MONASH University

MONASH
INFORMATION
TECHNOLOGY

Update, Delete and Transaction Management

GROUP OF EIGHT AUSTRALIA

MODIFYING ROWS USING UPDATE AND DELETE

MONASH University

UPDATE

- Changes the value of existing data.
- For example, at the end of semester, change the mark and grade from null to the actual mark and grade.

```
UPDATE table
SET column = (subquery) [, column = value, ...]
[WHERE condition];
```

```
UPDATE enrolment
SET mark = 80,
      grade ='HD'
WHERE sno = 112233
  and ……
```

```
UPDATE enrolment
SET mark = 85
WHERE unit_code = (SELECT unit_code FROM unit WHERE
                              unit_name='Introduction to databases')
          AND mark = 80;
```

DELETE

- Removing data from the database

```
DELETE FROM table
[WHERE condition];
```

```
DELETE FROM enrolment
WHERE sno='112233'
      AND
          unit_code= (SELECT unit_code FROM unit
                      WHERE unit_name='Introduction to Database' )
      AND
          semester='1'
      AND
          year='2012';
```

TRANSACTIONS

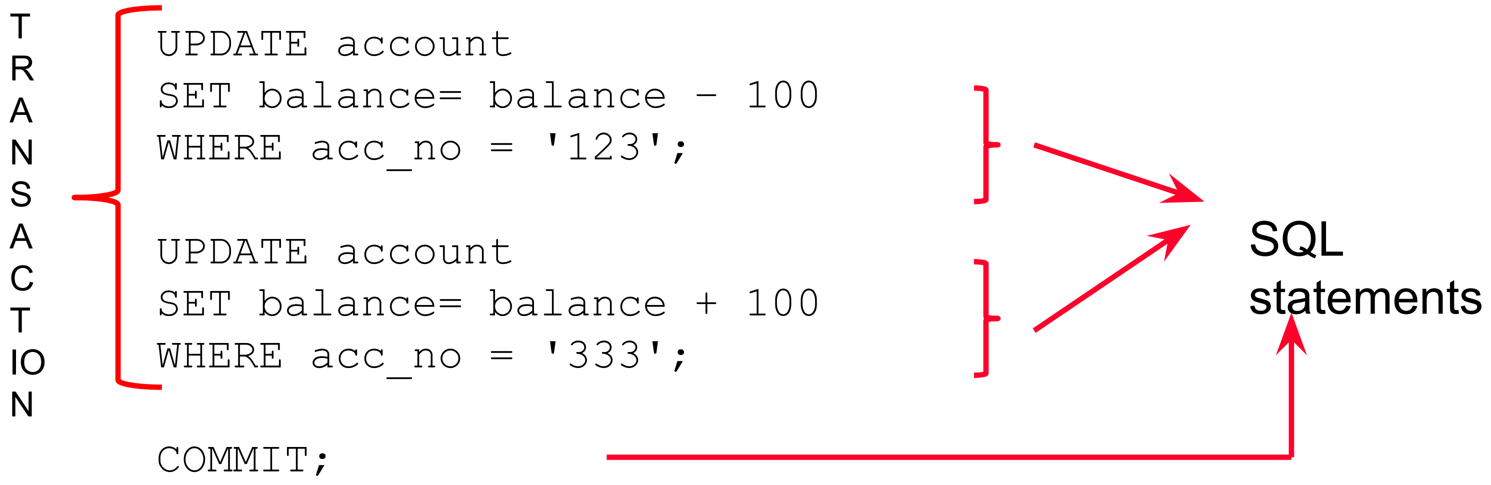
Transactions

- Consider the following situation.

  Sam is transferring $100 from his bank account to his friend Jim's.

  – Sam's account should be reduced by 100.
  – Jim's account should be increased by 100.

