# Oracle

## 表空间、段、区和块

Oracle的逻辑结构包括表空间（tablespace）、段（segment）、区（extent）、数据块（data block）

Oracle数据库在逻辑上是由多个表空间组成的；

表空间中存储的对象叫段，比如数据段、索引段、回退段；

段由区组成，区是磁盘分配的最小单位。段的增大时通过增加区的个数实现；

每个区的大小是数据块大小的整数倍，区的大小可以不相同；

数据块是数据库中最小的I/O单位，同时也是内存数据缓冲区的单位，及数据文件存储空间单位。块的大小由参数DB\_BLOCK\_SIZE设置，其值应设置为操作系统块大小的整数倍。

### 数据块（Block）

数据块Block是Oracle存储数据信息的最小单位。Oracle也就是通过数据块屏蔽不同操作系统存储结构的差异。

Oracle所有对数据的操作和空间分配，实际上都是针对数据块Block的操作。我们从数据表中搜索出一行，实际中oracle就会从内存缓冲区（或硬盘）中读取到该行所在的数据块，在返回这数据块的指定数据行。Oracle无论是在缓冲区，还是在硬盘，进行数据操作的最小单位也就是数据块。

数据块是有大小的，创建数据库时通过参数设置。

数据块的大小是通过kb字节个数来指定的，默认为8KB。

查看block大小语句：show parameter db\_block\_size;

NAME TYPE VALUE

------------------------------------ -----------

db\_block\_size integer 8192 //1024×8

### 区（extent）

区extent是比数据块大一级的存储结构，表示的是一连串连续的数据块集合。我们知道，物理存储通常是随机的读写过程，即使在同一个文件里，我们也不能保证相同的一个信息是存储在绝对连续的物理存储空间的。

在进行存储数据信息的时候，oracle将分配数据块进行存储，但是不能保证所有分配的数据块都是连续的结构。所以出现分区extent，表示一系列连续的数据块集合。

视图dba\_extents（或者all\_extents、user\_extents）是我们研究分区结构和存储构成的重要手段。

### 数据段（segment）

数据段是分区extent的上层组织单位，一个数据库对象对应一个segment，数据库对象归属在不同的schema（owner）上的。

数据段是与数据库对象相对应，一般一个数据库对象对应一个数据段。多个extent对应一个数据段，每个数据实际上就是数据库一个对象的代表。、

数据段的类型是多样的。任何种类的数据库对象，本质上都是一种数据段。数据表、索引、回滚、聚集这些都是数据段的一种表现形式。同时，数据段是在数据对象创建的时候就已经创建出来，随着对象体积的增大，而不断分配多个extents进行管理。

### 表空间（tablespace）

表空间tablespace是存储结构中的最高层结构。建立一个表空间的时候，是需要制定存储的文件。一个表空间可以指定多个数据文件，多个文件可以在不同的物理存储上。

## 表的基本操作+merge+truncate

（1）创建表create

create table tableName(colName dataType, ...n)[tablespace tableSpaceName]

（2）修改表alter

alter table tableName

[add (colName dataType, ...n)] --添加列

[modify (colName dataType, ...n)] --修改列

[drop drop\_clause] --删除列或约束条件

例如：

alter table emp4 add (test varchar2(10),test2 number); --添加多列

alter table emp4 modify (test varchar2(20),test2 varchar2(20)); --修改多列

alter table emp4 drop (test,test2); --删除多列

（3）删除表drop

drop table tableName.

（4）在往表中插入记录时，如果需要插入某列的值为空，则值必须置为null，如果列值指定为该列的默认值，则用default。

（5）merge语句，使用该语句可以实现对表的更新或插入

merge into tableName using tableName on(join\_condition)

when matched then update set...

when not matched then insert(...) values(...)

例如：

merge into student a using student b on(a.id>b.id)

when matched then update set age=age+2

when not matched then insert(id,name,age,no) values(b.id,b.name,b.age,b.no);

（6）删除表记录之delete和truncate

delete语法：delete from tableName [where condition]

truncate语法：truncate table tableName

truncate删除tableName表中的所有记录，使用truncate可以释放占用的数据块表空间。truncate删除是不能回滚的，而delete删除是可以回滚的。正因为如此使用truncate删除所有记录的速度比用delete删除所有记录的速度快。

