By David Itshak
shaked19@gmail.com

http://www.ildba.co.il/author/cimid/
http://www.sqlserver.co.il/?cat=940

Global Hebrew Virtual PASS Chapter :
https://www.youtube.com/watch?v=x4hGjYGBfkc
https://www.youtube.com/watch?v=eJO8G9if3EY

SqlSaturday Israel 2016 :
Reference and Credits

Apress Oracle Database Transactions and Locking Revealed (2014)
Chapter 6: Redo and Undo

RMAN Recipes for Oracle Database 12c: A Problem-Solution Approach, 2nd Ed.
Chapter 13: Performing Flashback Recovery

Oracle® Database Concepts
12c Release 1 (12.1)
E41396-13
https://docs.oracle.com/database/121/CNCPT/toc.htm
13.3 Overview of Checkpoints
13.4 Overview of Instance Recovery

Oracle® Database
Backup and Recovery User's Guide
12c Release 1 (12.1)
E50658-07
https://docs.oracle.com/database/121/BRADV/toc.htm
Why Backups Are So Important?

JournalSpace Drama: All Data Lost Without Backup, Company Deadpooled

Posted Jan 3, 2009 by Robin Wauters

Blogging platform JournalSpace (which I’d never heard of to date) has ceased to be, following a wipe-out of the main database for which there was no back-up in place. According to the JournalSpace blog, the database was overwritten as a result of a malicious act from a disgruntled ex-employee.

It was the guy handling the IT (and, yes, the same guy who I caught stealing from the company, and who did a slash-and-burn on some servers on his way out) who made the choice to rely on RAID as the only backup mechanism for the SQL server. He had set up automated backups for the HTTP server which contains the PHP code, but, inscrutibly, had no backup system in place for the SQL data. The ironic thing here is that one of his hobbies was telling everybody how smart he was.
Why Backups Are So Important?
핑ואו ה-11 באוקטובר
התקאות

Police

[Image of police officers standing near the water with a view of the city and a large plume of smoke in the background.]
Worst Practices

1. Failing to keep offsite backup copies of data
2. Failing to monitor your backup job effectively
3. Using manual or obsolete procedure
4. No written plans
5. Lack of backup integrity (backing up files, not DB)
6. Not testing the backup (no dry run)
7. Doing no backup at all
DRP

Disaster Recovery Plan

The document includes processes, policies, and procedures that are essential for disaster recovery.

It includes planning for the renewal of applications, data, hardware, and communication, such as networks and other information technology elements.

Examples: fire, earthquake, flood, explosion, building collapse, cancelation, and so forth.

It includes a comprehensive set of measures to safeguard and communicate information and data.

Guidelines: alerts, updates, facts, and reports.

The system is a communication network that manages and distributes information.
מדדים

(RTO) Recovery Time Objective

dיחומין העיר לעזרה של מערכות לעבר זמינות של מערכות

(RPO) Recovery Point Objective

dיחומין העיר לעזרה של מערכות לעבר סכום של מערכות

Service Level Agreement

Mean time MTTR (Mean Time Between Failure) MTTF

(High availability)

דימויות בוגרות

- זמינות גבוהה
- חוסר זמינות = downtime
BCP – Business Continuity Plan

- DRP הוא קבוצת משנה של תחולים גדולים ייחודי - הבטחת הרציפות

- העסקת הבטחת הרציפות היא קבוצת משנה של תחולים גדולים ייחודי - הבטחת הרציפות

- כולל בנוסח תכונת של היבטים שאינים קשורים ל테כנולוגיה המודרנית

- כל썰 בתוכנית אונס ענייני מענה מתכנית, מתקנים, תכשורת

- דחיפה משמעותית בעקבות פיגועי 11 בספטמבר - חוקים המחייבים

- בחינת משמעויות התוכנית (בחיי חולים למשלו) לניהול תכניות התאוששות מאסונות

- אפקטיביות

- בין 2%-4% מתכונות הוכנות אונס תכניות התאוששות המודרנית

- אפקטיביות
הפעלה מאתר מרוחק חלופי

השתית עם מענה לצרכים מינימליים בזת חירום: מערכות

נדושות, תקשורת, תחנות עבודה וכולל
Step 1: Baseline

SAN or NAS Attached hosts

Immediate Write Acknowledgement

Source

Baseline copy of source volume(s)

LAN/WAN

OR

Immediate Write Acknowledgement

Step 2: Updates

SAN or NAS Attached hosts

Immediate Write Acknowledgement

Source

Periodic updates of changed blocks

LAN/WAN

Target
Disaster Recovery Process

- Data Centric Environments
- "Mean Time To Recovery" (MTTR) after an event
-LAN/WAN
- Restoration of data from backup tapes can take days

DRP (redirect)
(resync backwards after source restoration)
Oracle Data Guard Architecture

