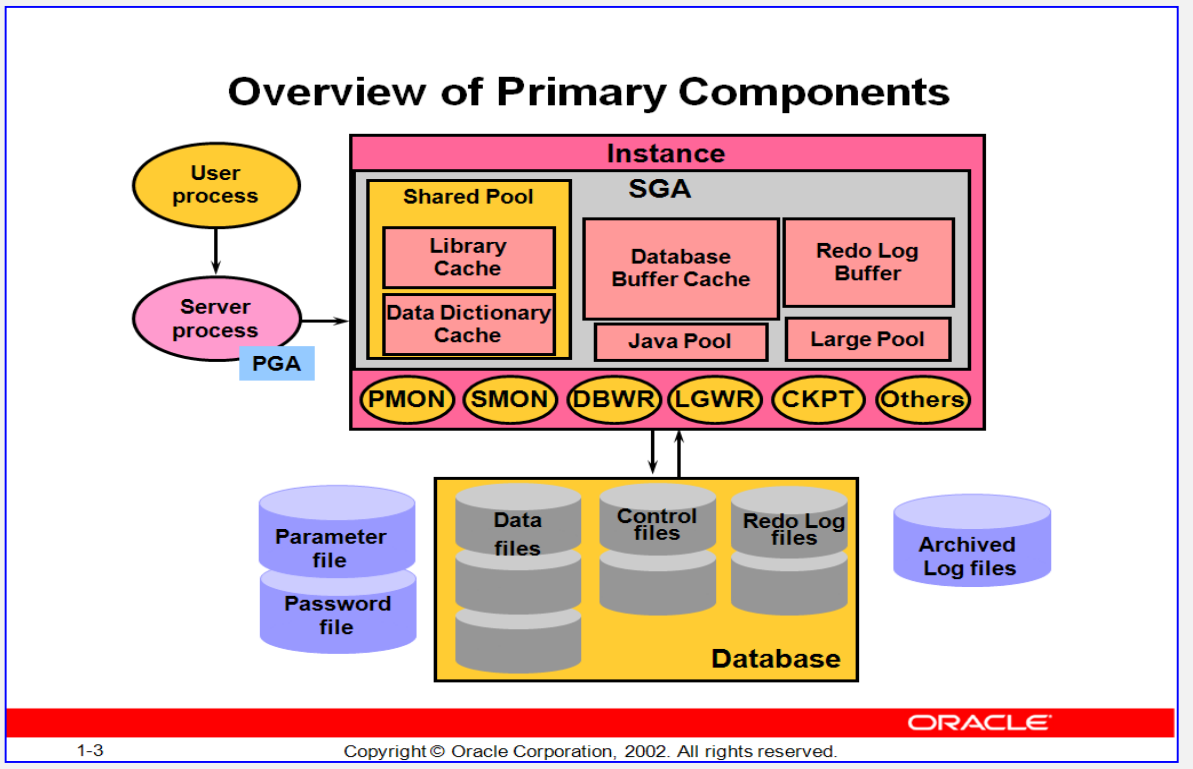
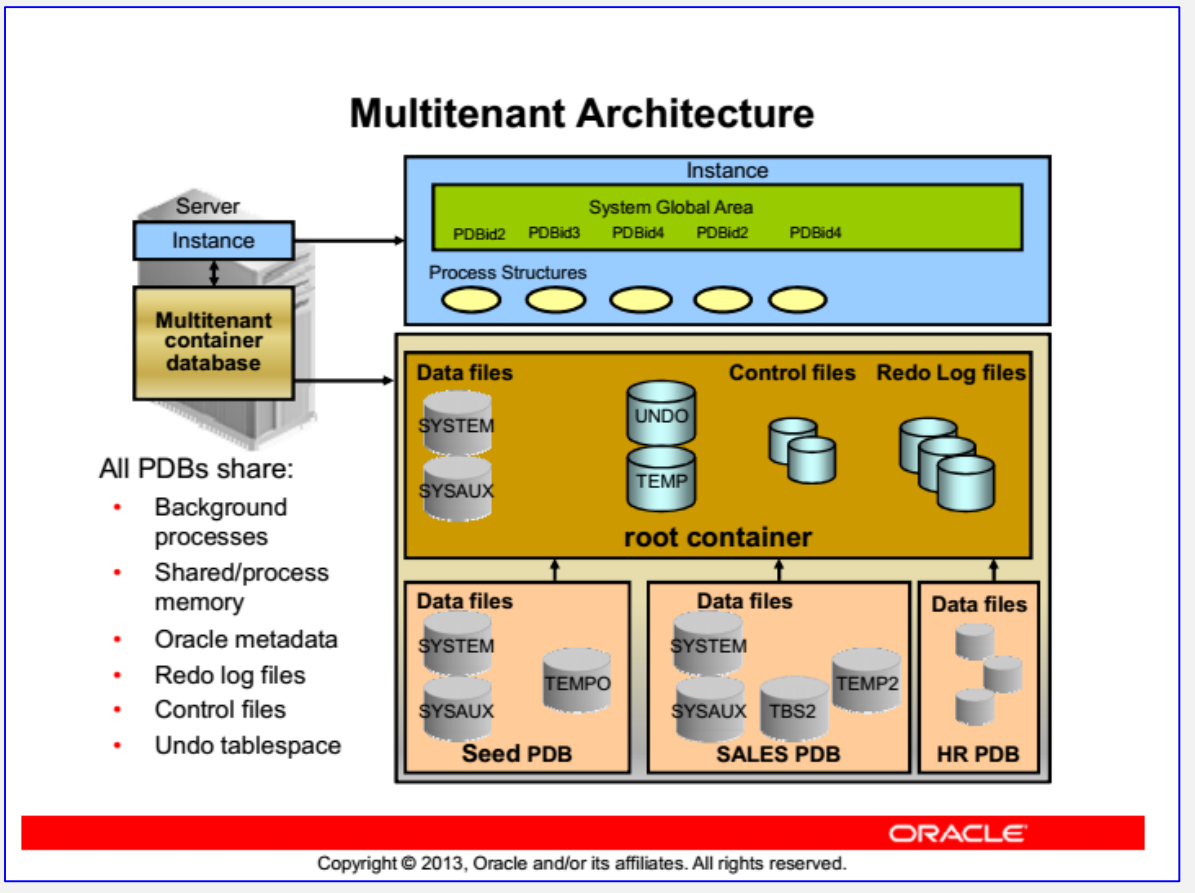
Oracle11g 体系-课堂笔记

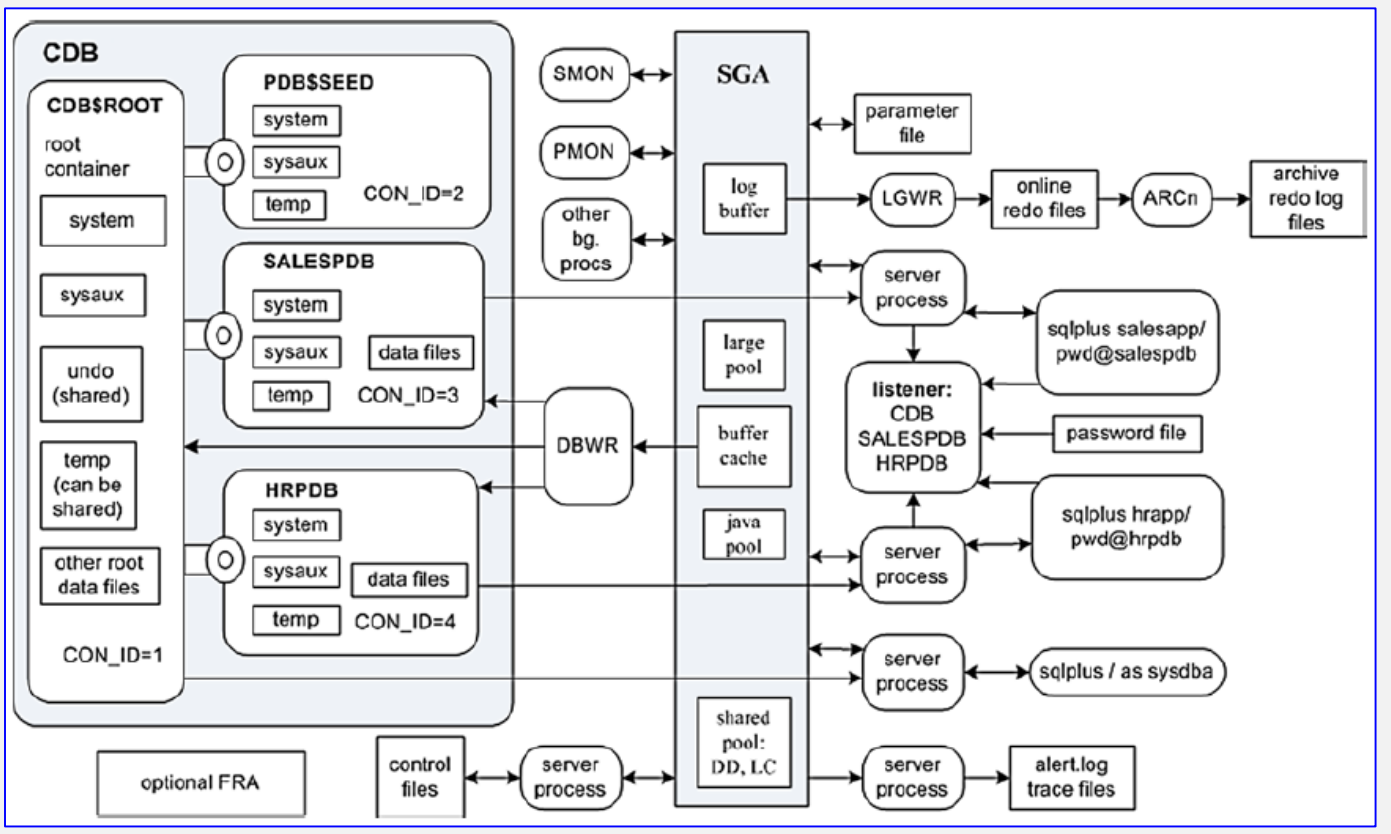
第一部分 体系架构

第一章：实例与数据库

1.1 Oracle 基础架构及应用环境







1.1.1 Oracle Server 的基本结构

1）oracle server：database + instance

2）database：data file、control file、redolog file

3）instance: an instance access a database

4）oracle memory: sga + pga

5）instance：sga + backgroud process

1.1.2 系统全局区 SGA：

1）在一个 instance 只有一个 sga

2）sga 为所有 session 共享，随着 instance 启动而分配

3）instance down ， sga 被释放

1.2 SGA 的基本组件

1）shared pool

共享池是对 SQL、PL/SQL 程序进行语法分析、编译、执行的内存区域。

共享池由库缓存（library cache）,和数据字典缓存（data dictionary cache）以及结果缓存（result cache）等组成。

共享池的大小直接影响数据库的性能。

关于 shared pool 中的几个概念

①library cache：

sql 和 plsql 的解析场所，存放 sql/plsql 语句代码，以及它们的执行计划。以备其他用户共享

②data dictionary cache

存放重要的数据字典信息，以备其他用户共享使用

③server result cache：

存放服务器端的 SQL 结果集及 PL/SQL 函数返回值

④User Global Area (UGA)：

共享服务器连接模式下如果没有配置 large pool，则 UGA 属于 SGA 的 shared pool， 专用连接模式时 UGA 属于 PGA

2）database buffer cache（PPT-II-328）

用于存储从磁盘数据文件中读入的数据，为所有用户共享。

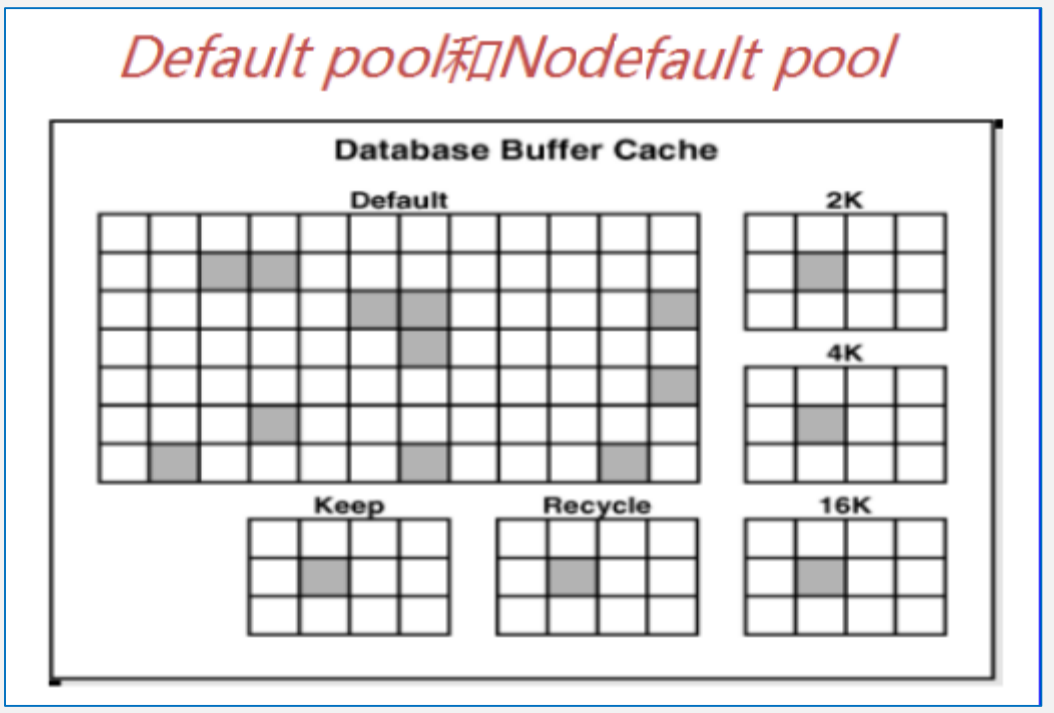
服务器进程（server process)负责将数据文件的数据从磁盘读入到数据缓冲区中，当后续的请求需要这些数据时如果在内存中找到，则不需要再从磁盘读取。

数据缓冲区中被修改的数据块（脏块）由后台进程 DBWR 将其写入磁盘。

数据缓冲区的大小对数据库的读取速度有直接的影响。

要弄明白 Database Buffer Cache 中的几个 cache 概念：

① Buffer pool=(default pool)+(nodefault pool)



其中：

default pool（参数 db_cache_size）是标准块存放的内存空间大小，SGA 自动管理时此参数不用设置。使用 LRU 算法清理空间

nodefault pool：对应的参数有

db_nk_cache_size 指定非标准块大小内存空间，比如 2k、4k、16k、32k。

db_keep_cache_size 存放经常访问的小表或索引等。

db_recycle_cache_size 与 keep 相反，存放偶尔做全表扫描的大表的数据。

①如何指定使用某个表调入 nodefault pool

SQL> alter table scott.emp1 storage(buffer_pool keep);

SQL>

select segment_name,buffer_pool from dba_segments where segment_name='EMP1';

②default pool 对应的参数是 db_cache_size 与标准块 default block 是配套的，如果 default block 是 8k, db_cache_size 这个参数将代替 db_8k_cache_size。

③如果要建立非标准块的表空间，先前要设定 db buffer 中的与之对应的 db_nk_cache_size 参数。

第一步，先指定 db buffer 里的 16k cache 空间大小。

SQL> alter system set db_16k_cache_size=8m;

第二步，建立非标准块表空间

SQL> create tablespace tbs_16k datafile '/u01/oradata/prod/tbs16k01.dbf' size 10m blocksize 16k;

SQL> select TABLESPACE_NAME,block_size from dba_tablespaces;

3）redo log buffer

以日志条目（redo entries）方式记录了数据库的所有修改信息(包括 DML 和 DDL），目的是为数据库恢复，日志条目首先产生于日志缓冲区，日志缓冲区较小，一般缺省值在 3M-15M 之间，它是以字节为单位的。

日志缓冲区的大小启动后就是固定不变的，如要调整只能通过修改参数文件后重新启动生效。不能动态修改！不能由 SGA 自动管理！

4）large pool（可选）

为了进行大的后台批处理操作而分配的内存空间，主要用于共享服务器的 session memory（UGA），RMAN 备份恢复以及并行查询等操作。有助于降低 shared pool 碎片。

5）java pool（可选）

为了 java 虚拟机及应用而分配的内存空间，包含所有 session 指定的 JAVA 代码和数据。

6）stream pool（可选）

为了 stream process 而分配的内存空间。stream 技术是为了在不同数据库之间共享数据，因此，它只对使用了 stream 数据库特性的系统是重要的。

1.3 Oracle 的进程:

1）user process：

客户端的 process，访问数据库分为三种形式，①sql*plus, ②应用程序，③web 方式（EM）

①sql*plus 可以执行 sql 和 plsql 请求，是典型的客户端进程。

linux 作为客户端：可以使用 ps 看到 sqlplus 关键字:

$ ps -ef |grep sqlplus

windows 作为客户端：可以通过查看任务管理器看到 sqlplus 用户进程:

C:\Documents and Settings\prod>sqlplus sys/system@ as sysdba

②应用程序

例如：通过 java 程序直接嵌套 sql 语句，或调用 Oracle 存储过程。

③web 方式

例如：使用 OEM 登录、管理数据库。

$emctl start dbconsole

2）server process：

服务器端的进程，user process 不能直接访问 Oracle，必须通过相应的 server process 访问实例，进而访问数据库。

[oracle@prod ~]$ ps -ef |grep LOCAL

在 linux 下看到的 server process, (LOCAL=YES)是本地连接，(LOCAL=NO)是远程连接。

可以在 oracle 查看 V$process 视图，它包括了当前所有的后台进程和服务器进程。

SQL> select pid,program,background from v$process;

background 字段为 1 是 background process，其余都是 server process

3）background process

基本的后台进程有

1）smon：系统监控进程

①当实例崩溃之后，Oracle 会自动恢复实例。②释放不再使用的临时段。

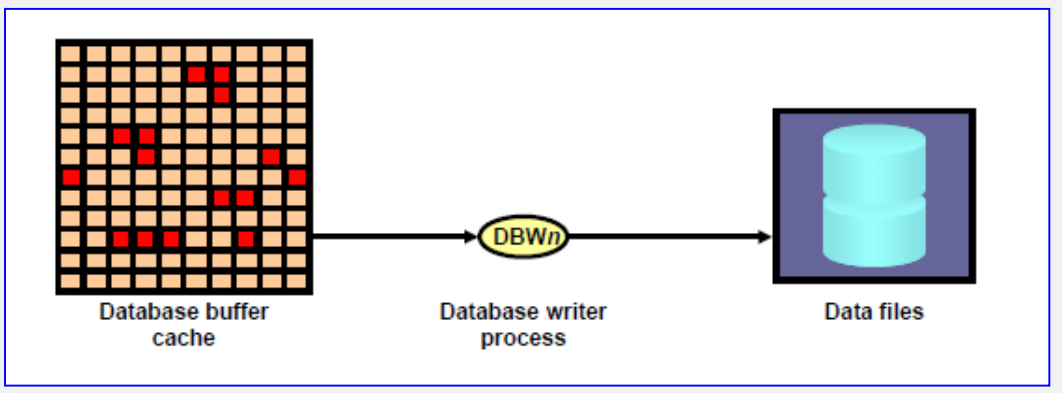
2）pmon：进程监控

①当 user process 失败时，清理出现故障的进程。释放所有当前挂起的锁定。释放服务器端使用的资源

②监控空闲会话是否到达阈值

③动态注册监听

3）dbwn：数据写入进程



1、将变更的数据缓冲区的脏 buffer 写入数据文件中。

2、释放数据缓冲区空间。

3、触发条件：

①ckpt 发生

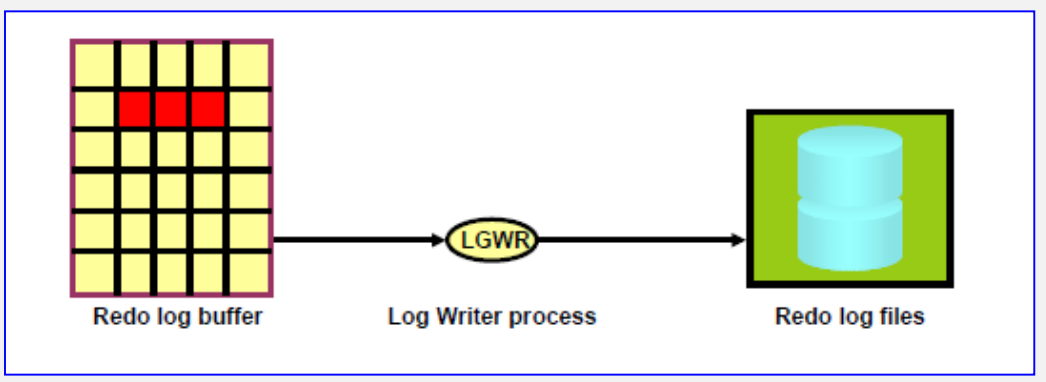
②脏块太多时(阈值)

③db_buffer 自由空间不够时

④表空间 read only/offline/backup 模式等

以上 4 个状况之一发生时，dbwn 都会被触发

4）lgwr：写日志条目



1、将日志缓冲区中的日志条目写入日志文件。

2、不像 DBWR 可以有多个进程并行工作，LGWR 只有一个工作进程

3、触发条件：

①commit

②online redo switch

③3 秒

④三分之一满（或 1M 满）

⑤先于 dbwr 写（先记后写，必须在 dbwr 写脏块之前写入日志，保证未提交数据都能回滚）

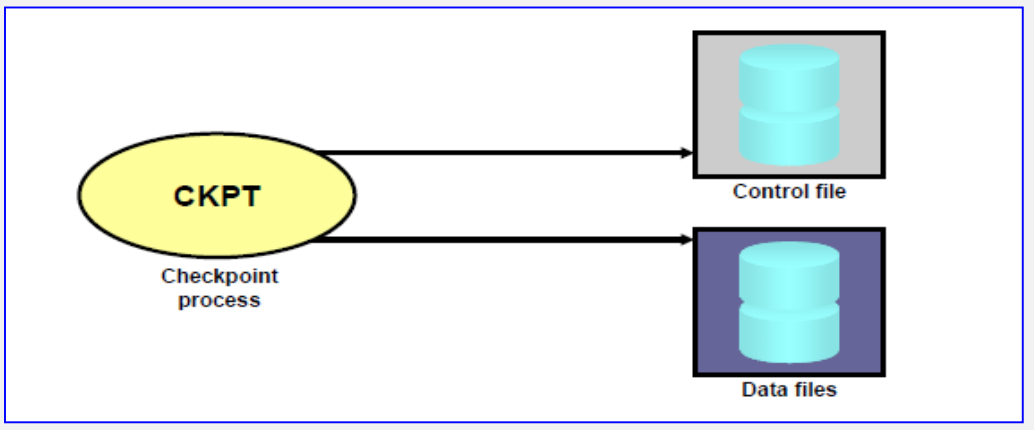
以上 5 个状况之一发生时， lgwr 都会记日志

fast commit 和 group commit

第一个事务 commit 时发生写日志的动作，在这个过程中如果又有其他事务也 commit，

则等到第一个事务写日志完成后，其他事务以 group commit 的方式一次性一起 commit 写日志.

5）ckpt：生成检查点



作用：通知或督促 dbwr 写脏块

1、完全检查点：保证数据库的一致性。

2、增量检查点：不断更新控制文件中的检查点位置，当发生实例崩溃时，可以尽量缩短实例恢复的时间。

3、局部检查点：特定的操作下，如针对某个表空间 read only/offline、Shrink 数据文件、ALTER TABLESPACE BEGIN BACKUP 等 。

6）arcn：归档当前日志

归档模式下，发生日志切换时，把当前日志组中的内容写入归档日志，作为历史日志提供数据库的 recovery

1,4 PGA 的基本组件

1）程序全局区(Program Global Area)的作用

①缓存来自服务器进程和后台进程的数据和控制信息。

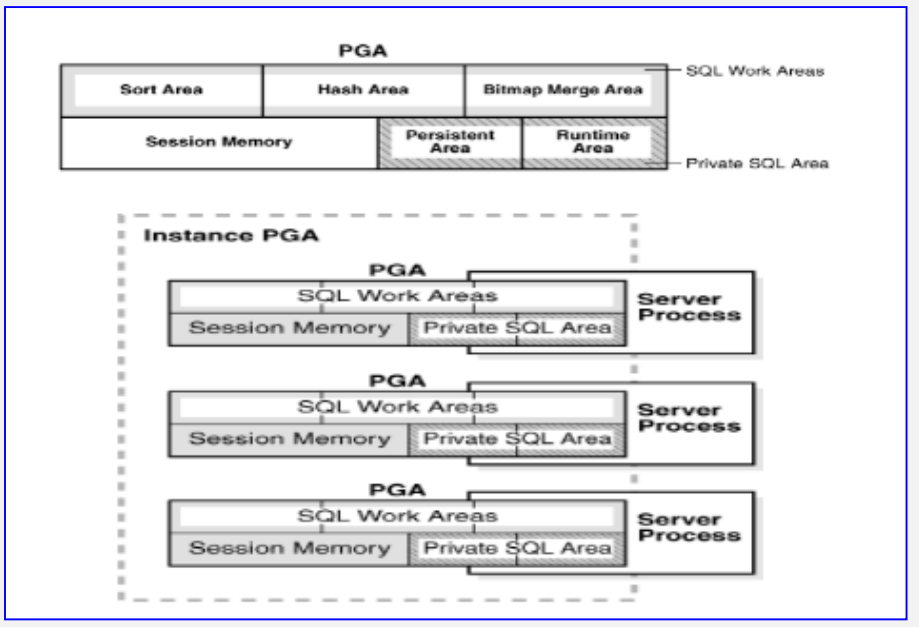
②提供排序、hash 连接

③不提供 session 之间的共享，

④PGA 在进程创建时被分配，进程终止时被释放。所有进程的 PGA 之和构成了 PGA 的大小。

PGA 的管理是比较复杂的，9i 后，Oracle 推荐使用 PGA 自动管理，屏蔽了 PGA 的复杂性。

2）PGA 的结构：



①SQL 工作区（SQL Work Area）：有几个子区 1、Sort Area, 2、Harh Area 3、Bitmap Merge Area

作用：排序操作(order by/group by/distinct/union 等)，多表 hash 连接，位图连接，创建位图

②会话空间（Session Memory）

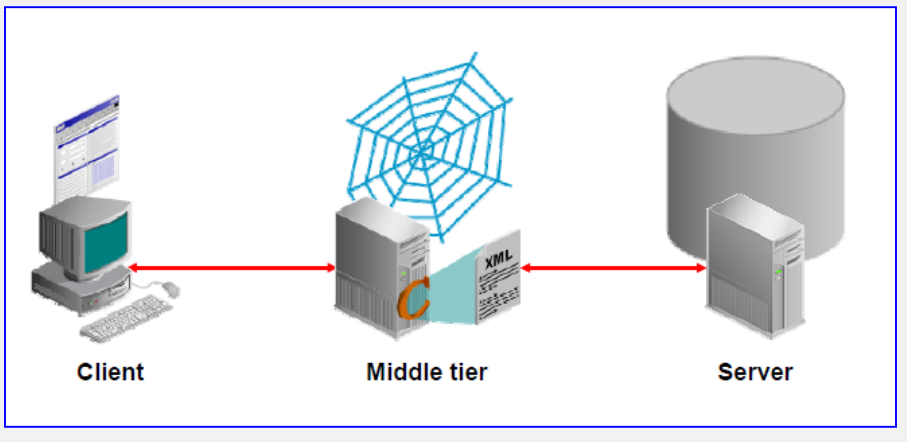
作用：存放 logon 信息等会话相关的控制信息

③私有 SQL 区域（Private SQL Area）

作用：存储 server process 执行 SQL 所需要的私有数据和控制结构，如绑定变量，它包括固定区域和运行时区域

④游标区域（Cursor Area）：sql 游标、PLSQL 游标使用的就是这块区域

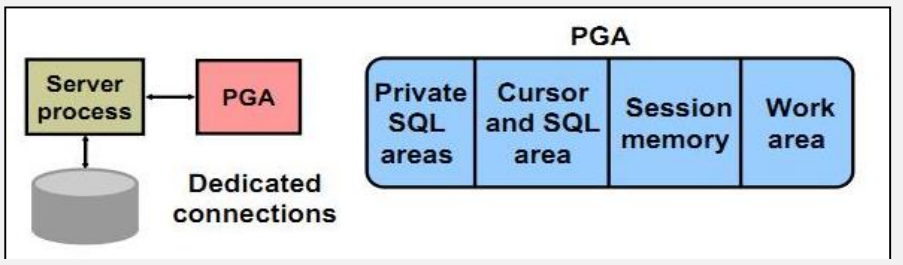
1.5 连接方式

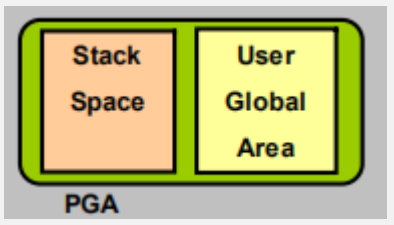


1）专用连接模式(dedicated)

