chapitre 05

▼ Understanding “Change”

When a DML statement (e.g., INSERT, UPDATE, DELETE) is executed in an Oracle database, it starts a transaction. The changes made by the transaction are initially visible only to the session that made them. These changes become visible to other sessions only after a COMMIT is issued. If the transaction is rolled back, the changes are discarded.

Here is a simplified explanation of how changes are processed:

1. **Reading and Buffering**: Oracle reads the relevant data blocks from disk into the buffer cache.
2. **Undo Segment**: Before modifying the data blocks, Oracle writes a "before-image" of the data to the undo segment to enable rollback if necessary.
3. **Log Buffer**: Changes made to both data and undo blocks are written to the log buffer.
4. **Redo Logs**: The log buffer is periodically written to the online redo log files, ensuring that changes can be reapplied or rolled back during instance recovery.

The process ensures data consistency and durability, allowing for recovery in case of a failure.

▼ Differentiating Undo and Redo

Undo and redo are essential for managing changes in an Oracle database, serving different purposes:

- **Undo Data**:
    - Stores the old values before changes are made.

    - Used for rolling back transactions, ensuring read consistency, instance recovery, and flashback operations.
    - Dynamic in size, growing as needed.
- **Redo Data**:
    - Records all changes made to the database.
    - Used for rolling forward changes during recovery, archiving for point-in-time recovery, and analyzing changes with tools like Log Miner.
    - Stored in a fixed-size redo log buffer, periodically written to redo log files.

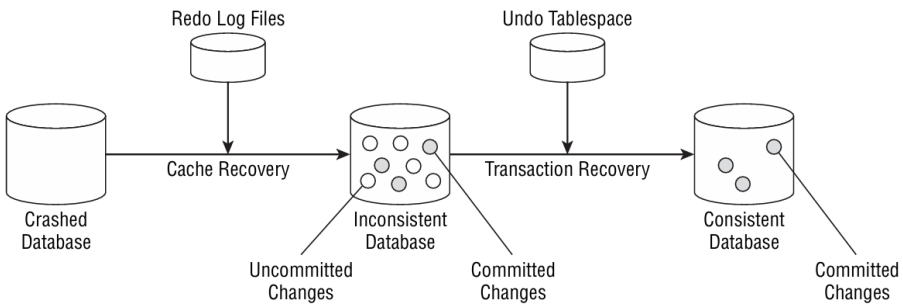
In summary, undo data allows for reversing changes and maintaining consistency, while redo data ensures changes can be reapplied or recovered after a failure.

▼ Undo and Redo Application During Instance Recovery

When an Oracle database is shut down cleanly, all changes in memory are written to disk, so no recovery is needed upon startup. However, if the shutdown is abrupt (e.g., SHUTDOWN ABORT) or the instance crashes, instance recovery is required to ensure data consistency. This recovery process, known as crash recovery, involves two main phases:

1. **Cache Recovery (Rolling Forward)**: Oracle applies all changes recorded in the redo log files since the last checkpoint to the data files. This includes both committed and uncommitted transactions, ensuring no committed changes are lost.
2. **Transaction Recovery (Rolling Back)**: Oracle then uses undo information to reverse any uncommitted changes applied during the roll-forward phase.

The database can be opened after the cache recovery phase is complete, but the rollback of uncommitted transactions continues in the background.



Instance recovery using undo and redo

Undo provides read consistency in Oracle databases by storing the previous values of data before any DML (Data Manipulation Language) changes are committed. When a user issues a query, the database uses undo data to present a consistent view of the data as it was before any ongoing transactions started. This ensures that users do not see uncommitted changes made by others during their queries.

For example, if User A starts a transaction and makes changes to a table, and then User B issues a SELECT query against the same table, User B will see the data as it was before User A's changes. The undo data allows the database to reconstruct the previous values of the data for User B's query.

Additionally, read consistency can be extended to an entire transaction using the `SET TRANSACTION READ ONLY` statement, ensuring that all queries within the transaction see a consistent snapshot of the data as it was before the transaction began.

In summary:

- **Read Consistency**: Ensures queries see data as it was before any uncommitted changes.
- **Undo Segments**: Store previous values of data for read consistency and rollback operations.

- **Read-Only Transactions**: Use `SET TRANSACTION READ ONLY` to ensure consistent data view throughout the transaction.

Configuring and Monitoring Undo in Oracle Databases

Automatic vs. Manual Undo Management:

- **Automatic Undo Management (Recommended)**: Use the initialization parameters `UNDO_MANAGEMENT` (set to `AUTO`), `UNDO_TABLESPACE`, and `UNDO_RETENTION` to configure and manage undo data.
- **Manual Undo Management**: Not recommended. Requires setting `UNDO_MANAGEMENT` to `MANUAL` and managing rollback segments manually.