Production Database

Network

Sync or Async
Redo Shipping

Physical Standby

Open R/O

Redo Apply

Backup

Redo Shipping
יאורו

• אטר ראשית מסונכרן לאתרי משניים

קרח אסון באטר ראשית

יאורו מוכר

سجنור מופסקNESS, storage באטר הופך

לחדות פעיל בסיסי R/W

הנחנוים לשירותיםซอֹף לפעילות

פעילים
Types of DB Failures
Types of DB Failures

Statement Failure

- A program attempts to enter invalid data into an Oracle table.
- Programmatic logical errors.
- Long data insertion job or a data import job to fail midway because there is no more room to put in the data

User Process Failure

- User performing an abnormal disconnect or performing a terminal program error and losing the session connection.
- As a DBA, there is not much you need to do
- Oracle background processes will roll back any committed changes to the data and release the locks that were held by the abnormally disconnected user session. The user will have to reconnect after the abrupt termination.
Types of DB Failures: User Error

– User Accidentally dropping a table or wrongly modify or delete data from a table.

– Use flashback table or the new Oracle 12c RMAN command recover table to restore a table to a previous point in time.

– Use the flashback drop feature to recover an accidentally dropped table.

– If the transaction isn’t committed yet, you can simply rollback the unwanted changes or Oracle’s LogMiner tool.

UPDATE BANK_TBL SET BALANCE=500 WHERE ACCOUNT_NO=123456
Types of DB Failures: Media Failure

- When you lose a disk or a disk controller fails
- Any failure to read from or write to a disk.
- Ex: A head crash, a file corruption, and the overwriting or deletion of a data file.
- Require performing a media recovery with the help of backups of the data files and archived redo logs.
Types of DB Failures: Network Failure

- Oracle Net listener, network interface card (NIC), or the network connection has failed.
- DBA must configure multiple network cards and a backup network connection and backup listener to protect against these errors.
- You can use the connect-time failover feature to protect against a network failure
Types of DB Failures: Instance Failure

Reasons:

1. A hardware failure, a power failure, or an emergency shutdown procedure.

2. When the key Oracle background process, such as PMON, shuts down because of an error
   – Oracle will perform an automatic instance or crash recovery at this point.
   – Oracle will automatically perform a rollback of the uncommitted transactions by using data from the undo segments and will roll forward the committed changes it finds in the online redo log files.
   – Once the uncommitted changes are backed out and the committed changes are rolled forward, the data files are in sync again and will contain only committed data.
Types of DB Failures: Instance Failure

- First phase: cache recovery or rolling forward reapplying all of the changes recorded in the online redo log to the data files.
- Because rollback data is recorded in the online redo log, rolling forward also regenerates the corresponding undo segments.
Types of DB Failures: Instance Failure

- After rolling forward, data blocks contain all committed changes recorded in the online redo log files.
- Files could also contain uncommitted changes that were either saved to the data files before the failure, or were recorded in the online redo log and introduced during cache recovery.
Types of DB Failures: Instance Failure

- After the roll forward, any changes that were not committed must be undone.
- Oracle Database uses the checkpoint position, which guarantees that every committed change with an SCN lower than the checkpoint SCN is saved on disk.
- Oracle Database applies undo blocks to roll back uncommitted changes in data blocks that were written before the failure or introduced during cache recovery.
- This phase is called rolling back or transaction recovery
Methods of Data Protection

• Backup types
  – Physical backup are copies of physical database files
  – Logical backups are backups for logical data such as tables or Stored procedures

• Backup methods:
  – Oracle tools: Rman; EM Cloud control 12C
  – User-managed backup and recovery (OS commands, 3rd party tools)

• Backup-related procedures
  – Data archival: Copying data to long-term storage
  – Data transfer: ETL or OLAP scenarios
Oracle Database 12C backup and Recovery Tools

• Recovery Manager (Rman) command line tool
• Oracle Enterprise Manager Cloud Control 12C
  – GUI front-end to Rman
  – Data Recovery Advisor (DRA) – Automated corruption detection and repair utility (Rman interface)
• Oracle Data Pump:
  – PL/SQL packages to run from the command line
  – Perform logical import and exports
Preliminary Data Protection tasks

• Enable ARCHIVELOG Mode
  – Periodically archives (backs up) the online redo log files

• Backup UNDO tablespace

• Specify the fast recovery area

• Configure Flashback features

• Multiplex and back up the control file

• Relocating your data files to different disks:
  Backup on showers disks

• Back up server parameter file
Flow of data changes through an Oracle instance and database
Incomplete & Complete Recovery

• **Restore**
  – RMAN restore command is used to retrieve data files, control files, archived redo log files, and server parameter files (spfiles) from backup sets.
  – it will reconstruct an exact copy of the data file as it was when it was backed up.

• **Recovery** is the process of applying transactions from incremental backups (if using) and the redo files (archive and/or online) to the data files.

• **Complete recovery** means you can recover all transactions that were committed

• **Incomplete recovery** means that you cannot restore all committed transactions.
  – Required when you don’t have all the redo required to apply all committed transactions to the data files.
  – To restore the database to a previous state to recover data that was accidentally deleted.
  – Initiated with the recover database until command .

