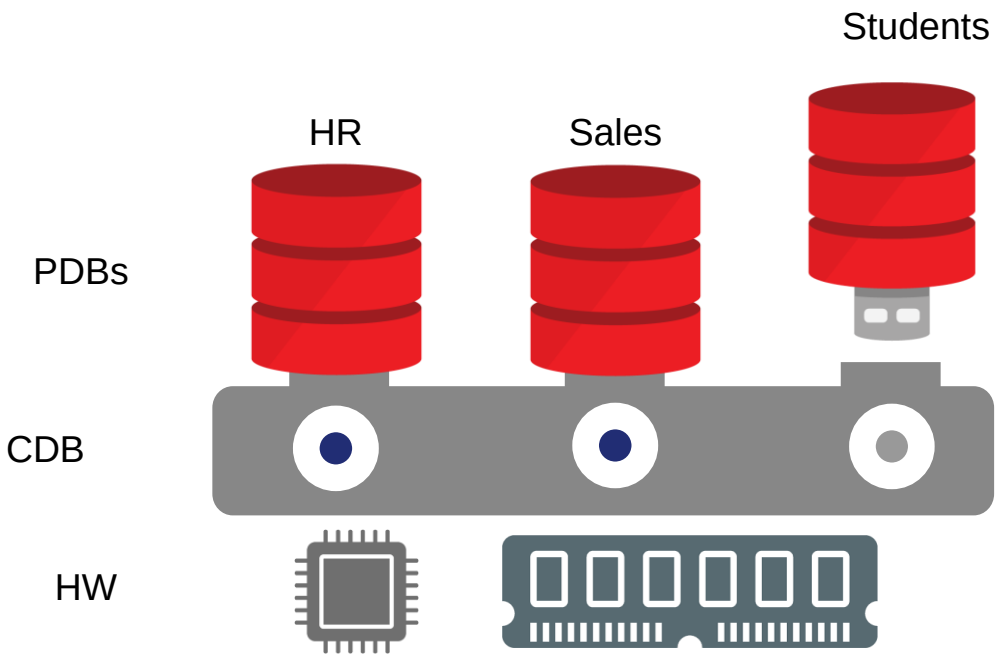
DATABASE ADMINISTRATION

Pluggable Database

Multitenant

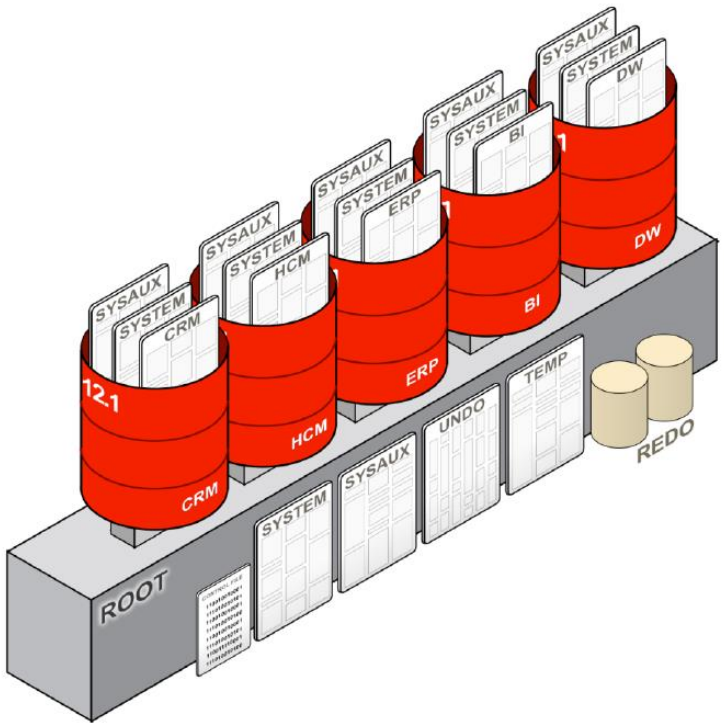
What is Oracle Multitenant



- Consolidation
- Easier management
- Easier deployment
- Higher portability
- Extra cost
- Extra complexity
- Competitors with same architecture: PostgreSQL, MS SQL

"Oracle Multitenant enables an Oracle Database to function as a container database (CDB). A CDB consolidates multiple pluggable databases (PDB), a portable collection of schemas, schema objects, and non-schema objects. Whether deployed on-premises or in the cloud, with Oracle Multitenant, applications run unchanged in self-contained PDBs, improving resource utilization, management, and overall security."
Multitenant | Oracle

