MONASH University

MONASH
INFORMATION
TECHNOLOGY

Update, Delete and Transaction Management

Workshop Q&A 2021S2

GROUP OF EIGHT AUSTRALIA

MODIFYING ROWS USING UPDATE AND DELETE

UPDATE

- Changes the value of existing data.
- For example, it has been observed that when the TRAINING data for the drone system was added the description had an error and the hours for the course are incorrect.

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

```
UPDATE training
SET train_desc = 'DJI Hobby Drone Training',
       train_hrs = 5
WHERE train_code = 'DJIHY'
```

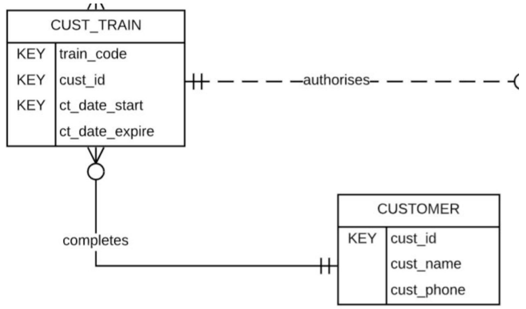
Update the drone cost/hr by 10% for all DJI Inspire 2 drones purchased after the 31st March 2021

```
UPDATE drone
SET drone_cost_hr  = drone_cost_hr * 1.1
WHERE dt_code = (SELECT dt_code FROM drone_type WHERE
                                dt_model = 'DJI Inspire 2')
          AND drone_pur_date > to_date('31-Mar-2021','dd-Mon-yyyy');
```

DELETE

- Removing data from the database

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



HiFlying Drones have a number of customers who have registered but never attended a training course, remove these customers from the CUSTOMER table

```
DELETE FROM customer
WHERE cust_id NOT IN (SELECT DISTINCT cust_id FROM cust_train);
```

TRANSACTIONS

Transactions

- Consider the following situation.

  When a drone returns from a rental two activities are required:

  - *the return date is recorded, **and***
  - *the drone flight time is updated to reflect the time the customer has flown*

Assume that rental number 239 was returned on the 27th April 2021 after having been flown for 120.3 minutes as part of this rental (the data is read from the drone). The SQL involved is:

T
R
A
N
S
A
C
T
IO
N

```
UPDATE rental
SET rent_in = to_date('27-Apr-2021','dd-Mon-yyyy')
WHERE rent_no = 239;

UPDATE DRONE
SET drone_flight_time = drone_flight_time + 120.3
WHERE drone_id = (SELECT drone_id FROM rental
                  WHERE rent_no = 239);
COMMIT;
```

SQL statements

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

Q1. Given the following transaction:

```
UPDATE rental
SET rent_in = to_date('27-Apr-2021','dd-Mon-yyyy')
WHERE rent_no = 239;

UPDATE DRONE
SET drone_flight_time = drone_flight_time + 120.3
WHERE drone_id = (SELECT drone_id FROM rental  WHERE rent_no = 239);

COMMIT;
```

If the power for the database is lost after the first update to the RENTAL table, this (multiple answers possible):

A. Can be ignored, no action is necessary
B. Is a atomic property issue
C. Is an isolation issue
D. Is a consistency issue
E. Is a durability issue

MONASH University

Consistency - Example

- Assume that the server lost its power after the execution of the first step of the drone return transaction. We now have a drone back but with the incorrect flight time - the database is inconsistent
- Consistency properties ensure, when the database is recovered, that RENTAL and DRONE tables will be returned to their states before the drone return process started.
- The last consistent state is *when the return step of the transaction has not been started.*

