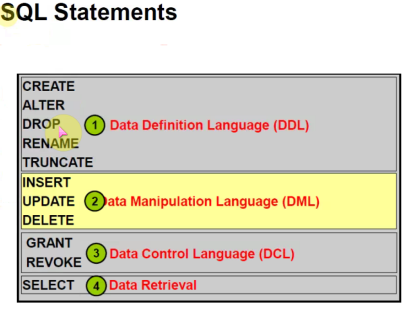
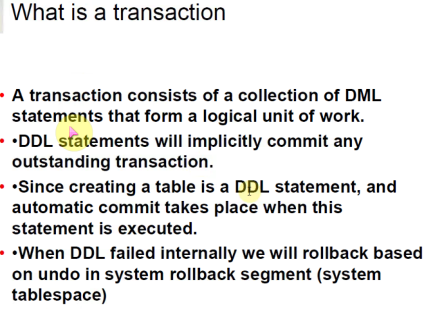
数据库中sql语句的类型



select merge可以看作是DML



1 事务：是oracle数据库中的逻辑工作单元，由一组DML语句组成一个有序的操作。比如说典型的转账业务

事务有ACID特性。

多个事务并发操作需要锁lock，lock是一种资源！

创建环境：

create table test\_transaction (id number,name varchar2(22),salary number);

alter table test\_transaction drop column salary;

alter table test\_transaction add account number;

insert into test\_transaction values (1,'aaa',5000);

insert into test\_transaction values (2,'bbb',6000);

实验：查看一个transaction =》v$transaction

SQL> select \* from test\_transaction;

        ID     SALARY NAME                   DAT

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

         1       5000 aaa                    06-JUL-05 12.00.00.000000 AM

         2       5500 bbb                    06-JUL-05 12.00.00.000000 AM

update test\_transaction set salary=6000 where id=1;

update test\_transaction set salary=6500 where id=2;

另一个会话查看

SQL> select addr,xidusn,xidslot,xidsqn,xid from v$transaction;

ADDR                 XIDUSN    XIDSLOT     XIDSQN XID

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

00007FFFA48573C8          7         28       1787 07001C00FB060000

SQL> select \* from v$rollname;

       USN NAME

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

     0 SYSTEM

     1 \_SYSSMU1\_1240252155$

     2 \_SYSSMU2\_111974964$

     3 \_SYSSMU3\_4004931649$

     4 \_SYSSMU4\_1126976075$

     5 \_SYSSMU5\_4011504098$

     6 \_SYSSMU6\_3654194381$

     7 \_SYSSMU7\_4222772309$

     8 \_SYSSMU8\_3612859353$

     9 \_SYSSMU9\_3945653786$

    10 \_SYSSMU10\_3271578125$

最后

rollback;

DML==>MANUAL COMMIT

DDL==>AUTO COMMIT

sqlplus的设置：

SQL> show all

autocommit OFF

ddl语句之前之后都会做commit，为什么？？？

==》ddl内部做dml（insert into tab$/segm$等）更改数据字典表，为避免字典表lock所以自动commit。

ddl使用系统回滚段(v$rollname  SYSTEM )

ddl内部过程如下：

commit;

create table a(id int);

=>dml insert into segm$()

               insert into obj$()

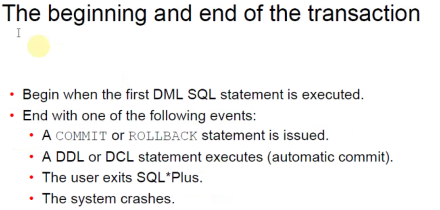
               insert into tab$()

               insert into col$()

               dml >lock

=>commit;  》释放lock

思考：dump online logfile 看一个ddl操作！！

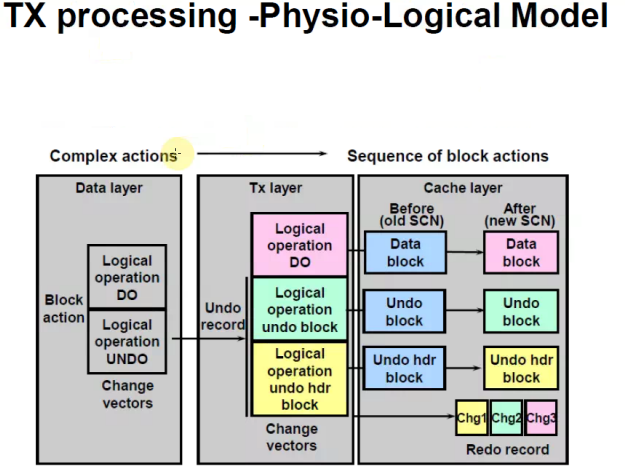


事务以第一个可执行的DML语句开始，在提交或回滚时结束。

the user  exits SQL\*Plus  ==>auto commit

system crashes or kill session ==>auto rollback

###############################################



很难，很难讲明白！！！！！！！！！！！！！！！！

oracle 处理事务的模型

需要结合redo/undo来理解。参考 redo knowledge and undo knowledge and space（storage）manager

如果用的是mysql ，其功能简单，则做好在应用层来控制事务，这个应用层事务控制器就是参考的oracle知识，淘宝的很多mysql应用就是这么做的！！

比如说：

update test\_trasaction set name='b' where name='a';