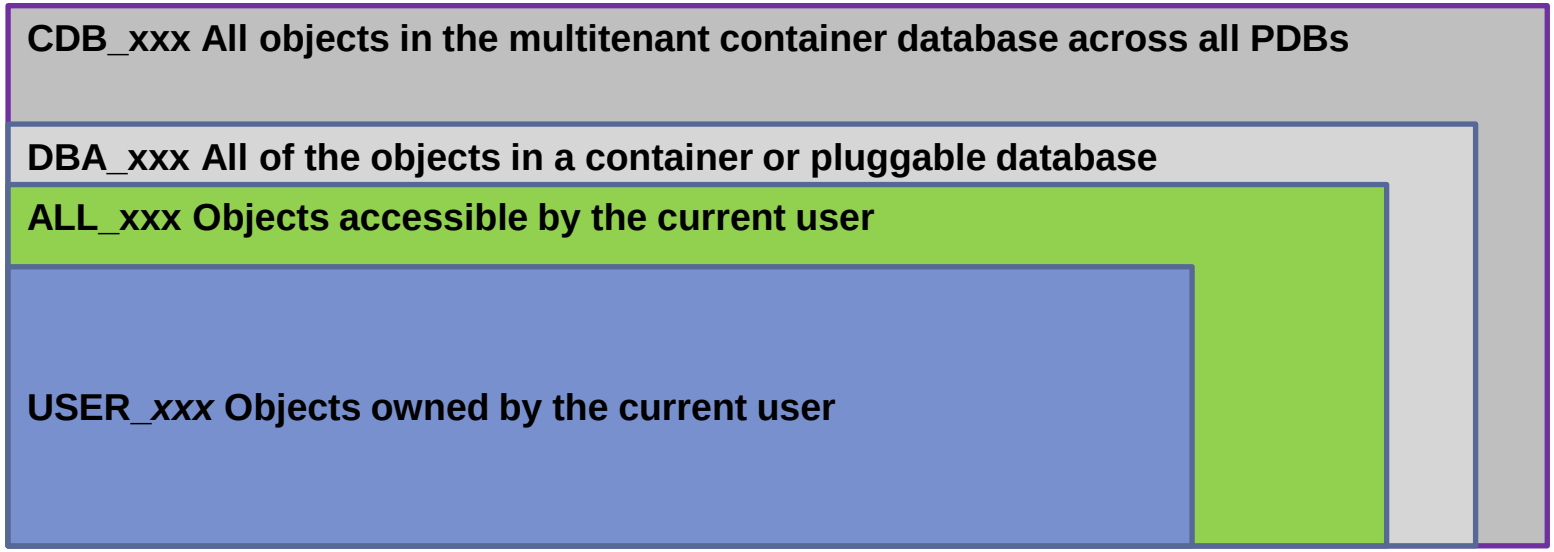
Shared and local resources



- Pluggable databases has its own set of datafiles and tablespaces
- Pluggable databases share the UNDO, REDO, CF and background processes
- They can have local users or common users
- Resource manager at CDB and PDB level
- Support for Real Application Clusters

Data dictionary changes

- A new level of data dictionary views was introduced



DATABASE ADMINISTRATION

T3: TRANSACTIONS AND CONCURRENCY [C7]

ACID Principles

CHAPTER PLAN

- 1. Transactions: general principles
  - ACID Properties: isolation principle
  - Transaction Isolation Level and SQL
  - Locking mechanisms
  - SQL Commands for transactional control

- 2. Oracle Mechanisms and structures to manage transaction in concurrent context
  - Oracle Structures to manage resource with access concurrent
    - Oracle Sessions
    - UNDO Segments
    - Redo log Files
  - Oracle Locking System
    - Oracle Isolation Levels
    - Lock types

1. Transactions: General Principles

- TRANSACTION concept:
  - Data Operation Set defined as an ATOMIC-unit-of-work regarding data consistency (ensuring data integrity), cancelling and recovery strategy.

- ACID Principles:
  - *Atomicity*: “all or nothing” behavior or indivisible database update operation set;
  - *Consistency*: a transaction starts and ends with a consistent database state:
    - consistent state: all the integrity constraints are validated;
  - ***Isolation***: multiple (parallel) transactions allowed, but each individual transactions runs *independently* by not interfering with other concurrent transactions;
  - *Durability*: transaction commit confirmation will ensure database persistent state (state of data will survive to further eventual failures).

SQL Isolation Levels

Isolation Levels	Dirty Read	Non repeatable Read	Phantom Read
Read uncommitted	✓	✓	✓
Read committed	X	✓	✓
Serializable	X	X	X

Transaction Isolation Strategies

- Lock strategy
  - Active transactions will acquire a set of concurrency marks (locks) that will be released at the end of transaction processing.
  - Locking *granularity*: column, row, page (block), table, tablespace, database.
  - Types of locks: shared lock, exclusive lock, update lock.
- Timestamp strategy
  - Every new transaction could by sequentially marked with their start timestamp.
  - Conflict resolution could be done by using transaction ageing ordering principle (older transactions will have higher priority).

Transaction Deadlocks

- Deadlocks occur when at least two transactions enter in a stale state waiting each other lock releases.
- Deadlock resolution strategies:
  - *Timeouts*.
  - *Prevention* by inferential computing of potential deadlocks before acquiring actual locks.
  - *Arbitrage* of database engine on active transaction graph: inferential computing to undo one or more active transactions and rolling database recovery.