• Oracle detects which data files need media recovery by comparing the system change number (SCN) information in the control file and the corresponding SCN information in the data file headers.
Performing Complete Recovery

- Oracle detects which data files need media recovery by comparing the system change number (SCN) information in the control file and the corresponding SCN information in the data file headers.

- **SCN Oracle Startup Checks**

<table>
<thead>
<tr>
<th>Condition on Startup</th>
<th>Oracle Behavior</th>
<th>DBA Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF checkpoint SCN &lt; Data file</td>
<td>“Control file too old”</td>
<td>Restore a newer control file or recover with the</td>
</tr>
<tr>
<td>checkpoint SCN</td>
<td>error</td>
<td>using backup controlfile clause.</td>
</tr>
<tr>
<td>CF checkpoint SCN &gt; Data file</td>
<td>Media recovery</td>
<td>Most likely a data file has been restored from</td>
</tr>
<tr>
<td>checkpoint SCN</td>
<td>required</td>
<td>a backup. Recovery is now required.</td>
</tr>
<tr>
<td>CF checkpoint SCN = Data file</td>
<td>Start up normally</td>
<td>None</td>
</tr>
<tr>
<td>Database in mount mode, instance</td>
<td>Crash recovery</td>
<td>None. Oracle automatically performs crash</td>
</tr>
<tr>
<td>thread status = OPEN</td>
<td>required</td>
<td>recovery.</td>
</tr>
</tbody>
</table>
Performing Database-Level Recovery

$ rman target /
RMAN> startup mount;
RMAN> restore database;
RMAN> recover database;
RMAN> alter database open.

• How It Works
• The restore database command will restore every data file in your database.
• When you issue the recover database command, RMAN will automatically apply redo to any data files that need recovery.
• The recovery:
  • Incremental backup pieces (applicable only if using incremental backups)
  • Archived redo log files (generated since the last backup or last incremental backup that is applied)
  • Online redo log files (current and unarchived)
Performing Flashback Recovery

- **Flashback Database**: Rewind a DB to an earlier time
- **Undropping a table**: When the table is dropped, it’s renamed and placed recycle bin. You can reinstate the dropped table using only one simple command.
- **Flashing back a table**: You can pull the changes made to the row data from the undo segments, as long as they are available in the undo segments. When you pull the older versions of the table, you can reinstate the entire table to a point in time in the past using these past images.
- **Recovering a single table from backup**: You have dropped a table. Now it is gone from the recycle bin. Recover a single table, or a single partition of with a single command.
Flashback Database

- Rewind a DB to an earlier time
- Enable flashback on DB

```sql
SQL> alter database flashback on;
```

Database altered.

- Additional logs to be created during the database operation: flashback logs with the regular archived logs.
- Flashback logs record changes to DB blocks exclusively for the purpose of rolling back the block changes.
- When you flash DB back to a point in the past, these flashback logs are read and applied to the database blocks to undo the changes.
- Entire DB is transported to that point in time.
- You can flash back to the following:
  1. A specific point in time, specified by date and time
  2. A specific SCN number
  3. The last resetlogs operation
  4. A named restore point

Flashback works both back and forth.
Oracle Flashback

• Fast & Easy querying/disaster tool
• We can view past states of DB Objects and optionally return DB objects to those previous states
• DB manage undo segments in undo tablespace (AUM)

Flashback VS Redo
Flashback Database to a specific SCN

- Check the SCN of DB now

```sql
SQL> select current_scn 2 from v$database;
CURRENT_SCN
---------
1137633
```

```bash
rman target=/
RMAN> shutdown immediate
RMAN> startup mount
```

- Flash DB back to your desired SCN

```bash
RMAN> flashback database to scn 1050951;
Starting flashback at 03-AUG-12 ....
Finished flashback at 03-AUG-12
RMAN> alter database open read only;
Database opened.
```

- Check the data in the table, if OK open DB. You can start the flashback process again to flash back to a different SCN.

```bash
RMAN> alter database open resetlogs;
database opened
```
Recovering a Dropped Table

1. Log on to the database as the table owner.
2. Check whether the table exists in the recycle bin. Issue the SQL*Plus command show recyclebin:

   SQL> show recyclebin

3. Revive the table from the recycle bin by issuing the following SQL*Plus command:

   SQL> flashback table test to before drop;

Flashback complete.