Assume that Jim’s account number is '333'. The transfer of money from Sam’s to Jim’s account will be written as the following SQL transaction:



```
UPDATE account
SET balance= balance – 100
WHERE acc_no = '123';

UPDATE account
SET balance= balance + 100
WHERE acc_no = '333';

COMMIT;
```

All statements need to be run as a single logical unit operation.

Transaction Properties

- A transaction must have the following properties:
  - **Atomicity**
    - all database operations (SQL requests) of a transaction must be entirely completed or entirely aborted
  - **Consistency**
    - it must take the database from one consistent state to another
  - **Isolation**
    - it must not interfere with other concurrent transactions
    - data used during execution of a transaction cannot be used by a second transaction until the first one is completed
  - **Durability**
    - once completed the changes the transaction made to the data are durable, even in the event of system failure

Consistency - Example

- Assume that the server lost its power during the execution of the money transfer transaction, only the first statement is completed (taking the balance from Sam’s).
- Consistency properties ensure that Sam’s account will be reset to the original balance because the money has not be transferred to Jim’s account.
- The last consistent state is *when the money transfer transaction has not been started.*

Durability - Example

- Assume the server lost power after the commit statement has been reached.

- The durability property ensures that the balance on both Sam’s and Jim’s accounts reflect the completed money transfer transaction.

Transaction Management

- Follows the ACID properties.
- Transaction boundaries
  - Start
    - first SQL statement is executed (eg. Oracle)
    - Some systems have a BEGIN WORK type command
  - End
    - COMMIT or ROLLBACK
- Concurrency Management
- Restart and Recovery.

Concurrency

Serial and ***Interleaved*** transactions.

Time:

T0	T1
Read(X)	
X=X+1	
Write(x)	
	Read(Y)
	Y=Y*2
	Write(Y)
	Read(x)
	X=X+2
	Write(X)

Serial

T0	T1
Read(X)	
	Read(Y)
	Y=Y*2
X=X+1	
Write(x)	
	Write(Y)
	Read(x)
	X=X+2
	Write(X)

Interleaved (non Serial)

MONASH University

The impact of interleaved transactions

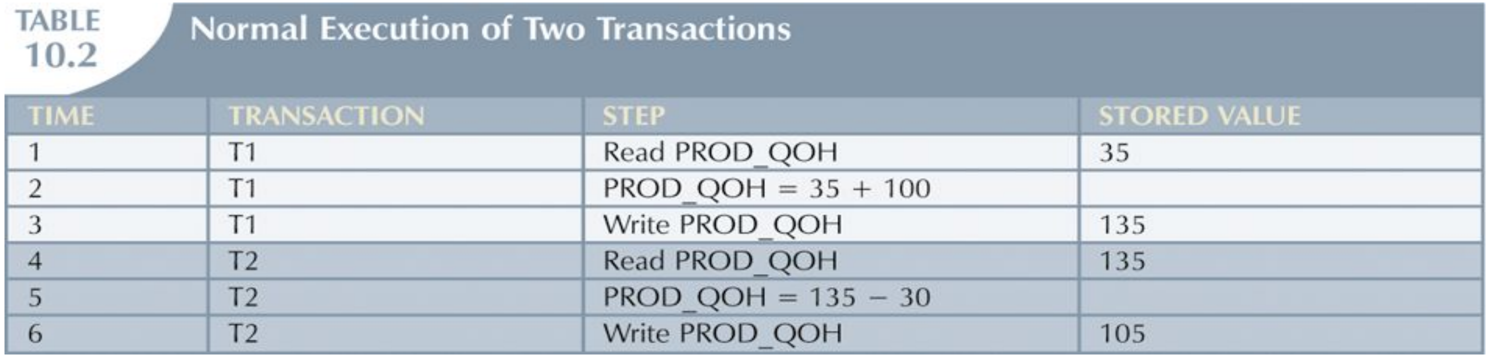


TABLE 10.2 Normal Execution of Two Transactions

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH = 135 − 30	
6	T2	Write PROD_QOH	105