SQL Commands

- *Setting transaction properties (isolation level):*
  - SET|START TRANSACTION
    - READ ONLY | READ WRITE
    - ISOLATION LEVEL
      - READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE
  - SET CONSTRAINTS ALL | <constraint name> [{ <comma> <constraint name> }...]
    - DEFERRED | IMMEDIATE
- *Setting transaction segments*:
  - SAVEPOINT <savepoint specifier>
  - RELEASE SAVEPOINT <savepoint specifier>
- *Confirmation or cancelling transaction:*
  - COMMIT [WORK]
  - ROLLBACK [WORK]
    - TO SAVEPOINT <savepoint specifier>

Practice C7_1

Steps	Notes	SQL Scripts
1. Deadlock scenario	SELECT … FOR UPDATE FOR	C7_P1.1_SELECT_DeadLock_A.sql C7_P1.2_SELECT_DeadLock_B.sql
2. Isolation Level Test Case: READ COMMITED	SET TRANSACTION ISOLATION LEVEL	C7_P1.1_SELECT_DeadLock_A.sql C7_P1.2_SELECT_DeadLock_B.sql
3. Isolation Level Test Case: SERIALIZABLE	SET TRANSACTION ISOLATION LEVEL	C7_P1.1_SELECT_DeadLock_A.sql C7_P1.2_SELECT_DeadLock_B.sql

Extra:

- Oracle Live SQL - Tutorial: Read Phenomena & Isolation Levels: Databases for Developers
- On Transaction Isolation Levels | Oracle Magazine

2. Oracle Transaction Management

- Oracle transactional support - structures and mechanisms:
  - Sessions: user and server processes
  - Memory Structures: PGA
  - UNDO Segments
  - Redo Log Files
- Oracle locking system
  - Oracle Isolation Levels
  - Oracle Lock Types

2.1 Oracle Transactional Support Structures

- Oracle user sessions:
  - User accounts privileges (DBA_USERS, DBA_SYS_PRIVS, DBA_TAB_PRIVS)
    - system privileges → to start Oracle sessions: CREATE SESSION
    - object privilegess → to access and update tables and indexes
  - Monitor transactional activity
    - V$SESSION: username, [sid, serial#], saddr, osuser, program, sql_id
    - V$TRANSACTION: addr, start_time, ses_addr
    - V$SQL: sql_id, sql_text, executions, users_executing, first_load_time, command_type
    - V$LOCK: addr, sid, type, lmode
- Oracle transactional supporting data structures:
  - UNDO data segments
  - REDO log entries

REDO and UNDO data structures

- REDO structures:
  - Used to *recover* database state from the last (physically) persistent copy and re-applying all transactional operations until last time when database was available.
- UNDO structures:
  - Used to *restore* database state at some previous state (before transaction) by executing the necessary restoring operations against transactional update operations.
- REDO vs. UNDO:
  - REDO: forward transactional operations;
  - UNDO: backward transactional operations.

REDO entries

- REDO Investigating Tool: DBMS_LOGMNR
- Dynamic view to analyze REDO data: V$LOGMNR_CONTENTS

```
DBMS_LOGMNR.ADD_LOGFILE('
     D:\APP\CATALIN.STRIMBEI\ORADATA\ORCL\REDO03.LOG');
DBMS_LOGMNR.START_LOGMNR(
     OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
     DBMS_LOGMNR.COMMITTED_DATA_ONLY);

SELECT timestamp, SCN, OPERATION, TABLE_NAME, ROW_ID, USERNAME, SQL_REDO
FROM V$LOGMNR_CONTENTS;

DBMS_LOGMNR.END_LOGMNR;
```

```
SELECT timestamp, SCN, OPERATION,
            TABLE_NAME, ROW_ID, USERNAME, SQL_REDO
FROM V$LOGMNR_CONTENTS;
```

Statement Output | Query Result | Query Result 1

All Rows Fetched: 303 in 0.012 seconds

	TIMESTAMP	SCN	OPERATION	TABLE_NAME	ROW_ID	USERNAME	SQL_REDO
289	20-APR-11	3368469	INSERT	INCASARI	AAAR9fAAGAAAAL7AAB	VINZARI01	insert into "VINZARI01"."INCASARI"("CODINC","DATAINC","TIPINC","CODDOC","NRDOC","DATADOC") va
290	20-APR-11	3368469	INSERT	INCASARI	AAAR9fAAGAAAAL7AAC	VINZARI01	insert into "VINZARI01"."INCASARI"("CODINC","DATAINC","TIPINC","CODDOC","NRDOC","DATADOC") va
291	20-APR-11	3368469	INSERT	INCASARI	AAAR9fAAGAAAAL7AAD	VINZARI01	insert into "VINZARI01"."INCASARI"("CODINC","DATAINC","TIPINC","CODDOC","NRDOC","DATADOC") va
292	20-APR-11	3368469	INSERT	INCASARI	AAAR9fAAGAAAAL7AAE	VINZARI01	insert into "VINZARI01"."INCASARI"("CODINC","DATAINC","TIPINC","CODDOC","NRDOC","DATADOC") va
293	20-APR-11	3368469	INSERT	INCASARI	AAAR9fAAGAAAAL7AAF	VINZARI01	insert into "VINZARI01"."INCASARI"("CODINC","DATAINC","TIPINC","CODDOC","NRDOC","DATADOC") va
294	20-APR-11	3368469	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAA	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1234','1111','53996');
295	20-APR-11	3368469	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAB	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1234','1118','101975');
296	20-APR-11	3368469	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAC	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1235','1112','125516');
297	20-APR-11	3368470	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAD	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1236','1117','9754');
298	20-APR-11	3368470	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAE	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1236','1118','100000');
299	20-APR-11	3368470	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAF	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1236','1120','7315');
300	20-APR-11	3368470	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAG	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1237','1117','9754');
301	20-APR-11	3368470	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAH	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1238','1113','106275');
302	20-APR-11	3368470	INSERT	INCASFACT	AAAR9fAAGAAAAL7AAI	VINZARI01	insert into "VINZARI01"."INCASFACT"("CODINC","NRFACT","TRANSA") values ('1239','1117','3696');
303	20-APR-11	3368471	COMMIT	(null)	AAAAAAAAAAAAAAAAAA	VINZARI01	commit;