The table is now available in the database.
Querying the History of a Table Row (Flashback Query)

• Find all the changes to the row for ACCNO 3760 in table ACCOUNTS:

```sql
select acc_status, versions_starttime, versions_startscn, versions_endtime, versions_endscn, versions_xid, versions_operation
from accounts
versions between scn minvalue and maxvalue
where accno = 3760
order by 3
```

![Table showing version history]

• using timestamps:

```sql
versions between timestamp to_date('7/12/2012 12:00:00', 'mm/dd/yyyy hh24:mi:ss')
```

• Using SCNs:

```sql
versions between SCN 1000 and 2000
```
Querying the History of a Table Row (Flashback Query)

Flashback Query Pseudocolumns:

<table>
<thead>
<tr>
<th>Pseudo Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSIONS_STARTTIME</td>
<td>This is the timestamp when this version of the row became effective. This is the commit time after the row was changed.</td>
</tr>
<tr>
<td>VERSIONS_STARTSCN</td>
<td>This is the SCN when this version became effective.</td>
</tr>
<tr>
<td>VERSIONS_ENDTIME</td>
<td>This is the timestamp when the version became old, replaced by a new version. This is the time of commit after the row was changed.</td>
</tr>
<tr>
<td>VERSIONS_ENDSCN</td>
<td>This is the SCN when the row’s version was changed.</td>
</tr>
<tr>
<td>VERSIONS_XID</td>
<td>This is the transaction ID that changed the row’s version. This can be joined with the XID column of the dictionary view FLASHBACK_TRANSACTION_QUERY to show the transaction that made this change. The view FLASHBACK_TRANSACTION_QUERY also shows other relevant details of the transaction, such as who did it, when, and so on.</td>
</tr>
<tr>
<td>VERSIONS_OPERATION</td>
<td>This is the abbreviated activity code—I, U, or D—for Insert, Update, or Delete that resulted in this version of the row.</td>
</tr>
</tbody>
</table>

```
A VERSIONS_STARTTIME VERSIONS_STARTSCN VERSIONS_ENDTIME   VERSIONS_ENDSCN
VERSIONS_XID   V
- ------------------ -------------------------- --------------------------
--------------------------
A 12-JUL12 04.38.57 PM 1076867 12-JUL12 04.39.03 PM 1076870
02002F00DB010000 U
I 12-JUL12 04.39.03 PM 1076870 12-JUL12 04.39.12 PM 1076874
08001B00DB010000 U
A 12-JUL12 04.39.12 PM 1076874
07002B0068010000 U
A 12-JUL12 04.38.57 PM 1076867
```
Flashing Back a Specific Table

• This prepares the table for flashback

```sql
SQL> alter table accounts enable row movement;
Table altered.
```

• Flash the table back to a specific timestamp:

```sql
SQL> flashback table accounts to timestamp to_date ('12-JUL-12 18.23.00', 'dd-MON-YY hh24.mi.ss');
Flashback complete.
```

• You can flash back to a specific SCN as well

• If you have gone too far into the past, you can flash “forward” using the same flashback statement
Recovering a Specific Table from Backup

- You can specify the point in time using timestamp or SCN

```
RMAN> connect target '/ as sysdba'
RMAN> recover table SCOTT.ACCOUNTS
  2> until time 'sysdate-1/60/24'
  3> auxiliary destination '+DG1';
```

```
RMAN> recover table SCOTT.ACCOUNTS
  2> until scn 2012991
  3> auxiliary destination '+DG1';
```

- High-level steps in that operation:
  1. Create another database instance.
  2. Restore from the backup the tablespace containing that table along with system, sysaux and undo tablespaces in a different location.
  3. Open that restored database.
  4. Export the table.
  5. Import into the main database.
  6. Drop this temporary database and delete the instance.
Block Media Recovery

- When RMAN first detects missing or corrupt redo records during block media recovery, it does not immediately signal an error because the block undergoing recovery may create one later in the redo stream.

- After block recovery begins, RMAN discovers that change 120 is missing from the redo stream, either because the log block is corrupt or because the log cannot be found. RMAN continues recovery if block 13 is re-created later in the redo stream.

- Assume that in change 140 a user drops the table employees stored in block 13, allocates a new table in this block, and inserts data into the new table. At this point, the database formats block 13 as a new block. Recovery can now proceed with this block even though some redo preceding the recreation operation was missing.
Appendix & Extra Material
Redo and Undo : Transaction Rollback

• When a ROLLBACK statement is issued, the database uses undo records to roll back changes made to the database by the uncommitted transaction.

• During recovery, DB rolls back any uncommitted changes applied from the online redo log to the data files.

• Undo records provide read consistency by maintaining the before image of the data for users accessing data at the same time that another user is changing it.
How Redo and Undo Work Together

• What happens during the processing of an INSERT with regard to redo and undo generation

• How Oracle uses this information in the event of failures at various points in time?

• **Undo information, stored in undo tablespaces or undo segments, is protected by redo as well.**
  – undo data is treated just like table data or index data—changes to undo generates some redo is logged to the log buffer and then the redo log file.
  – **Why?**

• **Undo data is added to the undo segment and is cached in the buffer cache, just like any other piece of data would be.**
Example INSERT-UPDATE-DELETE-COMMIT Scenario

• created a table with an index as follows:

```sql
create table t(x int, y int);
create index ti on t(x);
```

• what might happen with a set of statements like this:

```sql
insert into t (x,y) values (1,1);
update t set x = x+1 where x = 1;
delete from t where x = 2;
```

• What happens if the system fails at various points in the processing of these statements?