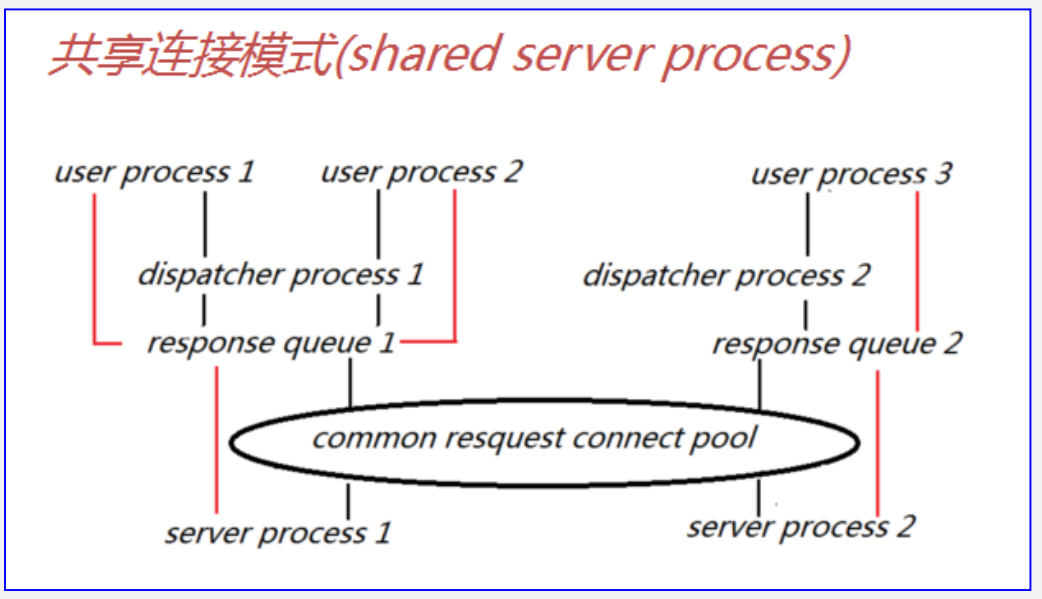
对于客户端的每个 user process，服务器端都会出现一个 server process，会话与专用服务器之间存在一对一的映射（一根绳上的两个蚂蚱）。

专用连接的 PGA 的管理方式是私有的。Oracle 缺省采用专用连接模式。





2）共享连接模式(shared)



多个 user process 共享一个 server process。

①,共享服务器实际上就是一种连接池机制（connectionpooling）,连接池可以重用已有的超时连接，服务于其它活动会话。但容易产生锁等待。此种连接方式现在已经很少见了(具体配置方法见第十四章 Oracle 网络）。

②所有调度进程(dispatcher)共享一个公共的请求队列(resquest queue)，但是每个调度进程都有与自己响应的队列(response queue)。

③在共享服务器中会话的（UGA）存储信息是在 SGA 中的，而不像专用连接那样在 PGA 中存储信息，这时的 PGA（非 UGA 部分））的存储结构为堆栈空间。

3）**驻留连接池模式**（database resident connection pooling，简称 DRCP）：

适用于必须维持数据库的永久连接。结合了专用服务器模式和共享服务器模式的特点。它使用连接代理（而不是专用服务器）连接客户机到数据库，优点是可以用很少的内存处理大量并发连接（11g 新特性，特别适用于 Apache 的 PHP 应用环境）。

第二章：实例管理及数据库的启动/关闭

2.1 参数文件

2. 1. 1 概念

1）instance 在启动阶段读取初始化参数文件（init parameter files)。该文件管理实例相关启动参数 。

基本初始化参数：大约 10-20 个左右（见联机文档）

初始化参数：300 个左右

隐含参数：Oracle 不推荐使用

2）两种形式

①动态参数：可以直接在内存中修改，并对当前 instance 立即生效

②静态参数。必须修改参数文件，下次启动后生效

SQL> select distinct issys_modifiable from v$parameter;

ISSYS_MODIFIABLE

IMMEDIATE 动态参数

FALSE 静态参数

DEFERRED 延迟参数，session 下次连接有效

3）会话级可修改和系统级可修改

1）alter session set

2）alter system set

2. 1. 2 两种参数文件

1）pfile（parameter file）

特点：

①必须通过文本编辑器修改参数，便于一次修改多个参数。

②缺省的路径及命名方式：$ORACLE_HOME/dbs/initSID.ora

2）spfile（system parameter file）

特点：

①二进制文件，不可以通过编辑器修改。通过 Linux 命令 strings 可以查看内容。

②路径及命名方式: $ORACLE_HOME/dbs/spfileSID.ora

③修改 spfile 文件的方法：

alter system set 参数=值 [scope=memory|spfile|both]

①scope=memory 参数修改立刻生效，但不修改 spfile 文件。

②scope=spfile 修改了 spfile 文件，重启后生效。

③scope=both (也可以省略)。

要更改静态参数，只能更改参数文件（pfile,spfile）,然后从新启动实例，才能使该参数生效

SQL> select name,ISSES_MODIFIABLE,ISSYS_MODIFIABLE from v$parameter where name='sql_trace';

NAME ISSES ISSYS_MOD

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

sql_trace TRUE IMMEDIATE

这个结果表示 sql_trace 参数在 session 级别可以改，在 system 级也可以 both 修改（动态参数）。

3）读取参数文件 pfile 和 spfile 相互生成

pfile 和 spfile 可以相互生成： 数据库即使不启动，pfile 和 spfile 也可以相互生成。

SQL>create pfile from spfile

SQL>create spfile from pfile

SQL>create pfile from memory;

SQL>create spfile from memory;

注意两点：

使用 spfile 启动后不能 create spfile from。

使用 pfile 启动后不能 alter system set 修改 spfile。

4）启动时默认优先 spfile，其次 pfile。

尽可能使用 spfile，pfile 一般留做备用，特殊情况也可以使用 pfile 启动:

SQL> startup pfile=$ORACLE_HOME/dbs/initprod.ora

如果 pfile 不是缺省命名或放在其他路径，则指定命令路径和文件名即可。

SQL> startup pfile=/home/oracle/mypfile

怎样知道实例是 spfile 启动还是 pfile 启动的

SQL> show parameter spfile

NAME TYPE VALUE

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

spfile string /u01/oracle/dbs/spfile.ora

如果 value 有值，说明数据库启动时读的是 spfile

另一个办法是看 v$spparameter（spfile 参数视图）中的参数 memory_target 的 isspecified 字段值，如果是 TRUE 说明是 spfile 启动的（考点）

SQL> select name,value,isspecified from v$spparameter where name like 'memory_target';

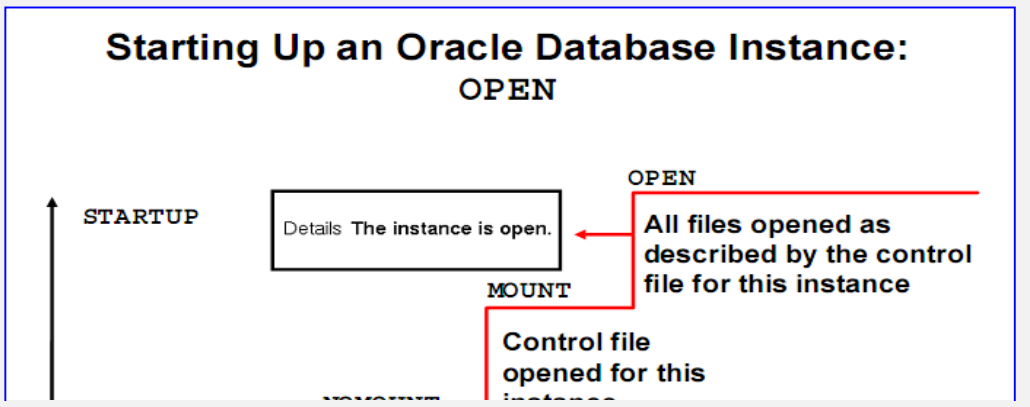
NAME VALUE ISSPECIFIED

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

memory_target 423624704 TRUE

• 2.2 数据库启动与关闭

2. 2. 1 实例启动分为三个阶段



1）nomount 阶段：读取 init parameter

SQL> select status from v$instance;

STATUS

STARTED

2）mount 阶段：读取控制文件

SQL> select status from v$instance;

STATUS

MOUNTED

3）open 阶段：

1、检查所有的 datafile、redo log、 group 、password file 正常

2、检查数据库的一致性（controlfile、datafile、redo file 的检查点是否一致）

SQL> select file#,checkpoint_change#,last_change# from v$datafile;　从控制文件读出

SQL> select file#,checkpoint_change# from v$datafile_header;　从数据文件读出

注意：启动时 last_change#不为空说明之前是干净的关闭数据库

SQL> select status from v$instance;

STATUS

OPEN

2. 2. 2 启动数据库时的一些特殊选项

startup force;　相当于 shutdown abort 后再接 startup

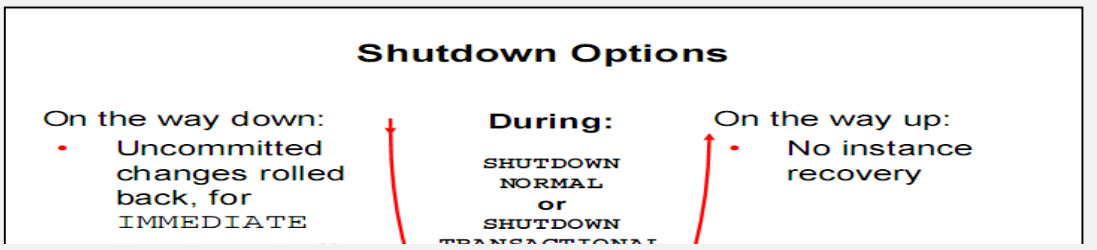
startup upgrade　只有 sysdba 能连接

startup restrict　有 restrict session 权限才可登录，sys 不受限制

alter system enable restricted session;　open 后再限制

alter database open read only;　scn 不会增长

2. 2. 3 实例关闭：



①shutdown normal　拒绝新的连接，等待当前会话结束，生成检查点

②shutdown transactional　拒绝新的连接，等待当前事务结束，生成检查点

③shutdown immediate　拒绝新的连接，未提交的事务回滚，生成检查点

④shutdown abort　不生成检查点，下次启动需要做 instance　recovery

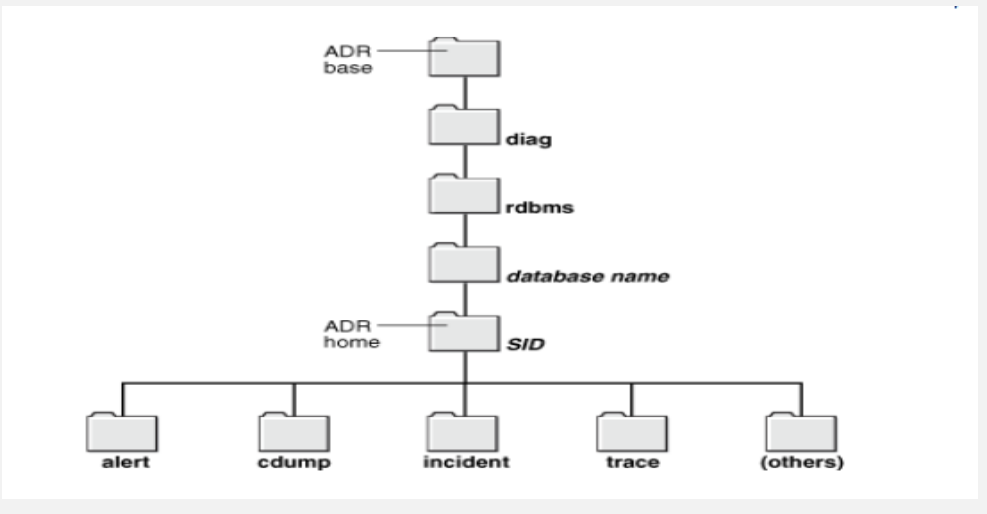
Shutdown Modes

Shutdown Modes	A	I	T	N
Allows new connections	No	No	No	No
Waits until current sessions end	No	No	No	Yes
Waits until current transactions end	No	No	Yes	Yes
Forces a checkpoint and closes files	No	Yes	Yes	Yes

2.3 自动诊断信息库 ADR

ADR（Automatic Diagnostic Repository）是 11g 新特性

存储在操作系统下的一个目录（树）结构



包括：①告警日志文件，②跟踪文件，③健康记录，④DUMP 转储文件等

SQL> show parameter diagnostic_dest

NAME　TYPE　VALUE

---

diagnostic_dest　string　/u01

$ORACLE_BASE=/u01，它也是 ADR 的基目录，如果你没有设置 ORACLE_BASE 环境变量，Oracle 给你设置的 ADR 基目录是

$ORACLE_HOME/log

SQL> show parameter dump

在 oracle 11g 中，故障诊断及跟踪的文件路径改变较大，告警文件分别 xml 的文件格式和普通文本格式两种形式存在。这两份文件的位置分别是 V$DIAG_INFO 中的 Diag Alert 和 Diag Trace 对应的目录。

2. 3. 1 跟踪文件：

1）后台进程的跟踪文件（Bdump）

SID_processname_PID.trc 如：_lgwr_5616.trc

2）服务器进程的跟踪文件（Udump）

SID_ora_PID.trc 如：_ora_10744.trc

另外增加.trm（trace map)文件，记录 trc 文件的结构信息。

SQL> select * from v$diag_info;

2. 3. 2 告警日志

以文本格式保存告警日志。命名是：alter_SID.log，它包含通知性的消息，如数据库启动或关闭，以及有关数据库物理结构变化的信息,也包括一些内部错误信息等。

告警日志会不断增长，定期清理是必要的

$cat /dev/null > alert_prod.log 将文件清空

直接删掉也没有关系，下次启动会自动创建

$ tail -f /u01/diag/rdbms/prod/prod/trace/alert_prod.log

2.4 口令文件

2. 4. 1 登录认证方式

主要有以下四种

1）OS 认证：

特点：①Oracle 用户以本地登录。②Oracle 用户必须属于 DBA 组，③Oracle 的 sysdba 身份不验证密码。

如：sqlplua / as sysdba

2）口令文件认证：

是一种通过网络的远程认证方式，只有 sysdba 权限的用户可以使用口令文件，登录时必须输入密码和网络连接符。

如：sqlplus sys/oracle@ as sysdba

3）口令密码认证方式

普通用户是指没有 sysdba 权限的用户，比如 system 、scott,或者是 tim 什么的，不管是本地还是远程，普通用户登录时必须在数据库 open 下输入口令，它们的口令密码保管在数据字典里。

4）外部密码认证

如果配置了 os_authent_prefix 参数，如缺省值为'ops$'，当数据库中存在用户'ops$tim'，且对该用户启用了外部验证。那么在操作系统上以 tim 用户登录成功后，就可以直接键入 sqlplus / ，登录用户是 ops$tim，密码由操作系统外部提供，不是数据字典认证。（见 SQL 课程用户访问控制章节）

2. 4. 2 口令文件

是 sysdba 身份的用户的远程认证密码文件，主要用于 sys 用户（严格来说是具有 sysdba 系统权限的用户）的远程登录认证。

1）位置：$ORACLE_HOME/dbs/orapwSID，

2）可以通过 remote_login_passwordfile 参数控制是否生效

3）使用 orapwd 命令创建新的 sys 口令文件：

[oracle@cuug ~]$ cd $ORACLE_HOME/dbs

[oracle@cuug dbs]$ $orapwd file=orapwprod password=system entries=5 force=y

file=orapw+sid 的写法，不能拼错 entries 的含义是表示口令文件中可包含的 SYSDBA/SYSOPER 权限登录的最大用户数。Force 强制覆盖原文件

第三章：控制文件

3.1 功能和特点

1）记录数据库当前物理状态

2）维护数据库的一致性

3）是一个二进制小文件

4）在 mount 阶段被读取

5）记录 RMAN 备份的元数据

查看 database 控制文件位置：

SQL> show parameter control_file

SQL> select name from v$controlfile;

NAME

/u01/oradata/prod/control01.ctl

/u01/oradata/prod/control02.ctl

/u01/oradata/prod/control03.ctl

3.2 实时更新机制

①当增加、重命名、删除一个数据文件或者一个联机日志文件时，Oracle 服务器进程（Server Process）会立即更新控制文件以反映数据库结构的变化。

②日志写进程 LGWR 负责把当前日志序列号记录到控制文件中。

③检查点进程 CKPT 负责把校验点的信息记录到控制文件中。

④归档进程 ARCn 负责把归档日志的信息记录到控制文件中。

通过视图 v$controlfile_record_section 可以了解到控制文件中记录了大量的数据库当前状态信息

3.3 多元化

1）配置多个 control_files，控制文件最好是 3 个（最多 8 个）。多路复用指的是相互镜像。

一般配置方法

①修改 spfile 中的 control_files 参数（修改之前最好将 spfile 先备份一份）

②shutdown immediate

③复制控制文件，Oracle 建议将多个控制文件分配在不同的物理磁盘上。

2）三个 control 文件要一致（同一版本，scn 相同），本来就是镜像关系

3）可以将控制文件 dump 出来，在跟踪文件中观察一下控制文件的内容。

```
alter session set events 'immediate trace name controlf level 12';

select * from v$diag_info;
```

3.4 备份与重建

3. 4. 1 备份

指对控制文件的实时备份，用于恢复数据文件。

```
SQL> alter database backup controlfile to '/u01/oradata/prod/con.bak';
```

数据库打开时是不能 cp 控制文件的。

3. 4. 2 重建

可以在 mount 或 open 模式生成一个 trace 文件，方便重建控制文件:

①文件内容在 Default Trace File 中

```
SQL> alter database backup controlfile to trace;
```

②或者 存到自己起的文件名里

```
SQL>alter database backup controlfile to trace as '/u01/oradata/prod/con.trace';
```

3.5 恢复与重建

3. 5. 1 恢复控制文件方法

控制文件一旦损坏，系统将不能正常工作。受损的控制文件会记录在告警日志中，恢复或重建控制文件必须使系统在 NOMOUNT 下

1）单个文件损坏了：参照多元化章节，通过简单复制解决。

2）所有的控制文件丢失：

①如果有 binary 控制文件备份，利用备份恢复控制文件，

②如果没有备份，利用 trace 脚本文件重新创建控制文件（代价：丢失归档记录信息和 RMAN 信息）

3. 5. 2 重建控制文件示例

第一步、Mount 或 open 下生成 trace 脚本

```
SQL>alter database backup controlfile to trace as '/u01/oradata/prod/con.trace';
```

第二步、启动到 nomount 状态下准备执行 trace 脚本

SQL> startup force nomount

第三步、执行重建控制文件语句

SQL>CREATE CONTROLFILE REUSE DATABASE "prod" NORESETLOGS ARCHIVELOG

MAXLOGFILES 16

MAXLOGMEMBERS 3

MAXDATAFILES 100

MAXINSTANCES 8

MAXLOGHISTORY 292

LOGFILE

GROUP 1 '/u01/oradata/prod/redo01.log' SIZE 50M,

GROUP 2 '/u01/oradata/prod/redo02.log' SIZE 50M,

GROUP 3 '/u01/oradata/prod/redo03.log' SIZE 50M

-- STANDBY LOGFILE

DATAFILE

'/u01/oradata/prod/system01.dbf',

'/u01/oradata/prod/sysaux01.dbf',

'/u01/oradata/prod/users01.dbf',

'/u01/oradata/prod/example01.dbf',

'/u01/oradata/prod/test01.dbf',

'/u01/oradata/prod/undotbs01.dbf'

CHARACTER SET ZHS16GBK

;

可以看到执行后三个控制文件又重新建立了。这时数据库已在 mount 下

说明：这个重建控制文件的过程主要有两大部分内容：

第一部分是脚本中的可见信息：

1）定义 db_name,

2）指定几个参数限定控制文件的最大值，

3）在线日志的物理信息，

4）数据文件的物理信息，

5）使用的字符集。

第二部分是隐含的不可见信息，比如 SCN 信息，重建复制了当前所有数据文件头部的最新 SCN 信息复制到了控制文件中。以便接下来打开数据库。

SQL> select file#,checkpoint_change# from v$datafile;

SQL> select file#,checkpoint_change# from v$datafile_header;

第四步、打开数据库

SQL> alter database open;

第五步、添加临时数据文件信息（脚本中的最后一行）

SQL> ALTER TABLESPACE TEMP ADD TEMPFILE '/u01/oradata/prod/temp01.dbf'

SIZE 30408704 REUSE AUTOEXTEND ON NEXT 655360 MAXSIZE 32767M;

第四章：redo 日志

4.1 作用和特征

作用：数据 recovery

特征：

1）记录数据库的变化（DML、DDL)

2）用于数据块的 recover

3）以组的方式管理 redo file，最少两组 redo，循环使用

4）和数据文件存放到不同的磁盘上，需读写速度快的磁盘(比如采用 RAID10)

5）日志的 block 和数据文件的 block 不是一回事

4.2 日志组及切换

1）最少两组，最好每组有两个成员（多路复用），并存放到不同的磁盘(不同路径)上，大小形同，互相镜像

2）日志在组写满时将自动切换

①归档模式：将历史日志连续的进行保存。

②非归档：历史日志被覆盖

③切换产生 checkpoint（考虑性能有延迟），通知 dbwn 写脏块 并且更新控制文件

3）在归档模式，日志进行归档，并把相关的信息写入 controlfile

4.3 添加日志组和成员

1）三个重要视图

SQL> select * from v$log;

```
SQL> select * from v$log;

    GROUP#    THREAD#  SEQUENCE#      BYTES    MEMBERS ARC STATUS           FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ------------- --------------------
         1          1         61   52428800          1 YES INACTIVE              4385173 2016-02-03 22:00:40
         2          1         62   52428800          1 NO  CURRENT               4420439 2016-02-03 23:22:38
         3          1         60   52428800          1 YES INACTIVE              4327551 2016-02-03 08:59:51
```

SQL> select * from v$logfile;

```
SQL> select * from v$logfile;

    GROUP# STATUS  TYPE    MEMBER                                          IS_
---------- ------- ------- ------------------------------------------------ ---
         3         ONLINE  /u01/oradata/prod/redo03.log                     NO
         2         ONLINE  /u01/oradata/prod/redo02.log                     NO
         1         ONLINE  /u01/oradata/prod/redo01.log                     NO
```

SQL> select name,sequence# from v$archived_log;

```
SQL> select name,sequence# from v$archived_log;

NAME                                     SEQUENCE#
---------------------------------------- ----------
/u01/arch/arch_1_884761251_7.log                  7
/u01/arch/arch_1_884761251_8.log                  8
/u01/arch/arch_1_884761251_9.log                  9
```

2）增加一个组 group4,

SQL> alter database add logfile group 4 '/u01/oradata/prod/redo04.log' size 50m;

```
SQL> select group#, member from v$logfile order by group#;

    GROUP# MEMBER
---------- ------------------------------------------------
         1 /u01/oradata/prod/redo01.log
         2 /u01/oradata/prod/redo02.log
         3 /u01/oradata/prod/redo03.log
         4 /u01/oradata/prod/redo04.log
```

3）添加日志组的成员

为每个组增加一个 member（一共是 4 个组）

先建好目录，准备放在/u01/disk2/prod/下

[oracle@cuug ~]$ mkdir -p /u01/disk2/prod

SQL>alter database add logfile member

'/u01/disk2/prod/redo01b.log' to group 1,

'/u01/disk2/prod/redo02b.log' to group 2,

'/u01/disk2/prod/redo03b.log' to group 3,

'/u01/disk2/prod/redo04b.log' to group 4;

```
SQL> select group#,member,status from v$logfile;

    GROUP# MEMBER                                          STATUS
---------- ------------------------------------------------ -------
         3 /u01/oradata/prod/redo03.log
         2 /u01/oradata/prod/redo02.log
         1 /u01/oradata/prod/redo01.log
         4 /u01/oradata/prod/redo04.log
         1 /u01/disk2/prod/redo01b.log                      INVALID
         2 /u01/disk2/prod/redo02b.log                      INVALID
         3 /u01/disk2/prod/redo03b.log                      INVALID
         4 /u01/disk2/prod/redo04b.log                      INVALID
```

STATUS 是 INVALID，说明 member 还没有同步好。

SQL> alter system switch logfile;　　至少做 4 次切换，消除 invalid。这步很必要。

4.4 v$log 视图的状态

STATUS 列有四种状态：

①unused:　新添加的日志组，还没有使用

②inactive　日志组对应的脏块已经从 data buffer 写入到 data file，可以覆盖

③active:　日志组对应的脏块还没有完全从 data buffer 写入到 data file，含有实例恢复需要的信息，不能被覆盖

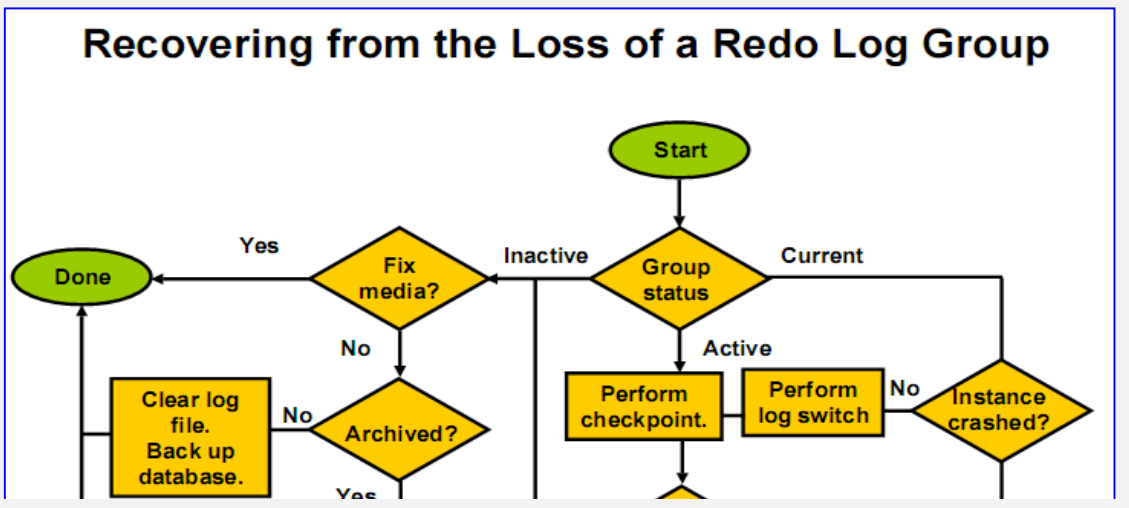
④current:　当前日志组对应的脏块还没有全部从 data buffer 写入到 data file，含有实例恢复需要的信息，不能被覆盖

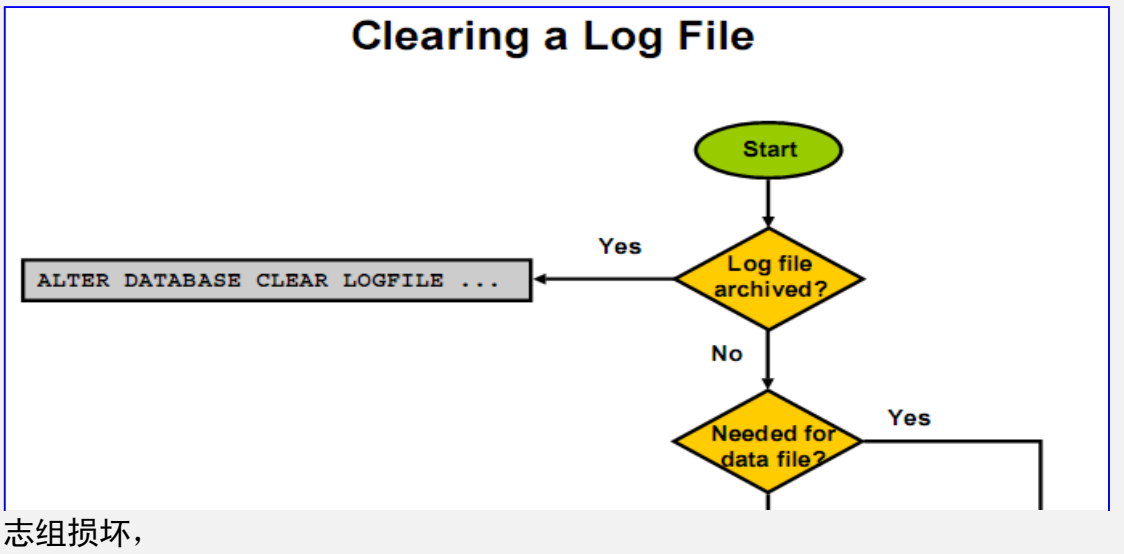
THREAD：　线程在单实例的环境下，thread#　永远是 1

SEQUENCE#　日志序列号。在日志切换时会递增

FIRST_CHANGE#　在每个日志组对应一个 sequence 号，其中首条日志条目的第一个的 scn。

4.5 日志恢复





4.5.1 inactive 日志组损坏，

假如日志组 4 损坏，状态 inactive。解决很简单，重建日志组即可

SQL> alter database clear logfile group 4;　这里 clear 的意思是重建 group4 的文件。

4.5.2 current 日志组丢失。

本例日志组 1 状态是 CURRENT 状态的，现在模拟当前日志组损坏

[oracle@prod]$ rm /u01/oradata/prod/redo01.log

[oracle@prod]$ rm /u01/disk2/prod/redo01b.log

SQL> alter system switch logfile;　　切换几次，触动它一下。

告警日志会记录有关信息

暂时好像没有什么问题发生，继续切换，当 current 又转会到 group1 时，session 死！

当前日志损坏的问题比较复杂，见上图可以分以下几种情况讨论

1）数据库没有崩溃

第一步，可以做一个完全检查点，将 db buffer 中的所有 dirty buffer 全部刷新到磁盘上。

SQL> alter system checkpoint;

第二步，尝试数据库在打开状态下进行不做归档的强制清除。

SQL> alter database clear unarchived logfile group n;

数据库此时为打开状态，这步若能成功，一定要做一个新的数据库全备，因为当前日志无法归档，归档日志 sequence 已无法保持连续性。全备的目的就是甩掉之前的归档日志。

2）数据库已经崩溃，只能做传统的基于日志的不完全恢复或使用闪回数据库。

SQL> recover database until cancel;

SQL> alter database open resetlogs;

3）如果之前没有可用的备份，或问题严重到任何方法都不能 resetlogs 打开数据库，为了抢救数据，考虑最后一招使用 Oracle 的隐含参数：_allow_resetlogs_corruption=TRUE