UNDO data

- UNDO Investigating tool: mecanismul **FLASHBACK QUERY**
- Analyze UNDO data:
  - On dynamic view:
    - FLASHBACK_TRANSACTION_QUERY
  - On base tables with SQL clauses:
    - SELECT * FROM table
      - ***AS OF SCN nnn***;
    - SELECT * FROM tabela
      - ***AS OF TIMESTAMP 'dd/mm/yyyy hh24:mi:ss.FF'***;
    - SELECT * FROM tabela
      - ***VERSIONS BETWEEN scn1 AND scn2;***

SELECT START_SCN, OPERATION, TABLE_NAME, ROW_ID, UNDO_SQL
FROM FLASHBACK_TRANSACTION_QUERY;

Query Result | Statement Output | Query Result 1

Fetched 50 rows in 0.735 seconds

	START_SCN	OPERATION	TABLE_NAME	ROW_ID	UNDO_SQL
1	3368447	DELETE	JUDETE	AAAR84AAAAAAAAAAAA	insert into "VINZARI01"."JUDETE"("JUD","JUDET","REGIUNE") values ('VS','Vaslui','Moldova');
2	3368447	DELETE	JUDETE	AAAR84AAAAAAAAAAAA	insert into "VINZARI01"."JUDETE"("JUD","JUDET","REGIUNE") values ('VN','Vrancea','Moldova');
3	3368447	DELETE	JUDETE	AAAR84AAAAAAAAAAAA	insert into "VINZARI01"."JUDETE"("JUD","JUDET","REGIUNE") values ('TM','Timis','Banat');
4	3368447	DELETE	JUDETE	AAAR84AAAAAAAAAAAA	insert into "VINZARI01"."JUDETE"("JUD","JUDET","REGIUNE") values ('SV','Suceava','Moldova');
5	3368447	DELETE	JUDETE	AAAR84AAAAAAAAAAAA	insert into "VINZARI01"."JUDETE"("JUD","JUDET","REGIUNE") values ('NT','Neamt','Moldova');
6	3368447	DELETE	JUDETE	AAAR84AAAAAAAAAAAA	insert into "VINZARI01"."JUDETE"("JUD","JUDET","REGIUNE") values ('IS','Iasi','Moldova');
7	3368447	DELETE	CODURI_POSTALE	AAAR86AAGAAAAW1AAM	insert into "VINZARI01"."CODURI_POSTALE"("CODPOST","LOC","JUD") values ('706400','Birlad','VS');
8	3368447	DELETE	CODURI_POSTALE	AAAR86AAGAAAAW1AAK	insert into "VINZARI01"."CODURI_POSTALE"("CODPOST","LOC","JUD") values ('706500','Vaslui','VS');
9	3368447	DELETE	CODURI_POSTALE	AAAR86AAGAAAAW1AAL	insert into "VINZARI01"."CODURI_POSTALE"("CODPOST","LOC","JUD") values ('705300','Focsani','VN');
10	3368447	DELETE	CODURI_POSTALE	AAAR86AAGAAAAW1AAP	insert into "VINZARI01"."CODURI_POSTALE"("CODPOST","LOC","JUD") values ('701900','Timisoara','TM');
11	3368447	DELETE	CODURI_POSTALE	AAAR86AAGAAAAW1AAN	insert into "VINZARI01"."CODURI_POSTALE"("CODPOST","LOC","JUD") values ('705800','Suceava','SV');
12	3368447	DELETE	CODURI_POSTALE	AAAR86AAGAAAAW1AAO	insert into "VINZARI01"."CODURI_POSTALE"("CODPOST","LOC","JUD") values ('705550','Roman','NT');
13	3368447	DELETE	CODURI_POSTALE	AAAR86AAGAAAAW1AAJ	insert into "VINZARI01"."CODURI_POSTALE"("CODPOST","LOC","JUD") values ('701150','Pascani','IS');
14	3368447	DELETE	CODURI_POSTALE	AAAR86AAGAAAAW1AAI	insert into "VINZARI01"."CODURI_POSTALE"("CODPOST","LOC","JUD") values ('700505','Iasi','IS');
15	3368447	DELETE	CLIENTI	AAAR8+AAGAAAAX1AAN	insert into "VINZARI01"."CLIENTI"("CODCL","DENCL","CODFISCAL","ADRESA","CODPOST","TELEFON") va

SELECT with FlashBack

FlashBack from UNDO	Transactional data versions
SELECT * FROM JUDETE AS OF SCN 3460502;	
SELECT * FROM JUDETE AS OF SCN 3460513;	
SELECT * FROM JUDETE AS OF SCN 3460558;	