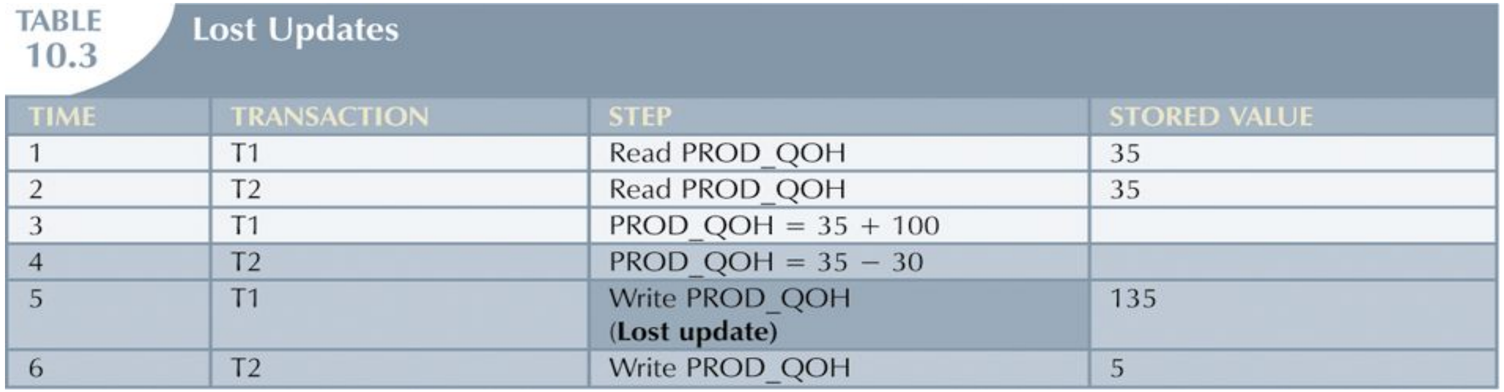


TABLE 10.3 Lost Updates

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 − 30	
5	T1	Write PROD_QOH **(Lost update)**	135
6	T2	Write PROD_QOH	5

Concurrency Management - Solution

- Locking mechanism.
  - A mechanism to overcome the problems caused by interleaved transactions.
- A lock is an indicator that some part of the database is temporarily unavailable for update because:
  - one, or more, other transactions is reading it, or,
  - another transaction is updating it.
- A transaction must acquire a lock prior to accessing a data item and locks are released when a transaction is completed.
- Locking, and the release of locks, is controlled by a DBMS process called the Lock Manager.

Lock Granularity

- Granularity of locking refers to the size of the units that are, or can be, locked. Locking can be done at
  - database level
  - table level
  - page level
  - record level

    Allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page.
  - attribute level