1. What happens if the buffer cache fills up?
2. What happens if we ROLLBACK at any point?
3. What happens if we succeed and COMMIT?
The INSERT

• The initial INSERT INTO T statement will generate both redo and undo.
• The undo generated will be enough information to make the INSERT “go away.”
• The redo generated by the INSERT INTO T will be enough information to make the INSERT “happen again.”
Hypothetical Scenario: The System Crashes Right Now

- System crashes before a COMMIT is issued or before the redo entries are written to disk.
- **Everything is OK.**
- SGA is wiped out, but we don’t need anything that was in the SGA.
- It will be as if this transaction never happened when we restart.
- None of the blocks with changes got flushed to disk, and none of the redo got flushed to disk.
- **We have no need of any of this undo or redo to recover from an instance failure.**
Hypothetical Scenario: The Buffer Cache Fills Up Right Now

- **DBWN** must make room and our modified blocks are to be flushed from the cache.
- DBWN will start by asking LGWR to flush the redo entries that protect these database blocks. Before DBWN can write any of the blocks that are changed to disk, LGWR must flush (to disk) the redo information related to these blocks.
- At this point, We have generated some modified table and index blocks. These have associated undo segment blocks, and all three types of blocks have generated redo to protect them.
Hypothetical Scenario: The Buffer Cache Fills Up Right Now Cont

• The redo log buffer is flushed *at least* every three seconds, when it is one-third full or contains 1MB of buffered data, or whenever a COMMIT or ROLLBACK takes place.

• It is very possible that at some point during our processing, the redo log buffer will be flushed

• We’ll have modified blocks representing uncommitted changes in the buffer cache and redo for those uncommitted changes on disk. This is a very normal scenario that happens frequently.
The UPDATE

- The UPDATE will cause much of the same work as the INSERT to take place.
- This time, the amount of undo will be larger; we have some “before” images to save as a result of the UPDATE.
- Dark rectangle in the redo log file represents the redo generated by the INSERT, the redo for the UPDATE is still in the SGA and has not yet been written to disk.
Hypothetical Scenario: The System Crashes Right Now

• Let’s assume that our redo generated from the INSERT statement is on disk (in the redo log file) and redo generated from the UPDATE is in cache.
• Redo for the UPDATE was only in the log buffer and never made it to disk (and was wiped out when the system crashed). That’s okay, the transaction was never committed and the data files on disk reflect the state of the system before the UPDATE took place.
• However, the redo for the INSERT was written to the redo log file. Therefore Oracle would “roll forward” the INSERT.
• WE would end up with modified undo blocks (information on how to undo the INSERT), modified table blocks (right after the INSERT), and modified index blocks (right after the INSERT).
Redo and Undo: Transaction Rollback

• When a ROLLBACK statement is issued, the database uses undo records to roll back changes made to the database by the uncommitted transaction.

• During recovery, DB rolls back any uncommitted changes applied from the online redo log to the data files.

• Undo records provide read consistency by maintaining the before image of the data for users accessing data at the same time that another user is changing it.
How Redo and Undo Work Together

• What happens during the processing of an INSERT with regard to redo and undo generation

• How Oracle uses this information in the event of failures at various points in time?

• **Undo information, stored in undo tablespaces or undo segments, is protected by redo as well.**
  
  – undo data is treated just like table data or index data—changes to undo generates some redo is logged to the log buffer and then the redo log file.

  – **Why ?**

• **Undo data is added to the undo segment and is cached in the buffer cache, just like any other piece of data would be.**
Example INSERT-UPDATE-DELETE-COMMIT Scenario

• created a table with an index as follows:

```sql
create table t(x int, y int);
create index ti on t(x);
```

• what might happen with a set of statements like this:

```sql
insert into t (x,y) values (1,1);
update t set x = x+1 where x = 1;
delete from t where x = 2;
```

• What happens if the system fails at various points in the processing of these statements?

1. What happens if the buffer cache fills up?
2. What happens if we ROLLBACK at any point?
3. What happens if we succeed and COMMIT?
The INSERT

- The initial INSERT INTO T statement will generate both redo and undo.
- The undo generated will be enough information to make the INSERT “go away.”
- The redo generated by the INSERT INTO T will be enough information to make the INSERT “happen again.”
Hypothetical Scenario: The System Crashes Right Now

- System crashes before a COMMIT is issued or before the redo entries are written to disk.
- Everything is OK.
- SGA is wiped out, but we don’t need anything that was in the SGA.
- It will be as if this transaction never happened when we restart.
- None of the blocks with changes got flushed to disk, and none of the redo got flushed to disk.
- We have no need of any of this undo or redo to recover from an instance failure.
Hypothetical Scenario: The Buffer Cache Fills Up Right Now

- **DBWn must make room and our modified blocks are to be flushed from the cache.**

- **DBWn will start by asking LGWR to flush the redo entries that protect these database blocks. Before DBWn can write any of the blocks that are changed to disk, LGWR must flush (to disk) the redo information related to these blocks.**

- **At this point, We have generated some modified table and index blocks. These have associated undo segment blocks, and all three types of blocks have generated redo to protect them.**
Hypothetical Scenario: The Buffer Cache Fills Up Right Now Cont

• The redo log buffer is flushed *at least every three seconds*, when it is one-third full or contains 1MB of buffered data, or whenever a COMMIT or ROLLBACK takes place.