Durability - Example

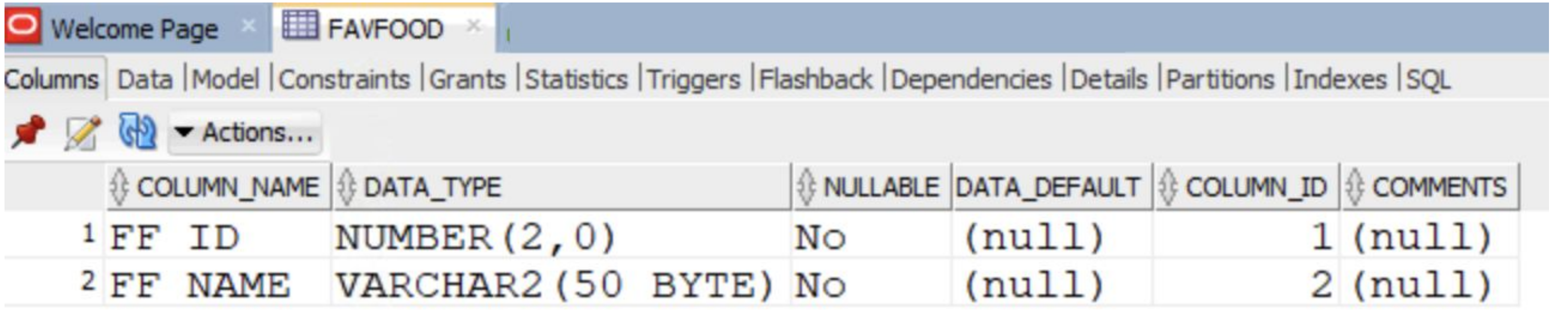
- Assume the server lost power after the commit statement has been reached
  - Note that the change may not have been written to disk, it only existed in memory prior to the power loss

- The durability property ensures that the drones RENTAL rent_in date and DRONE drone_flight_time are maintained as the correct values when the server is restarted.

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.

Insert into a table FAVFOOD


Welcome Page
FAVFOOD
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL
Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	FF_ID	NUMBER(2,0)	No	(null)	1	(null)
2	FF_NAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)

Please note your insert is being monitored and recorded, we will be displaying who entered what soon

Q2. Given two transactions:

T1 – R(X), W(X)
T2 – R(Y), W(Y), R(X), W (X)

Where R(X) means Read(X) and W(X) means Write(X)

How many different sequences (schedules) are possible if serial execution is involved

A. 1
B. It depends on the DBMS
C. 6
D. 2

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)

Q3. Transaction T1 is calculating the total flight time of all drones while T2 is, at the same time, returning a drone and updating the drone_flight_time for a drone which has been returned. Calculating the total flight time involves read only access to the database.

This is an example of:

A. Lost Update
B. Uncommitted Data
C. Inconsistent Retrieval
D. None of these, this action causes no problems

The impact of interleaved transactions - Inconsistent Retrieval

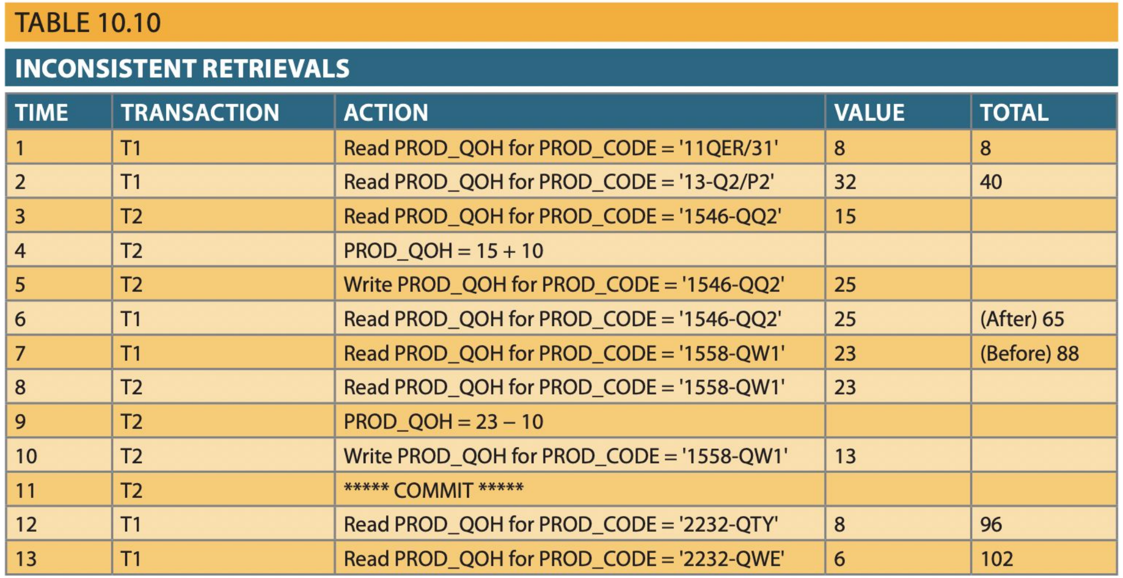


TABLE 10.10

INCONSISTENT RETRIEVALS

TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 – 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

Other possible issues:

- Lost Updates (covered in online Workshop)
- Uncommitted data (data read which is later rolled back - see text)

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.

Q4. A database using locking to support concurrency control will implement (multiple answers are possible):

A. Read locks
B. Wait locks
C. Timed locks
D. Commit locks
E. Write locks

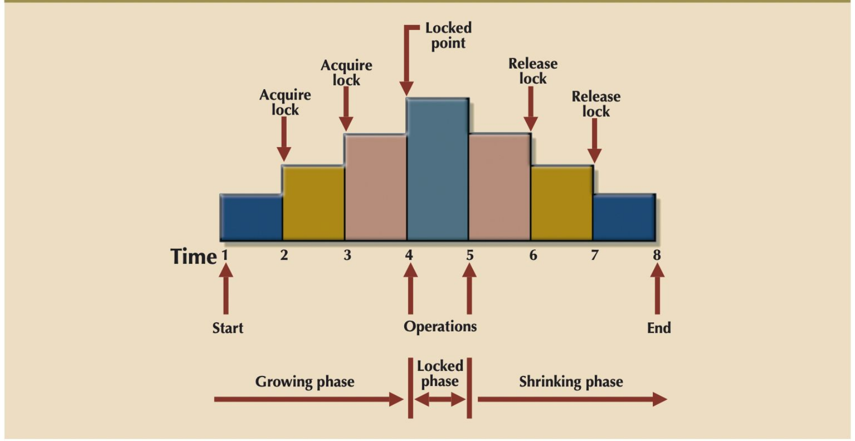
Lock Types

- 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**

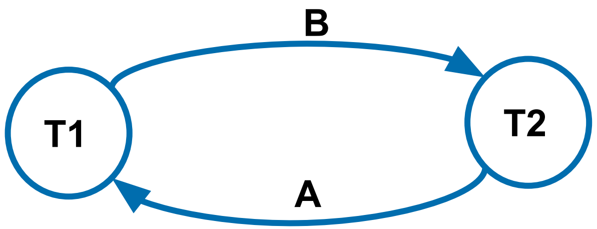
Two-Phase locking (2PL) to Ensure Serialisability

- Growing phase where locks are acquired
  - Once all locks acquired -> locked point
  - Transaction data changes made during locked point
- Shrinking phase where locks are released
  - Transaction cannot obtain a new lock during this phase

FIGURE 10.7 TWO-PHASE LOCKING PROTOCOL



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 requesting a lock is aborted and restarted if it would cause a deadlock
- Deadlock avoidance
  - 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