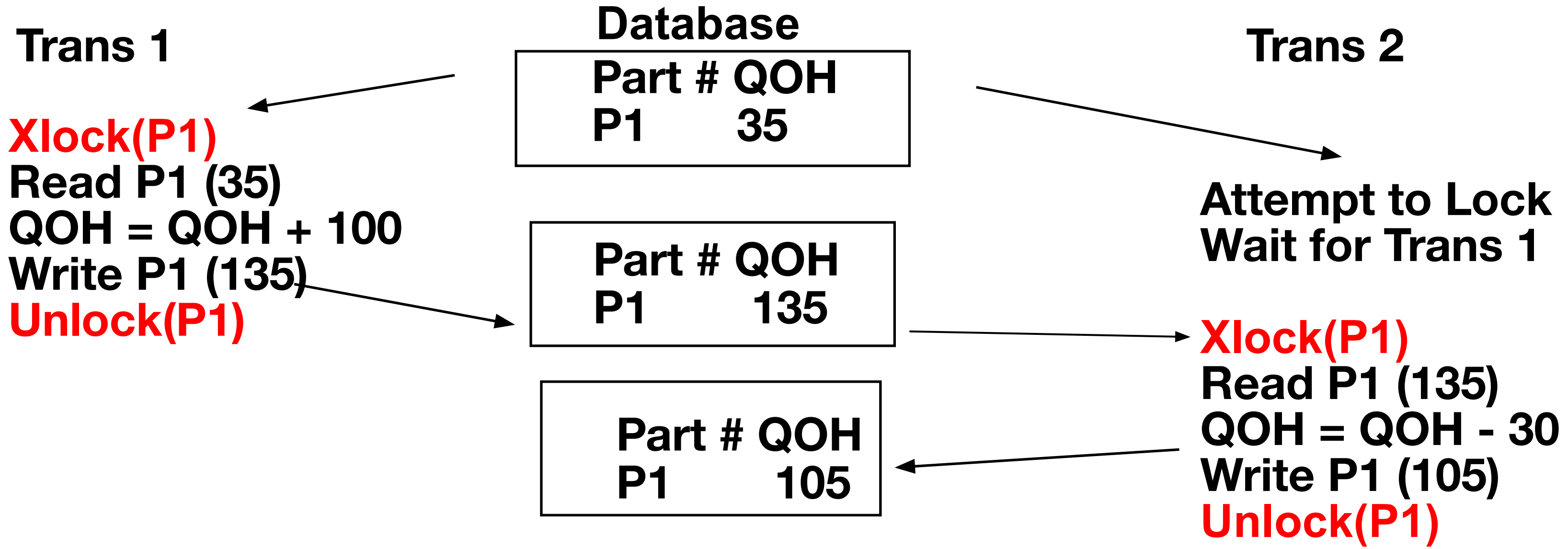
    Allows concurrent transactions to access the same row, as long as they require the use of different attributes within that row.

Lock Types

- Shared lock. Multiple processes can simultaneously hold shared locks, to enable them to read without updating.
  - if a transaction **T**$_i$ has obtained a shared lock (denoted by **S**) on data item **Q**, then **T**$_i$ can **read** this item but not **write** to this item
- Exclusive lock. A process that needs to update a record must obtain an exclusive lock. Its application for a lock will not proceed until all current locks are released.
  - if a transaction **T**$_i$ has obtained an exclusive lock (denoted **X**) on data item **Q,** then **T**$_i$ can both **read** and **write** to item **Q**

Exclusive Locks – Example 1

- Write-locked items
    - require an Exclusive Lock
    - a single transaction exclusively holds the lock on the item



Trans 1

Xlock(P1)
Read P1 (35)
QOH = QOH + 100
Write P1 (135)
Unlock(P1)