Oracle 不推荐使用这个隐含参数

该参数的含义是：允许数据库在不致性的情况下强制打开数据库。_

在不一致状态下强行打开了数据库后，建议做一个逻辑全备。

4.5.3 active 日志组损坏

做检查点切换，如成功，按照 inactive 损坏处理。否则，按 current 损坏处理。

4.6 使日志恢复到原来的配置

1）删除日志组

SQL> alter database drop logfile group 4;　　状态是 current 和 active 的组不能删除

2）删除成员

SQL> alter database drop logfile member '/u01/disk2/prod/redo01b.log';　　状态是 current 组不能删除成员，需要切换一下。

3）清理物理日志文件。　　删除物理文件需要 rm

第五章：归档日志

5.1 归档和非归档的区别

1）归档会在日志切换时，备份历史日志，对于 OLTP 系统都应考虑归档模式，以便数据库能支持热备，并提供数据库完全恢复和不完全恢复(基于时间点）

2）归档会启用 ARCn 的后台进程、也会占用磁盘空间

3）非归档适用某种静态库、测试库、或者可由远程提供数据恢复的数据库。非归档只能冷备，且仅能还原最后一次全备。

5.2 设置归档模式

1）SQL> shutdown immediate　　一定要干净的关闭数据库

2）SQL> startup mount　　启动到 mount 下

3）SQL> alter database archivelog;　　设置归档方式

4）SQL> archive log list;　　查看归档状态

5）SQL> alter database open;　　打开数据库

5.3 路径及命名方法

1）Oracle 判断归档目的地时按如下顺序优先择取

①log_archive_dest_n 值或 log_archive_dest 值

②db_recover_file_dest 参数指定的位置

③$ORACLE_HOME/dbs 位置

2）路径可以通过 archive log list 命令显示

```
SYS@ prod>archive log list;
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            /u01/arch
Oldest online log sequence     11
Next log sequence to archive   13
Current log sequence           13
SYS@ prod>
```

Archive destination（存档终点）有两种情形

①缺省是 USE_DB_RECOVERY_FILE_DEST，其含义是采用参数 db_recovery_file_dest 参数的定义，即闪回恢复区

②采用参数 log_archive_dest_n 或 log_archive_dest 指定的路径

SQL> show parameter archive

NAME	TYPE	VALUE
archive_lag_target	integer	0
log_archive_config	string	
log_archive_dest	string	
log_archive_dest_1	string	location=/u01/arch
log_archive_dest_10	string	
log_archive_dest_2	string	
log_archive_dest_3		

...

log_archive_duplex_dest string

log_archive_format string arch_%t_%r_%s.log

SQL> show parameter db_recovery

3）首先来看这两个参数：

log_archive_dest_n

log_archive_format

log_archive_dest_n（n:1-10） 表示可以有 10 个目标路径存放归档日志（镜像关系），即可以多路复用 10 个归档日志的备份。如上显示我只使用了 log_archive_dest_1，也就是说只有一套归档日志，没有做镜像。

参数设定的格式如下：

SQL> alter system set log_archive_dest_1='location=/u01/arch';

把历史日志归档到本机目录下 （location 代表本机）

SQL> alter system set log_archive_dest_2='service=standby';

远程备份，把历史日志备份到网络连接符为 standby 的数据库上。（service 代表远程），配置 DG 时有用。

log_archive_format 是定义命名格式的,一般考虑使用下面三个内置符号（模板）

%t thread# 日志线程号

%s sequence 日志序列号

%r resetlog 代表数据库的周期

参数设定的格式如下：

SQL> alter system set log_archive_format ='arch_%t_%r_%s.log' scope=spfile;

4）几个参数和选项的说明：

log_archive_dest

log_archive_duplex_dest

log_archive_dest_n

LOG_ARCHIVE_MIN-SUCCEED_DEST

log_archive_dest 和 log_archive_duplex_dest 这两个参数已经弃用了，它们能完成两路复用（镜像）但只能指定本机 location，无法指定远程。

一旦使用 log_archive_dest_n，log_archive_dest 参数就失效了。

LOG_ARCHIVE_MIN-SUCCEED_DEST 和 MANDATORY 选项

例：指定在线日志重用之前对归档满足要求

LOG_ARCHIVE_DEST _1 = ‘LOCATION=/disk1/arch MANDATORY’
LOG _ARCHIEVE_DEST_2 = ‘SERVICE=PROD'
LOG_ARCHIVK_DEST_3 = ‘LOCATION=/disk3/arch'
LOG_ARCHIVE_MIN-SUCCEED_DEST = 2

考点：dest1 必须成功，dest2-3 要有一个成功，如果没有 dest1 没有 MANDATORY 选项，dest1-3 任意两个成功

5.4 归档进程和手动切换

1）归档进程

$ ps -ef |grep ora_arc

oracle 1215 2435 0 13:26 pts/2 00:00:00 grep ora_arc

oracle 31796 1 0 13:00 ? 00:00:00 ora_arc0_

oracle 31798 1 0 13:00 ? 00:00:00 ora_arc1_

ARCn 就是归档进程，n 最多可达 30 个，由 log_archive_max_processes 参数指定。

2）手动切换日志：

第一种：SQL> alter system switch logfile; 仅切换当前实例，适用归档和非归档。

第二种：SQL> alter system archive log current; 在 RAC 下切换所有实例，仅适用于归档模式。

第六章：日志挖掘

6.1 作用

Oracle LogMiner 是一个非常有用的分析工具

可以轻松获得 Oracle 在线/归档日志文件中的具体内容，

可以解析出所有对于数据库操作的 DML 和 DDL 语句。

特别适用于调试、审计或者回退某个特定的事务。

由一组 PL/SQL 包和一些动态视图组成

一个完全免费的工具。没有提供任何的图形用户界面（GUI）

6.2 方法

6.2.1 DML 日志挖掘

1）添加 database 补充日志

SQL>ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;

注意：为了避免日志遗漏，这步要先执行。

2）确定要分析的日志范围，添加日志，分析

SQL>execute dbms_logmnr.add_logfile(logfilename=>'日志',options=>dbms_logmnr.new);

提供第一个要加载的日志文件

SQL>execute dbms_logmnr.add_logfile(logfilename=>'追加日志',options=>dbms_logmnr.addfile);

可以反复添加补充多个日志文件

3）执行 logmnr 分析

SQL>execute dbms_logmnr.start_logmnr(options=>dbms_logmnr.dict_from_online_catalog);

4）查询分析结果，

SQL>select username,scn,timestamp,sql_redo from v$logmnr_contents where seg_name='表名';

可以设置时间格式，也可以在显示方式里再确定格式.

5）关闭日志分析

SQL>execute dbms_logmnr.end_logmnr;

示例演示

6.2.2 DDL 日志挖掘

1）建立一个存放 dict.oral 的目录，设置参数 utl_file_dir 指定该目录，

$ mkdir /home/oracle/logmnr

SQL> alter system set utl_file_dir='/home/oracle/logmnr' scope=spfile;

2）建立数据字典文件 dict.ora

SQL> execute dbms_logmnr_d.build('dict.ora','/home/oracle/logmnr',dbms_logmnr_d.store_in_flat_file);

3）添加日志分析

SQL> execute dbms_logmnr.add_logfile(logfilename=>'日志文件',options=>dbms_logmnr.new);

SQL> execute dbms_logmnr.add_logfile(logfilename=>'追加日志',options=>dbms_logmnr.addfile);

4）执行分析

SQL>

execute dbms_logmnr.start_logmnr(dictfilename=>

'/home/oracle/logmnr/dict.ora',options=>dbms_logmnr.ddl_dict_tracking);

5）查看分析结果

SQL> select username,scn,to_char(timestamp,'yyyy-mm-dd hh24:mi:ss'),sql_redo from v$logmnr_contents WHERE USERNAME ='SCOTT' and lower(sql_redo) like '%table%';

6）关闭日志分析

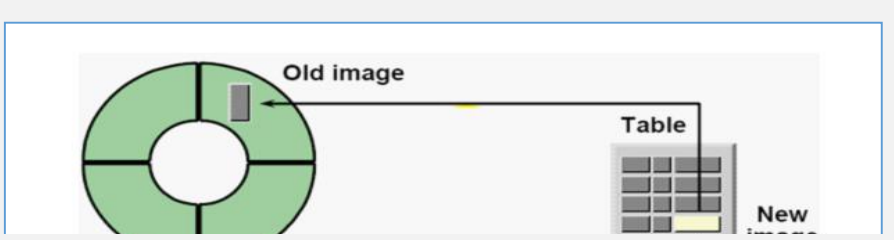
SQL> execute dbms_logmnr.end_logmnr;

示例将在备份恢复课程中演示。

第七章：管理 undo

7.1 Undo 作用

使用 undo tablespace 存放从数据缓存区读出的数据块的前镜像



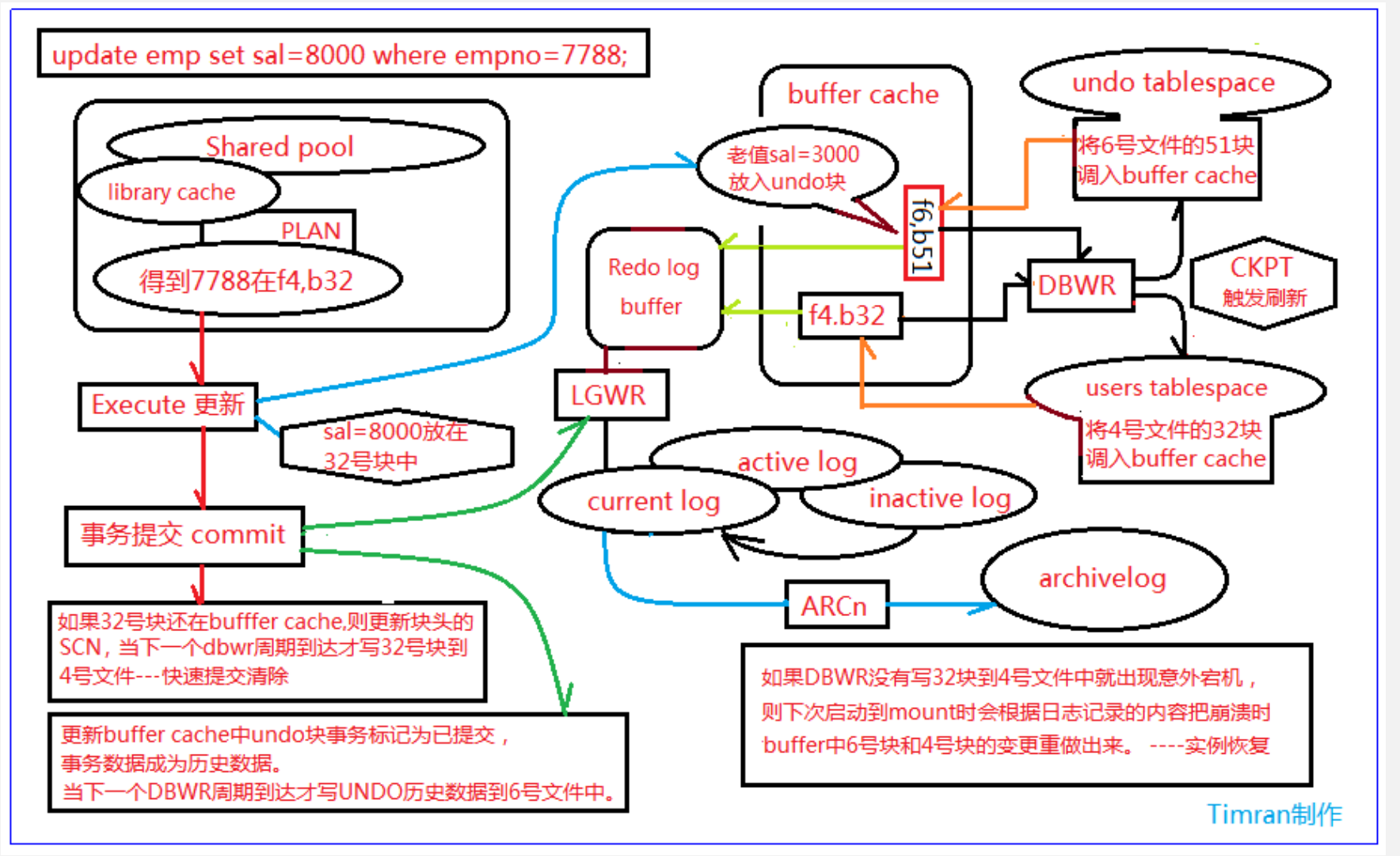
提供以下四种情况所需要的信息

1）回滚事务：rollback

2）读一致性：正在做 DML 操作的数据块，事务结束前，其他用户读 undo 里面的数据前镜像

3）实例的恢复：instance recover(undo -->rollback)

4）闪回技术 ：flashback query、flashback table 等



7.2 Undo 的参数

7.2.1 三个基本参数

SYS>show parameter undo

NAME	TYPE	VALUE
undo_management	string	AUTO
undo_retention	integer	900
undo_tablespace	string	UNDOTBS1

7.2.2 参数说明

1）undo_management

①manaual 仅在维护时使用

②auto 使用 undo tablespace 管理 undo

2）undo_retention 指定保留期，希望在这个期间 commit 的 undo 数据不要覆盖

3）undo_tablespace 当前使用的 Undo 表空间

7. 2. 3 建立一个 undo 表空间

可以建立多个 undo 表空间，但只有一个是使用中的 undo 使用中的 undo tablespace 不能 offline 和 drop

建立一个新的 undo 表空间

SQL>

create undo tablespace undotbs2 datafile '/u01/oradata/prod/undotbs02.dbf' size 50m autoextend on;

查看 undo tablespace

SQL> select tablespace_name,status,contents from dba_tablespaces;

查看当前正在使用的 undo tablespace 回滚段

SQL> select * from v$rollname;

```
       USN NAME
---------- ------------------------------
         0 SYSTEM
         1 _SYSSMU1_1363316212$
         2 _SYSSMU2_1363316212$
         3 _SYSSMU3_1363316212$
         4 _SYSSMU4_1363316212$
         5 _SYSSMU5_1363316212$
         6 _SYSSMU6_1363316212$
         7 _SYSSMU7_1363316212$
         8 _SYSSMU8_1363316212$
         9 _SYSSMU9_1363316212$
        10 _SYSSMU10_1363316212$
```

7. 2. 4 切换 undo

SQL> alter system set undo_tablespace=undotbs2;　　动态参数，修改立即生效

SQL> select * from v$rollname;

```
       USN NAME
---------- ------------------------------
         0 SYSTEM
        11 _SYSSMU11_1357956213$
        12 _SYSSMU12_1357956213$
        13 _SYSSMU13_1357956213$
        14 _SYSSMU14_1357956213$
        15 _SYSSMU15_1357956213$
        16 _SYSSMU16_1357956213$
        17 _SYSSMU17_1357956213$
        18 _SYSSMU18_1357956213$
```

19 _SYSSMU19_1357956213$

20 _SYSSMU20_1357956213$

SQL> drop tablespace undotbs1 including contents and datafiles;　　删除未激活 undo

7.3 Undo 空间重用机制

undo 数据的 4 种状态

①active:

表示 transaction 还没有 commit，不可覆盖，

②unexpired:

由已经 commit 或 rollback 的数据转换而来的状态，在保留期内，尽量不覆盖（非强制）

③expired:

由 unexpired 转换而来，其中的数据是超过 undo 保留期的，随时可以再分配（覆盖）。

④free:

分配了但未使用过。一般是 undo 最初创建和扩展时出现，它不是一个常态的。

undo 的状态转换体现了 undo 空间的循环使用原理：分配---》冻结--->回收---》再分配

7.4 关于 AUM

1）什么是 AUM

Oracle10gR2 以后引入了一个新的自动调整 undo retention 的特性，

目的是尽量避免两个 Undo 错误

ora-30036 错误---空间不足

ora-01555 错误---快照太旧

11g 缺省设置为 AUM(Auto Undo Management)

AUM 下 current undo retention 是自动调整的

SYS@ prod>select begin_time,tuned_undoretention from v$undostat;

BEGIN_TIME　　TUNED_UNDORETENTION

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

2015-10-11 20:17:20　　1784

2015-10-11 20:07:20　　1723

2015-10-11 19:57:20　　1119

2）AUM 的两种工作方式

①autoextend off 下，忽略 undo_retention 参数，TUNED_UNDORETENTION 参照 undo 表空间大小和 undo 统计信息，

缺点：空间给定不合理时，产生 UNDO 告警，不能完全避免 ora-30036，ora-01555，若 UNDO 增加尺寸，又可能造成 TUNED_UNDORETENTION 增加。

②autoextend on 下，参考 undo_retention 作为下限值，TUNED_UNDORETENTION 期内，以扩展空间代替覆盖 unexpired，基本避

免了 ora-30036，ora-01555

缺点：表空间可能过度膨胀。

3）关闭 AUT 模式(Oracle 不推荐)

如果要关闭 undo 自动调优，可以使用隐含参数

设置隐含参数_undo_autotune=false

4）Undo 的 guarantee 属性

通常情况下，unexpired 数据并不绝对保证在 retention 期内不会覆盖，必要时可考虑设置在保留期强制不覆盖的 guarantee 属性,同时应该使 undo autoextend on

SQL> select tablespace_name,status,contents,retention from dba_tablespaces;

缺省配置下 undo retention 是 noguarantee

guarantee 属性可以修改

SQL> alter tablespace undotbs2 retention guarantee;　保证在 retention 期间不允许被覆盖

SQL> alter tablespace undotbs2 retention noguarantee;

7.5 undo 信息的查询

1） v$session　查看用户建立的 session

2） v$transaction　当前的事务

3） v$rollname　undo 段的名称

4） v$rollstat　undo 段的状态

5） v$undostat　查看每 10 分钟的统计数据

6） dba_undo_extents　查看 undo 段中不同状态的空间占用

7） dba_rollback_segs　数据字典里记录的 undo 段状态

第八章：检查点（checkpoint)

8.1 什么是 checkpoint

checkpoint 是数据库的一个内部事件，检查点激活时会触发数据库写进程(DBWR)，将数据缓冲区里的脏数据块写到数据文件中。其作用有两个方面：

1）保证数据库的一致性，这是指将脏数据从数据缓冲区写出到硬盘上，从而保证内存和硬盘上的数据是一致的。

2）缩短实例恢复的时间，实例恢复时，要把实例异常关闭前没有写到硬盘的脏数据通过日志进行恢复。如果脏块过多，实例恢复的时间也会过长，检查点的发生可以减少脏块的数量，从而减少实例恢复的时间。

8.2 检查点分类

①完全检查点 database checkpoint

②增量检查点 incremental checkpoint

③局部检查点 partial checkpoint

8. 2. 1 完全检查点工作方式：

记下当前的 scn，将此 scn 之前所有的脏块一次性写完，再将该 scn 号同步更新控制文件和数据文件头。

触发完全检查点的四个操作

①正常关闭数据库:shutdown immediate

②手动检查点切换:alter system checkpoint;

③日志切换：alter system switch logfile;

④数据库热备模式：alter database begin backup;

示例 1：

验证以上概念可以做一下 alter system checkpoint，然后观察 v$datafile 和 v$datafile_header 中 scn 被更新。

示例 2 研究一下日志切换:alter system switch logfile;

设置 FAST_START_MTTR_TARGET<>0，查看 v$log 视图中的 active 状态几分钟后会变成 inactive 状态，说明了什么？确认该操作也更新了控制文件和日志文件头部的 SCN。

8. 2. 2 增量检查点概念及相关参数：

8.2.2.1 概念：

1）被修改过的块，在 oracle 中都被统称为脏块.脏块按照首次变脏的时间顺序被一个双向链表指针串联起来,这称做检查点队列。

2）当增量检查点发生时，DBWR 就会被触发,沿着检查点队列的顺序将部分脏块刷新到磁盘上，每次刷新截止的那个块的位置就叫检查点位置，检查点位置之前的块,都是已经刷新到磁盘上的块。而检查点位置对应的日志地址（RBA）又总是被记录在控制文件中。如果发生系统崩溃，这个最后的检查点位置就是实例恢复运用日志的起点。

3）增量检查点使检查点位置前移。进而缩短实例恢复需要的日志起点和终点之间的距离，触发增量检查点越频繁，实例恢复的时间越短，但数据库性能受到频繁 IO 影响会降低。

4）增量检查点不会同步更新数据文件头和控制文件的 SCN。

8.2.2.2 与增量检查点有关的几个知识点

1）FAST_START_MTTR_TARGET 参数：

这个参数是考点。它给出了你希望多长时间完成恢复实例。

此参数单位为秒，缺省值 0，范围 0-3600 秒，根据这个参数，Oracle 计算出在内存中累积的 dirty buffer 所需的日志恢复时间，如果日志累计到达一定量，则增量检查点进程被触发。该参数如果为 0，ORACLE 则会根据 DBWN 进程自身需要尽量减少写入量，这样虽然实现了性能最大化，但实例恢复时间可能会比较长。

早期还有几个有关增量检查点的参数，

①log_checkpoint_interval

规定了 redo 日志积累多少 block 后激活增量检查点，对用户来讲要给出这个参数不太方便，所谓 block 指的是 os block，而不是 oracle block。

②log_checkpoint_timeout 给一个触发增量检查点的间隔，单位是秒。

如果设置了 FAST_START_MTTR_TARGET 参数，就不要用早期的一些参数，容易引起冲突。

如果将 fast_start_mttr_target 设置为非 0，1）将启用检查点自动调整机制。2）log_checkpoint_interval 参数将被覆盖掉。

2）90% OF SMALLEST REDO LOG（内部机制）

从上次切换后算起，累计日志为一个日志组大小的 90%时，做一次检查点切换。

3）每 3s 查看 checkpoint 队列脏块的写出进度

这个 3 秒是 Oracle 强调的增量检查点特性之一。注意，3s 并不触发检查点，它只是记录当时的检查点位置，并将相关信息写入到 controlfile。

8. 2. 3 实例恢复有关的顾问

使能实例恢复顾问（MTTR Advisory）。需要设置两个参数

1）STATISTICS_LEVEL　置为 typical（缺省）或者 all

2）FAST_START_MTTR_TARGET　置为非零值

为了合理的设置 MTTR 参数，可以参考视图 v$instance_recovery 的估算值

```
SQL>select recovery_estimated_ios,actual_redo_blks,target_redo_blks,target_mttr,estimated_mttr from v$instance_recovery;
```

8. 2. 4 局部检查点

对于仅限于局部影响的操作，可以触发局部检查点。

比如：表空间 offline,数据文件 offline,删除 extent,表 truncate,alter tablespace begin backup 等。

第九章：实例恢复机制

9.1 什么是实例恢复

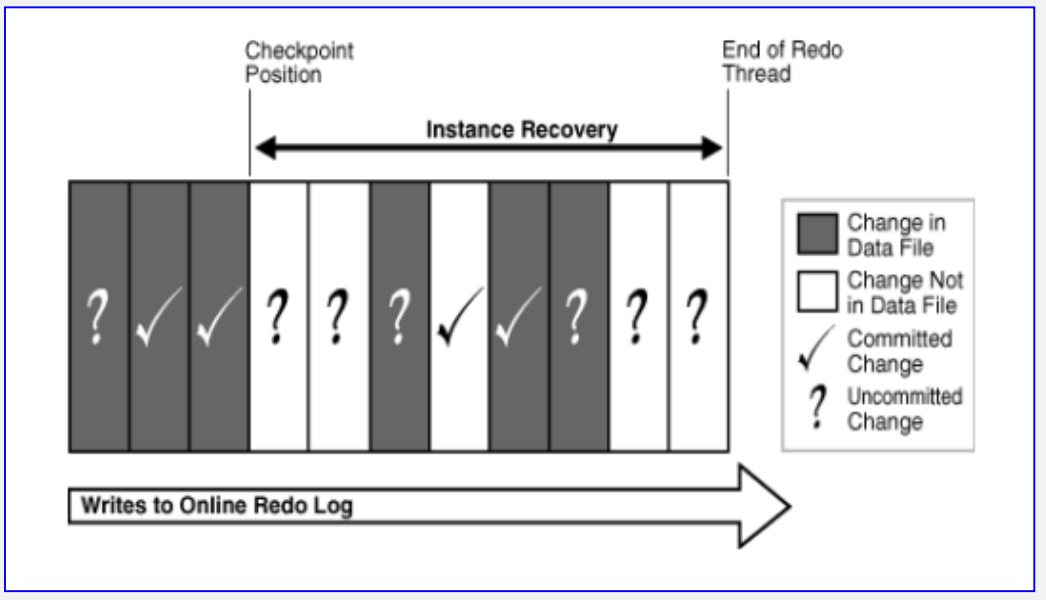
有了 redo,undo 和增量 ckpt 概念后，有助于更好的理解实例恢复

1）背景：当实例崩溃发生时，内存数据丢失，而当时的内存中 db buffer 和磁盘上的 datafile 内容不一致。

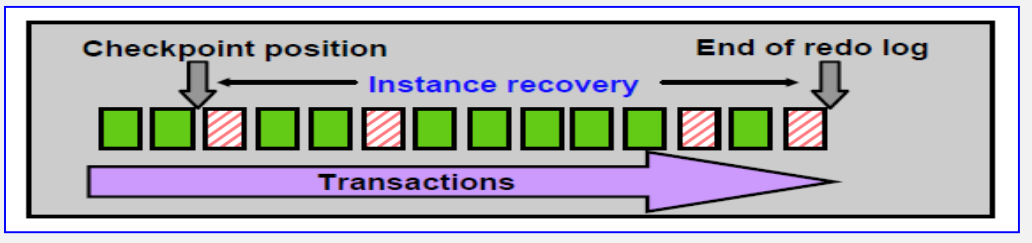
2）要解决两个问题：

①重新构成崩溃时内存中还没有保存到磁盘的已 commit 的变更块。

②回滚掉已被写至数据文件的 uncommit 的变更块。



9.2 增量检查点发挥的作用



每当增量检查点触发时，一部分 dirty buffer 被刷新到磁盘，并记录了最后一次检查点位置。当实例恢复时，Oracle 首先从控制文件里找到最后一次检查点位置，这个位置其实就是实例恢复时运用日志的起点（RBA）。然后是 smon 监控下的一系列动作：

1）roll forward

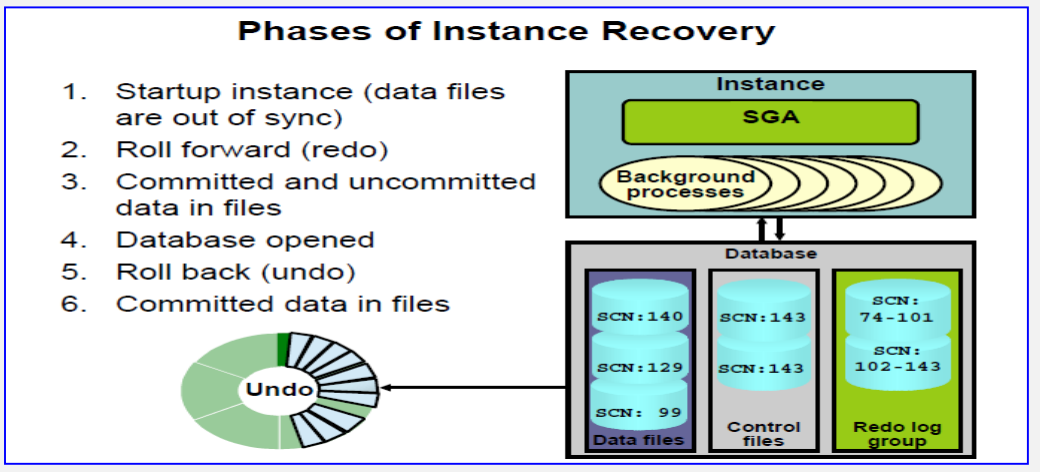
利用 redo，将检查点位置之后的变更，包括 commit 和 uncommit 的都前滚出来了，然后统统写到磁盘(datafile)里。

2）open

用户可以连接进来，访问数据库。

3）roll back

回滚掉数据文件中未提交的数据



第十章：手工创建数据库

要求：参考联机文档，建立一个数据库，名称：emrep

第一步，编辑.bash_profile 环境变量

$cp .bash_profile .bash_profile.prod

$vi .bash_profile

```
ORACLE_BASE=/u01
ORACLE_HOME=$ORACLE_BASE/oracle
ORACLE_SID=emrep
PATH=$ORACLE_HOME/bin:$PATH
export ORACLE_BASE ORACLE_HOME ORACLE_SID PATH
```

第二步，建立必要的目录

$mkdir -p /u01/admin/emrep/adump

$mkdir -p /u01/oradata/emrep

第三步，编辑 pfile，指定基本参数

$vi $ORACLE_HOME/dbs/initemrep.ora

```
db_name='EMREP'
memory_target=500m
control_files = ('/u01/oradata/emrep/control01.ctl','/u01/oradata/emrep/control02.ctl')
audit_file_dest='/u01/admin/emrep/adump'
db_block_size=8192
```

diagnostic_dest='/u01'

undo_tablespace='UNDOTBS1'

第四步， 建立口令文件、spfile

$orapwd file=$ORACLE_HOME/dbs/orapwemrep password=oracle entries=5

SQL> startup nomount pfile=/u01/oracle/dbs/initemrep.ora

SQL> create spfile from pfile;

SQL> startup force nomount;