• It is very possible that at some point during our processing, the redo log buffer will be flushed.

• we’ll have modified blocks representing uncommitted changes in the buffer cache and redo for those uncommitted changes on disk. This is a very normal scenario that happens frequently.
The UPDATE

• The UPDATE will cause much of the same work as the INSERT to take place.
• This time, the amount of undo will be larger; we have some “before” images to save as a result of the UPDATE.
• Dark rectangle in the redo log file represents the redo generated by the INSERT, the redo for the UPDATE is still in the SGA and has not yet been written to disk.
Hypothetical Scenario: The System Crashes Right Now

- Let’s assume that our redo generated from the INSERT statement is on disk (in the redo log file) and redo generated from the UPDATE is in cache.
- Redo for the UPDATE was only in the log buffer and never made it to disk (and was wiped out when the system crashed). That’s okay, the transaction was never committed and the data files on disk reflect the state of the system before the UPDATE took place.
- However, the redo for the INSERT was written to the redo log file. Therefore Oracle would “roll forward” the INSERT.
- We would end up with modified undo blocks (information on how to undo the INSERT), modified table blocks (right after the INSERT), and modified index blocks (right after the INSERT).
Hypothetical Scenario: The System Crashes Right Now Cont.

- To roll back the uncommitted INSERT, Oracle will use the undo it just rolled forward (from the redo and now in the buffer cache) and apply it to the data and index blocks, making them look as they did before the INSERT took place.

- Now everything is back the way it was. The blocks that are on disk may or may not reflect the INSERT (it depends on whether or not our blocks got flushed before the crash).

- If the blocks on disk do reflect the INSERT, then the INSERT will be undone when the blocks are flushed from the buffer cache. If they do not reflect the undone INSERT, so be it—they will be overwritten later anyway.
Hypothetical Scenario: The Application Rolls Back the Transaction

• At this point, Oracle will find the undo information for this transaction either in the cached undo segment blocks (most likely) or on disk if they have been flushed (more likely for very large transactions).

• It will apply the undo information to the data and index blocks in the buffer cache, or if they are no longer in the cache request, they are read from disk into the cache to have the undo applied to them.

• These blocks will later be flushed to the data files with their original row values restored.

• This scenario is much more common than the system crash. It is useful to note that during the rollback process, the redo logs are never involved.

• The only time redo logs are read for recovery purposes is during recovery and archival. This is a key tuning concept: redo logs are written to. Oracle does not read them during normal processing. As long as you have sufficient devices so that when ARCn is reading a file, LGWR is writing to a different device, there is no contention for redo logs.

• Many other databases treat the log files as “transaction logs.” They do not have this separation of redo and undo. For those systems, the act of rolling back can be disastrous—the rollback process must read the logs their log writer is trying to write to. They introduce contention into the part of the system that can least stand it. *Oracle’s goal is to make it so that redo logs are written sequentially, and no one ever reads them while they are being written.*
The DELETE

- Again, undo is generated as a result of the DELETE, blocks are modified, and redo is sent over to the redo log buffer.
- This is not very different from before. In fact, it is so similar to the UPDATE.
The DELETE

- Oracle will flush the redo log buffer to disk. The modified blocks are in the buffer cache; maybe some of them have been flushed to disk.
- *All of the redo* necessary to replay this transaction is safely on disk and the changes are now permanent. If we were to read the data directly from the data files, we probably would see the blocks as they existed *before* the transaction took place, as DBWn most likely has not yet written them.
- That’s OK—the redo log files can be used to bring those blocks up to date in the event of a failure. The undo information will hang around until the undo segment wraps around and reuses those blocks. Oracle will use that undo to provide for consistent reads of the affected objects for any session that needs them.
Redo

- Redo log files are crucial to the Oracle database.

- These are the transaction logs for the database.

- Oracle maintains two types of redo log files: online and archived.

- They are used for recovery purposes;

- Their main purpose in life is to be used in the event of an instance or media failure.
Online Redo Logs

- Oracle needs to store in the online redo logs any information that would be required to perform a crash recovery.
- At time 1, Block A is read from Data File AA into the buffer cache and modified.
- At time 2 the redo-change vector information (how the block changed) is written to the log buffer.
- At time 3 the log-writer process writes the Block A change-vector information to online redo log 1.
- At time 4 a log switch occurs, and online redo log 2 becomes the current online redo log.

Redo protected until the modified (dirty) buffer is written to disk.
Redo

• All database changes generate redo
  – Records changes made to
    • Data and index segments
    • Undo segments
    • Data dictionary
    • Control files (indirectly)

• Redo is used:
  – During recovery of database
    • Instance recovery
    • Media recovery
Redo and Undo : Transaction Rollback

• When a ROLLBACK statement is issued, the database uses undo records to roll back changes made to the database by the uncommitted transaction.