JUD	JUDET	REGIUNE

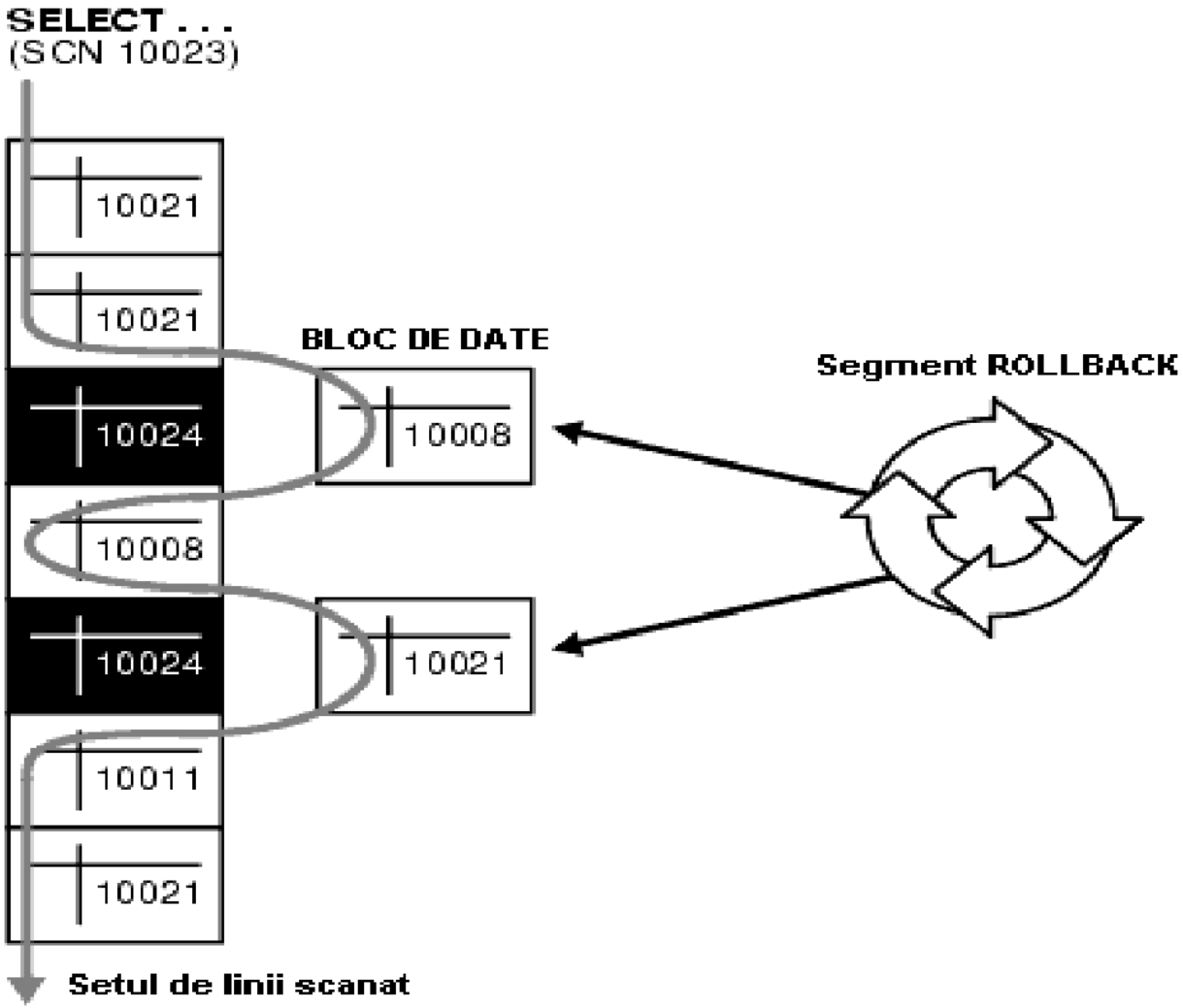
	JUD	JUDET	REGIUNE
1	IS	Iasi	Moldova
2	NT	Neamt	Moldova
3	SV	Suceava	Moldova
4	TM	Timis	Banat
5	VS	Vaslui	Moldova

	JUD	JUDET	REGIUNE
1	IS	Iasi	Moldova
2	NT	Neamt	Moldova
3	SV	Suceava	Moldova

```
SELECT jud, judet, regiune,
versions_startscn, versions_endscn, versions_operation
FROM JUDETE VERSIONS BETWEEN SCN 3460502 AND 3460588;
```

	JUD	JUDET	REGIUNE	VERSIONS_STARTSCN	VERSIONS_ENDSCN	VERSIONS_OPERATION
1	IS	Iasi	Moldova	3460509	(null)	I
2	NT	Neamt	Moldova	3460509	(null)	I
3	SV	Suceava	Moldova	3460509	(null)	I
4	TM	Timis	Banat	3460583	(null)	I
5	TM	Timis	Banat	3460554	(null)	D
6	TM	Timis	Banat	3460509	3460554	I
7	VN	Vrancea	Moldova	3460554	(null)	D
8	VN	Vrancea	Moldova	3460509	3460554	I
9	VS	Vaslui	Moldova	3460583	(null)	I
10	VS	Vaslui	Moldova	3460554	(null)	D
11	VS	Vaslui	Moldova	3460509	3460554	I