涉及到3部分

data layer==》data block   reference spface（storage） know

       logical operation do（a》b）

       logical operation undo==》tx，undo data

       change vectors==》redo record

tx layer ==》trasaction  reference transacton know

       logical operation do（a》b）

       undo record ==》tx uba

       change vectors==》redo record

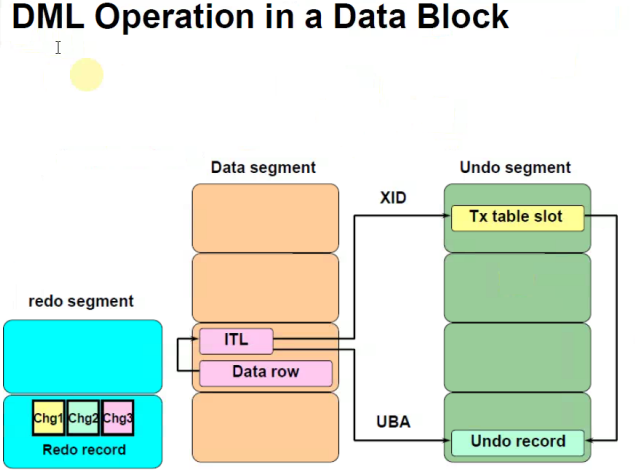
cache layer  ==》referance redo know

       data block（a>b）a是old，b是new

       undo：undo block放undo（a），undo hdr block存放transaction

       redo：change vector

3部分显示同样的操作，不同的阶段！！



update test\_transaction set name='b' where name='a';

先产生redo： redo record(非imu)==》参考redo kenow

然后在rollback segment header》tx table （xid，UBA）》slot生成事务》按照uba生成undo record

A transaction begins when a slot is allocated in the transaction table at the head of a rollback segment. This is the

physical manifestation of a transaction. The transaction identifier (Txid) is a pointer to this location.

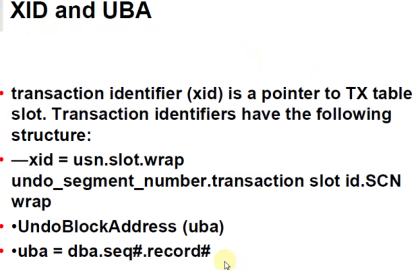
然后实现update

最后dbwn更改块，data segment header有itl：存储了xid，uba

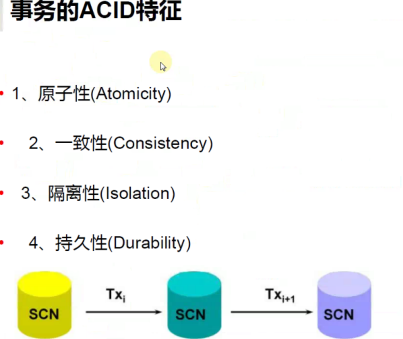
数据就放在了data row

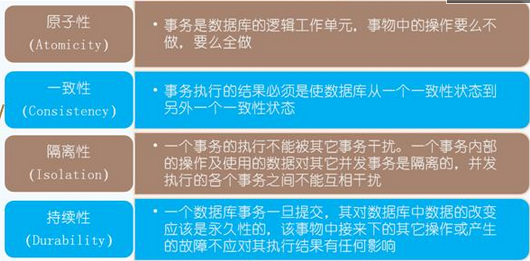
########################################################

xid and uba



一个事务本质上有ACID 4特性





undo保障atomicity：要么commit要么rollback

undo保障consistency，如 select 读一至性

undo实现isolation，不同会话事务隔离

redo保障durability

没有事务隔离性可能发生的情况

脏读dirty read：当一个事务读取另一个事务尚未提交的修改时，产生脏读。

如：事务1更新了记录，但没有提交，事务2读取了更新后的行，然后事务T1回滚，现在T2读取无效。

不可重复读unrepeatable read：同一查询在同一事务中多次进行，在此期间，由于其他事务提交了对数据的修改或删除，每次返回不同的结果

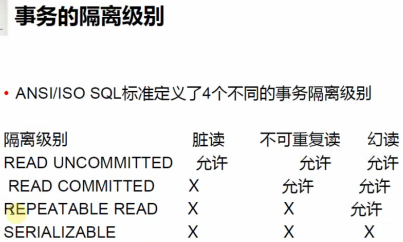
如：事务1读取记录时，事务2更新了记录并提交，事务1再次读取时可以看到事务2修改后的记录；