第五步， 建立数据库

```
SQL> CREATE DATABASE emrep
    USER SYS IDENTIFIED BY oracle
    USER SYSTEM IDENTIFIED BY oracle
    LOGFILE GROUP 1 '/u01/oradata/emrep/redo01.log' SIZE 30M,
                GROUP 2    '/u01/oradata/emrep/redo02.log' SIZE 30M,
                GROUP 3    '/u01/oradata/emrep/redo03.log' SIZE 30M
    MAXLOGFILES 5
    MAXLOGMEMBERS 5
    MAXLOGHISTORY 1
    MAXDATAFILES 100
    CHARACTER SET AL32UTF8
    NATIONAL CHARACTER SET AL16UTF16
    EXTENT MANAGEMENT LOCAL
    DATAFILE '/u01/oradata/emrep/system01.dbf' SIZE 325M REUSE
    SYSAUX DATAFILE '/u01/oradata/emrep/sysaux01.dbf' SIZE 325M REUSE
    DEFAULT TABLESPACE users
        DATAFILE '/u01/oradata/emrep/users01.dbf'
        SIZE 100M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED
    DEFAULT TEMPORARY TABLESPACE temp1
        TEMPFILE '/u01/oradata/emrep/temp01.dbf'
        SIZE 20M REUSE
    UNDO TABLESPACE undotbs1
        DATAFILE '/u01/oradata/emrep/undotbs01.dbf'
        SIZE 100M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

第六步，运行脚本，生成必要的数据字典和 PLSQL 包

[oracle@cuug emrep]$ sqlplus / as sysdba

SQL>startup

生成必要的数据字典

SQL> @?/rdbms/admin/catalog.sql

生成必要 PLSQL 包

SQL> @?/rdbms/admin/catproc.sql

SQL>conn system/oracle

必要的授权

SQL> @?/sqlplus/admin/pupbld.sql

至此，手工创建一个基本的数据库完毕

第七步，设定归档模式（可选）

SQL> archive log list;

数据库日志模式	非存档模式
自动存档	禁用
存档终点	/u01/oracle/dbs/arch
最早的联机日志序列	9
当前日志序列	11

$mkdir -p /u01/arch2

SQL> create pfile from spfile;

SQL> alter system set log_archive_dest_1='location=/u01/arch2';

SQL> alter system set log_archive_format='arch_%t_%r_%s.log' scope=spfile;

SQL> shutdown immedaite;

SQL> startup mount

SQL> alter database archivelog;

SQL> alter database open;

检查 emrep 数据库工作正常

SQL>alter system switch logfile;

第八步，删除 emrep（可选）

使用 dbca，为了让 dbca 识别出 emrep 数据库，需要先编辑 oratab 配置文件

$vi /etc/oratab

最后一行添加如下信息

#

prod:/u01/oracle:N

emrep:/u01/oracle:N

注：dbca 仅仅是删除 emrep 数据库，可能没有清除 emrep 相关的目录，需要再手工清理一下。

第二部分 存储架构

第十一章：数据字典

11.1 什么是数据字典

1）Oracle 提供了大量的内部表，它们记录了数据库对象信息。可以将这些内部表划分为两种主要类型：静态的数据字典表和动态的性能表。这些内部表是由 oracle 维护的，它们都是只读表。用户包括 sys 都不能修改，只能查看。

2）Oracle 数据库字典通常是在创建和安装数据库时被创建的

11.2 数据字典内容

1）数据库中所有模式对象的信息，如表、视图、簇、及索引等。
2）分配多少空间，当前使用了多少空间等。
3）列的缺省值。
4）约束信息的完整性。
5）Oracle 用户的名字。
6）用户及角色被授予的权限。
7）用户访问或使用的审计信息。
8）其它产生的数据库信息

11.3 数据字典组成

1）**数据字典表**：是 Oracle 存放系统数据的表。这些表属于 SYS 用户。用以存储表、索引、约束以及其他数据库结构信息，通常以$结尾，如 tab$,obj$,ts$,aud$等。

2）**内部表（X$）**：Oracle 的核心，官网不做说明, Oracle 通过大量 X$建立起大量视图，仅供用户 select

3）**数据字典视图**：数据字典表上创建，通常分为三类 dba_, all_, user_

4）**动态性能视图**（V$）：实时更新反应当前实例的状态，官网对 V$视图有详尽的说明。

实际工作中最常用的是数据字典视图和动态性能视图：

广义概念中：v$也属于数据字典范畴。因为 v$的结构也是在创建数据库的时候通过执行脚本完成的。与数据字典视图不同的是：v$数据源不是来自 system 表空间，而是来自内存或控制文件，它在实例启动后被填充，在实例关闭后被清除。

11.4 查询静态和动态视图

1）DICT 表记录了所有静态视图和动态视图的名称

SQL> select * from dict where table_name='DBA_OBJECTS';

SQL> select count(*) from dict;

2 ）静态数据字典视图： （dba_）

在数据库 open 状态下访问，可以通过静态视图了解 database 的架构(记录 database 的架构

dba_:存储所有用户对象的信息（默认只能有 sys/system 用户访问）

all_:存储当前用户能够访问的对象（包括用户所拥有的对象和别的用户授权访问的对象）的信息。

user_：存储当前用户所拥有的对象的相关信息。

3）动态性能视图（V$）

是维护当前实例信息的，由于不断的更新，所以也叫动态视图。其底层是一组虚拟的动态表称为 X$表，Oracle 不允许直接访问 X$表，而是在这些表上创建视图，然后再创建这些视图的同义词。

基表（x$)----------视图（v_$)----------同义词 v$-------用户访问

可以通过 v$fixed_table 视图 查到所有的动态视图的名称；用于调优和数据库监控。

SQL> select count(*) from v$fixed_table;

从 Oracle8 开始， GV$视图开始被引入，其含义为 Global V$,GV$的产生是为了满足 OPS 环境的需要，除了一些特例以外，每个 V$视图都有一个 GV$视图存在。

4）如何让普通用户访问 dba_xxx 和 v$的三种授权方法

第一种

SQL>grant select on dba_objects to scott; 将对象权限授予 scott

SQL>grant select on v_$log to scott; 授予 Scott 用户 v_$对象权限

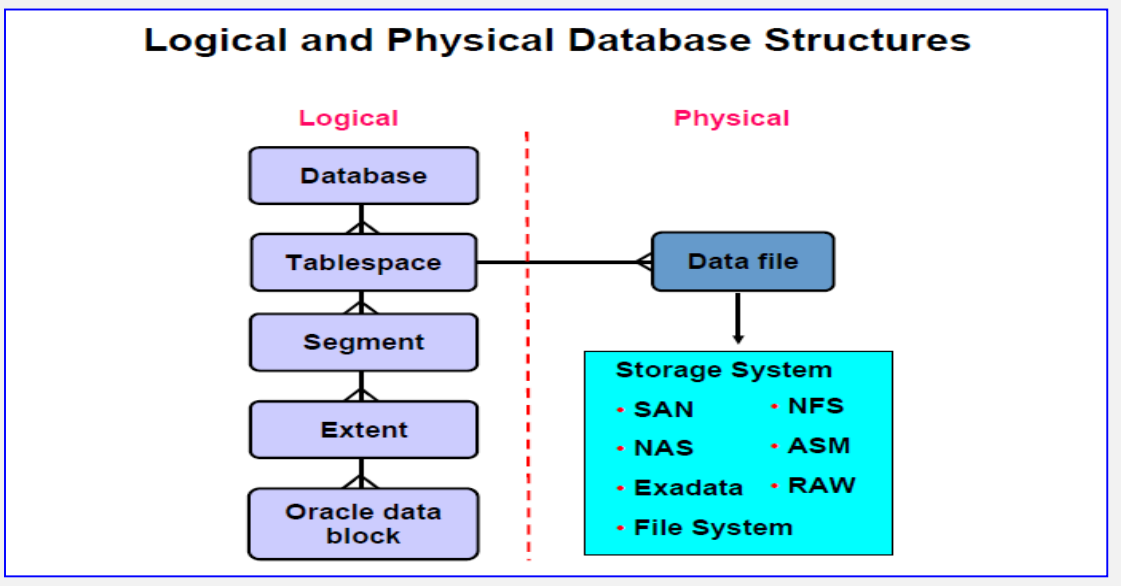
第二种 SQL>grant select any dictionary to scott; 将系统权限授权给 scott

第三种

SQL>grant select any table to scott;

SQL>alter system set O7_DICTIONARY_ACCESSIBILITY=true scope=spfile;

第十二章：逻辑存储架构



12.1 TABLESPACE(表空间）

12. 1. 1 类型

①PERMANENT 永久表空间

②UNDO 撤销表空间

③TEMPORARY　临时表空间

12. 1. 2 **管理方式：**

重点是段的管理方式和区的管理方式是在建立表空间时确定的。

段管理方式有 AUTO 和 MANUAL 两种，区管理方式有本地管理和字典管理（已淘汰）两种。

SQL> select tablespace_name,contents ,extent_management,segment_space_management from dba_tablespaces;

TABLESPACE_NAME	CONTENTS	EXTENT_MAN	SEGMEN
SYSTEM	PERMANENT	DICTIONARY	MANUAL
SYSAUX	PERMANENT	LOCAL	AUTO
TEMP	TEMPORARY	LOCAL	MANUAL
USERS	PERMANENT	LOCAL	AUTO
EXAMPLE	PERMANENT	LOCAL	AUTO
UNDO_TBS01	UNDO	LOCAL	MANUAL
TMP01	TEMPORARY	LOCAL	MANUAL
TEST	PERMANENT	DICTIONARY	MANUAL

注意两点：

1）如果 system 表空间是数据字典管理，其他表空间可以是数据字典管理或 local 管理（默认）

2）字典管理可以转换成本地管理，但是对于系统表空间，要求执行一些附加步骤，比较麻烦。

SQL>execute dbms_space_admin.tablespace_migragte_to_local('tablespacename');

12. 1. 3 基本操作

1）建立表空间

SQL> create tablespace a datafile '/u01/oradata/prod/a01.dbf' size 10m;

利用 oracle 提供的 dbms_metadata.get_ddl 包看看缺省值都给的是什么？

SQL> set serverout on;

SQL>

```
declare
aa    varchar2(2000);
begin
select dbms_metadata.get_ddl('TABLESPACE','A') into aa FROM dual;
dbms_output.put_line(aa);
end;
/
```

结果：

```
  CREATE TABLESPACE "A" DATAFILE
  '/u01/oradata/prod/a01.dbf' SIZE 10485760
```

LOGGING ONLINE PERMANENT BLOCKSIZE

8192

EXTENT MANAGEMENT LOCAL AUTOALLOCATE

SEGMENT SPACE MANAGEMENT AUTO

PL/SQL 过程已成功完成。

关注最后一行，两个重要信息是:(1)区本地管理且自动分配空间，(2)段自动管理。

dbms_metadata.get_ddl 也可以查看表，('TABLE','EMP','SCOTT')替换('TABLESPACE','B')试试。

SQL>

create tablespace b datafile '/u01/oradata/prod/b01.dbf' size 10m

extent management local uniform size 128k

segment space management manual

同上，调 dbms_metadata.get_ddl 包看 Oracle 对该语句的 ddl 操作是：

CREATE TABLESPACE "B" DATAFILE

'/u01/oradata/prod/a01.dbf' SIZE 10485760

LOGGING ONLINE PERMANENT BLOCKSIZE

8192

EXTENT MANAGEMENT LOCAL UNIFORM SIZE 131072 SEGMENT SPACE MANAGEMENT MANUAL、

最后一行信息是：区本地管理且统一分配 128K，段手动管理。如果在建表时使用缺省说明，则该表将服从其表空间的这些定义，

2） 删除表空间

1、表空间的删除和 offline

SQL>drop tablespace test including contents and datafiles;

contents 包括控制文件和数据字典信息，datafiles 是物理数据文件。

数据库 OPEN 下不能删除的表空间是

①system ②active undo tablespace ③default temporary tablespace ④default tablespace

数据库 OPEN 下不能 offine 的表空间是

①system ②active undo tablespace ③default temporary tablespace

3）查看表空间大小

SQL> select TABLESPACE_NAME,sum(bytes)/1024/1024 from dba_data_files group by tablespace_name;

4）查看表空间空闲大小

SQL> select TABLESPACE_NAME,sum(bytes)/1024/1024 from dba_free_space group by tablespace_name;

TABLESPACE_NAME	SUM(BYTES)/1024/1024
UNDOTBS1	98.4375
SYSAUX	14.625
USERS	48.1875

SYSTEM 1.875

EXAMPLE 31.25

5）查看表空间（数据文件）是否自动扩展

SQL> col file_name for a40;

SQL> select file_name,tablespace_name,bytes/1024/1024 mb,autoextensible from dba_data_files;

6）建立大文件（bigfile）的表空间

①small file，在一个表空间可以建立多个数据文件（默认）

②bigfile ：在一个表空间只能建立一个数据文件 （最大可达 32T），简化对数据文件管理。

SQL> create bigfile tablespace big_tbs datafile '/u01/oradata/prod/bigtbs01.dbf' size 100m;

试图在该表空间下增加一个数据文件会报错

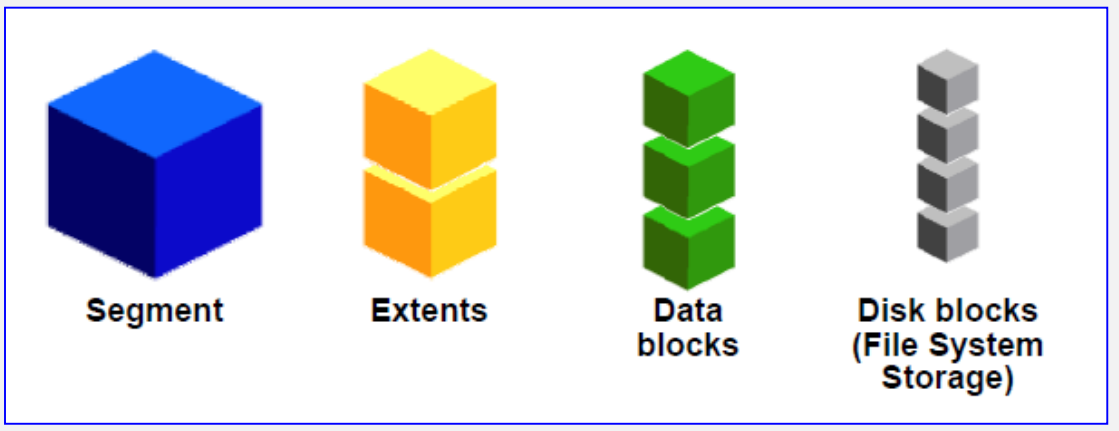
SQL> alter tablespace big_tbs add datafile '/u01/oradata/prod/bigtbs02.dbf' size 100m;

报错：ORA-32771: cannot add file to bigfile tablespace

查看大文件表空间：

SQL> select name,bigfile from v$tablespace;

12.2 SEGMENT(段）



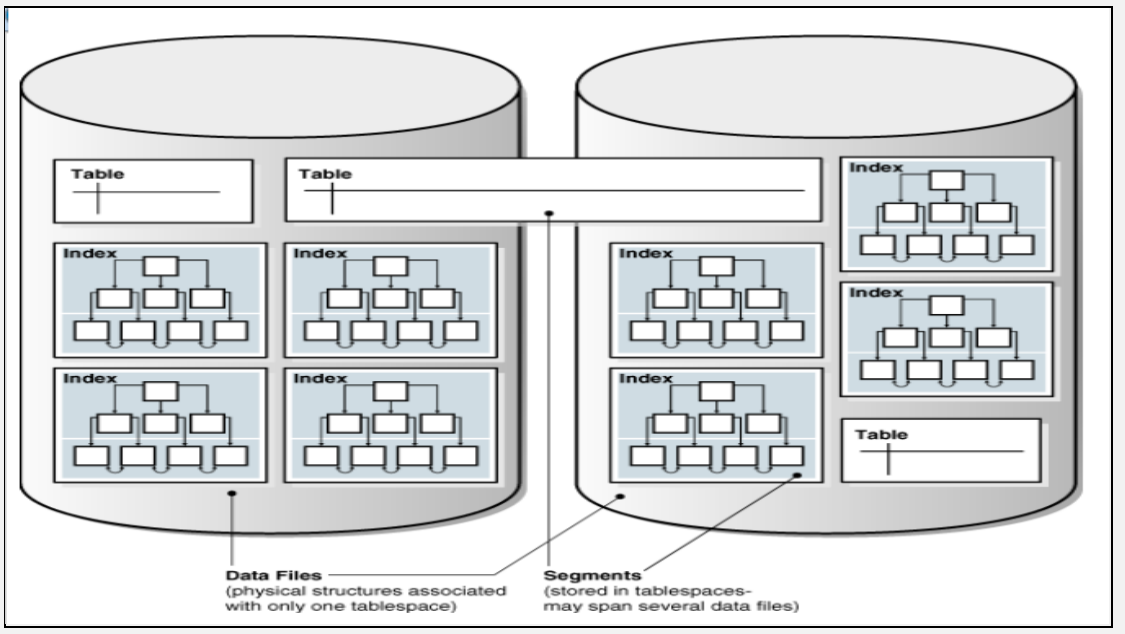
12. 2. 1 特点：

1）表空间在逻辑上可以对应多个段，物理上可以对应多个数据文件，一个段比较大时可以跨多个数据文件。

2）创建一个表，ORACLE 为表创建一个（或多个）段，在一个段中保存该表的所有表数据(表数据不能跨段）。

3）段中至少有一个初始区。当这个段数据增加使得区（extent）不够时，将为这个段分配新的后续区。

4）延迟段参数，deferred_segment_creation 默认是 TRUE，当建立表后不立即建立表段以及分配初始区



12. 2. 2 段管理方式：

1）自动管理方式，简称 ASSM(Auto Segment Space Management)　采用位图管理段的存储空间

原理：简单说就是每个段的段头都有一组位图（5 个位图），位图描述每个块的满度，根据满度的不同将每个块登记到相应的位图上，位图自动跟踪每个块的使用空间（动态），5 个位图的满度按如下定义：满度 100%，75%、50%、25%和 0%，比如块大小为 8k，你要插入一行是 3k 的表行，那么 oracle 就给你在满度 50%的位图上找个登记的可插入的块。

ASSM 的前提是 EXTENT MANAGEMENT LOCAL，在 ORACLE9I 以后，缺省状态为自动管理方式，ASSM 废弃 pctused 属性。

2）手工管理方式，简称 MSSM(Manual Segment Space Management)　采用 FREELIST(空闲列表）管理段的存储空间

原理：这是传统的方法，现在仍然在使用，涉及三个概念 freelist、pctfree 和 pctused，

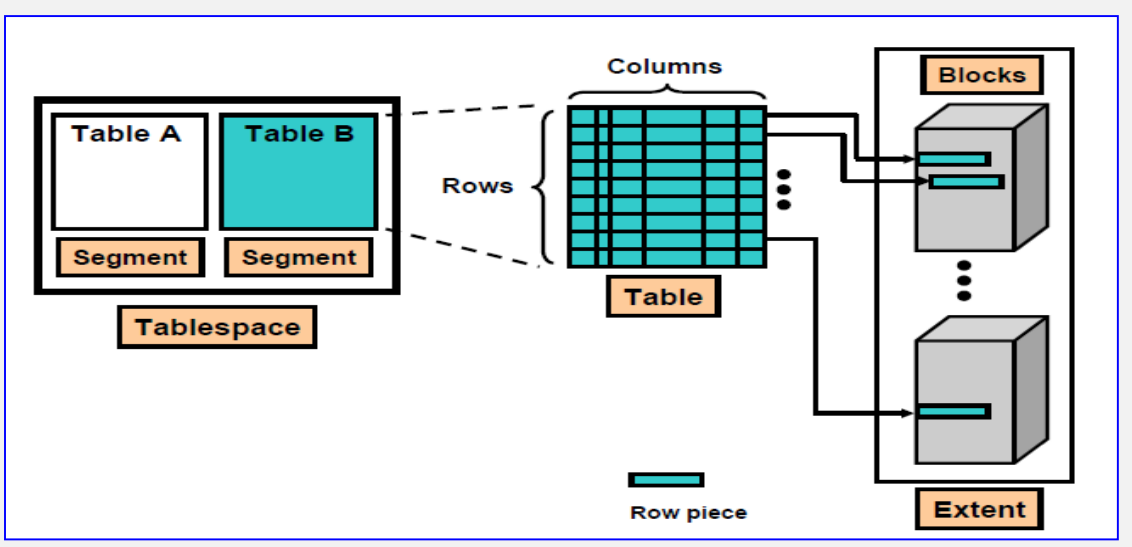
①freelist：空闲列表中登记了可以插入数据的可用块，位置在段头，插入表行数据时首先查找该列表。

②pctfree：用来为一个块保留的空间百分比，以防止在今后的更新操作中增加一列或多列值的长度。达到该值，从 freelist 清除该块信息。

③pctused：一个块的使用水位的百分比，这个水位将使该块返回到可用列表中去等待更多的插入操作。达到该值，该块信息登录到 freelist。

这个参数在 ASSM 下不使用。ASSM 使用位图状态位取代了 pctused

12. 2. 3 表和段的关系



1）一般来讲，一个单纯的表就分配一个段，但往往表没那么单纯，比如表上经常会有主键约束，那么就会有索引，索引有索引段，还有分区表，每个分区会有独立的段，再有就是 Oracle 的大对象，如果你的表里引用 blob,clob,那么这个表就又被分出多个段来。

测试：

SQL> conn / as sysdba

SQL> grant connect,resource to tim identified by tim;

SQL> conn tim/tim

SQL> select * from user_segments;

SQL> create table t1 (id int);

SQL> select segment_name from user_segments;

SQL> create table t2 (id int constraint pk_t2 primary key, b blob, c clob);

SQL> select segment_name from user_segments;

2）延迟段。顾名思义，创建表的时候并不马上建立相应的段。

Oracle11gR2 又增加了一个新的初始化参数 DEFERRED_SEGMENT_CREATION(仅适用未分区的 heap table），此参数默认 TRUE，当 create table 后并不马上分配 segment，仅当第一个 insert 语句后才开始分配 segment。这对于应用程序的部署可能有些好处。(PPT-II-476-478)

也可以使局部设置改变这一功能(覆盖 DEFERRED_SEGMENT_CREATION=TRUE)，在 create table 语句时加上 SEGMENT CREATION 子句指定。如：

SQL>create table scott.t1(id int,name char(10)) SEGMENT CREATION IMMEDIATE TABLESPACE TB1;

create table t2(id int,name char(10)) SEGMENT CREATION deferred;

12.3 EXTENT（区）

12. 3. 1 特点：

区是 ORACLE 进行存储空间分配的最小单位。一个区是由一系列逻辑上连续的 Oracle 数据块组成的逻辑存储结构。一个区不可以跨数据文件，段中第一个区叫初始区，随后分配的区叫后续区。

12. 3. 2 管理方式：

1）字典管理：在数据字典中管理表空间的区空间分配。Oracle 8i 以前只有通过 uet$和 fet$的字典管理。

缺点：某些在字典管理方式下的存储分配有时会产生递归操作，并且容易产生碎片，从而影响了系统的性能，现在已经淘汰了。

2）本地管理：在每个数据文件中使用位图管理空间的分配。表空间中所有区（extent)的分配信息都保存在该表空间对应的数据文件的头部。

优点：速度快，存储空间的分配和回收只是简单地改变数据文件中的位图，而不像字典管理方式还需要修改数据库。无碎片，更易于 DBA 维护。

12. 3. 3 表和区的关系：

当建立表的时候建立段，然后自动分配相应的 extent（1 个或者多个），亦可以手工提前分配 extent（用于需大量插入数据的表）

实验：查看段的初始区分配情况

sys:

SQL> create tablespace test datafile '/u01/oradata/prod/test01.dbf' size 10m;

SQL> create table scott.t1 tablespace test as select * from scott.dept;

SQL> col segment_name for a20;

SQL> select segment_name,file_id,extent_id,blocks,block_id,bytes/1024/1024 mb from dba_extents where segment_name='T1';

SQL> Insert into scott.t1 select * from scott.t1;

SQL> commit;

```
SQL> select segment_name,file_id,extent_id,blocks,block_id,bytes from dba_extents where segment_name='T1';

SEGMENT_NAME            FILE_ID  EXTENT_ID     BLOCKS   BLOCK_ID      BYTES
-------------------- ---------- ---------- ---------- ---------- ----------
T1                            6          0          8          9      65536
T1                            6          1          8         17      65536
T1                            6          2          8         25      65536
T1                            6          3          8         33      65536
T1                            6          4          8         41      65536
T1                            6          5          8         49      65536
T1                            6          6          8         57      65536
T1                            6          7          8         65      65536
T1                            6          8          8         73      65536
T1                            6          9          8         81      65536
T1                            6         10          8         89      65536
T1                            6         11          8         97      65536
T1                            6         12          8        105      65536
T1                            6         13          8        113      65536
T1                            6         14          8        121      65536
T1                            6         15          8        129      65536
T1                            6         16        128        137    1048576
T1                            6         17        128        265    1048576
T1                            6         18        128        393    1048576
```

一个extent内的块（逻辑编号）是连续的

默认extent大小\autoallocate即动态决定分配块的大小

删除掉四分之三数据，看看 extent 是否收回

SQL> delete scott.t1 where deptno>15;

已删除 98304 行

SQL> commit;

重新插入一倍的数据，看看 extent 是否增加

SQL>insert into scott.t1 select * from scott.t1;

已创建 32768 行。

思考一下上面的步骤说明了什么？

12.3.4 预先分配空间

可以根据需要预先分配一些 extent，减少并发分配时可能发生申请区块的争用。

SQL>alter table scott.t1 allocate extent (datafile '/u01/oradata/prod/test01.dbf' size 5m);

注意：预分配的空间一定是在表空间可达到的 size 范围内

回收 free extent，使用 deallocate,

SQL> alter table scott.t1 deallocate unused;

注意：只能收回从未使用的 extent

12.4 BLOCK(数据块）

12.4.1 OracleBlock 的构成

BLOCK 是 Oracle 进行存储空间 IO 操作的最小单位。构成上分为 block header、free space、data

数据块头部：

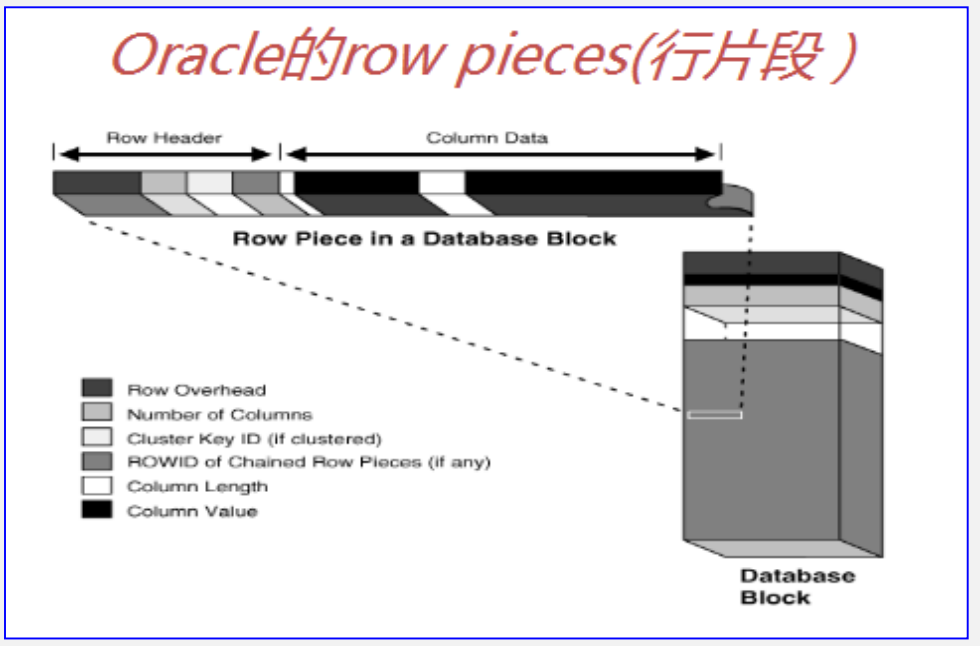
①ITL：事务槽，可以有多个 ITL 以支持并发事务，每当一个事务要更新数据块里的数据时，必须先得到一个 ITL 槽，然后将当前

事务 ID，事务所用的 undo 数据块地址，SCN 号，当前事务是否提交等信息写到 ITL 槽里。

②initrans ：初始化事务槽的个数，表默认 1, index 默认为 2；

③maxtrans： 最大的事务槽个数 （默认 255）

④ROW DIR: 行目录, 指向行片段行起始和结束的偏移量。



考点：使块头增加的可能情况是，row entries 增加，增加更多的事务槽（ITL）空间。

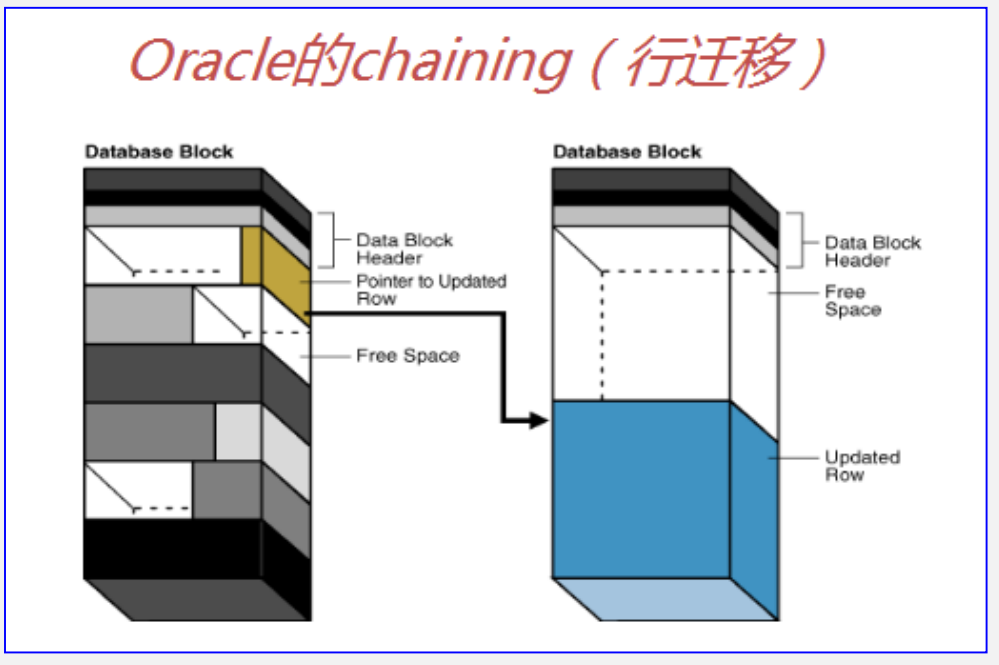
12. 4. 2 行链接和行迁移

1）什么是行链接和行迁移

①行链接：指一行存储在多个块中的情况，即行链接是跨越多块的行。

②行迁移：指一个数据行由于 update 语句导致当前块被重新定位到另一个块（那里有充足的空间）中，但在原始块中会保留一个指针。原始块中的指针是必需的，因为索引的 ROWID 项仍然指向原始位置。

行迁移是 update 语句当 pctfree 空间不足时引起的，它与 insert 和 delete 语句无关。



2）如何知道发生了行链接或行迁移

查看 dba_tables 的 AVG_ROW_LEN 列和 CHAIN_CNT 列。当 CHAIN_CNT 有值时，看 AVG_ROW_LEN，它表示行的平均长度（byte），如果 AVG_ROW_LEN<块大小，发生的是行迁移，否则可能有行链接。

测试：

SCOTT：