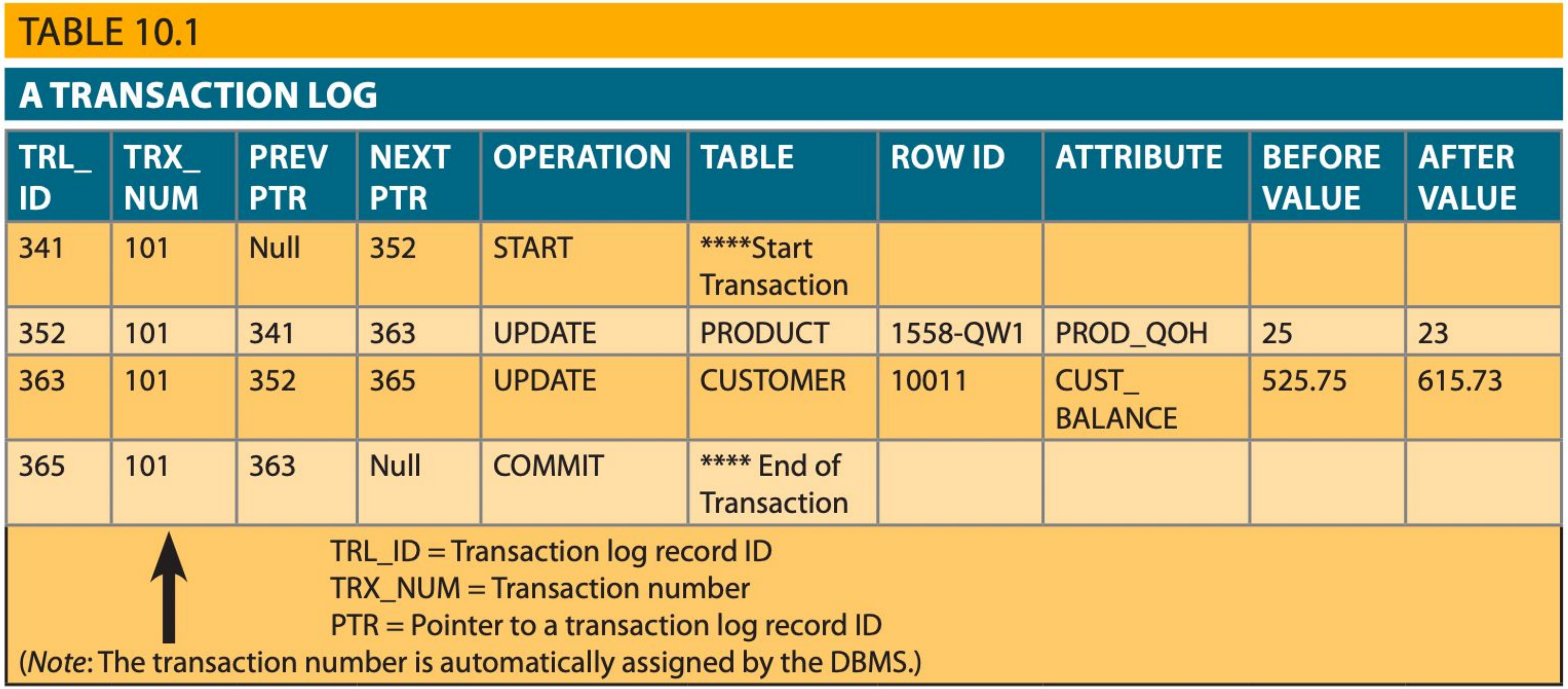


TABLE 10.1

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
PTR = Pointer to a transaction log record ID

