

Topic 7

Insert, Update, Delete (DML) and Transaction Management

Workshop 2025 S1



DML: ADDING TUPLES/ROWS TO A TABLE

SQL Commands COMMIT and ROLLBACK

From the topic 2 Applied activities:

```
INSERT INTO student (  
    studid,  
    studfname,  
    studlname,  
    studdob,  
    studaddress,  
    studphone,  
    studemail  
) VALUES (  
    13390148,  
    'Brier',  
    'Kilgour',  
    TO_DATE('21-Feb-1995', 'dd-Mon-yyyy'),  
    '79776 Dryden Plaza, Moorabbin',  
    '6981280319',  
    'Brier.Kilgour@student.monash.edu'  
);  
  
COMMIT;
```

COMMIT makes the changes to the database permanent

ROLLBACK will undo/remove the changes

COMMIT/ROLLBACK only applicable to INSERT/UPDATE and DELETE

INSERT

- Adding data to a table in a database.
- SYNTAX:

```
INSERT INTO table [(column [, column...])]
VALUES (value [, value...]);
```

```
INSERT INTO drone (drone_id, drone_pur_date, drone_pur_price,
                  drone_flight_time, drone_cost_hr, dt_code)
VALUES (200, '10/Dec/2022', 1200.10, 0, 120, 'DIN2');
```

- note missing attributes from attribute list will be assigned NULL

or (all attributes required):

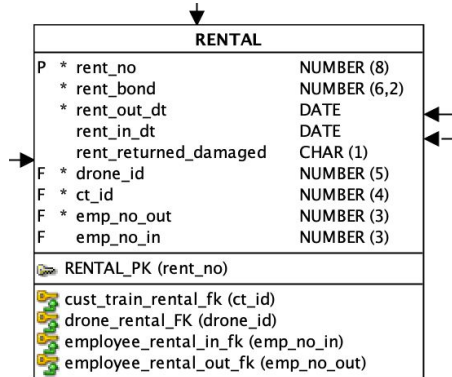
```
INSERT INTO drone VALUES (200, '10/Dec/2022', 1200.10, 0, 120, null, 'DIN2');
```

DRONE		
P *	drone_id	NUMBER (5)
*	drone_pur_date	DATE
*	drone_pur_price	NUMBER (7,2)
*	drone_flight_time	NUMBER (6,1)
*	drone_cost_hr	NUMBER (6,2)
	drone_decom_date	DATE
F *	dt_code	CHAR (4)
DRONE_PK (drone_id)		
drone_type_drone_fk (dt_code)		

Inserting DATES into a table

- '10/Dec/2022'
 - this is a character string, not a date - we understand by agreement that it represents a date but it is NOT a date datatype
 - what date is this: 10/12/2022 - 10th Dec 2022 or the 12th October 2022
 - it depends on your country
- Oracle converts strings to dates using its preset National Language Support (NLS) rules, however ***we must not depend on this***
- **TO_DATE** is a special Oracle function which converts a string to a date based on a *user supplied pattern*
 - convert a date
 - to_date('10 Dec 2022','dd Mon yyyy')
 - convert a date and time
 - to_date('10/12/2022 17:00','dd/mm/yyyy hh24:mi')
 - convert a time
 - to_date('17:00','hh24:mi')

Inserting DATES into a table continued



The diagram shows the structure of the RENTAL table. It lists columns with their data types and constraints. Arrows point to specific columns: an arrow points to the table name 'RENTAL', an arrow points to the 'rent_no' column, and two arrows point to the 'rent_out_dt' and 'rent_in_dt' columns.

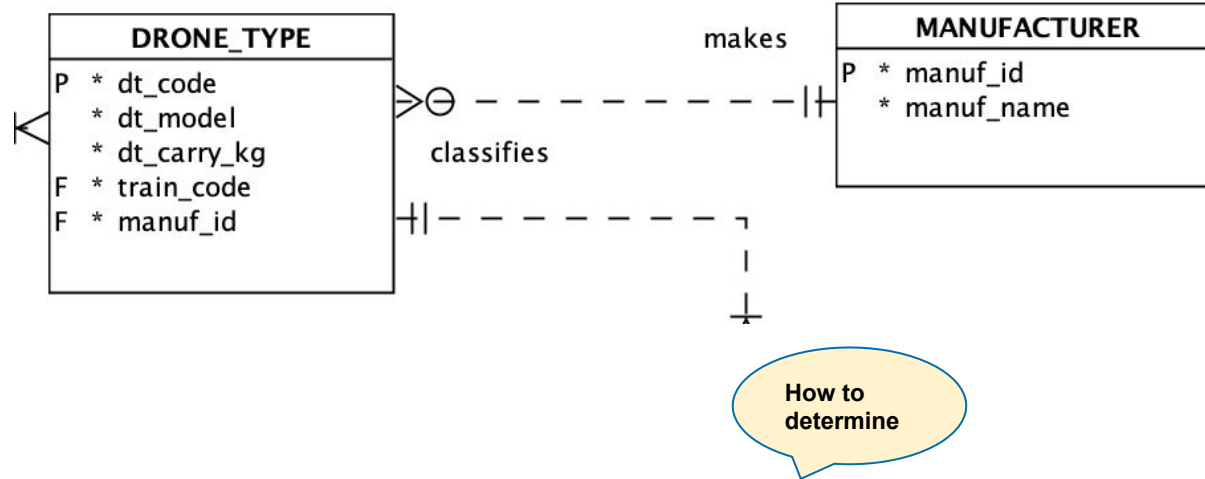
RENTAL		
P *	rent_no	NUMBER (8)
*	rent_bond	NUMBER (6,2)
*	rent_out_dt	DATE
	rent_in_dt	DATE
	rent_returned_damaged	CHAR (1)
F *	drone_id	NUMBER (5)
F *	ct_id	NUMBER (4)
F *	emp_no_out	NUMBER (3)
F	emp_no_in	NUMBER (3)
RENTAL_PK (rent_no)		
cust_train_rental_fk (ct_id)		
drone_rental_fk (drone_id)		
employee_rental_in_fk (emp_no_in)		
employee_rental_out_fk (emp_no_out)		