```
SQL> create table t1 (c1 varchar2(20));
SQL>
begin
for i in 1..1000 loop
insert into t1 values(null);
end loop;
end;
/
```

先分析一下 t1 表，确定无行迁移

```
SQL> analyze table t1 compute statistics;
SQL> select pct_free,pct_used,avg_row_len,chain_cnt,blocks from user_tables where table_name='T1';
PCT_FREE     PCT_USED AVG_ROW_LEN    CHAIN_CNT        BLOCKS
---------- ---------- ----------- ---------- ----------
10                          3              0                  5
```

使用了 5 个块

```
SQL> select distinct file#,block# from v$bh a,user_objects b where a.objd=b.object_id and b.object_name='T1' order by 2;
```

v$bh 视图可以显示出 t1 表一共分配了 8 个块，具体是那些块。

填充这些空列，再分析 t1,有了行迁移

```
SQL> update t1 set c1='prod is my name';
SQL> commit;
SQL> analyze table t1 compute statistics;
SQL> select pct_free,pct_used,avg_row_len,chain_cnt,blocks from user_tables where table_name='T1';
PCT_FREE     PCT_USED AVG_ROW_LEN    CHAIN_CNT        BLOCKS
---------- ---------- ----------- ---------- ----------
          10                         22            865             13
```

说明 1000 行中有 865 行发生了行迁移，使用的块也增加了。

3）怎样确定那些行发生了行迁移

```
$ sqlplus / as sysdba
SQL> @/u01/oracle/rdbms/admin/utlchain.sql
SQL> analyze table scott.t1 LIST CHAINED ROWS;
SQL> select count(*) from chained_rows;
  COUNT(*)
----------
        865
SQL> select table_name, HEAD_ROWID from chained_rows where rownum<=3;
TABLE_NAME                          HEAD_ROWID
```

```
------------------------------ ------------------

T1                                       AAASC4AAEAAAAIfABQ

T1                                       AAASC4AAEAAAAIfABR

T1                                       AAASC4AAEAAAAIfABS

SQL> Select dbms_rowid.ROWID_RELATIVE_FNO(rowid) fn,

dbms_rowid.rowid_block_number(rowid) bn, rowid,c1 from scott.t1 where rowid='AAASPhAAEAAAAIdABQ';

            FN                BN ROWID                    C1

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

             4               541 AAASPhAAEAAAAIdABQ prod is my name

SYS@ prod>drop table chained_rows;
```

4）解决行迁移有很多方法

可以根据上例 chained_rows 表中提供的 rowid，将 t1 表中的那些记录删除，然后在重新插入。

这里使用 move 解决，简单些：

```
SQL> alter table t1 move;
```

使用 shrink 解决不了行迁移

move 表后，再分析 t1,行迁移消失。

```
SQL> analyze table t1 compute statistics;

SQL> select pct_free,pct_used,avg_row_len,chain_cnt,blocks from user_tables where table_name='T1';

PCT_FREE     PCT_USED AVG_ROW_LEN    CHAIN_CNT         BLOCKS

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

10                         19              0              6
```

手工方法解决行迁移

```
SQL>@/u01/oracle/rdbms/admin/utlchain.sql

SQL>analyze table scott.t1 LIST CHAINED ROWS;

SQL>create table scott.t2 as select * from scott.t1 where rowid in (select HEAD_ROWID from chained_rows);

SQL>delete table scott.t1 where rowid in (select HEAD_ROWID from chained_rows);

SQL>insert into scott.t1 select * from sott.t2;

SQL>drop table scott.t2;
```

5）行链接实验

```
SCOTT@ prod>create table t1 (c1 varchar2(4000),c2 varchar2(4000));

SCOTT@ prod>insert into t1 values(lpad('a',4000,'*'),lpad('b',4000,'*'));

SCOTT@ prod>commit;

SCOTT@ prod>analyze table t1 compute statistics;

SCOTT@ prod>select table_name, AVG_ROW_LEN,CHAIN_CNT from user_tables where table_name='T1';

TABLE_NAME                           AVG_ROW_LEN    CHAIN_CNT
```

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

T1 8015 1

SYS@ prod>create tablespace ttt datafile '/u01/oradata/prod/ttt01.dbf' size 10m blocksize 16k;

SCOTT@ prod>alter table t1 move tablespace ttt;

SCOTT@ prod>analyze table t1 compute statistics;

SCOTT@ prod>select table_name, AVG_ROW_LEN,CHAIN_CNT from user_tables where table_name='T1';

TABLE_NAME AVG_ROW_LEN CHAIN_CNT

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

T1 8009 0

12. 4. 3 表和数据块的关系

1）什么是高水位线？

高水位线（high-water mark，HWM）

在数据库中，如果把表想象成从左到右依次排开的一系列块，高水位线就是曾经包含了数据的最右边的块。原则上 HWM 只会增大，即使将表中的数据全部删除，HWM 也不会降低。

2）HWM 有利有弊，

优点：可以使 HWM 以下的块重复利用

缺点：使用全表扫描时要读取 HWM 以下的所有 block，耗费更多的 IO 资源。

12. 4. 4 如何降低 HWM

多种方法可以降低 HWM：①移动表，②收缩表，③导入导出表，④在线重定义表

1）移动表

move 方法，将表从一个表空间移动到另一个表空间（也可以在本表空间内 move)。

语法：alter table t1 move [tablespace users];

优点：可以清除数据块中的碎片,降低高水位线。适用 MSSM 和 ASSM

缺点：

①move 需要额外（一倍）的空间。

②move 过程中会锁表，其他用户不能在该表上做 DML 或 DDL 操作。

③move 之后，相关索引都不可用了，表上的索引需要重建。

测试 move 后索引不可用

SCOTT:

SQL>create table emp1 as select * from emp;

SQL>create index emp1_idx on emp1(ename);

SQL>select table_name,index_name,status from user_indexes where table_name='EMP1';

TABLE_NAME INDEX_NAME STATUS

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

EMP1 EMP1_IDX VALID

SQL>alter table emp1 move;

SQL>select table_name,index_name,status from user_indexes where table_name='EMP1';

TABLE_NAME INDEX_NAME STATUS

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

EMP1 EMP1_IDX UNUSABLE

SQL>alter index EMP1_IDX rebuild;

SQL>select table_name,index_name,status from user_indexes where table_name='EMP1';

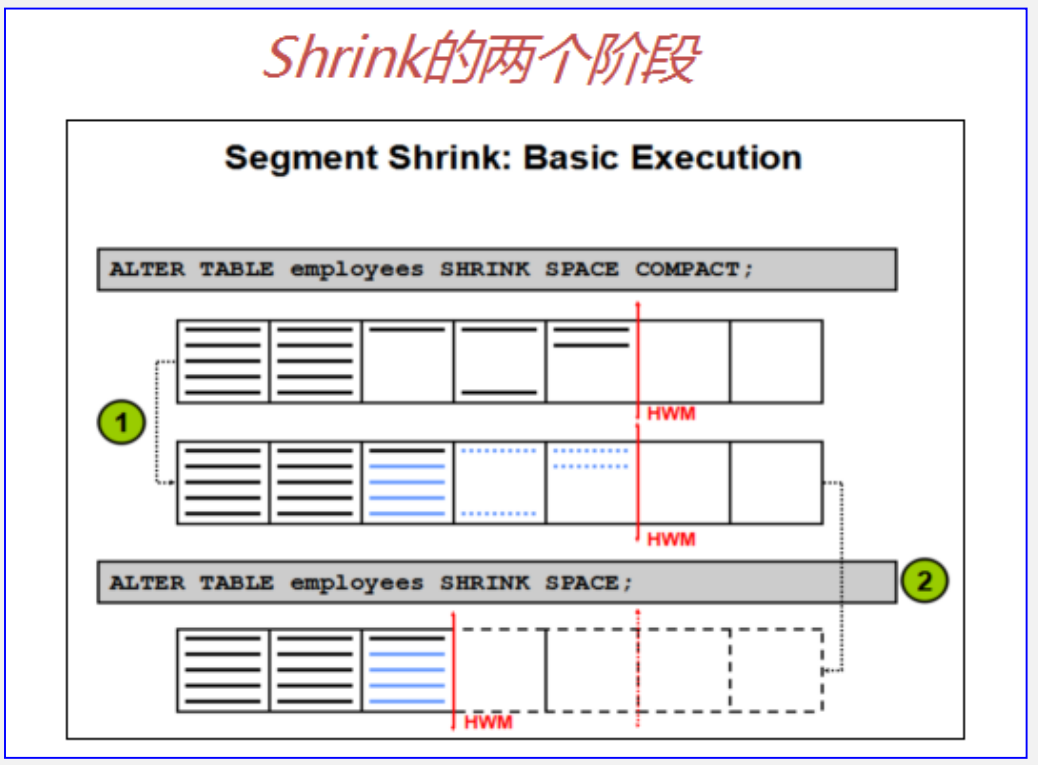
TABLE_NAME INDEX_NAME STATUS

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

EMP1 EMP1_IDX VALID

2）收缩表

Shrink 方法，也叫段重组，表收缩的底层实现的是通过匹配的 INSERT 和 DELETE 操作。



语法：alter table t2 shrink space [cascade][compact];

优点：使用位图管理技术，①降低热块，②更合理的重新利用空闲块

缺点：①要求段管理是 ASSM 方式，②表上启用 row movement。

它分两个不同的阶段：压缩阶段和降低 HWM 阶段。(PPT-II-491)

第一阶段：发出 alter table t2 shrink space compact 命令;这是压缩阶段。在业务高峰时可以先完成这样步骤

第二阶段：再次 alter table t2 shrink space; 因压缩阶段工作大部分已完成，将很快进入降低 HWM 阶段，DML 操作会有短暂的锁等待发生。

测试：

SQL>create table scott.t2 as select * from dba_objects;

scott：

SQL>select max(rownum) from t2;

SQL> analyze table t2 compute statistics;

SQL> select table_name,blocks,empty_blocks,num_rows from user_tables where table_name='T2';

TABLE_NAME BLOCKS EMPTY_BLOCKS NUM_ROWS

T2 1039 113 68875

Blocks：表示使用过的块，即低于 HWM 的块数量。

empty_blocks：表示 extent 分配了，但从未使用过的块，即高于 HWM 的块数量

两项之和 1039+113=1152 是这个段分配的块数

SQL> select segment_name,blocks from user_segments where segment_name='T2';

SEGMENT_NAME BLOCKS

T2 1152

删除 40000 行

SQL>delete t2 where rownum<=40000;

SQL>commit;

SQL> analyze table t2 compute statistics;

SQL> select table_name,blocks,empty_blocks,num_rows from user_tables where table_name='T2';

TABLE_NAME BLOCKS EMPTY_BLOCKS NUM_ROWS

T2 1039 113 28875

num_rows 已经减掉了 40000 条， 但 blocks 并没有减少， 说明 HWM 没有下降。

做 shrink

SQL>alter table t2 enable row movement; 使能行移动

第一步：压缩阶段

SQL>alter table t2 shrink space compact;

SQL>analyze table t2 compute statistics for table;

SQL> select table_name, blocks, empty_blocks, num_rows from user_tables where table_name='T2'; HWM 不会降低。

第二步：降低 HWM 阶段

SQL> alter table t2 shrink space;

SQL> analyze table t2 compute statistics;

SQL> select table_name,blocks,empty_blocks,num_rows from user_tables where table_name='T2';

TABLE_NAME BLOCKS EMPTY_BLOCKS NUM_ROWS

T2 426 22 28875

12.5 临时表空间

12. 5. 1 用途:

用于缓存排序的数据（中间结果）

可以建立多个临时表空间，但默认的临时表空间只能有一个且不能 offline 和 drop。temp 表空间是 nologing 的（不记日志）。

```
SQL> select file_id,file_name,tablespace_name from dba_temp_files;

FILE_ID FILE_NAME                                          TABLESPACE_NAME
---------- -------------------------------------------------- ------------------------------
1          /u01/oradata/prod/temp01.dbf                    TEMP

SQL> col name for a60;

SQL> select file#,name ,bytes/1024/1024 from v$tempfile;

FILE# NAME                                                 BYTES/1024/1024
---------- -------------------------------------------------- ---------------
1       /u01/oradata/prod/temp01.dbf                       100
```

12. 5. 2 基本操作

1）建立临时表空间 temp2。增加或删除 tempfile。

```
SQL> create temporary tablespace temp2 tempfile '/u01/oradata/prod/temp02.dbf' size 10m;

SQL> alter tablespace temp2 add tempfile '/u01/oradata/prod/temp03.dbf' size 5m;

SQL> select file_id,file_name,tablespace_name from dba_temp_files;

FILE_ID FILE_NAME                                          TABLESPACE_NAME
---------- -------------------------------------------------- ------------------------------
1          /u01/oradata/prod/temp01.dbf                    TEMP
2          /u01/oradata/prod/temp02.dbf                    TEMP2
3          /u01/oradata/prod/temp03.dbf                    TEMP2
```

将 temp2 里删掉一个 tempfile。

```
SQL> alter tablespace temp2 drop tempfile '/u01/oradata/prod/temp03.dbf';

SQL> select file_id,file_name,tablespace_name from dba_temp_files;
```

2） 查看默认的临时表空间

```
SQL> select * from database_properties where rownum<=5;
```

3） 指定用户使用临时表空间

```
SQL> alter user scott temporary tablespace temp2;
```

4） 切换默认的临时表空间

```
SQL> alter database default temporary tablespace temp2;
```

12. 5. 3 临时表空间组（10g 新特性）

在很多情况下，会有多个 session 使用同一个用户名去访问 Oracle，而临时表空间又是基于用户的，那么可以建立一个临时表空间组，组中由若干临时表空间成员构成，从而可以提高单个用户多个会话使用临时表空间的效率。

1）临时表空间组无法显式创建，组是通过第一个临时表空间分配时自动创建。

SQL> alter tablespace temp1 tablespace group tmpgrp;

SQL> alter tablespace temp2 tablespace group tmpgrp;

SQL> select * from dba_tablespace_groups;

GROUP_NAME TABLESPACE_NAME

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

TMPGRP TEMP1

TMPGRP TEMP2

2）将临时表空间组设成默认临时表空间，实现负载均衡。

SQL> alter database default temporary tablespace tmpgrp;

3）要移除表空间组时，该组不能是缺省的临时表空间。

SQL> alter database default temporary tablespace temp;

SQL> alter tablespace temp1 tablespace group '';

SQL> alter tablespace temp2 tablespace group '';

4）当组内所有临时表空间被移除时，组也被自动删除。

SQL> select * from dba_tablespace_groups;

no rows selected

SQL> drop tablespace temp2 including contents and datafiles;

考点：某个 tempfile 坏掉使得 default temporary tablespace 不能正常工作，数据库不会 crash，解决的办法是 add 一个新的 tempfile，然后

再 drop 掉坏的 tempfile.（default temporary tablespace 不能 offline，但 temporary file 可以 offline）

12.6 如何调整表空间的尺寸

表空间的大小等同它下的数据文件大小之和

当发生表空间不足的问题时常用的 3 个解决办法：

1）增加原有数据文件大小（resize）

2）增加一个数据文件（add datafile）

3）设置表空间自动增长（autoextend)

示例:

SQL> create tablespace prod datafile '/u01/oradata/prod/prod01.dbf' size 5m;

SQL> create table scott.test1 (id int) tablespace prod;

SQL> insert into scott.test1 values(1);

SQL> insert into scott.test1 select * from scott.test1;

SQL> /

SQL> /

报错：ORA-01653: unable to extend table SCOTT.TEST1 by 8 in tablespace prod

1）用第一种方法扩充表空间

SQL> alter database datafile '/u01/oradata/prod/prod01.dbf' resize 10m;

SQL> insert into scott.test1 select * from scott.test1;

SQL> /

SQL> /

报错：ORA-01653: unable to extend table SCOTT.TEST1 by 128 in tablespace prod

2）用第二种方法扩充表空间：

SQL> alter tablespace prod add datafile '/u01/oradata/prod/prod02.dbf' size 20m;

SQL> insert into scott.test1 select * from scott.test1;

SQL> /

SQL> /

报错：ORA-01653: unable to extend table SCOTT.TEST1 by 128 in tablespace prod

3）用第三种方法扩充表空间：

SQL> alter database datafile '/u01/oradata/prod/prod01.dbf' autoextend on next 10m maxsize 500m;

SQL> insert into scott.test1 select * from scott.test1;

SQL> drop tablespace prod including contents and datafiles;

12.7 可恢复空间分配 Resumable

当我们往一个表里面插入大量数据时，如果某条 insert 语句因表空间的空间不足（没有开启自动扩展），会报 ORA-01653:无法扩展空间的错误，该条 SQL 语句会中断，浪费了时间及数据库资源。为防范这个问题，Oracle 设计了 resumable。

1）功能：

在 resumable 开启的情况下，如果 Oracle 执行某条 SQL 申请不到空间了，比如数据表空间，undo 表空间，temporary 空间等，则会将该事务的语句挂起（suspended)，等

你把空间扩展后，Oracle 又会使该 insert 语句继续进行。

2）设置方法

可以通过两个级别设置 resumable

①system 级别：初始化参数 RESUMABLE_TIMEOUT 非 0，这将使数据库中所有 session 使用可恢复的空间分配

②session 级别：alter session enable|disable resumable [TIMEOUT]; 这将为当前 session 设置可恢复的空间分配。因为 resumable 是有资源消耗代价的, 所以 session 级的 resumable 是比较实用的。

3）参数 RESUMABLE_TIMEOUT 的用法

单位为秒,

RESUMABLE_TIMEOUT=0, enable session 时应该指定 TIMEOUT。否则使用缺省值 7200 秒。

RESUMABLE_TIMEOUT<>0，enable session 时可以省略 TIMEOUT，此时指定 TIMEOUT 会覆盖掉参数 RESUMABLE_TIMEOUT 值。

示例：

session 1:

1）建个小表空间，固定 2m 大小，然后建个表属于这个表空间

SQL> create tablespace small datafile '/u01/oradata/prod/small01.dbf' size 2m;

SQL> create table scott.test(n1 char(1000)) tablespace small;

2）向这个表插入数据，表空间满了，使 for 语句没有完成循环，2000 条语句整体失败。

SQL> begin

for i in 1..2000 loop

insert into scott.test values('this is test');

end loop;

commit;

end;

/

报错：ORA-01653: 表 SCOTT.TEST 无法通过 128 (在表空间 SMALL 中) 扩展

这个 128 是块数，表明最后一次需要 1M 的 extent，没有成功.

SQL> select count(*) from scott.test;

COUNT(*)

0

因为没有在循环体内 commit，所以插入的记录全部回滚了。

3）使能 resumable 功能

SQL> alter session enable resumable;

4）再重复第 2）步，会话被挂起；

session 2:

5）查看视图的有关信息

SQL> select session_id,sql_text,error_number from dba_resumable;

SESSION_ID SQL_TEXT ERROR_NUMBER

--- --- ---

136 INSERT INTO SCOTT.TEST VALUES('this is test') 1653

SQL> select sid,event,seconds_in_wait from v$session_wait where sid=136;

SID EVENT SECONDS_IN_WAIT

--- --- ---

136 statement suspended, wait error to be cleared 1

6）加扩表空间，看到 session1 里挂起的会话得以继续并成功完成了 2000 条语句的插入。

SQL> alter tablespace small add datafile '/u01/oradata/prod/small02.dbf' size 4m;

SQL> select count(*) from scott.test;

COUNT(*)

2000

7）查看 EM 告警日志报告了以上信息。验证结束后可以 disable resumable，并删除 small 表空间及数据文件。

session 1:

SQL> alter session disable resumable;

SQL> drop tablespace small including contents and datafiles;

第十三章：表的类型

13.1 表的类型

①堆表：heap table ：数据存储时，行的位置和插入数据的先后顺序无关。采用全表扫描时依据 rowid 排序进行。

②分区表 表>2G

③索引组织表（IOT）

④簇表

⑤临时表

⑥压缩表

13.2 表分区及其种类

13.2.1 Range Partitioning（范围分区）

1）建立分区表

SCOTT:

```
SQL>create table sale(
product_id varchar2(5), sales_count number(10,2))
partition by range(sales_count)
(
   partition p1 values less than(1000),
   partition p2 values less than(2000),
   partition p3 values less than(3000)
);
```

2）查看信息：

```
select TABLE_NAME,PARTITION_NAME,HIGH_VALUE from user_tab_partitions where table_name='SALE';
insert into sale values('1',600);
insert into sale values('2',1000);
insert into sale values('3',2300);
insert into sale values('4',6000);
commit;
select * from sale partition(p1);
select * from sale partition(p2);
```

3）增加一个分区

alter table sale add partition p4 values less than(maxvalue);

再看一下，可以插入 6000 值了

SQL>select TABLE_NAME,PARTITION_NAME,HIGH_VALUE from user_tab_partitions where table_name='SALE';

查看分区基本面的视图

SQL>select * from user_part_key_columns where name='SALE';　　查看分区列信息的视图

insert into sale values('4',6000);

4）看一下段的分配

SQL> select segment_name,segment_type,partition_name from user_segments;

5）删除分区、插入分区，合并分区

SQL>alter table sale drop partition p2;　　删去分区 2

SQL>alter table sale split partition p3 at (2000) into (partition p2,partition p3);　　插入分区 2

SQL>insert into sale values(2,1800);

SQL>commit

SQL>alter table sale merge partitions p2,p3 into partition p3;　　合并分区，数据还在

SQL>alter table sale split partition p3 at (2000) into (partition p2,partition p3); 恢复原来的 4 分区

6）update 操作

默认情况下，如果对分区表的分区字段做超范围（跨段）update 操作，会报错——ORA-14402:。如果一定要改，可以通过打开表的 row movement 属性来完成。

SQL> select rowid,t1.* from sale partition(p1) t1;

ROWID　　PRODU SALES_COUNT

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

AAASvUAAEAAAAGVAAA 1　　600

SQL> update sale set sales_count=1200 where sales_count=600;

报错：ORA-14402: 更新分区关键字列将导致分区的更改

SQL> alter table sale enable row movement;

SQL> update sale set sales_count=1200 where sales_count=600;

SQL> select rowid,t1.* from sale partition(p2) t1;

ROWID　　PRODU SALES_COUNT

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

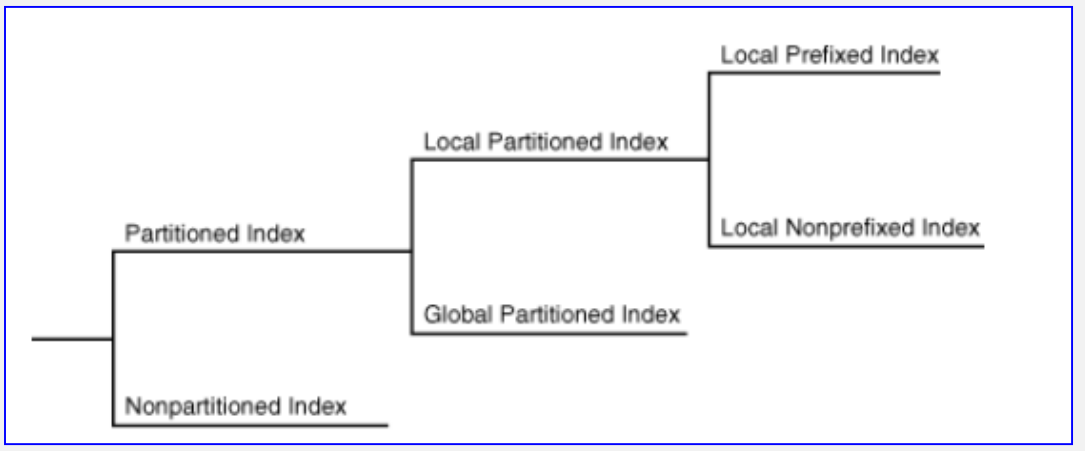
AAASvVAAEAAAAGdAAA 2　　1000

AAASvVAAEAAAAGdAAB 1　　1200

一般来说范围分区的分区字段使用数字类型或日期类型，使用字符类型的语法是可以的，实际工作中使用较少。这或许跟 values less than 子句有关。

13. 2. 2 分区索引

一般使用分区都会建立索引，分区索引有 local 与 global 之分。



所谓前缀索引 Local Parfixed Index，是指组合索引键值列的 first column 使用的是索引的分区列。

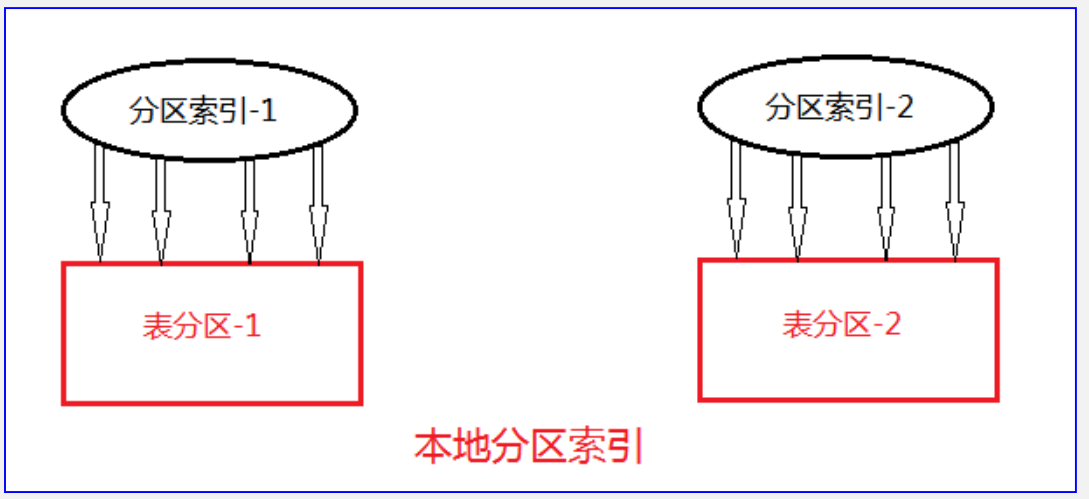
具体来说，

1）当本地索引列只有一列，而该列正是表的分区列时。

2）如果本地索引列是组合多列，那么第一列必须是表分区列。

1）本地索引 Local：

特点：一个索引分区对应一个表分区，表分区边界就是索引分区边界。更新一个表分区时仅仅影响该表分区对应的索引分区。同样，删除一个分区时对应的索引分区也被删除，增加一个分区时索引分区也自动增加。



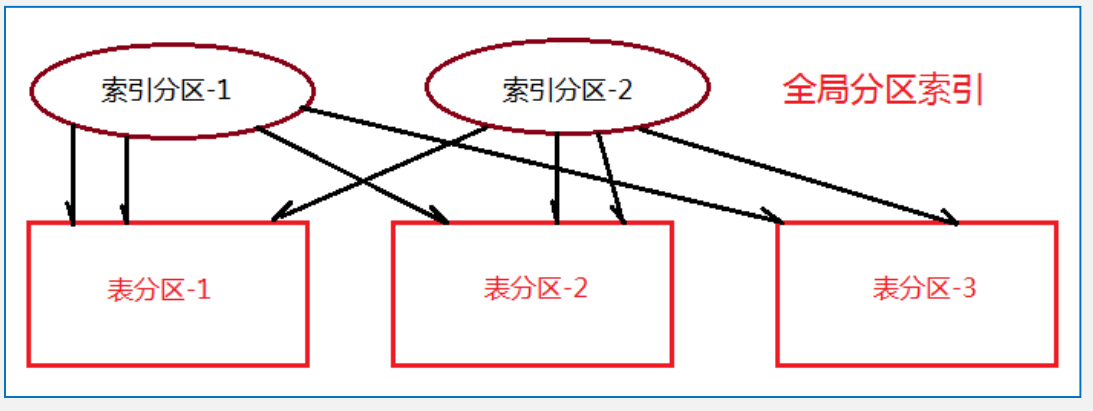
SQL>create index sale_idx on sale(sales_count) local;

SQL>select INDEX_NAME,PARTITION_NAME,HIGH_VALUE from user_ind_partitions;

删除一个分区，其中的数据全部清除，并且包括相关索引等

SQL> alter table sale drop partition p3;

2）全局索引 global：



特点：

①索引分区不与表分区对应，索引分区数可以和表分区数一样多，也可以不一样多，

但索引 key 和索引分区列必须重合（否则报 ORA-14038: GLOBAL 分区索引必须加上前缀）。

②一定要将 maxvalue 关键字做上限。

③删除表分区要带上 update indexes 子句，否则全局索引分区 UNUSABLE