SQL SELECT execution within concurrent transactional context



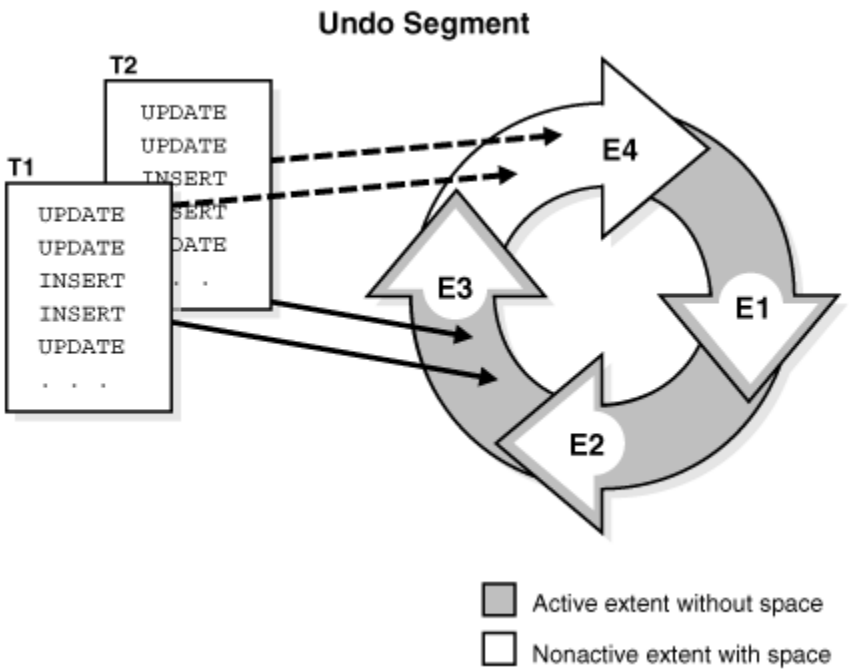
UNDO|ROLLBACK Segment Types

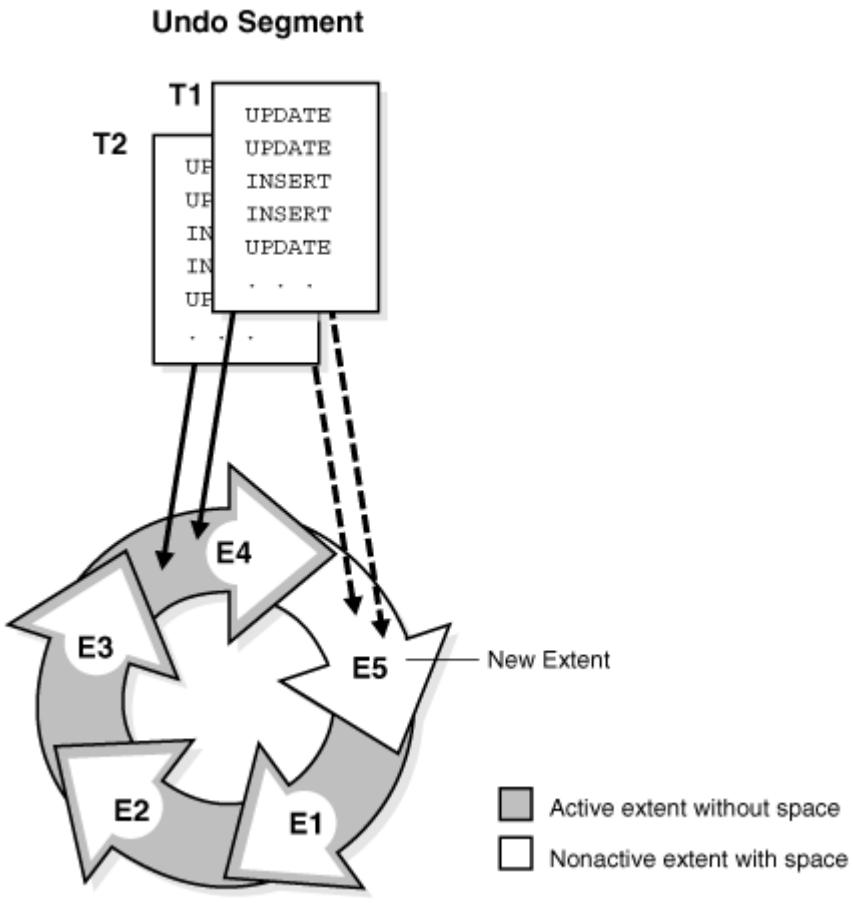
- **UNDO Segment Types**:
  - ***System*** – for database dictionary transactions;
  - ***Non-system;***
  - ***Private*** – exclusive for a single instance;
  - ***Public*** – shared between instances (RAC | GRID Cluster Architectures).
- UNDO Segments activating modes:
  - *auto* on instance start process
- [undo_management=auto]
- or [undo_management=auto *rollback_segments=…,…*];
  - explicit by admin statement:

    ```
    ALTER ROLLBACK SEGMENT roll_seg_name ONLINE;
    ```
- UNDO Segments allocating modes:
  - *automat* on new session start process;
  - manually – by explicit statement before transaction start:

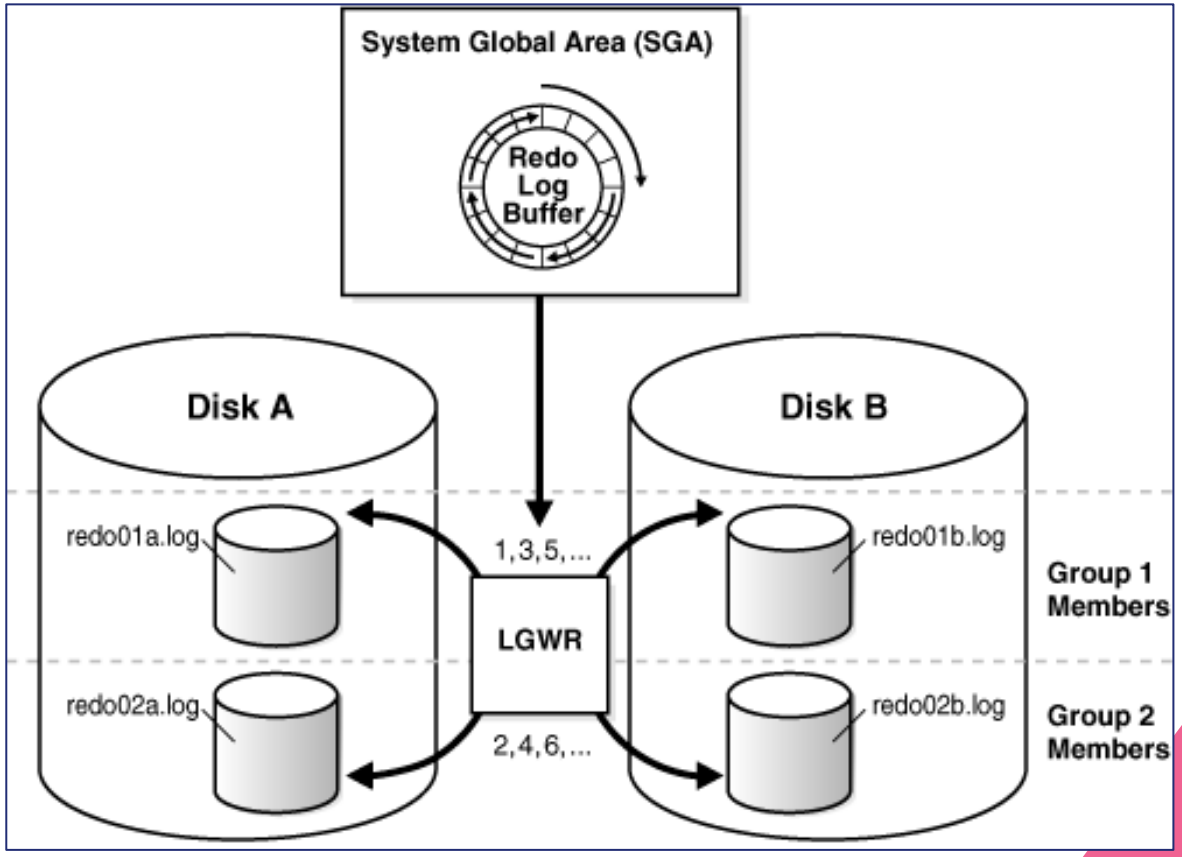
    ```
    SET TRANSACTION ROLLBACK SEGMENT roll_seg_name;
    ```

UNDO Segment structure and dynamics





REDO Data Structures



Administration: REDO LOG members and groups

```
ALTER DATABASE [database]
ADD LOGFILE [GROUP integer] filespec
[, [GROUP integer] filespec … ]
```

```
ALTER DATABASE [database]
ADD LOGFILE MEMBER
    [ ‘filename’ [REUSE]
    [,‘filename’ [REUSE]] …
TO {GROUP integer | (‘filename’ [,‘filename’] …)}
```

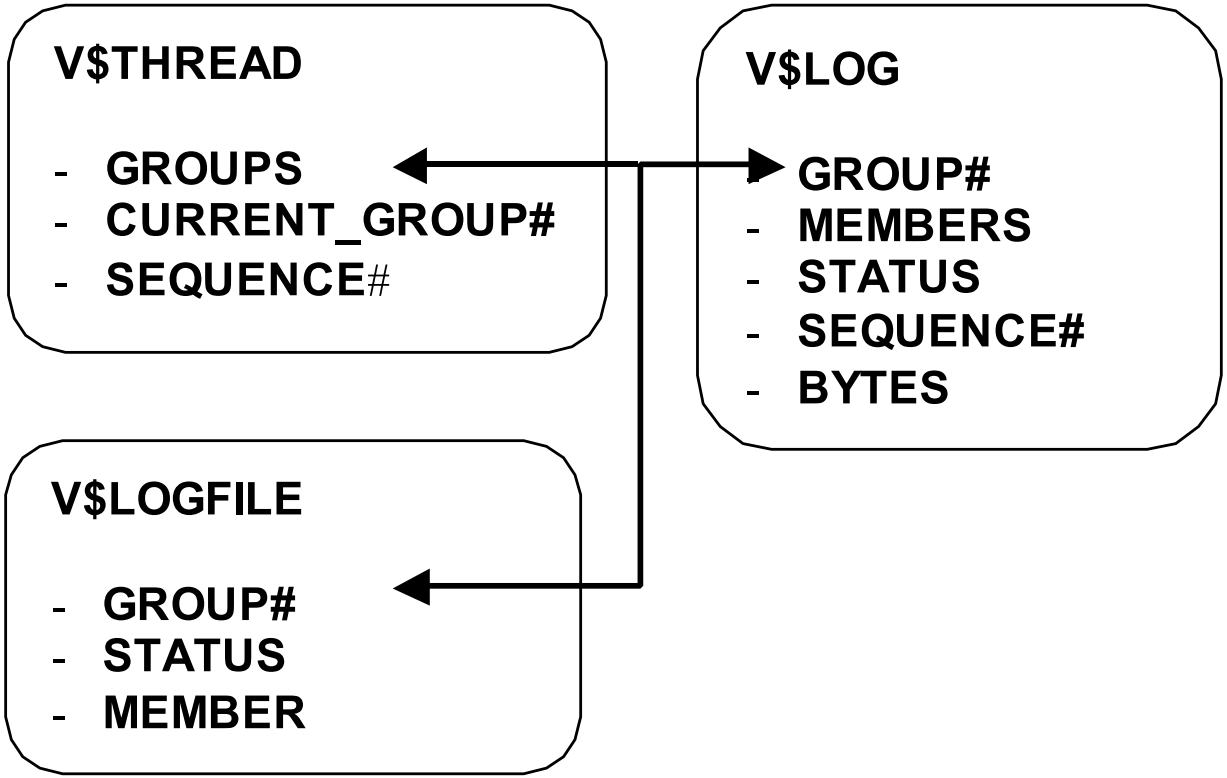
Administration: REDO LOG members and groups

```
ALTER DATABASE [database]
DROP LOGFILE
{GROUP integer | (‘filename’ [,‘filename’] …)}
[, {GROUP integer | (‘filename’ [,‘filename’] …)}] …
```

```
ALTER DATABASE [database]
DROP LOGFILE MEMBER
     ‘filename’
    [,‘filename’] …
```

```
ALTER DATABASE [database]
CLEAR LOGFILE
{GROUP integer | (‘filename’ [,‘filename’] …)}
[, {GROUP integer | (‘filename’ [,‘filename’] …)}] …
```