## 索引

索引作用：提高对表的查询速度；对表有关列的取值进行检查；在使用order。

索引分类：B\*树索引、位图索引

B\*树索引：一棵树，有分支和叶，分支相当于书的大目录，叶则相当于具体的书页。Oracle用B\*树机制存储索引条目，可以保证用最短路径访问键值。

位图索引：主要用于节省空间，减少Oracle对数据块的访问，它采用位图偏移方式来与表的行id号对应，采用位图索引一般是重复值太多的表字段。位图索引在实际密集型OLTP(数据事务处理)中用得比较少，因为OLTP会对表进行大量的删除、修改、添加操作，Oracle每次进行这些操作的时候都会对要操作的数据块加锁，以防止多人操作容易产生的数据块锁等待甚至死锁现象。而在OLAP(数据分析处理)中应用位图有优势，因为OLAP中大部分是对数据库的查询操作，而且一般采用数据仓库技术，所以大量数据采用位图索引节省空间比较明显。

提高查询速度的方法还有在表上建立主键，主键与唯一索引的差别在于唯一索引可以空，主键为非空.

创建索引：

create [unique | bitmap] index [schema.]indexName

on [schema.]tableName(colName[ASC|DESC], ...n)

删除索引：

drop index indexName;

## 数据完整性

数据完整性分类：域完整性、实体完整性和参照完整性

### 域完整性

又称列完整性，是指确定一个数据集对某列是否有效和确实是否允许空值。域完整性通常是通过使用有效性检查来实现的，还可以通过限制数据类型、格式或者可能的取值范围来实现

Oracle通过check约束实现域完整性。

create table tableName(

colName dataType check(checkExpression),

colName dataType constraint checkName check(checkExpression)

);

例如：

create table person(

id int primary key,

sex varchar(4) check(sex='男' or sex='女'),

age int default 0 constraint ageCheck check(age>=0 and age<=125)

)

利用SQL语句在修改时创建check约束:

alter table tableName add(constraint checkName check(checkExpression), ...n)

例如：

alter table person

add(constraint sexCheck check(sex='男' or sex='女'),

age int check(age>=0 and age<=125));

利用SQL语句删除check约束:

alter table tableName drop constraint checkName;

例如：

alter table person drop constraint sexCheck;

### 实体完整性

又称行完整性，它要求表中的每一行都有一个唯一的标识符，这个标识符就是主关键字。

通过索引、unique约束和primary key约束可以实现实体完整性。

当为表创建primary key时，Oracle会自动为primary key对应的列创建唯一索引，实现数据的唯一性。

如果要确保一个表中的非主键列的值不可重复，可以在该列加上unique约束，unique约束也会自动产生索引。

primary key约束和unique约束的区别如下：

1. 一个数据表只能有一个primary key约束，但可以有多个unique约束
2. primary key对应字段的值不能为null，而unique约束对应列的值可以为null
3. primary key对应字段或字段的组合的值是不能重复的，而unique约束列的值为null的时候是可以重复的，也就是说unique约束列可以有多个null值

创建对应的约束：

create table tableName(

col1 dataType primary key,

col2 dataType unique,...n);

create table tableName(

col1 dataType, col2 dataType, ...n,

primary key(col1, col2), unique(col3, col4));

alter table tableName add(

constraint pkName primary key(column1, column2),

constraint uniqueName unique(column3, column4));

例如：

create table person(id int primary key, name varchar(30) unique, age int);

create table person(id int, name varchar(30), age int, primary key(id), unique(name));

alter table person add(

constraint personPk primary key(id),

constraint nameUnique unique(name));

删除primary key和unique约束：

alter table tableName drop constraint constraintName;

例如：

alter table person drop constraint personPk;

### 参照完整性

又称引用完整性，它可以保证从表（参照表）中的数据与主表（被参照表）中的数据一致性。

参照完整性通过定义从表的外键与主表的主键相对应来实现。可以利用foreign key定义从表中的外键，利用primary key定义主表中的主键。

创建外键：

create table messagge(id int primary key, fileId int references file(id));

alter table message add(

constraint messageFk foreign key(fileId) references file(id));

删除外键：

alter table message drop constraint messageFk;

## 表查询中需要注意的一些问题

1. 在select语句中同时使用where、group by和having子句时的执行顺序：where->group by->having
2. count(\*)会统计表中的null计算在内,而count(colName)会忽略对应字段的null记录
3. 以join关键字指定的连接主要有四种：

（1）inner join是内连接（默认），inner可省略，把所有满足条件的记录列出来。

（2）left outer join是左外连接，outer可省略，它除了列出所有满足条件的记录外，还包括左表的所有行。

（3）right outer join是右外连接，outer可省略，它除了列出所有满足条件的记录外，还包括右表的所有行。

（4）full outer join是全外连接，outer可省略，，它除了列出所有满足条件的记录外，还包括左表和右表的所有行。

1. select中union子句

[SQL语句1] UNION [SQL语句2]