Database

Part #	QOH
P1	35

Part #	QOH
P1	135

Part #	QOH
P1	105

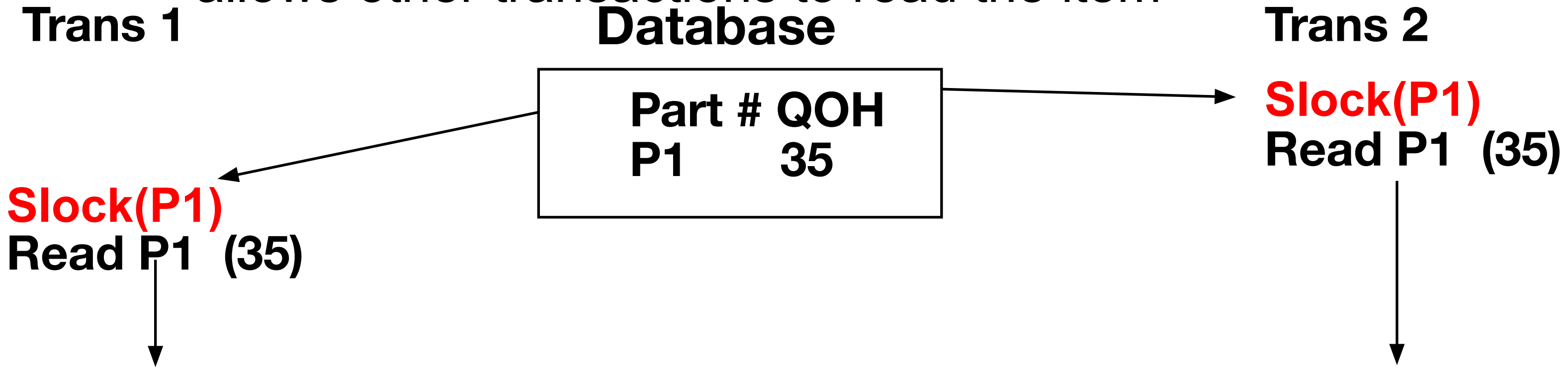
Trans 2

Attempt to Lock
Wait for Trans 1

Xlock(P1)
Read P1 (135)
QOH = QOH - 30
Write P1 (105)
Unlock(P1)

Shared Locks – Example 2

- Read-locked items
  - require a Shared Lock
  - allows other transactions to read the item



- **Shared locks** improve the amount of concurrency in a system

If **Trans 1** and **Trans 2** only wished to read **P1** with no subsequent update they could both apply an **Slock** on **P1** and continue

Lock - Problem

- Deadlock.

Scenario:

- Transaction 1 has an exclusive lock on data item A, and requests a lock on data item B.
- Transaction 2 has an exclusive lock on data item B, and requests a lock on data item A.

Result: Deadlock, also known as “deadly embrace”.

Each has locked a resource required by the other, and will not release that resource until it can either commit, or abort. Unless some “referee” intervenes, neither will ever proceed.

Dealing with Deadlock

- Deadlock prevention
  - A transaction must acquire all the locks it requires before it updates any record.
  - If it cannot acquire a necessary lock, it releases all locks, and tries again later.

- Deadlock detection and recovery
  - Detection involves having the Lock Manager search the Wait-for tables for lock cycles.
  - Resolution involves having the Lock Manager force one of the transactions to abort, thus releasing all its locks.

Dealing with Deadlock

- If we discover that the system is in a state of deadlock, some of the transactions causing the deadlock must be aborted. Choosing which transaction to abort is called as *victim selection*.
- The algorithm for victim selection should generally avoid selecting transactions that have been running for a long time and that have performed many updates, and should try instead to select transactions that have not made any changes or that are involved in more than one deadlock cycle in the wait-for graph.

Database Restart and Recovery

- Restart
  - Soft crashes
    - loss of volatile storage, but no damage to disks. These necessitate restart facilities.
- Recovery
  - Hard crashes
    - hard crashes - anything that makes the disk permanently unreadable. These necessitate recovery facilities.
- Requires transaction log.

Transaction Log

- The **log**, or journal, tracks all transactions that update the database. It stores
  - For each transaction component (SQL statement)
    - Record for beginning of transaction
    - Type of operation being performed (update, delete, insert)
    - Names of objects affected by the transaction (the name of the table)
    - “Before” and “after” values for updated fields
    - Pointers to previous and next transaction log entries for the same transaction
    - The ending (COMMIT) of the transaction

The log should be written to a **multiple** separate physical devices from that holding the database, and must employ a force-write technique that ensures that every entry is immediately written to stable storage, that is, the log disk or tape.