Administration: REDO LOG members and groups



Practice C7_2

Steps	Notes	SQL Scripts
1. Get REDO records from LOGs files	DBMS_LOGMNR.START_LOGMNR V$LOGMNR_CONTENTS	C7_P2.1_REDO_UNDO.sql
2. Get UNDO info from FLASHBACK data	SELECT ... FROM FLASHBACK_TRANSACTION_QUERY SELECT … AS OF SCN SELECT … AS OF TIMESTAMP	C7_P2.1_REDO_UNDO.sql

2.2 Oracle Locking Mechanisms

- Oracle locking modes:
  - Exclusive Mode:
    - First transaction that successfully acquire the lock will exclusively access database resource until lock release (at the end of transaction: rollback or commit).
  - Share Mode:
    - Some transactions could share (to some extent) some database resources (as of reading operations), meanwhile other transactions could not acquire an exclusive lock and all conflictual operations (as for writing) will be rejected.
- Oracle DML locking levels:
  - TX (row level) - maximum concurrency level, maximum granularity;
  - TM (table level) - from DML operations to prevent DDL operations that could change some definitions of the tables under update.

Oracle Table Lock Types

- Row Share - RS
- Row Exclusive - RX
- Share - S
- Share Row Exclusive - SRX
- Exclusiv - X

Oracle Row and Table Lock Modes

DML Statement	Row Lock	Table Lock
SELECT ... FROM table		
INSERT INTO table ...	X	RX
UPDATE table ...	X	RX
DELETE FROM table ...	X	RX
SELECT ... FROM table ... FOR UPDATE OF ...	X	RS
LOCK TABLE table IN ...		
ROW SHARE MODE		RS
ROW EXCLUSIVE MODE		RX
SHARE MODE		S
SHARE EXCLUSIVE MODE		SRX
EXCLUSIVE MODE		X

Oracle Table Lock Modes Compatibilities

Compatible	RS	RX	S	SRX	X	Obs
RS	Yes	Yes	Yes	Yes	No	
RX	Yes	Yes	No	No	No	
S	Yes	No	Yes	No	No	Queries No UPD
SRX	Yes	No	No	No	No	Queries No UPD
X	No	No	No	No	No	Queries No UPD

Practice C7_3

Steps	Notes	SQL Scripts
1. Table Lock: Share Lock Mode	LOCK TABLE IN SHARE MODE	C7_P3.1_Strategii_blocaje_ORCL_A.sql C7_P3.2_Strategii_blocaje_ORCL_B.sql
2. Table Lock: Share Exclusive Mode	LOCK TABLE IN SHARE ROW EXCLUSIVE MODE	C7_P3.1_Strategii_blocaje_ORCL_A.sql C7_P3.2_Strategii_blocaje_ORCL_B.sql
3. Table Lock: Share Row Mode	LOCK TABLE IN ROW SHARE MODE	C7_P3.1_Strategii_blocaje_ORCL_A.sql C7_P3.2_Strategii_blocaje_ORCL_B.sql

Oracle Live SQL - Tutorial: Update & Transactions: Databases for Developers

References

- Scott Jesse, Bill Burton, Bryan Vongray *Oracle Database 11g. Release 2 High Availability Maximize Your Availability with Grid Infrastructure, Oracle Real Application Clusters, and Oracle Data Guard, Second Edition*, 2011, The McGraw-Hill Companies, Inc. (Publisher)
- Robert G. Freeman, Matthew Hart, *Oracle RMAN 11g Backup and Recover, 2010*, The McGraw-Hill Companies, Inc. (Publisher)
- David C. Knox, Scott G. Gaetjen, Hamza Jahangir, Tyler Muth, Patrick Sack, Richard Wark, Bryan Wise, *Applied Oracle Security: Developing Secure Database and Middleware Environments*, 2010 by McGraw-Hill Companies, Inc
- Paul Wright, *Protecting Oracle Database 12c*, 2014 Apress
- Links
  - https://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm
  - http://databasefundas.blogspot.ro/2011/08/oracle-locks.html