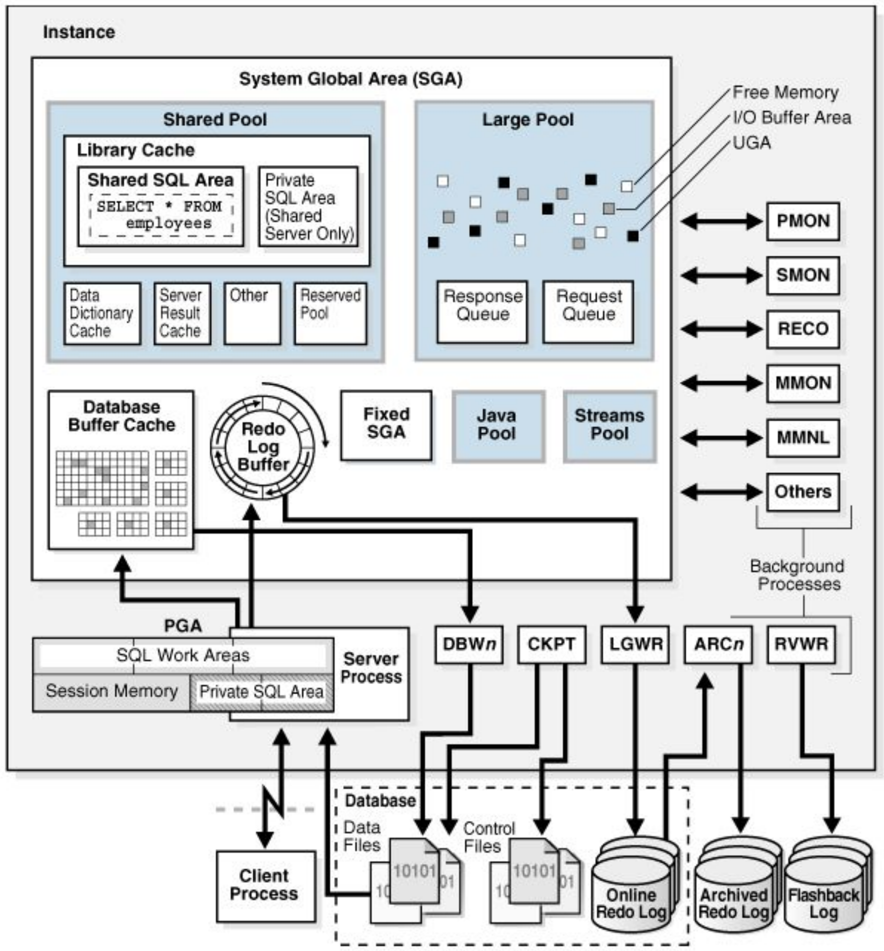
(*Note*: The transaction number is automatically assigned by the DBMS.)

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*

INSTANCE
(memory resident)



DATABASE
(on disk)

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.

DBMS recovery process using the ***write through*** method

Checkpoint

Transactions recorded in the log (green complete)

a	1	b	2	c	d	e	3	4	5	f	g	h	6	i	j	k	l	7	m

Step 1: Using the log compile REDO and UNDO lists

REDO list							**UNDO list**												
1	2	3	4	5	6	7	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**

Step 2: UNDO incomplete or rolled back transactions starting from newest

m	**l**	**k**	**j**	**i**	**h**	**g**	**f**	**e**	**d**	**c**	**b**	**a**

Step 3: REDO committed transactions starting from oldest

1	2	3	4	5	6	7

MONASH University

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

DBMS recovery process using the ***deferred write*** method

Checkpoint

Transactions recorded in the log (green complete)

a	1	b	2	c	d	e	3	4	5	f	g	h	6	i	j	k	l	7	m

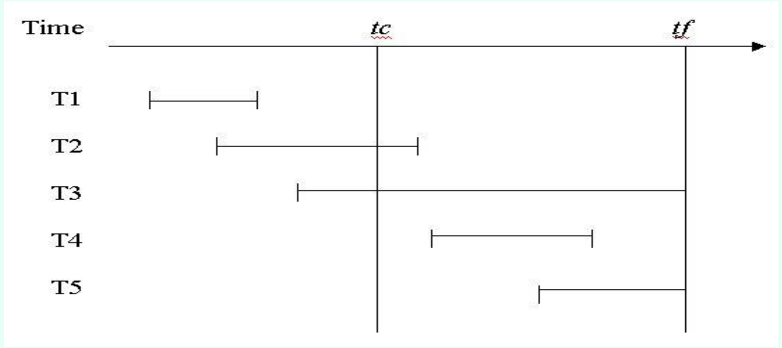
Step 1: Using the log compile REDO list

REDO list						
1	2	3	4	5	6	7

Step 2: REDO committed transactions starting from oldest

1	2	3	4	5	6	7

Q5. For a write through system, which transaction/s will need to be UNDONE (ROLLBACK)?



tc =time of checkpoint
tf = time of failure

A. T1
B. T2
C. T3
D. T4
E. T5
F. None of them

Non Volatile Storage 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**
- Known as ***Forward Recovery***