Sample Transaction Log

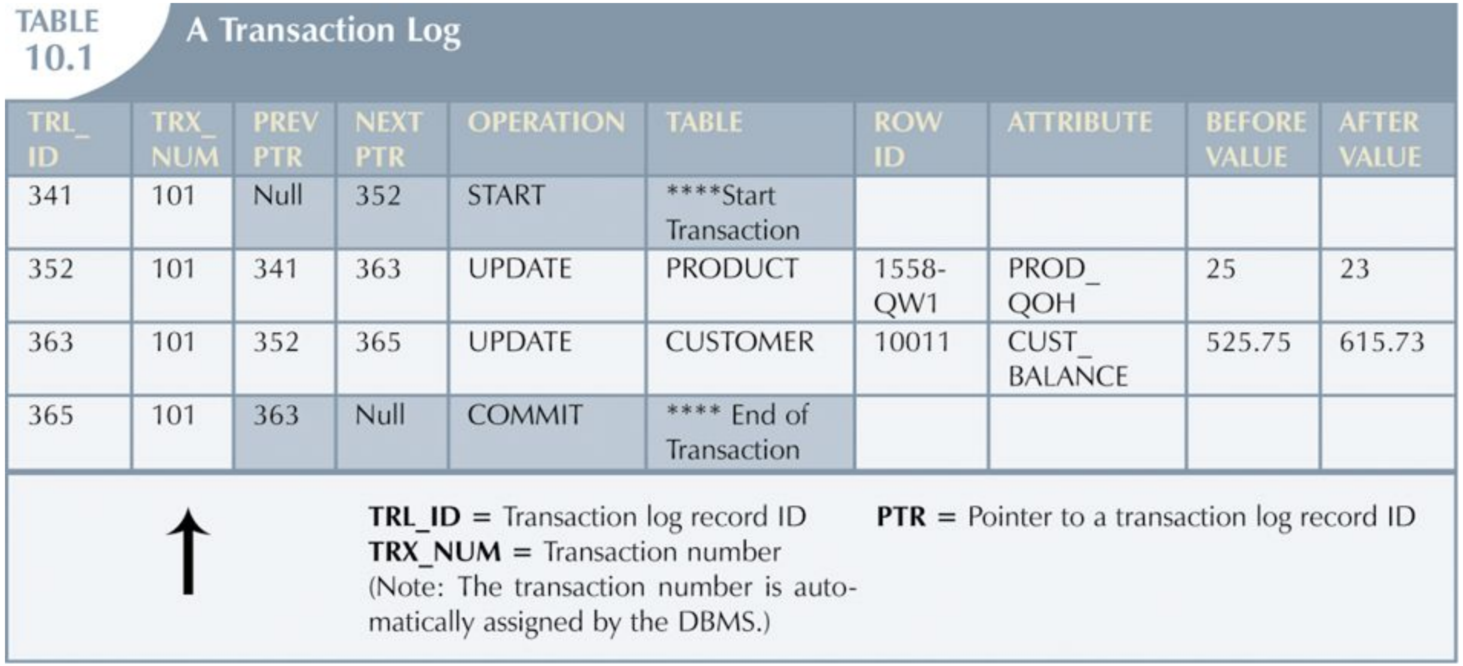


TABLE 10.1 A Transaction Log

TRL_ ID	TRX_ NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				

↑

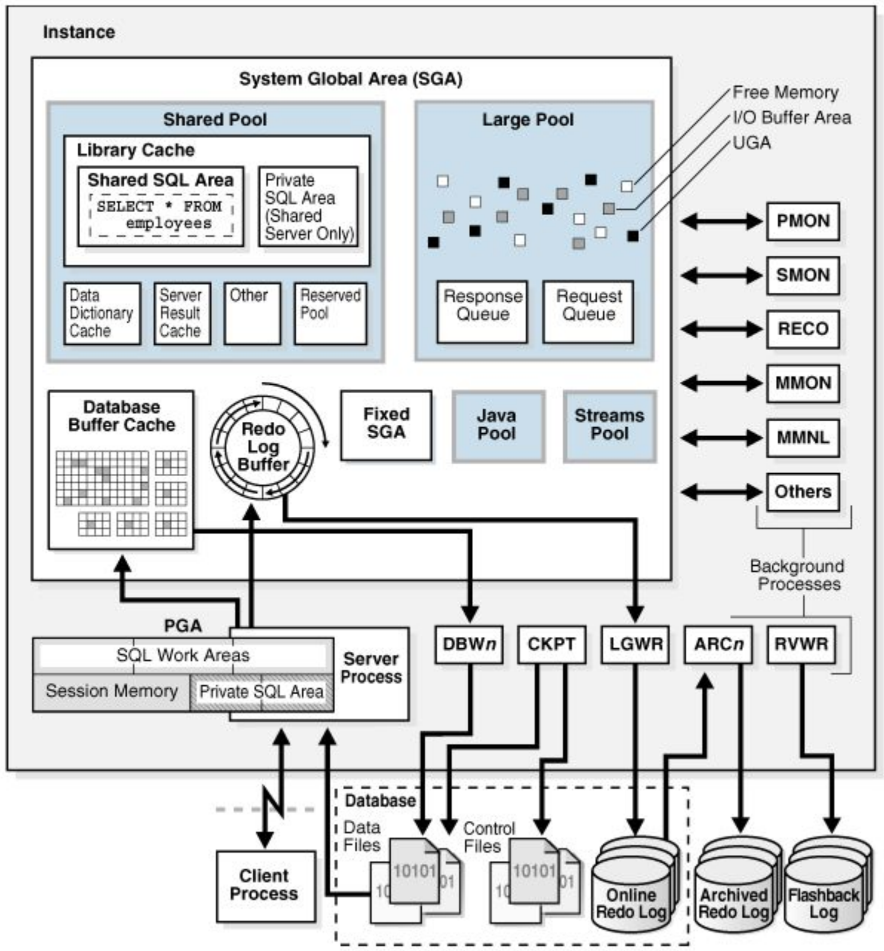
TRL_ID = Transaction log record ID
TRX_NUM = Transaction number
(Note: The transaction number is automatically assigned by the DBMS.)

PTR = Pointer to a transaction log record ID

Checkpointing

- Although there are a number of techniques for checkpointing, the following explains the general principle. A checkpoint is taken regularly, say every 15 minutes, or every 20 transactions.
- The procedure is as follows:
  - Accepting new transactions is temporarily halted, and current transactions are suspended.
  - Results of committed transactions are made permanent (force-written to the disk).
  - A checkpoint record is written in the log.
  - Execution of transactions is resumed.

Oracle database – *not examined*



Write Through Policy

- The database is immediately updated by transaction operations during the transaction's execution, before the transaction reaches its commit point
- If a transaction aborts before it reaches its commit point a ROLLBACK or UNDO operation is required to restore the database to a consistent state
- The UNDO (ROLLBACK) operation uses the log before values

Restart Procedure for Write Through

- Once the cause of the crash has been rectified, and the database is being restarted:
  - The last checkpoint before the crash in the log file is identified. It is then read forward, and two lists are constructed:
  - a REDO list containing the transaction-ids of transactions that were committed.
  - and an UNDO list containing the transaction-ids of transactions that never committed