Key Initialization Parameters:

1. **UNDO_MANAGEMENT**: Determines whether undo is managed manually or automatically. Set to `AUTO` for automatic management.
2. **UNDO_TABLESPACE**: Specifies the undo tablespace to use. Only one undo tablespace can be active at a time, but switching to a new tablespace will keep the old one active until all transactions using it are complete.
3. **UNDO_RETENTION**: Defines the retention period for undo data to support long-running queries and read consistency.

Switching Undo Tablespace:

- You can switch the active undo tablespace using `ALTER SYSTEM SET UNDO_TABLESPACE=undo_tbs_name`.

Monitoring Undo:

- Use views like `V$UNDOSTAT` to monitor undo space usage and calculate optimal size.

- **EM Database Express Undo Advisor**: A tool to help size or resize the undo tablespace.
- **RETENTION GUARANTEE**: Ensures that long-running queries have the necessary undo entries, even if it means DML transactions might fail.

Error Handling:

- **ORA-01650**: Not enough undo space for active transactions.
- **ORA-01555**: Snapshot too old, typically due to long-running queries and reused undo entries.

Example Commands:

```
-- Set automatic undo management
ALTER SYSTEM SET UNDO_MANAGEMENT = AUTO SCOPE=SPFILE;

-- Create or switch to a new undo tablespace
CREATE UNDO TABLESPACE undotbs2 DATAFILE 'undotbs2.dbf' SIZ
E 200M;
ALTER SYSTEM SET UNDO_TABLESPACE = undotbs2;

-- Check current undo tablespace
SHOW PARAMETER undo_tablespace;
```

By using automatic undo management, DBAs can reduce intervention, ensure read consistency, and leverage advanced features like flashback queries.

UNDO_RETENTION and Undo Management

- **UNDO_RETENTION**: Specifies how long (in seconds) committed undo data should be retained before it can be overwritten. This helps ensure read consistency and supports long-running queries.

- **Automatic Tuning**: Setting `UNDO_RETENTION` to zero enables automatic tuning, where Oracle adjusts retention to meet the needs of the longest-running query without necessarily extending the undo tablespace.
- **Guaranteed Undo Retention**: Using the `RETENTION GUARANTEE` clause ensures that the specified retention time is met, even if it means transactions may fail due to lack of space.

Categories of Undo Information:

1. **Uncommitted Undo**: Supports active transactions; never overwritten.
2. **Committed Undo**: Needed to satisfy `UNDO_RETENTION`; can be overwritten if required by new transactions.
3. **Expired Undo**: No longer needed and can be overwritten.

Example:

```
-- Show current UNDO_RETENTION
SHOW PARAMETER undo_retention;

-- Change UNDO_RETENTION to 12 hours (43200 seconds)
ALTER SYSTEM SET undo_retention = 43200;

-- Confirm change
SHOW PARAMETER undo_retention;
```

Flashback Operations: Dependent on undo data and automatic undo management, allowing various forms of data recovery and historical queries.

- **Flashback Query**: Query data as it existed in the past.
- **Flashback Version Query**: Query different versions of rows.
- **Flashback Transaction**: Rollback a transaction and its dependencies.
- **Flashback Transaction Query**: View changes at the transaction level.
- **Flashback Table**: Point-in-time recovery of a table.

Lock Modes and Their Usage

Oracle provides lock modes to manage concurrent access to database resources, ensuring data integrity during DML operations.

Types of Locks:

1. **Exclusive Lock (X)**: Prevents other transactions from accessing the locked resource. It is obtained during DML operations like INSERT, UPDATE, or DELETE.
2. **Shared Lock (S)**: Allows concurrent read access but prevents write access. Multiple transactions can acquire shared locks simultaneously.

Table Lock Modes:

- **ROW SHARE (RS)**: Allows concurrent access to the table but prohibits locking the entire table exclusively.
- **ROW EXCLUSIVE (RX)**: Allows concurrent access but prohibits SHARE locks; obtained automatically during DML operations.
- **SHARE (S)**: Allows read-only access, preventing updates. Required for creating indexes.
- **SHARE ROW EXCLUSIVE (SRX)**: Allows concurrent queries but prevents other SHARE locks or updates.
- **EXCLUSIVE (X)**: Most restrictive; allows only queries and prohibits any DML operations.

Manual Locking:

- **SELECT ... FOR UPDATE**: Locks selected rows to prevent other transactions from modifying them.
- **NOWAIT**: Returns immediately if the resource is locked.
- **WAIT**: Waits for a specified time to acquire the lock.

Handling Lock Conflicts:

- Lock conflicts can occur due to long-running transactions or inappropriate locking levels. These conflicts may need manual resolution, especially if a transaction is left uncommitted.

Example:

```
-- Obtain a share row exclusive lock with a 60-second wait
LOCK TABLE hr.employees IN SHARE ROW EXCLUSIVE MODE WAIT 6
0;
```