$LOCKED\_OBJECT l,V$SESSION S WHERE l.SESSION\_ID=S.SID; |  |
| kill掉这个死锁的进程 | alter system kill session ‘sid,serial#’; | 其中sid为上面的session\_id |
| 如果还不能解决 | select pro.spid from v$session ses,v$process pro where ses.sid=XX and ses.paddr=pro.addr; | 其中sid用死锁的sid替换: exit；  ps -ef|grep spid  其中spid是这个进程的进程号，kill掉这个Oracle进程 |

## 4. 实例、用户、表空间、数据文件、临时数据文件、表



* 安装Oracle后会有默认的实例ORCL，可以不用再创建新的实例；
* 用户、表空间属于实例的逻辑概念，一个实例下的用户和表空间独立于其它实例。用户主要是权限上的考虑，表空间主要是数据文件存储上的考虑。
* 用户被授权于表空间，表空间在逻辑上整理了数据文件，数据文件只能属于某个表空间。
* 表可能分布在不同的数据文件中，表逻辑上归属于用户（同一表空间下不同用户可以有相同的表名），与表空间无关，所以当我们登录数据库的时候需要指定用户而不是表空间。

## 5. Oracle数据库优化

**物理优化：**

1). Oracle的运行环境（网络，硬件等）

2). 使用合适的优化器

3). 合理配置oracle实例参数

4). 建立合适的索引（减少IO）

根据记录量，索引也是很占空间的；删除标的部分记录时不会删除索引，只有记录全部删除才会删除索引。

5). 将索引数据和表数据分开在不同的表空间上（降低IO冲突）

6). 建立表分区，将数据分别存储在不同的分区上（以空间换取时间，减少IO）。表分区方式：记录字段的范围、时间、枚举取值、散列等方式分区。表分区应用场景：表大小超过2G；表中含历史数据、新数据增加到新的分区中。

**逻辑优化：**

1). **表拆分**。可以对表进行逻辑分割，如中国移动用户表，可以根据手机尾数分成10个表，这样对性能会有一定的作用。

2). **绑定变量**。Sql语句使用占位符语句，并且开发时候必须按照规定编写sql语句（如全部大写，全部小写等）oracle解析语句后会放置到共享池中。

如： select \* from Emp where name=? 这个语句只会在共享池中有一条，而如果是字符串的话，那就根据不同名字存在不同的语句，所以占位符效率较好。

3). **存储过程**。数据库不仅仅是一个存储数据的地方，同样是一个编程的地方，一些耗时的操作，可以通过存储过程等在用户较少的情况下执行，从而错开系统使用的高峰时间，提高数据库性能。

4). **sql写法**。尽量不使用\*号，如select \* from Emp，因为要转化为具体的列名是要查数据字典，比较耗时；使用Exits Not Exits 替代 In Not in ；

6). **事务处理**。合理使用事务，合理设置事务隔离性。数据库的数据操作比较消耗数据库资源的，尽量使用批量处理，以降低事务操作次数。

**重要说明：**

数据库的**垂直拆分和水平拆分**。垂直拆分、是解决表之间IO的问题，将数据量太大的表放到单独的server上；水平拆分、是解决单表IO的问题，对数据量太大的单表根据字段特征进行拆分成多个表。

**绑定变量的必要性**：oracle的sql执行过程为：1.语法检查，判断sql拼写是否符合语法；2. 语义检查，判断合法权限下相关对象是否存在；3.语句解析，利用内部算法生成解析树及执行计划；4.执行sql，返回结果。其中第三步比较耗时，是根据内部算法看当前sql是否已在library cache中，如果在的话则省略了优化器的相关工作，这个在大量查询时很重要。

## 6. oracle几个自定义结构变量

|  |  |  |
| --- | --- | --- |
| 类型 | 语法 | 描述 |
| record | Type recordName is record(  V1 data\_ype [not null] [:=defaultValue];  V2 data\_ype [not null] [:=defaultValue];)  eg:  declare  type recordTest is record(  var1 tableTest.volumn1%type,  var2 tableTest.volumn2%type  );  recordResult recordTest;  begin  select vol1, vol2 into recordResult from tableTest2;  dbms\_output.put\_line(recordResult.var1);  end; | 类似python中的元组，可以用来存储数据表的一行记录； |
| varray | type varrayName is varray(size) of data\_ype [not null]  eg:  declare  type varrayTest is varray(5) of varchar2(64);  varrayResult varrayTest;  begin  varrayResult := varrayTest('1', '2', '3', '4', '5');  for i in varrayResult.first .. varrayResult.last loop  DBMS\_OUTPUT.PUT\_LINE(varrayResult(i));  end loop;  end; | 类似c中的数组，成员支持下标取值，灵活易用； |
| table | type tableName is table of data\_ype [not null] index by [BINARY\_INTEGER|PLS\_INTEGER|VARRAY2]  eg:  declare  type recordTest is record(  var1 tableTest.volumn1%type,  var2 tableTest.volumn2%type  );  type tableTest is table of recordTest;  tableResult tableTest;  begin  select vol1, vol2 into bulk collect tableResult from tableTest2;  for i in tableResult.first .. tableResult.last loop  dbms\_output.put\_line(tableResult.var1);  end loop;  end | 对record类型的扩展，支持多行数据，可以用来存储数据表的查询结果。  **存储单列多行**：  type tableTest is table of varchar2(64);  **存储多列多行和ROWTYPE结合使用**：  type tableTest is table of tableTest%rowtype;  **存储多列多行和RECORD结合使用**：  Type tableTest is table of recordTest; |

## 7. 存储过程、函数、作业

**存储过程**：

create or replace **procedure** 存储过程名 (p1 in|out type, p2 in|out type) as|is

var1 type;

begin

null;

end;

**函数**：

create or replace **function** 函数名 (p1 type, p2 type) return type is

var1 type;

begin

null;

end;

二者的区别：

函数一定要有return返回值，存储过程没有；

函数可用在sql语句调用但不能单独执行，存储过程可以单独执行但不能用于sql调用。

此外二者在语法等方面，都基本相似。

**作业：**

variable test\_job number;

begin

dbms\_job.submit(:test\_job,'test\_procedure;',sysdate,'sysdate+1/1440');

end;

这里参数的解释：

job OUT binary\_ineger：出参，唯一标识一个工作

What IN varchar2：表示被执行的代码块对象名

next\_date IN date：初次运行此job的时间

interval IN varchar2：何时此job被重新运行

## 8.其它

* oracle的导入导出非常耗内存；
* oracle的排序对性能影响很大，如非必要就别排序；
* 表设计时，根据具体业务场景，不一定死扣范式；
* 外键的好处是提高数据质量，但坏处是增加了其它负担，而且需要注意一些问题，比如外键如果不加索引会有严重性能问题；
* oracle字段在设计时顺序就确定了，对于访问频繁的字段最好放前面，对性能有好处；
* 数据类型在设计是一定要确定好，另外不要用char, long这些数据类型；
* varchar2类型的长度，虽然不会影响实际记录数据的大小，但对于外部应用来说，是有影响的，比如weblogic等进程会根据类型长度预分配内存，这样可能导致这些进程的内存消耗过大；
* 表大分表、业务大分库，不过这些一般在设计阶段会容易些，因为没有数据负担；
* 主键不要有业务规则，主键仅标志记录唯一性即可，否则随业务膨胀会有影响；