```
SQL>
create index sale_global_idx on sale(sales_count) global
partition by range (sales_count)
(
partition p1 values less than(1500),
partition p2 values less than(maxvalue)
);
SQL>select index_name,status from user_ind_partitions;
SQL>alter table sale drop partition p2 update indexes; --索引不失效
（否则需要单独 rebuild：alter index SALE_GLOBAL_IDX rebuild partition p1;）
SQL>select index_name,status from user_ind_partitions;
SQL>alter index SALE_GLOBAL_IDX rebuild partition p1;
```

13.3 其他表分区

13.3.1 Hash Partitioning（散列分区，也叫 hash 分区）

实现均匀的负载值分配，增加 HASH 分区可以重新分布数据。

```
SQL>
create table my_emp(empno number, ename varchar2(10))
partition by hash(ename)
(
   partition p1, partition p2
);

SQL>select * from user_tab_partitions where table_name='MY_EMP';
SQL>select TABLE_NAME,PARTITIONING_TYPE,PARTITION_COUNT,STATUS from USER_PART_TABLES;

TABLE_NAME                     PARTITION PARTITION_COUNT STATUS
------------------------------ --------- --------------- --------
MY_EMP                         HASH                    2 VALID
```

插入 10000 条记录，看是否均匀插入。

```
begin
 for i in 1..10000 loop
```

```
    insert into my_emp values(i,'tim'||i);
  end loop;
end;
/
select count(*) from my_emp partition(P1);
select count(*) from my_emp partition(P2);
```

13.3.2 list 分区（列表分区）枚举法：

将不相关的数据组织在一起

```
create table personcity(id number, name varchar2(10), city varchar2(10))
partition by list(city)
(
    partition east values('tianjin','dalian'),
    partition west values('xian'),
    partition south values ('shanghai'),
    partition north values ('herbin'),
    partition other values (default)
);
SQL>select TABLE_NAME,PARTITION_NAME,HIGH_VALUE from user_tab_partitions where table_name='PERSONCITY';

TABLE_NAME                     PARTITION_NAME                 HIGH_VALUE
------------------------------ ------------------------------ ----------------------------------------------------------------------------------------
PERSONCITY                     EAST                           'tianjin', 'dalian'
PERSONCITY                     WEST                           'xian'
PERSONCITY                     SOUTH                          'shanghai'
PERSONCITY                     NORTH                          'herbin'
PERSONCITY                     OTHER                          default

SQL>select TABLE_NAME,PARTITIONING_TYPE,PARTITION_COUNT,STATUS from USER_PART_TABLES where TABLE_NAME='PERSONCITY';

TABLE_NAME                     PARTITION PARTITION_COUNT STATUS
------------------------------ --------- --------------- --------
PERSONCITY                     LIST                    5 VALID

insert into personcity values(1,'sohu','tianjin');
insert into personcity values(2,'sina','herbin');
```

```
insert into personcity values(3,'yahoo','dalian');
insert into personcity values(4,'360','zhengzhou');
insert into personcity values(5,'baidu','xian');
```

查看结果

```
select * from personcity partition(east);
```

13.3.3 Composite Partitioning（复合分区）

把范围分区和散列分区相结合或者 范围分区和列表分区相结合。

```
create table student(sno number, sname varchar2(10))
partition by range(sno)
subpartition by hash(sname)
subpartitions 4
(
   partition p1 values less than(1000),
   partition p2 values less than(2000),
   partition p3 values less than(maxvalue)
);
```

有三个 range 分区做基本分区，每个 range 分区会有 4 个 hash 子分区，一共有 12 个分区。

```
SQL> select TABLE_NAME,PARTITION_NAME,SUBPARTITION_COUNT,HIGH_VALUE
from user_tab_partitions where TABLE_NAME='STUDENT';
```

```
SQL> select TABLE_NAME,PARTITION_NAME,SUBPARTITION_NAME from user_tab_subpartitions where table_name='STUDENT';
```

13.3.4 Interval Partitioning （间隔分区）

实际上是由 range 分区引申而来，最终实现了 range 分区的自动化。

scott:

SQL>

```
create table interval_sales (s_id int, d_1 date)
partition by range(d_1)
interval (numtoyminterval(1,'MONTH'))
(
    partition p1 values less than ( to_date('2010-02-01','yyyy-mm-dd') )
);
SQL> insert into interval_sales values(1, to_date('2010-01-21','yyyy-mm-dd') );
SQL> insert into interval_sales values(2, to_date('2010-02-01','yyyy-mm-dd') );
```

越过 p1 分区上线，将自动建立一个分区

```
SQL>select TABLE_NAME,PARTITIONING_TYPE,INTERVAL from user_part_tables where TABLE_NAME='INTERVAL_SALES';
```

```
TABLE_NAME                     PARTITION INTERVAL
------------------------------ --------- ------------------------------
INTERVAL_SALES                 RANGE     NUMTOYMINTERVAL(1,'MONTH')

SQL> select partition_name,HIGH_VALUE,INTERVAL from user_tab_partitions;

PARTITION_NAME                 HIGH_VALUE                INTERVAL
------------------------------ ------------------------- ------------------------------
SYS_P33                        TO_DATE(' 2010-03-01      YES
                                00:00:00', 'SYYYY-M
                               M-DD HH24:MI:SS', 'N
                               LS_CALENDAR=GREGORIA

P1                             TO_DATE(' 2010-02-01      NO
                                00:00:00', 'SYYYY-M
                               M-DD HH24:MI:SS', 'N
                               LS_CALENDAR=GREGORIA
```

```
SQL> alter table interval_sales rename partition sys_p33 to p2;
```

可以重命名分区名

注意：interval (numtoyminterval(1,'MONTH'))的意思就是每个月有一个分区，每当输入了新的月份的数据，这个分区就会自动建立，而不同年的相同月份是两个分区。

分区重命名 alter table INTERVAL_SALES rename partition SYS_P33 to p2;

13.3.5 System Partitioning（系统分区）

这是一个人性化的分区类型，System Partitioning，在这个新的类型中，不需要指定任何分区键，数据会进入哪个分区完全由应用程序决定，即在 Insert 语句中决定记录行插入到哪个分区。

先建立三个表空间 tbs1,tbs2,tbs3， 然后建立三个分区的 system 分区表，分布在三个表空间上。

```
create table test (c1 int,c2 int)
partition by system
(
   partition p1 tablespace tbs1,
   partition p2 tablespace tbs2,
   partition p3 tablespace tbs3
);
```

现在由 SQL 语句来指定插入哪个分区：

```
SQL> INSERT INTO test PARTITION (p1) VALUES (1,3);

SQL> INSERT INTO test PARTITION (p3) VALUES (4,5);

SQL> select * from test;

        C1         C2
---------- ----------
         1          3
         4          5
```

注意：如果要删除以上表空间，必须先删除其上的分区表，否则会报错 ORA-14404: 分区表包含不同表空间中的分区。

13.3.6 Reference Partitioning（引用分区）

当两个表是主外键约束关联时，我们可以利用父子关系对这两个表进行分区。只要对父表做形式上的分区，然后子表就可以继承父表的分区。

如果没有 11g 的引用分区，你想在两个表上都建立对应的分区，那么需要使两表分别有相同名称的键值列。引用分区的好处是避免了在子表上也建立父表同样的一个分区键列，父表上的任何分区维护操作都将自动的级联到子表上。

```
SQL>
CREATE TABLE purchase_orders
   (po_id NUMBER(4),
    po_date TIMESTAMP,
    supplier_id NUMBER(6),
    po_total NUMBER(8,2),
    CONSTRAINT order_pk PRIMARY KEY(po_id))
PARTITION BY RANGE(po_date)
   (PARTITION Q1 VALUES LESS THAN (TO_DATE('2007-04-01','yyyy-mm-dd')),
    PARTITION Q2 VALUES LESS THAN (TO_DATE('2007-06-01','yyyy-mm-dd')),
    PARTITION Q3 VALUES LESS THAN (TO_DATE('2007-10-01','yyyy-mm-dd')),
    PARTITION Q4 VALUES LESS THAN (TO_DATE('2008-01-01','yyyy-mm-dd')));
```

父表做了一个 Range 分区（可对引用分区使用除间隔分区外的所有分区策略）

```
SQL>
CREATE TABLE purchase_order_items
   (po_id NUMBER(4) NOT NULL,
    product_id NUMBER(6) NOT NULL,
    unit_price NUMBER(8,2),
    quantity NUMBER(8),
    CONSTRAINT po_items_fk FOREIGN KEY (po_id) REFERENCES purchase_orders(po_id))
    PARTITION BY REFERENCE(po_items_fk);
```

注意：

①主表 po_date 键值列做范围分区，子表省略 po_date 列也想做相应的分区，那么使用引用分区吧。

②子表最后一句 PARTITION BY REFERENCE()子句给出了引用分区约束名，使用的是子表的外键约束名。

③子表的外键 po_id 列必须是 NOT NULL。这与通常外键可以为 NULL 是有区别的。

SQL> select TABLE_NAME,PARTITION_NAME,HIGH_VALUE from user_tab_partitions;

SQL> select TABLE_NAME,PARTITIONING_TYPE,REF_PTN_CONSTRAINT_NAME from user_part_tables;

13.3.7 Virtual Column-Based Partitioning（虚拟列分区）

虚拟列是 11g 的新特性：

1）只能在堆组织表（普通表）上创建虚拟列

2）虚拟列的值并不是真实存在的，只有用到时才根据表达式计算出虚拟列的值，磁盘上并不存放。

3）可在虚拟列上建立索引。

4）如果在已经创建的表中增加虚拟列时，若没有指定虚拟列的字段类型，ORACLE 会根据 generated always as 后面的表达式计算的结果自动设置该字段的类型。

5）虚拟列的值由 ORACLE 根据表达式自动计算得出，不可以做 UPDATE。

6）表达式中的所有列必须在同一张表。且不能使用其他虚拟列。

7）可把虚拟列当做分区关键字建立虚拟列分区表。

SQL>

```
create table emp1
   (empno number(4) primary key,
     ename char(10) not null,
     salary number(5) not null,
     bonus number(5)    not null,
     total_sal AS (salary+bonus))
partition by range (total_sal)
   (partition p1 values less than (5000),
     partition p2 values less than (maxvalue))
     enable row movement;

insert into emp1(empno,ename,salary,bonus) values(7788,'SCOTT',3000,1000);
insert into emp1(empno,ename,salary,bonus) values(7902,'FORD',4000,1500);
insert into emp1(empno,ename,salary,bonus) values(7839,'KING',5000,3500);
commit;
```

SQL> select * from user_tab_partitions;

SQL> select * from user_part_key_columns;

SQL> select * from emp1 partition (p1);

EMPNO	ENAME	SALARY	BONUS	TOTAL_SAL
7788	SCOTT	3000	1000	4000

SQL> select * from emp1 partition (p2);

EMPNO	ENAME	SALARY	BONUS	TOTAL_SAL
7902	FORD	4000	1500	5500
7839	KING	5000	3500	8500

SQL> update emp1 set bonus=500 where empno=7902;

在建表时就使能了行移动（enable row movement），当更新分区键值时就不会报错（ORA-14402: 更新分区关键字列将导致分区的更改），这时查看 7902 到 p1 区去了

13.3.8 More Composite Partitioning

在 10g 中，我们知道复合分区只支持 Range-List 和 Range-Hash，而在在 11g 中复合分区的类型大大增加，现在 Range，List，Interval 都可以作为 base level 分区，而 Second level 则可以是 Range，List，Hash，也就是在 11g 中可以有 3*3=9 种复合分区，满足更多的业务需求。

13.4 表的联机重定义

表联机重定义的最大好处是当表处于在线联机访问的情况下，能够保证表的一致性。并且其相关约束、索引、触发器等等可以自动迁移，无需重建。

主要的三个方面应用是：

①把普通的堆表重定义成分区表，

②把普通表和索引组织表互相转换生成

③把 MSSM 下的堆表迁移到 ASSM 下

示例：联机创建分区表：将 emp1 非分区表联机重定义为分区表，要求完成两个任务，使其按照 sal 分区（以 2500 为界），并去掉 comm 列。这个过程需要建立一个临时分区表 emp1_temp 完成复制转换。

sys 下执行

SQL>create table scott.emp1 as select * from scott.emp;

SQL>alter table scott.emp1 add constraint pk_emp1 primary key(empno);

SYS>alter table scott.emp1 add constraint chk_emp1 check (sal>500);

1）创建一个临时分区表:emp1_temp，含有 7 列（删去 comm 列），然后 range 分区，两个区以 sal=2500 为界。

SQL>

```
CREATE TABLE scott.emp1_temp
   (empno              number(4) not null,
```

```
    ename           varchar2(10),
    job             varchar2(9),
    mgr             number(4),
    hiredate        date,
    sal             number(7,2),
    deptno          number(2))
PARTITION BY RANGE(sal)
    (PARTITION sal_low VALUES LESS THAN(2500),
    PARTITION sal_high VALUES LESS THAN (maxvalue));
```

2）检查原始表是否具有在线重定义资格，（要求表自包含及之前没有建立实体化视图及日志）

```
SQL>
BEGIN
   DBMS_REDEFINITION.CAN_REDEF_TABLE('scott','emp1');
END;
/
```

3）启动联机重定义处理过程

```
SQL>
BEGIN
   dbms_redefinition.start_redef_table('scott','emp1','emp1_temp',
    'empno empno,
    ename ename,
    job job,
    mgr mgr,
    hiredate hiredate,
    sal sal,
    deptno deptno');
END;
/
SQL> select count(*) from scott.emp1_temp;
   COUNT(*)
----------
            14
SQL> select * from scott.emp1_temp partition(sal_low);
```

SQL> select * from scott.emp1_temp partition(sal_high);

这个时候 emp1_temp 的主键、索引、触发器和授权等还没有从原始表继承过来。

SQL> select table_name,constraint_name from user_constraints where table_name='EMP1_TEMP';

TABLE_NAME CONSTRAINT_NAME

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

EMP1_TEMP SYS_C0011111 这是非空约束

SQL>select table_name,index_name from user_indexes where table_name='EMP1_TEMP';

no rows selected 没有索引

4）复制依赖对象

这一步的作用是：临时分区表 emp1_temp 继承原始表 emp1 的全部属性：包括索引、约束和授权以及触发器。

SQL>

```
DECLARE
   log_error int;
BEGIN
   DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
uname      =>'SCOTT',
orig_table    =>'EMP1',
int_table    =>'EMP1_TEMP',
num_errors =>log_error
);
END;
/
```

SQL> select table_name,constraint_name,SEARCH_CONDITION from user_constraints where table_name='EMP1_TEMP';

TABLE_NAME CONSTRAINT_NAME SEARCH_CONDITION

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

EMP1_TEMP SYS_C0011111 "EMPNO" IS NOT NULL

EMP1_TEMP TMP$$_CHK_EMP10 sal>500

EMP1_TEMP TMP$$_PK_EMP10

SQL>select table_name,index_name from user_indexes where table_name='EMP1_TEMP';

TABLE_NAME INDEX_NAME

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

EMP1_TEMP TMP$$_PK_EMP10

5) 完成增量

```
begin
  dbms_redefinition.sync_interim_table('scott','emp1','emp1_temp');
end;
/
```

6) 完成重定义过程。

```
SQL> EXECUTE dbms_redefinition.finish_redef_table('scott','emp1','emp1_temp');
SQL> select table_name,partition_name,high_value from user_tab_partitions;

SQL>select table_name,constraint_name from user_constraints where table_name='EMP1';
TABLE_NAME                     CONSTRAINT_NAME
------------------------------ ------------------------------
EMP1                           SYS_C0011111          这个非空约束可以删掉
EMP1                           CHK_EMP1
EMP1                           PK_EMP1

SQL>select table_name,index_name from user_indexes where table_name='EMP1';
TABLE_NAME                     INDEX_NAME
------------------------------ ------------------------------
EMP1                           PK_EMP1
```

最后的 finish 这一步发生了什么事情：

①原始表的增量更改刷新到中间表，会有锁表，时间看数据量大小

②原始表 emp1 与临时分区表 emp1_temp 互换名称。

13.5 索引组织表

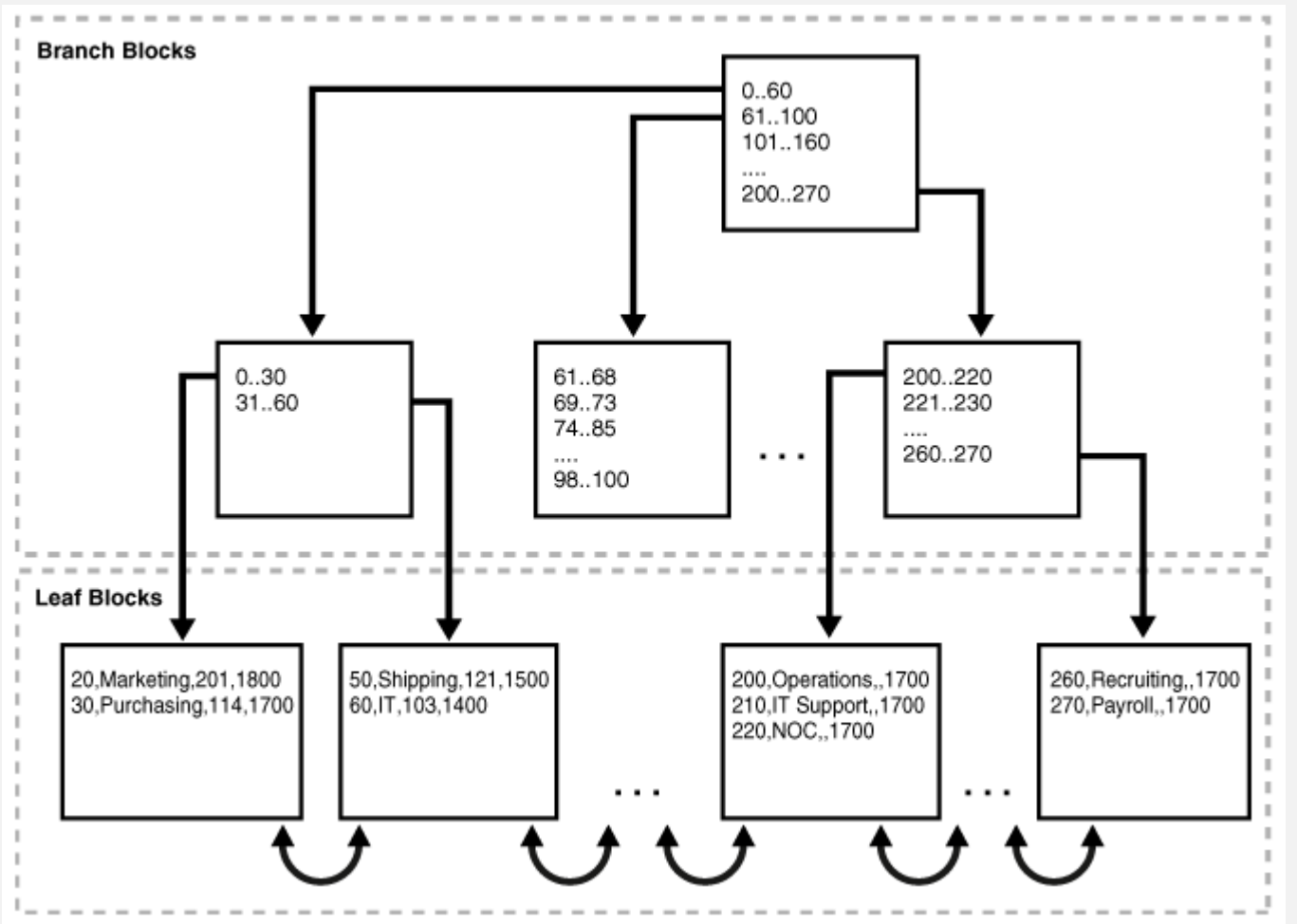
1）特点：

如果表经常以主键查询，并且较少的 DML 操作，可以考虑建立索引组织表（Index OrganizationTable）简称 IOT

Heap table 数据的存放是随机的，获取表中的数据时没有明确的先后之分，在进行全表扫描的时候，并不是先插入的数据就先获取。而 IOT 表是一个完全 B_tree 索引结构的表，表结构按照索引（主键）有序组织，因此数据存放在插入以前就已经确定了其位置。

由于 IOT 表是把普通表和索引合二而一了,这样在进行查询的时候就可以少访问很多基表的 blocks，但是插入和删除的时，速度比普通的表要慢一些。IOT 表适合做静态表。

IOT 表的叶块存储了所有表列，因为已按主键排序，所以叶子节点上不需要再存储 rowid。



2）溢出段的考虑

表列较多时，可以考虑设置溢出段。将主键和其他字段数据分开来存储以提高效率。

溢出段是个可选项，如果选择了溢出段，Oracle 将为一个 IOT 表分配两个段：一个是索引段，另一个是溢出段。

溢出段有两个子句 pctthreshold 和 including

说明：

pctthreshold 给块大小的百分比，当插入一行数据超出这个阈值时，就以这个阈值将索引 entry 一分为二，包含主键的一部分列值保留在索引段，而其他列值放入溢出段，即 overflow 到指定的存储空间去。

including 后面指定一个列名，将该列之前的列（包括该列）都放入索引段。其余的列在溢出段。

3）示例：

SQL> create table iot_prod(id int, name char(50), sal int, hiredate date, constraint pk_prod primary key (id))

organization index

pctthreshold 30 overflow including sal tablespace test;

索引段在 test 表空间，溢出段在 users 表空间

通过 user_segments 视图查看产生了两个段。

SQL> select tablespace_name,segment_name,segment_type,partition_name from user_segments;

SQL>select TABLE_NAME,IOT_TYPE from user_tables;　查看表类型

13.6 簇表(cluster table)：

1）特点

簇表可以实现多个表共享相同数据块。也可以将单表的具有相同列值的记录组织到同一数据块中，从而提高 IO 效率。

2）建立步骤：

①建立簇段 cluster segment

②基于簇，创建两个相关表，这两个表不建立单独的段，每个表都关联到 cluster segment 上。

③为簇创建索引，生成索引段。

3）示例：

TIM 下

```
SQL> create cluster cluster1(code_key number);

SQL> create table emp1 (empno int,ename char(10),deptno number) cluster cluster1(deptno);

SQL> create table dept1 (deptno number,loc char(10)) cluster cluster1(deptno);

SQL> create index index1 on cluster cluster1;
```

生成了 cluster1 段和 index1 段。

```
SQL> insert into emp1 select empno,ename,deptno from scott.emp;

SQL> insert into dept1 select deptno,loc from scott.dept;

SQL>select dbms_rowid.ROWID_RELATIVE_FNO(rowid) fn, dbms_rowid.rowid_block_number(rowid) bl, rowid,deptno
from emp1;

        FN         BL ROWID                 DEPTNO
---------- ---------- ------------------ ----------
         4        675 AAASO0AAEAAAAKjAAA         30
         4        675 AAASO0AAEAAAAKjAAB         30
         4        675 AAASO0AAEAAAAKjAAC         30
         4        675 AAASO0AAEAAAAKjAAD         30
         4        675 AAASO0AAEAAAAKjAAE         30
         4        675 AAASO0AAEAAAAKjAAF         30
         4        676 AAASO0AAEAAAAKkAAA         10
         4        676 AAASO0AAEAAAAKkAAB         10
         4        676 AAASO0AAEAAAAKkAAC         10
         4        679 AAASO0AAEAAAAKnAAA         20
         4        679 AAASO0AAEAAAAKnAAB         20
         4        679 AAASO0AAEAAAAKnAAC         20
         4        679 AAASO0AAEAAAAKnAAD         20
         4        679 AAASO0AAEAAAAKnAAE         20

14 rows selected.
```

SQL>select dbms_rowid.ROWID_RELATIVE_FNO(rowid) fn, dbms_rowid.rowid_block_number(rowid) bl, rowid,deptno from dept1;

FN	BL	ROWID	DEPTNO
4	675	AAASO0AAEAAAAKjAAA	30
4	676	AAASO0AAEAAAAKkAAA	10
4	678	AAASO0AAEAAAAKmAAA	40
4	679	AAASO0AAEAAAAKnAAA	20

比较一下 scott 下的 emp 表和 dept 表，就明白了。

如果将单个表组成簇表，则 where 后面有相同的谓词值优先保存到同一个块中。

```
select * from emp1 e,dept1 d where e.deptno=d.deptno and e.deptno=10;
```

4）相关操作

①查看簇的信息：

```
select * from user_clusters;
select * from user_clu_columns;
```

②删除簇：

```
drop table emp1
drop table detp1;
drop cluster cluster1;
```

最后删除簇

13.8 临时表

1）特点

Temporary Table 的数据存放在 PGA 或临时表空间下，它被每个 session 单独使用，即隔离 session 间的数据，每个 session 只能看到自己的数据。

①由于临时表使用 PGA，不使用 database buffer cache，，每个 session 是隔离的，，当 PGA 不够用时，使用临时表空间缓存数据。

②基于事务的临时段在事务结束后收回临时段，基于会话的临时段在会话结束后收回临时段，总之没有 DML 锁，没有约束，可以建索引，视图和触发器。

③临时表不需要恢复，所以不记录 redo，但 DML 操作会产生事务，所以有 UNDO 支持 rollback（对 commit 之后 DML），无需共享数据，减少了争用，所以速度快。

2）清除临时数据

①基于事务的临时段：在事务提交时，就会自动删除记录，on commit delete rows(缺省)。

②基于会话的临时段：当用户退出 session 时，才会自动删除记录, on commit preserve rows。

3）示例

scott：

```
SQL>create global temporary table tmp_student(sno int,sname varchar2(10), sage int) on commit preserve rows;
```

再用 Scott 开另一个 session

两边插入记录并提交，可以看到两个 session 各管各的数据，井水不犯河水！

要删除临时表，要所有 session 断开连接，再做删除。

13.9 只读表

在以前有只读表空间但没有只读表。11g 中增加了新特性----只读表。

SQL> alter table t read only;

SQL> update t set id=2;

报错：ORA-12081: 不允许对表 "SCOTT"."T" 进行更新操作

SQL> alter table t read write;

考点：只读表可以 drop，因为只需要在数据字典做标记，但是不能做 DML，另外，truncate 也不行，因为它们都在对只读表做写操作。

13.10 压缩表

1）目的

去掉表列中数据存储的重复值，提高空间利用率。对数据仓库类的 OLAP 有意义（频繁的 DML 操作的表可能不适用做压缩表）

可以压缩：堆表（若指定表空间则压缩该表空间下所有表），索引表，分区表，物化视图。

2）语法及压缩方法