• During recovery, DB rolls back any uncommitted changes applied from the online redo log to the data files.

• Undo records provide read consistency by maintaining the before image of the data for users accessing data at the same time that another user is changing it
Redo and Undo

• redo
  – Key to Oracle’s durability (recovery) mechanism is redo

• undo
  – Core to multiversioning (read consistency).

• Oracle uses redo to capture how transaction changed the data.
  – Allows you to replay the transaction (in the event of an instance crash or a media failure).

• Oracle uses undo to store the before image of a modified block.
  – Allows you to reverse or rollback a transaction.
Undo

- Oracle DB maintains records of the actions of transactions, collectively known as undo data.

- Opposite of redo.

- Undo information is generated by DB as you make modifications to data so that the data can be put back the way it was before the modifications took place.

- Redo is used to replay a transaction in the event of failure—to recover the transaction—undo is used to reverse the effects of a statement or set of statements.
Undo

• **Oracle Database uses undo data to:**
  – Roll back an active transaction
  – Recover a terminated transaction
  – Provide read consistency: support of multiversioning. Uncommitted changes cannot be seen by other sessions
  – Perform some logical flashback operations

• **Oracle Database stores undo data inside the database rather than in external logs.**

• **Undo data is stored in blocks that are updated just like data blocks, with changes to these blocks generating redo records.**
Undo Segments and Transactions

- When a transaction starts, DB assigns transaction to an undo segment, and to a transaction table, in current undo tablespace.
- Multiple active transactions can write concurrently to the same undo segment or to different segments.
- EX: transactions T1 and T2 can both write to undo segment U1, or T1 can write to U1 while T2 writes to undo segment U2.
- Undo segment form a ring. Transactions write to one undo extent, and then to the next extent in the ring, and so on in cyclical fashion.
Read Consistency and Undo Segments

• To manage the multiversion read consistency model, the database must create a read consistent set of data when a table is simultaneously queried and updated. Oracle Database achieves this goal through undo data.

• Guarantees that data returned by a single query is committed and consistent for a single point in time.
  • Depends on the transaction isolation level and the query:
    • In the read committed isolation level, point is time at which the statement was opened.
      – EX: if a SELECT statement opens at SCN 1000, then this statement is consistent to SCN 1000.
    • In a serializable or read-only transaction, this point is the time the transaction began.
      – EX: if a transaction begins at SCN 1000, and if multiple SELECT statements occur in this transaction, then each statement is consistent to SCN 1000.
    • In a Flashback Query operation (SELECT ... AS OF), SELECT statement explicitly specifies the point in time.
      – EX: you can query a table as it appeared last Thursday at 2 p.m.
Corruption

• Sometimes a software or hardware malfunction can corrupt data.

• Physical corruption:
  – Bad header
  – Block is Fractured/Incomplete
  – Block checksum is invalid
  – Block is misplaced

• Logical Corruption
  • Block contains a valid checksum and the structure below the beginning of the block is corrupt.
User Error

UPDATE BANK_TBL SET BALANCE=500
WHERE ACCOUNT_NO=123456
Source Of Data Loss
Media Failure

• Media failure is a physical problem with a disk that causes a failure of a read from or write to a disk that is required to run the database.

• Any database file can be vulnerable to a media failure.
Murphy’s laws

• If anything can go wrong, it will

• If anything just cannot go wrong, it will anyway

• If everything seems to be going well, you have obviously overlooked something
Reminder – Oracle Concepts
Oracle Database 12c Architecture
Oracle Database 12c Architecture:
Major background processes

- **DBWn**: The database writer writes blocks from the database buffer cache to the data files.
- **CKPT**: The checkpoint process writes checkpoint information to the control files and data file headers.
- **LGWR**: The log writer writes redo information from the log buffer to the online redo logs.
Database Writer Process (DBWn)

- Write modified (dirty) buffer in database buffer cache to disk
- Asynchronously while performing other process to advance checkpoint.
Backup and Recovery Instance Architecture

• Checkpoint Process
  – Signals the database writer process (DBWn) at each checkpoint.
  – Updates the data file headers with the checkpoint information.
  – Updates the control files with the checkpoint information.
Checkpoint Process (CKPT)

- Updates the control file and data file headers with checkpoint information and signals DBW to write blocks to disk.
- Checkpoint information includes the checkpoint position, **SCN**, location in online redo log to begin recovery, and so on.
Redo Log Buffer

- A **circular** buffer in SGA that stores redo entries describing changes made to DB.

- A redo record is a data structure that contains the information necessary to reconstruct, or redo, changes made to the database by DML or DDL operations.

- DB recovery applies redo entries to data files to reconstruct lost changes.

- DB processes copy redo entries from the user memory space to the redo log buffer in the SGA.