[SQL语句1] UNION ALL [SQL语句2]

union all是直接连接，取到得是所有值，记录可能有重复；union 是取唯一值，记录不重复

## 视图

视图是从一个或多个表或视图导出的表。视图是一个虚表，数据库不对视图所对应的数据进行实际存储，只存储视图的定义，对视图的数据进行操作时，系统根据视图的定义去操作与视图相关联的基表。

视图一旦被定义以后就可以像表一样被查询、修改、删除和更新。

视图特点：

（1）可以限制用户只能通过视图检索，这样就可以对最终用户屏蔽建表时底层的基表。

（2）为用户集中数据，简化用户的数据查询和处理。（有时我们所需要的数据可能分散在几个不同的表中，定义视图可以将它们集中在一起，从而方便我们对数据的查询和处理。）

（3）便于数据共享。各不同的用户不必都保存自己的数据，同样的数据只需保存一次。

创建视图：

create [or replace] view viewName[(col1, col2, ...n)]

as selectStatement [with check option[constraint constraintName]] [with read only]

例如：

对person表的name和sex列建立只读视图：

create or replace view personView as select name,sex from person with read only;

查询视图：

该语句就会查出视图personView里面的所有字段：

select \* from personView;

删除视图：

drop view viewName;

## PL/SQL

## 常用内置函数

to\_char()：转换为字符串

to\_number()：转换为数值类型

to\_date()：转换为日期类型

常用字符串函数：

length(value) : 返回value的长度，value可以是字符串、数字或表达式

lower(value) : 把给定字符串value变为小写

upper(value) : 把给定字符串value变为大写

string1 || string2 : 连接两个字符串为一个字符串

initcap(string) : 将字符串的首字母大写

replace(String,src,dest) : 将string中的src用dest代替

substr(string,start[,count]) : 从start位置开始在string中截取count个字符

统计函数：

avg([distinct] columnName) : 返回columnName中所有值的平均值

count([distinct] value) : 统计记录数，若使用distinct则只统计不同的非空数值。

max(columnName) : 返回columnName中的最大值

min(columnName) : 返回columnName中的最小值

sum(columnName) : 返回columnName中各值的总和

常用日期函数：

add\_months(date,number) : 返回date之后number个月的日期

last\_day(date) : 返回date所在月份的最后一天

months\_between(date1,date2) ：返回date2减去date1之后的月数

next\_day(date,'day') : 返回date后的day所在的日期，其中day表示全称的星期，如Monday

round(date,'format') : 把date四舍五入到指定的format格式

to\_char(date,'format') : 将日期date转换为format格式的字符串

to\_date(string,'format') : 将字符串string按照format转换成日期形式

to\_timestamp(string,'format')：将字符串string按照format转换成时间戳

trunc(date):将日期date的时分秒都设为0

## 用户自定义函数

参数类型：

in参数类型（默认）：将实参传递给形参，只能读不能写，函数返回时实参的值不改变；

out参数类型：函数内部可以被读写，函数返回时形参的值赋给实参；

in out参数类型：调用时，实参的值总是传递给形参，结束时，形参的值传递给实参。

创建函数：

create [or replace] function functionName(

param1 model dataType1, --model指in、out、in out模式

param2 model dataType2,

...

)

return returnDataType

is/as

begin

function\_body

return expression

end functionName;

例如：

create or replace function explainParameter -- 定义一个名为explainParameter的函数

(

inParam in char, -- 定义该参数类型为in参数类型，只能用于赋值

outParam out char, -- out参数类型，只能用于传值

inAndOutParam in out char -- in out参数类型，既能赋值，又能传值

)

return char -- 表示函数的返回类型为char类型

as -- 表示函数体部分

returnChar char; -- 这里声明返回变量为char类型的returnChar变量。

begin

inParam := 'Hello World'; --这是错误的，in类型的参数只能用来传值，不能赋值

outParam := 'Hello World'; --这是正确的，out类型的参数可以用来被赋值

inAndOutParam := 'Hello World'; --这是正确的，in out参数既可以用来传值，又可以被赋值

inAndOutParam := outParam; --这是错误的，out参数不能用来传值

return(returnChar); -- 返回returnChar，也可以不要括号，直接写return returnChar。

end explainParameter;

例：

CREATE OR REPLACE FUNCTION testFunc(num1 IN NUMBER, num2 IN NUMBER)

RETURN NUMBER

AS

num3 number;

num4 number;

num5 number;