```
{ COMPRESS [ BASIC | FOR { OLTP} ] | NOCOMPRESS }
```

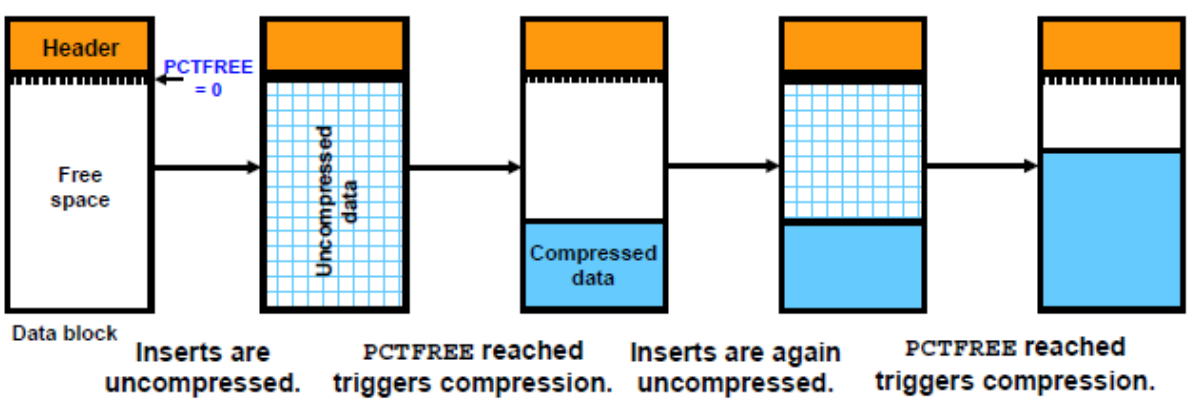
Compression Method	Compression Ratio	CPU Overhead	CREATE and ALTER TABLE Syntax	Typical Applications
Basic table compression	High	Minimal	COMPRESS [BASIC]	DSS
OLTP table compression	High	Minimal	COMPRESS FOR OLTP	OLTP, DSS

Advanced 11gR2 压缩较之前版本在语法上有了变化

①Basic table compression 对应的语法是：

CREATE TABLE ... COMPRESS BASIC;

替换 COMPRESS FOR DIRECT_LOAD OPERATIONS（旧）



TIM@ prod>create table emp1 as select * from scott.emp;

TIM@ prod>Insert into emp1 select * from emp1　　插入记录总是 28672

TIM@ prod>create table emp2 pctfree 0 compress basic as select * from scott.emp where 1=2;

TIM@ prod>insert /*+ append */ into emp2 select * from emp1;　　--直接路径法插入

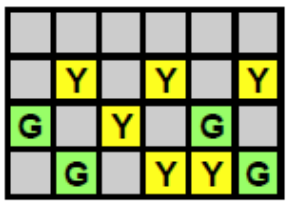
TIM@ prod>analyze table emp1 compute statistics;

TIM@ prod>analyze table emp2 compute statistics;

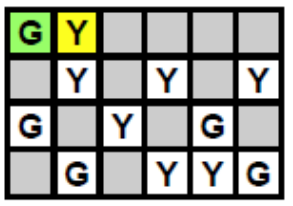
TIM@ prod>select table_name,pct_free,COMPRESSION,COMPRESS_FOR,blocks,empty_blocks,num_rows from user_tables;

TABLE_NAME	PCT_FREE	COMPRESS	COMPRESS_FOR	BLOCKS	EMPTY_BLOCKS	NUM_ROWS
EMP1	10	DISABLED		247	9	28672
EMP2	0	ENABLED	BASIC	50	6	28672

②OLTP table compression 针对 OLTP 的任何 SQL 操作。



Uncompressed block



OLTP compression with symbol table at the beginning of the block

CREATE TABLE ... COMPRESS FOR OLTP

代替 CREATE TABLE ... COMPRESS FOR ALL OPERATIONS（旧）

压缩的是 block 中的冗余数据，这对节省 db buffer 有益。例如一个表有 7 个 columns，5 rows，其中的一些 column 有重复的行值

2190,13770,25-NOV-00,S,9999,23,161

2225,15720,28-NOV-00,S,9999,25,1450

34005,120760,29-NOV-00,P,9999,44,2376

9425,4750,29-NOV-00,I,9999,11,979

1675,46750,29-NOV-00,S,9999,19,1121

压缩这个表后，存储形式成为如下，重复值用符号替代。

2190,13770,25-NOV-00,S,%,23,161

2225,15720,28-NOV-00,S,%,25,1450

34005,120760,*,P,%,44,2376

9425,4750,*,I,%,11,979

1675,46750,*,S,%,19,1121

那么自然要对这些符号做些说明，相当于有个符号表

Symbol	Value	Column	Rows
*	29-NOV-00	3	958-960

% 9999 5 956-960

第三部分 网络、审计、字符集

第十四章：Oracle 网络

14.1 Oracle Net 是什么

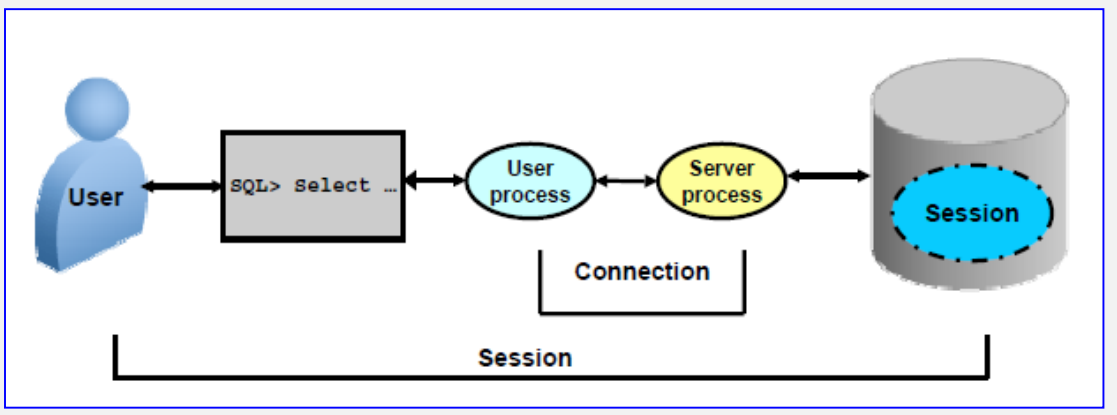
建立在通用的 TCP 和 IPC 之上的网络协议

1）服务器端的 listener (监听器)负责注册的 Oracle 的服务

2）远程客户端通过 Oracle Net 访问 Oracle 服务器端的服务

3）以 TCP 协议为基础的监听器必须描述三个基本要素：

①服务器的 IP 地址或主机名、②端口号、③服务名



14.2 配置文件

Oracle Net 配置文件都是文本形式的，可以通过 netca 或 netmgr 实用程序生成，也可以使用文本编辑器生成和修改。

1） Oracle Net 配置文件的路径

$ORACLE_HOME/network/admin/

2） 三个 Oracle Net 配置文件

①listener.ora 在服务器端的配置文件

②tnsnames.ora 在客户端的配置文件

③sqlnet.ora 描述连接方式的配置文件

14.3 轻松连接方式（ezconnect）

优点：不需要网络配置文件的描述。缺点：登录不方便

连接方法：只要服务器端启动默认的监听器 listener，并数据库打开（动态注册）

远程登录语法为：$Sqlplus 用户/密码@IP 地址:端口号/服务名

C:\Documents and Settings\prod>sqlplus scott/scott@192.168.3.88:1521/prod

14.4 动态注册

1）动态注册要点

①实例启动后，pmon 每分钟自动将服务名（service_name）注册到监听器中

也可以通过 alter system register 命令通知 pmon 立刻注册

②系统有一个默认的监听器叫做 LISTENER，端口号是 1521，如果利用它就可以不需要 listener.ora 配置文件

③动态注册要求实例至少启动到 mounted 状态，listener 监听器才能注册成功

2）如果不想使用默认的 listener 监听器，可以自定义一个动态监听器。需要下面完成两步操作：

①在 listener.ora 配置文件描述自定义监听器。如:

```
LSN1=
   (DESCRIPTION=
      (ADDRESS_LIST=
         (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.3.88)(PORT=1555))))
```

②更改 local_listener 参数

```
SQL> alter system set local_listener='(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.3.88)(PORT=1555))'
```

14.5 静态注册

1）静态注册要点

①静态注册必须在 listener.ora 中描述

②实例不必启动，静态监听器也能注册

③服务器启动静态监听后，可以通过远程启动数据库

2）静态注册的描述分为两部分内容

①网络三要素：①Protocal　②Host　③Port

②服务名描述：GLOBAL_DBNAME：全局数据库名（静态注册特征）

3）listener.ora 配置示例

```
LSN2 =
   (DESCRIPTION_LIST =
      (DESCRIPTION =
         (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.3.88)(PORT = 1522))
      )
   )

SID_LIST_LSN2 =
      (SID_LIST =
      (SID_DESC =
            (GLOBAL_DBNAME=prod )
            (ORACLE_HOME = /u01/oracle)
            (SID_NAME =prod)
```

```
        )
)
```

14.6 客户端配置文件 tnsnames.ora

1）可以描述登录多个服务器

2）示例

```
prod=
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.3.88)(PORT = 1522))
    )
    (CONNECT_DATA =
      (SERVICE_NAME =prod)
    )
  )
```

14.7 理解监听器

1) 实例以 service 的形式对外提供服务，监听器负责注册 service_name 。

2) Service_name 对外屏蔽了实例和数据库的复杂描述。

3) 客户端 tnsnames.ora 的 SERVICE_NAME 与服务器端 listener.ora 的 GLOBAL_DBNAME 等同。

4) listener.ora 中监听器的 SID_NAME 与实例的 SID 等同。

5) 一个$ORACLE_HOME 下只能对应一个 listener.ora 和 tnsnames.ora

6) 一个 listener.ora 可以描述多个监听器，一个监听器可以描述多个端口。

7) 一个 tnsnames.ora 可以描述多个网络连接符。

8) 一个实例可以对应多个 service_name。

14.8 共享连接配置：

14.8.1 sharded server mode 的几个参数

①dispatchers 调度进程数，一个 despatcher process 理论上可以支持 256 个 user process 请求。

②max_dispatchers dispatcher process 最大数量，dispatchers<=max_dispatchers

③shared_servers 共享服务器进程数，与 dispatcher process 数有关， 最大不超过 max_shared_servers

④max_shared_servers 所有 session 可使用的共享进程的最大值。

14.8.2 示例

1）要求 session 登录总数限制在 300 个以内，其中专用连接 100 个，共享连接 200 个。

SQL> alter system set sessions=300 scope=spfile;

SQL> alter system set shared_server_sessions=200;　　共享 user process 数量

SQL> alter system set processes=150 scope=spfile;　　服务器端 process 总数量

注：processes=专用连接后+后台进程数+（100*10%）约=150

2）要求分配 3 个 dispatcher，最大 dispatchers 能支持 10 个

SQL> alter system set dispatchers='(protocol=tcp)(dispatchers=3)';

SQL> alter system set max_dispatchers=10;

共享 server 为 10，最大共享 server 能支持 30 个

SQL> alter system set shared_servers=10;

SQL> alter system set max_shared_servers=30;

SQL> startup force;

SQL> show parameter dispatcher

NAME　　TYPE　　VALUE

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

dispatchers　　string　　(protocol=tcp)(dispatchers=3)

max_dispatchers　　integer　　10

SQL> show parameter shared_server

NAME　　TYPE　　VALUE

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

max_shared_servers　　integer　　30

shared_server_sessions　　integer　　200

shared_servers　　integer　　10

查看状态：

[oracle@prod ~]$ ps -ef |grep ora_d0

[oracle@prod ~]$ ps -ef |grep ora_s0

3）配置并启动两个监听器，

listener：　端口 1521，动态注册，支持共享连接模式（支持 200 个 user process）

lsn2：端口 1522，静态注册，支持专用连接模式（支持 100　个 user process）

配置 listener.ora

```
LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.3.88)(PORT = 1521))
  )

LSN2 =
  (DESCRIPTION =
```

```
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.3.88)(PORT = 1522))
  )

SID_LIST_LSN2 =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = prod)
      (ORACLE_HOME = /u01/oracle)
      (SID_NAME = prod)
    )
  )
```

配置 tnsnames.ora 如下：

```
prod_s=
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.3.88)(PORT = 1522))
    )
        (CONNECT_DATA =
          (SERVER = DEDICATED)
    (SERVICE_NAME =prod)
    )
  )

prod_d=
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.3.88)(PORT = 1521))
    )
    (CONNECT_DATA =
    (SERVER = SHARED)
    (SERVICE_NAME =prod )
    )
  )
```

注意：

①(SERVER = DEDICATED):　　表示此连接使用 DEDICATED SERVER MODE　缺省的配置。

②(SERVER = SHARED):　表示此连接使用 SHARED SERVER MODE，如果服务器端未配置 dispatchers，此连接失败

③ 不做任何说明：　如果服务器端配置了 dispatchers，SHARED SERVER MODE 优先。

4）启动 listener 监听器，并查看动态注册是否成功

$ lsnrctl start

$ lsnrctl status

启动 lsn2 监听器，并查看静态注册是否成功

$ lsnrctl start lsn2

$ lsnrctl status lsn2

5）远程连接，依次以两个 session 登录，

验证走专用连接

Session1:

C:\Users\Administrator>sqlplus system/oracle@prod_s

SQL>select sid from v$mystat where rownum=1;　得到当前 session 的 sid

SID

46

SQL> select sid,server,status from v$session where sid=46;

SID SERVER　STATUS

--- --- ---

46 DEDICATED　ACTIVE

验证走共享连接

Session2:

C:\Users\Administrator>sqlplus system/oracle@prod_d

SQL>select sid from v$mystat where rownum=1;　得到当前 session 的 sid

SID

37

SYS>select sid,server,status from v$session where sid=37;

SID SERVER　STATUS

--- --- ---

37 SHARED　ACTIVE

SQL> select sid,server,status from v$session where sid in (46,37);

SID SERVER　STATUS

--- --- ---

46 DEDICATED INACTIVE

37 SHARED INACTIVE

第十五章：数据库审计 audit

15.1 功能和类别：

1）功能：监控特定用户在 database 的 action（操作）

2）审计种类：

⑴标准数据库审计（①语句审计、②权限审计、③对象审计）

⑵基于值的审计（Value-Based，触发器审计）

⑶精细度审计（FGA）

15.2 启用审计（默认不启用）

1）有关参数

SQL> show parameter audit

NAME TYPE VALUE

---

audit_file_dest string /u01/admin/prod/adump

audit_sys_operations boolean FALSE

audit_syslog_level string

audit_trail string DB

SQL>

audit_trail 参数主要选项

1）none 不启用 audit

2）db 将审计结果放在数据字典基表 sys.aud$中，(一般用于审计非 sys 用户，也可以移出 system 表空间，在 system 表空间中的好处是方便检索）

还有一种 db 扩展，即 db,extended，可以包括绑定变量，CLOB 类型大对象等审计信息（考点）。

3）os 将审计结果存放到操作系统的文件里（由 audit_file_dest 指定的位置，一般用于审计 sys）

如何审计 sys

1）Oracle 强制审计 sys 用户的特权操作，如启动和关闭数据库，结果记录在参数 audit_file_dest 指向的.aud 文件中。

2）指定参数 audit_sys_operations = true 和 audit_trail = os

15.3 标准数据库审计的三个级别

查看标准审计结果可以通过视图 dba_audit_trail，该视图读取 aud$内容。

1）语句审计

按语句来审计，比如 audit table 会审计数据库中所有的 create table, drop table, truncate table 语句，执行成功或不成功都可审计。

SQL> audit table;

2）权限审计

按权限来审计，当用户使用了该权限则被审计，如执行 grant select any table to a; 当用户 a 访问了用户 b 的表时（如 select * from b.t;）

会用到 select any table 权限，故会被审计。用户访问自己的表不会被审计。

SQL> audit select any table;

3）对象审计

按对象审计，只审计 on 关键字指定对象的相关操作，如：aduit alter,delete,drop,insert on cmy.t by scott; 这里会对 cmy 用户的 t 表进行审计，但同时使用了 by 子句，所以只会对 scott 用户发起的操作进行审计。

SQL> audit update on scott.emp1 by access;

15.4 基于值的审计。

它拓展了标准数据库审计，不仅捕捉审计事件，还捕捉那些被 insert,update 和 delete 的值。由于基于值的审计是通过触发器来实现。所以你可以选择哪些信息进入审计记录，比如，只记录提交的信息，或不记录已改变的数据等。

15.5 精细审计 Fine Grained Auditing (FGA)。

15. 5. 1 特点：

拓展了标准数据库审计，捕捉准确的 SQL 语句。审计访问特定行或特定列。操作可以使用 dbms_fga 包。精细审计一般不包括 sys 用户，目前 EM 中只有标准数据库审计，还没有包括基于值的审计和精细审计。

15. 5. 2 示例：

sys:

SQL> create table scott.emp1 as select * from scott.emp;

SQL> grant all on scott.emp1 to tim;

1）添加一个精细度审计策略

```
SQL>begin
dbms_fga.add_policy(
object_schema=>'scott',
object_name=>'emp1',
policy_name=>'chk_emp1',
audit_condition =>'deptno=20',
audit_column =>'sal',
statement_types =>'update,select');
end;
/
```

2）测试

scott:

SQL> select * from emp1 where deptno=20;

tim:

SQL>update scott.emp1 set sal=8000 where empno=7902;

SQL>select empno,ename from scott.emp1 where deptno=20;　　缺少 sal 列，不审计

sys:

SQL> select empno,ename,sal from scott.emp1 where deptno=20;　　虽然符合条件，但默认不审计 sys

3）验证审计结果

SQL> conn /as sysdba

SQL>select db_user,to_char(timestamp,'yyyy-mm-dd hh24:mi:ss') "time" ,sql_text

from dba_fga_audit_trail;

DB_USER　　time　　SQL_TEXT

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

SCOTT　　2013-08-17 16:57:36　　select * from emp1 where deptno=20

TIM　　2013-08-17 16:57:52　　update scott.emp1 set sal=8000 where empno=7902

可以看出，必须同时满足了所有审计条件（前面定义的）才能入选。另外没有审计 SYS.

SQL> truncate table fga_log$;　　清除审计记录

4）删除 FGA 策略

SQL>exec dbms_fga.drop_policy(object_schema=>'scott',object_name=>'emp1',policy_name=>'chk_emp1');

第十六章：全球化特性与字符集

数据库的全球化特性是数据库发展的必然结果，位于不同地区、不同国家、不用语言而使用同一数据库越来越普遍。Oracle 数据库提供了对全球化数据库的支持，消除不同文字、语言环境、历法货币等所带来的差异、使得更容易、更方便来使用数据库。

16.1 全球化特性内容

Language support

Territory support

Character set support

Linguistic sorting

Message support

Date and time formats

Numeric formats

Monetary formats

16.2 字符集概念

在 Oracle 全球化特性中最重要的则是字符集。主要是讨论两个问题：

一是字符如何存储　　服务器端

二是字符如何显示　客户端

比如单个英文字符、单个阿拉伯数据字，#、$等，美国 ANSI 使用的标准字符集则使用了一个单字节(7 位)来表示。由于世界各国和各个地区使用的符号的多样性，仅有 2 的 7 次方（128）个单字节的码点是不够用的，因此就有需要多字节来表示各自不同的字符。

正是由于上述原因产生了不同的字符集的概念，如美国使用的为 US7ASCII，西欧则使用的是 WE8ISO8859P1，中国则是 ZHS16GBK。

为了统一世界各国字符编码，统一编码字符集的概念应运而生，这就是 Unicode。

在 Oracle 中，几种常用的 Unicode 为 UTF-8，AL16UTF16，AL32UTF8

16.3 字符集及分类

Oracle 支持两百多种字符集，包含了单字节、可变字节以及通用字符集等。

字符集通常根据使用的字节数来分类,主要分为以下几类

①单字节字符集，如 US7ASCII(7bit)，WE8ISO8859P1(8bit)，WE8DEC(8bit)

②可变长多字节字符集，如 JEUC，CGB2312-80

③固定长多字节字符集，`

16.4 Unicode 字符集

1）数据库字符集和国家字符集特性

Database Character Sets

主要是用作描述字符如何保存。

可存储列的类型为 CHAR,VARCHAR2,CLOB,LONG

National Character Set

主要是用于辅助 Database Character Set。因为早期的数据库中很多使用了单字节字符集，但随着业务的发展，需要使用到诸如 nchar,nvarchar 等 Unicode 字符或者需要扩展到世界各地存储不同的字符，因此辅助字符集应运而生。

可存储的类型为 NCHAR,NVARCHAR2,NCLOB

2）Oracle 支持的 Unicode 字符集

Character Set	Unicode Encoding	Database Character Set	National Character Set
UTF8	UTF-8	Yes	Yes(Oracle 9i and 10g only)
AL32UTF8	UTF-8	Yes	No
AL16UTF16	UTF-16	No	Yes

3) 字符集影响的数据类型

对于二进制数据类型，字符集的设置不影响该类型数据的存储，如视频、音频等

受影响的数据类型为：char,varchar2,nchar,nvarchar2,blob,clob,long,nclob

16.5 NLS 参数设定

1）服务器端：查看服务器中设定的 NLS 参数

SQL>select * from nls_database_parameters;

PARAMETER	VALUE
NLS_LANGUAGE	AMERICAN
NLS_TERRITORY	AMERICA
NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA
NLS_NUMERIC_CHARACTERS	.,
NLS_CHARACTERSET	ZHS16GBK
NLS_CALENDAR	GREGORIAN
NLS_DATE_FORMAT	DD-MON-RR
NLS_DATE_LANGUAGE	AMERICAN
NLS_SORT	BINARY
NLS_TIME_FORMAT	HH.MI.SSXFF AM
NLS_TIMESTAMP_FORMAT	DD-MON-RR HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT	HH.MI.SSXFF AM TZR
NLS_TIMESTAMP_TZ_FORMAT	DD-MON-RR HH.MI.SSXFF AM TZR
NLS_DUAL_CURRENCY	$
NLS_COMP	BINARY
NLS_LENGTH_SEMANTICS	BYTE
NLS_NCHAR_CONV_EXCP	FALSE
NLS_NCHAR_CHARACTERSET	AL16UTF16
NLS_RDBMS_VERSION	11.1.0.6.0

已选择 20 行。

2）客户端：查看环境变量中的 NLS_LANG 变量

NLS_LANG 变量为一个总控参数，控制了变量 nls_language 和 nls_territory 的行为，只要设定了该参数，则其它参数就相应确定了。

该参数的格式为：

NLS_LANG = language_territory.charset

在我们的虚机环境下，环境变量文件/home/oracle/.bash_profile 中描述了作为客户端的 NLS_LANG

NLS_LANG="simplified chinese"_china.ZHS16GBK，该参数分为几个部分来设定

第一部分为 language，为 simplified chinese。

第二部分为 territory，为 china。一二两部分必须用下划线连接。

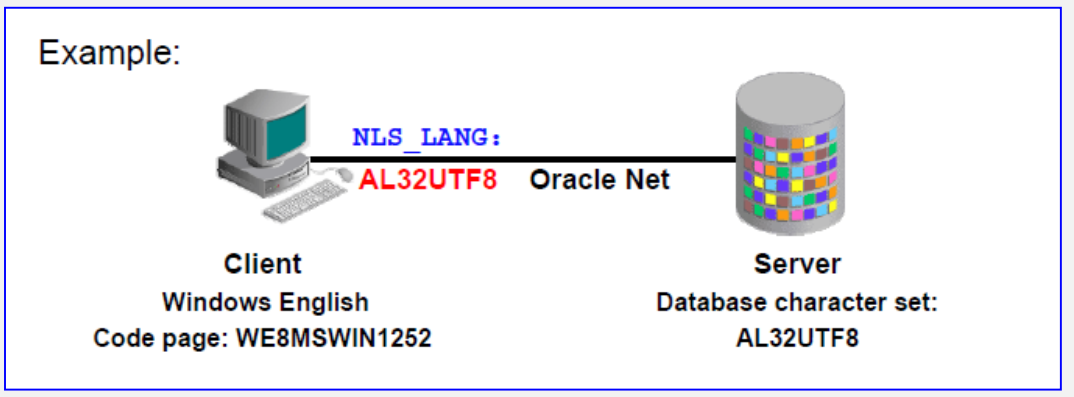
第三部分为 character set，为 zhs16gbk 二三两部分必须用小数点连接。

其含义是语言是简体中文，区域是中国，数据库字符集是 ZHS16GBK。

另外日期格式缺省的是 DD-MON-RR，我们单独定义了适合中国人使用的格式'YYYY-MM-DD HH24:MI:SS'

注意：当设置客户端的 NLS_LANG 字符集时必须考虑客户端的操作系统字符集

如果它们不一致，可能会造成无效的数据进入 SERVRE 数据库。见下图。



3）几个重要的参数：

①语言参数，nls_language：

受影响的参数有：

NLS_DATE_LANGUAGE

NLS_SORT

②区域参数，nls_territory：

受影响的参数有：

NLS_CURRENCY

NLS_DUAL_CURRENCY

NLS_ISO_CURRENCY

NLS_NUMERIC_CHARACTERS

NLS_DATE_FORMAT

NLS_TIMESTAMP_FORMAT

NLS_TIMESTAMP_TZ_FORMAT

③排序参数：nls_sort：

Order by 指定字段的排序方法，缺省的是 Binary，一般是支持单字节字符集 而多字节的字符集排序则引入 Linguistic Sorting

基于 Binary 排序，根据 encode 的二进制代码排序。

基于语言排序，又分单一语言和多重语言

16.6 更改数据库字符集

```
SQL> conn /as sysdba
SQL> shutdown immediate;
SQL> startup mount
SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;
SQL> ALTER SYSTEM SET JOB_QUEUE_PROCESSES=0;
SQL> ALTER SYSTEM SET AQ_TM_PROCESSES=0;
```

SQL> alter database open;

SQL> ALTER DATABASE CHARACTER SET ZHS16GBK;

ERROR at line 1:

ORA-12712: new character set must be a superset of old character set

新字符集必须为旧字符集的超集，这时我们可以跳过超集的检查做更改：

SQL> ALTER DATABASE character set INTERNAL_USE ZHS16GBK;

SQL> shutdown immediate

SQL> startup

第四部分 数据仓库管理

数据仓库以 OLAP 类型操作为主，这有别于 OLTP 类型的操作。

OLTP 体现的实时的事务处理，OLAP 可以看成是 OLTP 的历史数据“仓库”

OLAP 操作上主要体现为：

1）select 查询汇总为主，对事务性要求较少

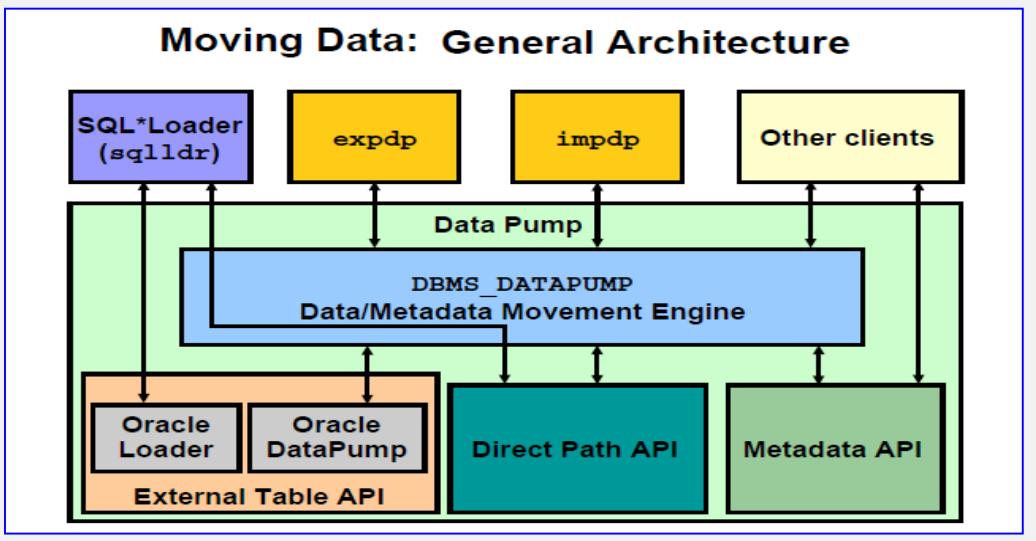
2）对数据快速复制、移动的需求

3）分布式查询的需求。

数据仓库的概念并不抽象，而且可操作性强，所以第四部分我们将以实操为主。

第十七章：数据移动

17.1 概念



1）数据移动源于数据仓库，它是逻辑对象层面的数据复制，数据移动有两种引擎：

①ORACLE_LOADER（Sqlload 引擎）

②ORACLE_DATAPUMP（数据泵引擎）

两个引擎的区别是：ORACLE_DATAPUMP 只能读取由它导出的文件，而 ORACLE_LOADER 可以读取任何它能解析的第三方文件格式。

2）数据移动主要包含两个方面内容

⑴创建外部表的方法，两种引擎都可以生成外部表数据。但用途和方法是不同的。

①Sqlload 引擎生成的外部表是文本格式的，支持跨平台的不同数据库间的数据移动。

②Data pump 引擎生成的外部表是二进制格式的。适用于 Oracle 平台的数据库之间快速数据移动。

⑵数据泵技术(expdp/impdp)

作为替代传统逻辑备份的导入导出，实现数据在逻辑层面的快速复制与恢复

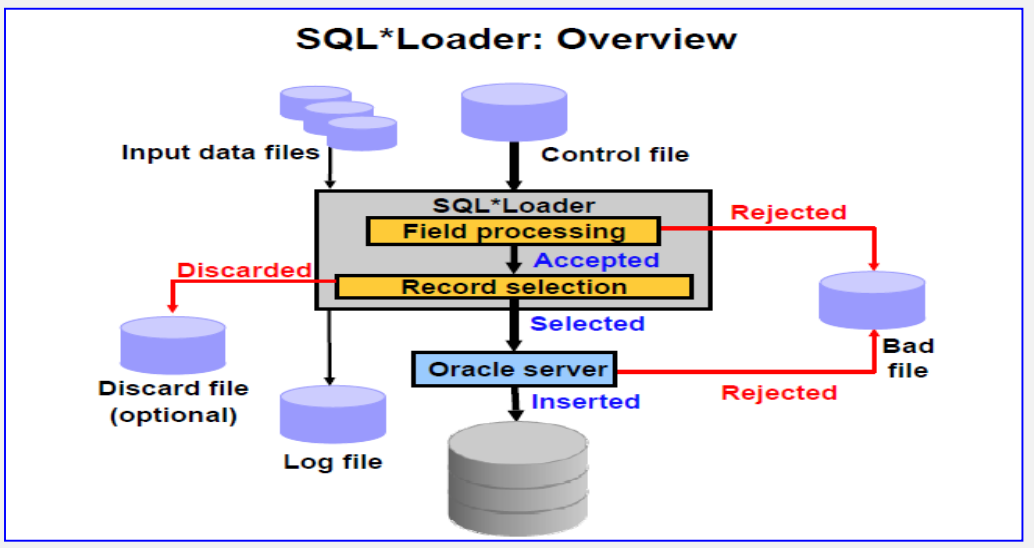
17.2 Directory(目录)

①创建外部表必须使用 Directory 指定外部表的目的地，目录是数据库对象，相当于把物理目录映射成一个逻辑目录名

②引入 directory 的好处是简化了在不同 OS 中对于物理目录路径的格式描述

③通过 Sqlload 和 Data pump 两种方法创建外部表时都必须使用指定的 directory

17.3 sql*loader



17.3.1 sql*loader 原理：

1）将外部数据（比如文本型）导入 oracle database。（可用于不同类型数据库数据迁移）

本质是在段（segment 表）重新 insert 记录

①conventional：将记录优先插入到 segment 的 HWM(高水位线）以下的块，要首先访问 bitmap ，来确定那些 block 有 free space

②direct path：将记录直接插入到 segment 的 HWM(高水位线）以上的从未使用过的块，绕过 db_buffer，不检查约束。还可以关闭 redo，也支持并行操作，加快插入速度。

传统插入数据和直接插入数据：

SQL> create table emp1 as select * from emp where 1=2;

SQL> insert into emp1 select * from emp;　　传统方式数据

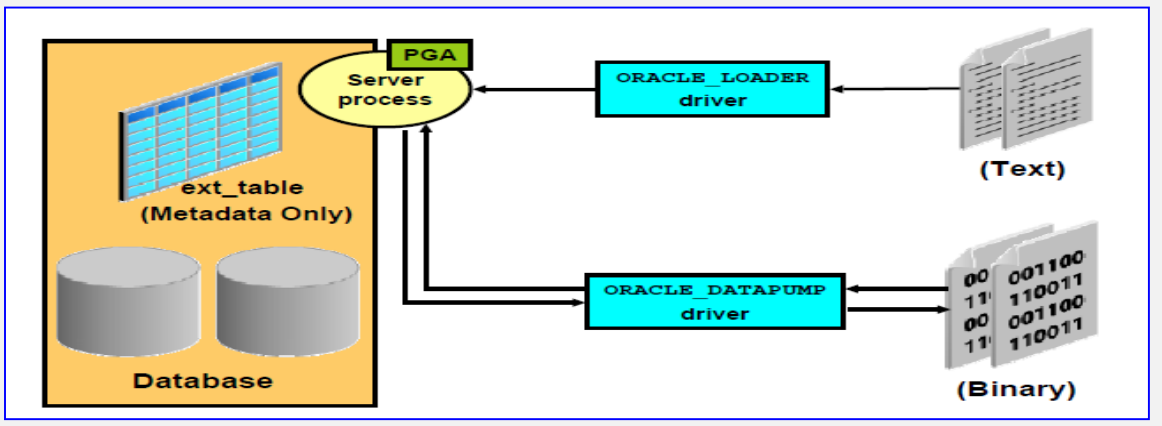
SQL> insert /*+ APPEND */ into emp1 select * from emp;　　直接方式数据，必须 commit 后才能查看数据

17.3.3 sql*loader 用法：

SQLLDR keyword=value [,keyword=value,...]

可以看帮助信息$/u01/oracle/bin/sqlldr(回车）

17.4 外部表示例



17.4.1 使用 ORACLE_LOADER 引擎建立外部表

步骤 1，模拟生成数据源

SQL>select empno||','||ename||','||sal||','||deptno from scott.emp;

步骤 2，建立目录，生成平面表（数据源）

$mkdir -p /home/oracle/dir1

$vi /home/oracle/dir1/emp1.dat　　粘贴步骤 1 的查询结果

步骤 3，建立 directory

名称为 dir1 指向物理目录/home/oracle/dir1,

SQL>create directory dir1 as '/home/oracle/dir1';

将 dir1 的对象权限授予 scott 和 tim 用户。

SQL>grant read,write on directory dir1 to scott,tim;

步骤 4，使用 ORACLE_LOADER 引擎创建外部表 emp1_ext

scott:

```
CREATE TABLE emp1_ext
      (EMPNO          NUMBER(4),
       ENAME           VARCHAR2(10),
       SAL                  NUMBER(7,2),
       DEPTNO        NUMBER(2))
   ORGANIZATION EXTERNAL
       (TYPE ORACLE_LOADER
        DEFAULT DIRECTORY dir1
            ACCESS PARAMETERS (FIELDS TERMINATED BY ",")
               LOCATION ('emp1.dat')
                  ) REJECT LIMIT UNLIMITED;
```

步骤 5，验证外部表

SQL> select * from emp1_ext;

REJECT LIMIT UNLIMITED 的意思是 select 时剔除不合格的行，而不会报错“ORA-30653: 已达到拒绝限制值”。

17.4.2 使用 ORACLE_DATAPUMP 引擎导出导入外部表

步骤 1，为 scott 用户建立外部表 emp2_ext

数据源是 emp2.dmp 文件，逻辑目录是 dir1。

```
SQL> CREATE TABLE emp2_ext
ORGANIZATION EXTERNAL
( TYPE ORACLE_DATAPUMP
   DEFAULT DIRECTORY dir1
   LOCATION ('emp2.dmp'))
AS SELECT empno,ename,sal,deptno FROM scott.emp ;
```

步骤 2，验证 scott 的外部表 emp2_ext

```
SQL> select * from scott.emp2_ext
```

步骤 3，为 tim 用户建立外部表 emp3_ext，同样读取数据源 emp2.dmp

tim:

```
SQL> CREATE TABLE emp3_ext
    (EMPNO      NUMBER(4),
     ENAME      VARCHAR2(10),
     SAL    NUMBER(7,2),
     DEPTNO NUMBER(2))
ORGANIZATION EXTERNAL
(
   TYPE ORACLE_DATAPUMP
   DEFAULT DIRECTORY dir1
   LOCATION ('emp2.dmp')
) ;
```

步骤 4，验证 tim 的外部表 emp3_ext

```
SQL>select * from tim.emp3_ext
```

17.4.3 使用 sqlldr 将 emp1.dat 导入到 scott 下的 emp1

步骤 1，建立 sqlldr 控制文件

```
$ vi /home/oracle/dir1/emp1.ctl
load data
infile '/home/oracle/dir1/emp1.dat'
insert                    --insert  插入表必须是空表，非空表用 append
into table emp1
fields terminated by ','
optionally enclosed by '"'
(empno,ename,sal,deptno)
```

步骤 2，在 scott 下建立 emp1 表（内部表结构），只要结构不要数据

SQL> create table scott.emp1 as select empno,ename,sal,deptno from scott.emp where 1=2;

步骤 3，ORACLE_LOADER 引擎导入（normal 方式）

$ cd /home/oracle/dir1

$ sqlldr scott/scott control=emp1.ctl log=emp1.log

步骤 4，验证结果

SQL> select * from scott.emp1;

上例的另一种形式是将数据源和控制文件合并在.ctl 里描述

[oracle@work sqlldr]$ vi emp.ctl

```
load data
infile *
append
into table emp1
fields terminated    by     ','
optionally enclosed by '"'
(empno,ename,sal,deptno)
begindata
7369,SMITH,800,20
7499,ALLEN,1600,30
7521,WARD,1250,30
```

[oracle@prod sqlload]$ sqlldr scott/scott control=emp.ctl log=emp.log

第十八章：逻辑备份（导出）与恢复（导入）

18.1 传统的导入导出 exp/imp：

18. 1. 1 概述

传统的导出导入程序指的是 exp/imp,用于实施数据库的逻辑备份和恢复。

导出程序 exp 将数据库中对象的定义和数据备份到一个操作系统二进制文件中。

导入程序 imp 读取二进制导出文件并将对象定义和数据载入数据库中

18. 1. 2 特点

传统的导出导入是基于客户端设计的。在$ORACLE_HOME/bin 下

导出和导入实用程序的特点有：

1)可以按时间保存表结构和数据

2)允许导出指定的表，并重新导入到新的数据库中

3)可以把数据库迁移到另外一台异构服务器上

4)在两个不同版本的 Oracle 数据库之间传输数据（客户端版本不能高于服务器版本）

5)在联机状态下进行备份和恢复

6)可以重新组织表的存储结构，减少行迁移及磁盘碎片

18.1.3 交互方式

使用以下三种方法调用导出和导入实用程序：

1，交互提示符：以交互的方式提示用户逐个输入参数的值。

2，命令行参数：在命令行指定执行程序的参数和参数值。

3，参数文件：允许用户将运行参数和参数值存储在参数文件中，以便重复使用参数

18.1.4 导入导出模式

导出和导入数据库对象的四种模式是：

1，数据库模式：导出和导入整个数据库中的所有对象

2，表空间模式：导出和导入一个或多个指定的表空间中的所有对象，10g 新增添可传输表空间。

3，用户模式：导出和导入一个用户模式中的所有对象

4，表模式：导出和导入一个或多个指定的表或表分区

18.2 导入导出示例

18.2.1 导入导出表

1）scott 导入导出自己的表，一般是从服务器导出到客户端(在 cmd 下操作)

SQL>create table emp1 as select * from emp;

SQL>create table dept1 as select * from dept;

C:\>exp scott/scott@prod file=d:empdept1.dmp tables=(emp1,dept1)

再导入 server 里 QL> drop table emp1 purge;

SQL> drop table dept1 purge;

C:\>imp scott/scott@prod file=d:empdept1.dmp

2）sys 导出 scott 表，

SYS 用户可以 exp/imp 其他用户的 object,是因为 SYS 含有 EXP_FULL_DATABASE 和 IMP_FULL_DATABASE 角色。

C:\>exp 'sys/system@prod as sysdba' file=d:sysscott.dmp tables=(scott.emp1,scott.dept1)

scott 导入（报错）

C:\>imp scott/scott@prod file=d:sysscott.dmp

报错：IMP-00013: 只有 DBA 才能导入由其他 DBA 导出的文件

IMP-00000: 未成功终止导入

C:\>imp 'sys/system@prod as sysdba' file=d:sysscott.dmp fromuser=scott

18.2.2 导入导出用户

当前用户 scott 导出自己的所有对象，注意仅仅导出的是 schema 的 object，也就是说这个导出不包括数据字典中的信息，比如用户账户，及原有的一些系统权限等等。

C:\>exp scott/scott@prod file=d:scott.dmp owner=scott 所有 segment name 的表才能导出，注意 deferred_segment_creation 问题

SQL> drop user scott cascade;

SQL> grant connect,resource to scott identified by scott;

C:\>imp scott/scott@prod file=d:scott.dmp

如果用 sys 来完成也可以使用如下命令：

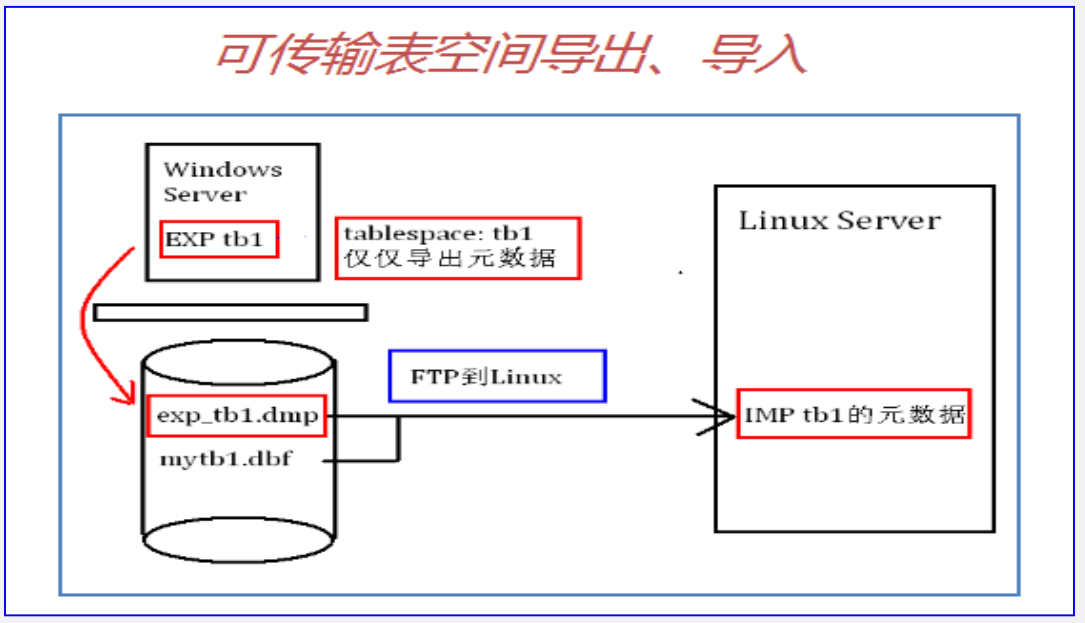
C:\>imp 'sys/system@prod as sysdba' file=d:scott.dmp fromuser=scott touser=scott

sys 用户也可以将导出的 scott 的内容导入给其他用户　必须是 sys 导出的

C:\>imp 'sys/system@prod as sysdba' file=d:scott.dmp fromuser=scott touser=tim

18. 2. 3 导入导出表空间

Oracle10g 后，引入了导入导出可传输表空间技术，使表空间的迁移更加快速高效



模拟场景：xp/orcl 到 linux/(中文字符集）可传输表空间的导入导出：

1）在 xp/orcl 上建立表空间

sys：

SQL>create tablespace tb1 datafile 'd:/mytb1.dbf' size 5m;

scott:

create table t1(year number(4),month number(2),amount number(2,1)) tablespace tb1;

insert into t1 values(1991,1,1.1);

insert into t1 values(1991,2,1.2);

insert into t1 values(1991,3,1.3);

insert into t1 values(1991,4,1.4);

commit;

2）导出 tb1 表空间,先设为只读；

sys:

SQL>alter tablespace tb1 read only;

xp:cmd 下

C:\>exp '/ as sysdba' tablespaces=tb1 transport_tablespace=y file=d:\exp_tb1.dmp

3）以 xmanager 把 exp_tb1.dmp 和 MYTB1.DBF 都上传到 linux/prod 里

目录如下：/u01/oradata/prod

4）在 linux 的$下执行导入

[oracle@prod ~]$ imp userid=\'/ as sysdba\' tablespaces=tb1 transport_tablespace=y file=/u01/oradata/prod/exp_tb1.dmp datafiles=/u01/oradata/prod/MYTB1.DBF

5）进入 linux/prod 下验证

sys:

SQL>select tablespace_name,status from dba_tablespaces;

SQL>select * from scott.t1;

6）重设回读写方式

SQL>alter tablespace tb1 read write;

说明：可传输表空间需要满足几个前提条件：

①原库和目标库字符集要一致。

②字符序有大端（big endian)和小端（little endian)之分，通过 v$transportable_platform 查看，如果不一致可以使用 rman 转换。

③compatible 10.0.0.或更高。

④迁移的表空间要自包含 （self contained）。

什么叫自包含：

当前表空间中的对象不依赖该表空间之外的对象。

例如：有 TEST 表空间,里面有个表叫 T1，如果在 T1 上建个索引叫 T1_idx,而这个索引建在 USERS 表空间上，由于 T1_idx 索引是依赖 T1 表的，

那么，TEST 表空间是自包含的，可以迁移，但会甩掉 T1_idx 索引，USERS 表空间不是自包含的，不符合迁移条件。

检查表空间是否自包含可以使用程序包

如上面的例子

SQL> execute dbms_tts.transport_set_check('TEST');

SQL> select * from TRANSPORT_SET_VIOLATIONS;

VIOLATIONS

ORA-39907: 索引 SCOTT.EMP1_IDX (在表空间 TEST 中) 指向表 SCOTT.EMP1 (在表空间 USERS 中)。

18. 2. 4 导出整个数据库

C:\>exp 'sys/system@prod as sysdba' file=d:full.dmp full=y

18.3 数据泵技术

18. 3. 1 **数据泵优点：**

1）（较传统的 exp/imp 速度提高 1-2 个数量级）

2）重启作业能改进性能力

3）并行执行能力

4）关联运行作业能力

5）估算空间需求能力

6）操作网络方式

18. 3. 2 数据泵组成部分：

①数据泵核心部分程序包：DBMS_DATAPUMP

②提供元数据的程序包：DBMS_MATADATA

③命令行客户机（实用程序）：EXPDP,IMPDP

18. 3. 3 数据泵文件：

①转储文件：此文件包含对象数据

②日志文件：记录操作信息和结果

③SQL 文件：将导入作业中的 DDL 语句写入 SQLFILE 指定的参数文件中

18. 3. 4 数据泵的目录及文件位置

以 sys 或 system 用户完成数据泵的导入导出时，可以使用缺省的目录 DATA_PUMP_DIR

SQL> select * from dba_directories;

如果设置了环境变量 ORACLE_BASE,则 DATA_PUMP_DIR 缺省目录位置是：

$ORACLE_BASE/admin/database_name/dpdump

否则是：

$ORACLE_HOME/admin/database_name/dpdump

18.4 数据泵示例

18. 4. 1

①使用数据泵默认的 directory

SYS@ prod>select * from dba_directories where directory_name='DATA_PUMP_DIR';

OWNER	DIRECTORY_NAME	DIRECTORY_PATH
SYS	DATA_PUMP_DIR	/u01/admin/prod/dpdump/

②为 scott 授予目录权限

SQL> grant read,write on directory DATA_PUMP_DIR to scott,tim;

18. 4. 2 数据泵导表

1）导出 scott 的 emp dept 表，导出过程中在 server 端有 MT 表出现 SYS_EXPORT_TABLE_01，导出完成后 MT 表自动消失

$expdp scott/scott directory=DATA_PUMP_DIR dumpfile=expdp_scott1.dmp tables=emp,dept

看看目录下的导出的文件

导入 scott 的表（导出的逆操作）

$impdp scott/scott directory=DATA_PUMP_DIR dumpfile=expdp_scott1.dmp

2）导出 scott 的 emp1 的数据，但不导出结构

$expdp scott/scott directory=DATA_PUMP_DIR dumpfile=expdp_scott1.dmp tables=emp1 content=data_only reuse_dumpfiles=y

导入 Scott 的表（导入的逆操作），只导入数据

SQL> truncate table emp1;

$impdp scott/scott directory=DATA_PUMP_DIR dumpfile=expdp_scott1.dmp tables=emp1 content=data_only

将 scott.dept 导入 tim 中

$ impdp userid=\'/ as sysdba\' directory=data_pump_dir dumpfile=empdept.dmp tables=scott.dept remap_schema=scott:tim

18. 4. 3 数据泵导用户

$ expdp system/oracle directory=DATA_PUMP_DIR dumpfile=scott.dmp schemas=scott

注意与 exp 的区别，schemas 代替了 owner 的写法

然后将所有对象导入 tim

$ impdp system/oracle directory=DATA_PUMP_DIR dumpfile=scott.dmp remap_schema=scott:tim

这步会重建 tim 用户，密码继承 scott 的密码，如果 tim 用户已存在，会有警告，可忽略。

18. 4. 4 数据泵可传输表空间（适用于大规模数据迁移）

①Win 端需要 directory 已经建立了 dir1

C:\>expdp '/ as sysdba' directory=dir1 dumpfile=tb1.dmp transport_tablespaces=tb1

②上传文件将 TB1.DMP 放入 linux 的/u01/admin/prod/dpdump/，MYTB1.DBF 放入数据库目录/u01/oradata/prod/下。

③Linux 端需要 directory 本例使用默认的数据泵 directory

$impdp userid=\'/ as sysdba\'

DIRECTORY=DATA_PUMP_DIR DUMPFILE='TB1.DMP' TRANSPORT_DATAFILES='/u01/oradata/prod/MYTB1.DBF'

18.5 数据泵直传示例

所谓直传就是通过网络在两个主机间导入导出（不落地），适用于大规模数据迁移

示例说明，**将 Linux 端导入 win 端**，可以表级、用户级、表空间级。

①需要建立一个 dblink，关于 dblink 内容见第十九章。

②数据泵使用缺省的目录 DATA_PUMP_DIR

③表空间直传比可传输表空间方式要慢的多，因为前者是 select 出来，insert 进去，而后者是 ftp 传输数据文件

④设置 flashback_scn 参数，是当导出时如果数据库仍有读写操作，若希望得到某时间点的数据，这需要 undo 保证数据一致性

1）win 端建立 dblink，使客户端能访问服务器端(win 端指向 Linux)

SQL> create public database link system_link connect to system identified by oracle using 'prod';

2）win 端直接使用 impdp，不需要 expdp。

①表级

C:\Users\Administrator>impdp system/oracle tables=tim.t1 network_link=system_link parallel=2　　win 端 tim 用户要存在

②用户级

C:\Users\Administrator>impdp system/oracle schemas=tim network_link=system_link parallel=2　　win 端 tim 用户可以不存在，数据泵建立

③表空间级

C:\Users\Administrator>impdp system/oracle tablespaces=test network_link=system_link parallel=2　　相关用户和表空间要存在

第十九章：物化视图

19.1 产生和作用

19.1.1 物化视图的产生

物化视图起源于数据仓库，早期的考虑是用于预先计算并保存表连接或聚集等耗时较多的操作的结果，这样，在执行查询时，就可以避免在基表上进行这些耗时的操作，从而快速的得到结果。生产实用中物化视图的功能逐步得到延伸，比如对某些重要的表进行**实时高级复制**，在一定程度上起到可以替代逻辑备份的作用。

19.1.2 物化视图的作用

1）提高查询效率-

物化视图和表一样可以直接进行查询。通过**查询重写**功能实现读写分离的作用，它是优化查询性能，提高系统工作效率的有效方法之一。

2）表级的高级复制

物化视图本身会存储数据，因此是物化了的视图。可以将物化视图复制到其他数据库，**刷新机制**提供了多种同步数据的策略，这样重要的表得以实时复制，如果是远程复制，也一定程度起到了灾备的作用。

19.2 物化视图基本功能

19.2.1 查询重写（Query Rewrite）：

查询重写是指当对物化视图的基表进行查询时，Oracle 会自动判断能否通过查询物化视图来得到结果。

使能查询重写：建立物化视图时可以指定 Enable Query Rewrite 关键字，若不指定默认为 DISABLE。

19.2.2 刷新（Refresh）：

指当基表发生了 DML 操作后，物化视图何时采用哪种方式和基表进行同步。

1）刷新模式：

①ON DEMAND：指物化视图按照用户需要进行刷新(提交后），比如设定刷新间隔。

也可以通过 dbms_mview.refres 包手工方式刷新。此刷新方式是默认的。

②ON COMMIT：当基表做提交时立即刷新物化视图，但基表 commit 的速度会受影响。一般用于查询重写。

③手动刷新(调用 dbms_mview 包)

2）刷新方法

①FAST 刷新采用增量刷新，需要物化视图日志，只刷新自上次刷新以后进行的修改。

②COMPLETE 刷新对整个物化视图进行完全的刷新。

③FORCE 方式，Oracle 在刷新时会去判断是否可以则采用 FAST 方式，否则采用 COMPLETE 的方式（比如物化视图日志损坏）。

④NEVER 指物化视图不进行任何刷新。

默认值是 FORCE ON DEMAND

19.2.3 物化视图日志

1）作用：为 fast（增量刷新）提供相关增量数据日志。

基表中的主键或 rowid 可以为 materialized log 唯一标识表行记录，这是 fast 刷新的必要条件。

如果基表有主键，那么基于主键的物化视图是首选。

如果基表无主键，可以建立基于 rowid 的物化视图。

基于 rowid 的快速刷新要在建立物化视图日志时做出说明。而基于主键的则可以缺省，不需要特别说明。

SQL> create materialized view log on test with rowid;

2）建立物化视图日志要点：

①一个基表只能建立一个 materialized log

②一个 materialized log 可以为多个物化视图提供数据

③一个 materialized log 可以选择基表中的某些列，包含这些列的物化视图才能支持 fast 刷新

④没有 materialized log 的基表只能支持物化视图的 comlete 刷新

⑤可以建立为 RowID 或 Primary Key 类型的。

⑥还可以选择是否包括 Sequence、Including New Values 以及指定列的列表

19.3 物化视图的操作

1）语法：

```
create materialized view [view_name]
refresh [fast|complete|force]
[
on [commit|demand] |
start with (start_time) next (next_time)
with [primary |rowid]
as
{创建物化视图用的查询语句}
```

2）如果创建物化视图，则必须具有访问主表、create MATERIALIZED VIEW 这两个权限

3）删除物化视图有两种顺序，一是先删除物化视图再删除日志；二是先删除日志再删除物化视图。

如果原数据表只被一个物化视图引用，那么可以采用先删除日志，再删除物化视图，这样删除的速度较快。

```
drop materialized view log on test_table;
drop materialized view mv_test_table;
```

4）查看物化视图的信息有一些视图，比如 user_mviews

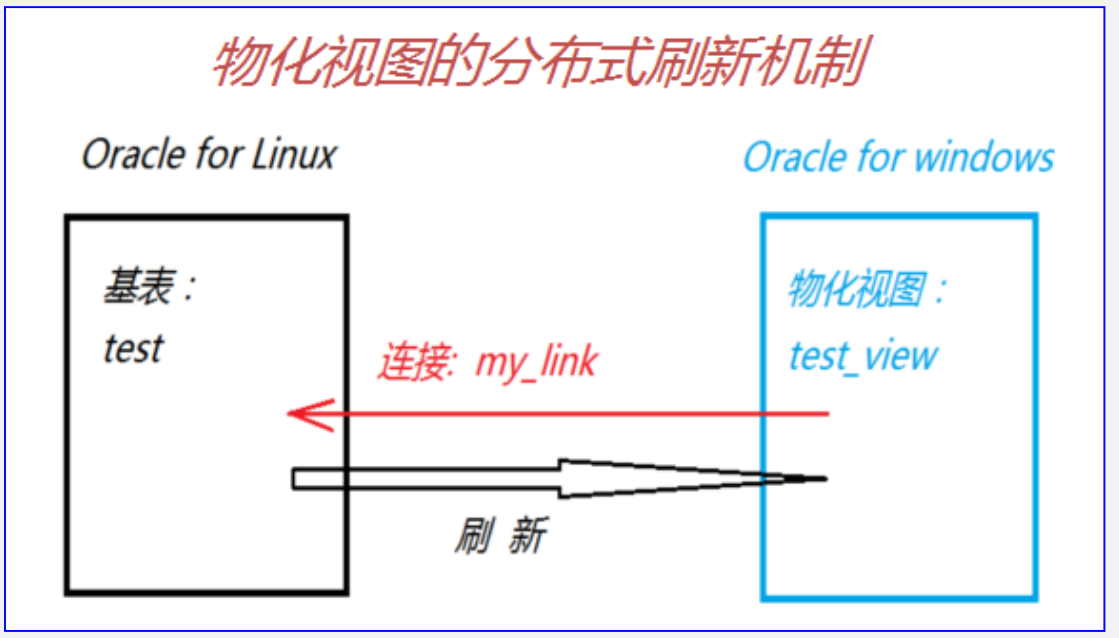
19.4 Database link

1）概述：

①database link 是定义一个数据库到另一个数据库的路径的对象，database link 允许你查询远程表及执行远程程序。在分布式环境里，database link 是必要的。

②database link 使用 Oracle Net 做单向连接，可由 SYS 创建 public database link，否则需要授权。

③视图 DBA_DB_LINKS 记录了 Database Link 信息



2）分类：

①Privte： 用户级

②Public ： 数据库级

③Globle ： 网络级

3）Database link 语法：

CREATE [SHARED][PUBLIC] database link link_name

[CONNECT TO [user][current_user] IDENTIFIED BY password]

[AUTHENTICATED BY user IDENTIFIED BY password]

[USING 'connect_string']

注 1: [PUBLIC] 选项表示这个 database link 所有用户都可以使用

注 2：[USING ‘网络连接符’] 这里描述的 tnsnames 里的描述。

19.5 物化视图示例

19. 5. 1 查询重写的物化视图

步骤 1: SCOTT 建立基本 test，和物化视图日志

SCOTT@ prod>create table test (id int,name char(10));

SCOTT@ prod>create materialized view log on test with rowid;

步骤 2: SYS 授权 SCOTT 建立物化视图权限

SQL> conn / as sysdba

SQL> grant create materialized view to scott;

步骤 3: SCOTT 建立 test 表的物化视图，使能查询重写

SQL> create materialized view test_view1 refresh fast with rowid **on commit** enable query rewrite as select * from test;

on commit 支持本地物化视图刷新，提交即刷新物化视图。

步骤 4: 测试：查询 test 表时 Oracle 自动选择了物化视图

SQL> insert into test values(1,'a');

SQL> insert into test values(2,'b');

SQL> commit;

SQL> set autotrace on;

SQL> select * from test;

.

19.5.2:分布式物化视图（间隔刷新）

本例的环境：

①删除之前的物化视图 test_view1 和物化视图日志，重建物化视图日志。

SCOTT@ prod>drop materialized view test_view1;

SCOTT@ prod>drop materialized view log on test;

SCOTT@ prod>create materialized view log on test with rowid; 重建物化视图日志

注：分布式物化视图不支持 on commit，所以之前的物化视图日志也无法支持远程刷新

②基表和物化视图日志在 Linux 平台（本地）

③物化视图在 Win7 平台（远程）

步骤 1: 建立 Win7 平台（远程）----linux 平台（本地）的 dblink,

C:\>sqlplus / as sysdba

SQL> create public database link my_link connect to scott identified by scott using 'prod';

步骤 2：验证 dblink，在 Win7 端 SCOTT 下操作：

SQL> select * from tab;

SQL> select * from tab@my_link;

步骤 3:授权 Win 端 SCOTT 建立物化视图的权限

SQL> conn / as sysdba

SQL> grant create materialized view to scott;

步骤 4：在 Win 端建立基于 test 表的 on demand 的物化视图 test_view2

SQL> conn scott/scott;

SQL> create materialized view test_view2 refresh fast

start with sysdate next sysdate+1/2880

with rowid

as select * from scott.test@my_link;

注：1440 分钟是 24 小时，1/1440 是 1 分钟，1/2880 就是 30 秒。

步骤 5: 验证物化视图可用

Win 端

SQL> select * from test_view2;

Linux 端

SQL>insert into test values(3,'c');

SQL>commit;

验证 Win 端 30 秒之内的数据是否插入成功

SQL> select * from test_view2;

19. 5. 3 建立分布式物化视图（手动刷新）

如何完成物化视图手动刷新

步骤 1: win 端 SCOTT 下操作

SQL> conn scott/scott;

SQL> create materialized view test_view3 refresh fast

with rowid

as select * from scott.test@my_link;

步骤 2：Linux 端插入数据

SQL> insert into test values(4,'d');

SQL> commit;

步骤 3：Win 端调用程序包完成手动刷新。

SQL> conn scott/scott;

SQL> exec dbms_mview.refresh ('test_view3','F');

注：’F’表示 fast 刷新。

19. 5. 4 聚合函数的物化视图

需要一些限制条件

1）物化视图日志需要 sequence(),和 including new values 子句记录聚合列信息

2）基表更改后不支持 fast 刷新（因为所有行将重新参与聚合函数运算）

例

SQL>create table emp1 as select * from emp;

SQL>create materialized view log on emp1 with rowid,sequence(sal,deptno) including new values;

本地：

SQL>create materialized view emp1_view1 refresh start with sysdate next sysdate+1/2880

as select deptno,sum(sal) sumsal from emp1 group by deptno;

远程：(win7)

SQL>create materialized view emp1_view2 refresh with rowid on demand

as select deptno,sum(sal) sumsal from emp1@my_link group by deptno;

SQL>exec dbms_mview.refresh('emp1_view2','?');

注：

①’?’表示 force 刷新，如果基表被更新过，将自动转为 complete 刷新方式

②sum(sal)聚合函数一定要有别名 sumsal

--完--