- Redo entries take up continuous, sequential space in the buffer.
Log Writer Process (LGWR)

Writes the redo log buffer to a redo log file on disk writes

- When user commit a transaction
- When the redo log buffer is one-third full
- Before a DBWn process writes modified buffers to disk
- Every 3 seconds
Online Redo Logs (ORLs)

- Crucial to the Oracle database
- Transaction logs for DB
- Two types of redo log files: online and archived
- Used for recovery purposes in the event of an instance or media failure.
Online Redo Log Switches

- The log writer is the background process responsible for writing transaction information from redo log buffer (in the SGA) to the online redo log files (on disk).
- Log writer flushes the contents of the redo log buffer when any of the following are true:
  - A COMMIT is issued.
  - A log switch occurs.
  - Three seconds go by.
  - Redo log buffer is one-third full.
  - Redo log buffer fills to one megabyte.
Reuse of Online Redo Log

- **At time 1**, Block A is read from Data File AA into the buffer cache and modified.
- **At time 2** the redo-change vector information (how the block changed) is written to the log buffer.
- **At time 3** the log-writer process writes the Block A change-vector information to online redo log 1.
- **At time 4** a log switch occurs, and online redo log 2 becomes the current online redo log.

**Reuse of Online Redo Log Files**
Redo

• All database changes generate redo
  – Records changes made to
    • Data and index segments
    • Undo segments
    • Data dictionary
    • Control files (indirectly)

• Redo is used:
  – During recovery of database
    • Instance recovery
    • Media recovery
Redo and Undo

• redo
  – Key to Oracle’s durability (recovery) mechanism is redo

• undo
  – Core to multiversioning (read consistency).

• Oracle uses redo to capture how transaction changed the data.
  – Allows you to replay the transaction (in the event of an instance crash or a media failure).

• Oracle uses undo to store the before image of a modified block.
  – Allows you to reverse or rollback a transaction.
Undo

• Oracle Database uses undo data to:
  – Roll back an active transaction
  – Recover a terminated transaction
  – Provide read consistency: support of multiversioning. Uncommitted changes cannot be seen by other sessions
  – Perform some logical flashback operations

• Oracle Database stores undo data inside the database rather than in external logs.

• Undo data is stored in blocks that are updated just like data blocks, with changes to these blocks generating redo records.
Backup and Recovery Instance Architecture

- **Log Writer Process (LGWR)**
  - Process is responsible for transferring the contents of the redo log buffer to the online redo log files.
- **LGWR writes to the online redo files**
  - At each commit
  - Every three seconds
  - When the redo log buffer is one-third full
Logging and Archiving

- Archiver (ARCn) is an optional background process and is in charge of archiving the filled online redo log files, before they can be overwritten by new data.
- ARCn is used only if you’re running your DB in archivelog mode.
Recovery

• From which events below can DBMS recover?
  – Wrong data entry
  – Disk failure
  – Fire / earthquake/ bankruptcy ...

• Systems crashes
  – Software errors
  – Power failures
## Recovery

<table>
<thead>
<tr>
<th>Type of Crash</th>
<th>Prevention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrong data entry</td>
<td>Constraints and Data cleaning</td>
</tr>
<tr>
<td>Disk crashes</td>
<td>Redundancy: e.g. RAID, archive</td>
</tr>
<tr>
<td>Fire, theft, bankruptcy</td>
<td>Buy insurance, Change jobs…</td>
</tr>
<tr>
<td>System failures</td>
<td>DATABASE RECOVERY</td>
</tr>
</tbody>
</table>

*Most frequent*
Performing Flashback Recovery

- **Flashback Database**: Rewind a DB to an earlier time
- **Flashback Query**: Run SELECTS on a past data states
- **Flashback version Query**: Compare past data states
- **Flashback Transaction Query**: Retrieve historical metadata from past
- **Flashback Table**: Rewind a table to an earlier state
- **Flashback Table**: Rewind a table to an earlier state
Sources of Data loss

• Media failure
  – Environmental issue
  – Physical Problems with a disk
  – Remediation: fault tolerance / redundancy.

• User error:
  – Accidental or malicious deletion
  – Remediation: Training, security policy

• Application error
  – Poorly written code can corrupt data blocks
  – A user session can crash
  – SQL statements can fail
  – Remediation: Code Review
Backup and Recovery Instance Architecture

- Online redo log files record all changes made to DB.
- Oracle uses a write-ahead protocol: logs are written to before the data files. Therefore, it is critical to always protect the online logs against loss by ensuring they are multiplexed.
- Any changes made to DB are first recorded in the redo log buffer, which is part of the SGA.
- Redo log files come into play when a database instance fails or crashes. Upon restart, the instance will read the redo log files looking for any committed changes that need to be applied to the data files.
- When you commit, Oracle ensures that what you are committing has first been written to the redo log files before these changes are recorded in the actual data files.
- The redo log is the ultimate source of truth for all changes to the data in an Oracle DB, since an instance failure before the changes are written to the data files means that the changes are only in the redo log files but not in the data files.