- The database is then rolled forward, using REDO logic and the after-images and rolled back, using UNDO logic and the before-images.

An alternative - Deferred Write

- The database is updated only after the transaction reaches its commit point
- Required roll forward (committed transactions redone) but does not require rollback

Recovery

- A hard crash involves physical damage to the disk, rendering it unreadable. This may occur in a number of ways:
  - Head-crash. The read/write head, which normally “flies” a few microns off the disk surface, for some reason actually contacts the disk surface, and damages it.
  - Accidental impact damage, vandalism or fire, all of which can cause the disk drive and disk to be damaged.
- After a hard crash, the disk unit, and disk must be replaced, reformatted, and then re-loaded with the database.

Backup

- A backup is a copy of the database stored on a different device to the database, and therefore less likely to be subjected to the same catastrophe that damages the database. (NOTE: A backup is not the same as a checkpoint.)
- Backups are taken say, at the end of each day’s processing.
- Ideally, two copies of each backup are held, an on-site copy, and an off-site copy to cater for severe catastrophes, such as building destruction.
- Transaction log – backs up only the transaction log operations that are not reflected in a previous backup of the database.

Recovery

- Rebuild the database from the most recent backup. This will restore the database to the state it was in say, at close-of-business yesterday.

- **REDO** all committed transactions up to the time of the failure - no requirement for **UNDO**