幻读phantom read：同一查询在同一事务中多次进行，由于其他提交事务所做的插入操作，虽然查询条件相同，每次返回的结果集却不同。

如：事务1读取记录时事务2增加了记录并提交，事务1再次读取时可以看到事务2新增的记录；

行。

事务的4种隔离级别



提交前读 ：所有的会话除了自己会话，读取的数据库都是提交后的

提交后读

重复读

串行顺序读

SERIALIZABLE

当前事务的所有语句只能看到这个事务中执行的第一个查询或者数据修改语句之前提交的行。

这是最严格的隔离级别，事务串行化的执

oracle 支持以下三种事务隔离级别（transaction isolation level）。

提交后读

串行顺序读

只读

如果想允许重复读使用如下语句

select \* from test\_transaction for update;

设置一个事务的隔离级别

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION READ ONLY;

设置会话的隔离级别

ALTER SESSION SET ISOLATION\_LEVEL=SERIALIZABLE;

ALTER SESSION SET ISOLATION\_LEVEL=READ COMMITTED;

有的数据库支持提交前读，即允许读取脏数据，如mysql

以下是事务的分类！



分布式事务涉及的两个数据库scn必须一致。所以db\_link会自动同步scn。这也就造成scn headroom问题。。。。。

2pc：two-phase-commit protocol

在2pc事务中， 其中一个数据库（通常是客户最初登陆的数据库）会成为协调器，这个站点会询问其他站点是否已经准备好提交。其他站点会报告就绪状态（y/n），如果有一个站点为n，则整个事务都会回滚。都为y则全部提交。

但是有一个时间窗口错误，即协调器站点收集了了其他站点的就绪状态后还需要把这个状态广播给所有的站点使他们知道是否可以提交，如果在这个时间窗口内发生网络或断电故障，站点2，3事物就会挂起（他们分别报告的自己的状态给1，1知道结果，1给准备好的节点发出commit 之前），形成可疑分布式事务（in-dout distributed transaction）,2,3无发关闭事务，他们在等待1发出结果通知。所以rac中通过RECO进程来解决这种事务错误，通过force选项的commit或rollback实现。

note： 不能再dblink上执行ddl操作

DBA\_2PC\_PENDING describes distributed transactions awaiting recovery.

DBA\_2PC\_NEIGHBORS describes incoming and outgoing connections for pending transactions.

select local\_tran\_id,in\_out,DBUSER\_OWNER,database,dbid from dba\_2pc\_neighbors;    --查看分布式事务使用的dblink

DBA\_DB\_LINKS describes all database links in the database. Its columns (except for PASSWORD) are the same as those in ALL\_DB\_LINKS.

ALL\_DB\_LINKS describes the database links accessible to the current user.

select host from dba\_db\_links where db\_link='';

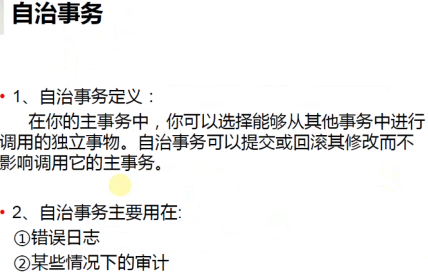
#######################################################

rollback force 'V817REP.BE.ORACLE.COM.89f6eafb.1.8.238';

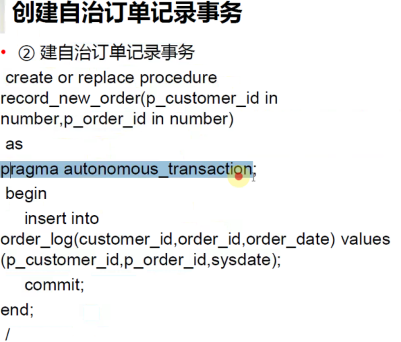
-OR-

rollback force '1.8.238';

execute DBMS\_TRANSACTION.PURGE\_LOST\_DB\_ENTRY('1.8.238');



整个事物失败==》我想记录失败的日志==》insert 日志表===》atomicity==》自治事务



利用自治事物，可以挂起当前执行的事务，开始一个新事物，完成事务工作后，继续提交或回滚父事物

自治事务是用pl/sql控制事物的新方法。