BEGIN

num3 := num1 + num2;

num4 := num1 \* num2;

num5 := num3 \* num4;

RETURN num5;

END;

函数调用：

select testFunc(1,2) from tableName;

num := testFunc(1,2);

## 游标

### 显式游标

步骤：声明游标🡪打开游标🡪读取游标🡪关闭游标

步骤一：声明游标

declare cursor moduleCursor is select name,parent from t\_module;

步骤二：打开游标

open cursorName;

其中，cursorName是游标名字，可使用系统变量%rowcount查看游标当前所在的位置。

示例，定义游标moduleCursor，然后打开游标，输出其当前所在的位置。

declare cursor moduleCursor is select name,parent from t\_module;

begin

open moduleCursor;

dbms\_output.put\_line(moduleCursor%rowcount);

end;

步骤三：读取数据

打开游标后，使用fetch语句从中读取数据

fetch cursorName[into variableName, …n]

示例代码：

declare

moduleName char(20);

moduleId number(3);

cursor moduleCursor is select id, name from t\_module;

begin

open moduleCursor;

dbms\_output.put\_line(moduleCursor%ROWCOUNT);

fetch moduleCursor into moduleId, moduleName;

--在进行found操作之前必须先进行一次fetch操作，否则游标不会指向结果集的开始，

while moduleCursor%found

loop

dbms\_output.put\_line('ModuleName: ' || moduleName || ' , ModuleId: ' || moduleId);

fetch moduleCursor into moduleId, moduleName;

end loop;

close moduleCursor;

end;

在读取游标的数据也可使用游标的for循环：

declare

moduleId number(3);

moduleName char(20);

cursor moduleCursor is select id, name from t\_module;

begin

--moduleCursorRecord是一个临时变量，代表当前的一条记录

for moduleCursorRecord in moduleCursor loop

moduleId := moduleCursorRecord.id;

moduleName := moduleCursorRecord.name;

dbms\_output.put\_line('ModuleId:' || moduleId || ', ModuleName:' || moduleName);

end loop;

end;

在对游标进行循环取数据时，使用for循环可以大大减少代码量，增强代码的可读性。此外，使用for循环的时候，不需要显式的打开和关闭游标。

步骤四：关闭游标

close cursorName;

需要注意的地方：

（1）用%found和%notfound检验游标成功与否。该属性表示当前的游标是否指向有效的一行，根据其返回值true/false检查是否结束该游标的使用。

（2）游标目标变量必须与游标select表中的列的数据类型一致。

（3）如果试图打开一个已打开或者关闭一个已关闭的游标时将会出现错误。因此，在进行打开或者关闭操作前，如果不知道游标的当前状态，可以使用%isopen进行检查，再根据其返回值执行相应的打开或者关闭操作。

（4）可以使用%rowcount获取当前游标所在的位置。

### 隐式游标

示例：

create or replace function moduleNameFunc(moduleId in number)

return char

as

moduleName char(20);

begin

select name into moduleName from t\_module where id = moduleId;

return moduleName;

end;

注意：

（1）每一个隐式游标必须有一个into语句。

（2）into语句后接收数据的变量的数据类型要与对应的列的数据类型一致。

（3）隐式游标一次只能返回一条数据。

## 异常

（1）预定义异常

declare

num1 number := 2;

num2 number := 0;

num3 number;

begin

num3 := num1/num2; --当执行这句的时候，系统捕捉到异常会自动转到异常处理部分，最后输出“不能除0”。

exception

when zero\_divide then

dbms\_output.put\_line('不能除0');

when too\_many\_rows then

dbms\_output.put\_line('too many rows');

end;

（2）自定义异常

可以先定义一个exception变量，然后使用raise语句来抛出一个异常，然后在捕捉异常的时候捕捉对应的抛出异常。

declare

selfDefineException exception; /\*定义异常变量\*/

num1 number := 2;

num2 number := 3;

num3 number;

begin

num3 := num1 + num2;

if num3=5 then

raise selfDefineException; /\*抛出自定义的异常\*/

end if;

exception

when selfDefineException then /\*捕获自定义的异常\*/

dbms\_output.put\_line("selfDefineException");

when zero\_divide then

dbms\_output.put\_line("不能除0");

end;

（3）使用others异常

declare

num1 number := 2;

num2 number := 0;

num3 number;

begin

num3 := num1/num2; --当执行这句的时候，系统捕捉到异常会自动转到异常处理部分，最后输出“不能除0”。

exception

when others then /\*这里直接捕捉others异常\*/

dbms\_output.put\_line('不能除0');

end;

## s