- Insert a rental into the RENTAL table:

```
INSERT INTO rental VALUES (30, 180, TO_DATE('23/AUG/2022  
13:00', 'DD/MON/YYYY HH24:MI'), NULL, NULL, 118, 18, 1, NULL);
```

```
INSERT INTO rental (rent_no, rent_bond, rent_out_dt, drone_id, ct_id,  
emp_no_out) VALUES (30, 180, TO_DATE('23/AUG/2022 13:00', 'DD/MON/YYYY  
HH24:MI'), 118, 18, 1);
```

How to determine new PK values?



INSERT INTO manufacturer VALUES (12, 'DJI');

INSERT INTO drone_type VALUES('DJIT', 'DJI Trello', 5, 'DJIHY', 12);

Using a SEQUENCE

- Oracle supports auto-increment of a numeric PRIMARY KEY.
 - SEQUENCE.
- Steps to use:
 - Create sequence

```
CREATE SEQUENCE manuf_seq
  START WITH 100 INCREMENT BY 1;
```
 - Access the sequence using two built-in variables (pseudocolumns):
 - NEXTVAL and CURRVAL
 - INSERT INTO manufacturer
VALUES(**manuf_seq.nextval**, 'Monash Drones');
 - INSERT INTO drone_type
VALUES('DJIT', 'DJI Trello', 5, 'DJIHY', **manuf_seq.currval**);
 - Note sequence value CANNOT be relied on after a COMMIT/ROLLBACK, they may work but cannot be relied on - thus DO NOT USE after COMMIT/ROLLBACK

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');
```

Sub (or nested) SELECT

```
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');
```

- This DML command makes use of a SUB SELECT (a complete select statement contained inside an SQL command, enclosed in brackets)
 - (SELECT dt_code FROM drone_type WHERE dt_model = 'DJI Inspire 2')
- Here this select is run first and returns the value of 'DIN2' which is substituted into the main SQL command, the statement thus becomes:

```
UPDATE drone
SET drone_cost_hr = drone_cost_hr * 1.1
WHERE dt_code = 'DIN2'
      AND drone_pur_date > to_date('31-Mar-2021','dd-Mon-yyyy');
```

How do we find a string in the database?

- SQL search is case sensitive

```
1 SELECT
2   *
3 FROM
4   drone.manufacturer
5 WHERE
6   manu_name = 'PARROT';
```

OUTPUT TERMINAL PORTS QUERY RESULT SQL HISTORY

All rows fetched: 0 in 0.054 seconds

	MANUF_ID	MANUF_NAME
--	----------	------------

MANUF_ID	MANUF_NAME
10	DJI Da-Jiang Innovations
20	Parrot
30	SwellPro

```
1 SELECT
2   *
3 FROM
4   drone.manufacturer
5 WHERE
6   manu_name = 'Parrot';
```

OUTPUT TERMINAL PORTS QUERY RESULT SQL HISTORY

All rows fetched: 1 in 0.046 seconds

	MANUF_ID	MANUF_NAME
1	20	Parrot

Since we **cannot "know"** the case of our data, SQL has two functions UPPER and LOWER used to modify case:

```
1 SELECT
2   *
3 FROM
4   drone.manufacturer
5 WHERE
6   LOWER(manu_name) = 'parrot';
```

OUTPUT TERMINAL PORTS QUERY RESULT SQL HISTORY

All rows fetched: 1 in 0.044 seconds

	MANUF_ID	MANUF_NAME
1	20	Parrot

```
1 SELECT
2   *
3 FROM
4   drone.manufacturer
5 WHERE
6   UPPER(manu_name) = 'PARROT';
```

OUTPUT TERMINAL PORTS QUERY RESULT SQL HISTORY

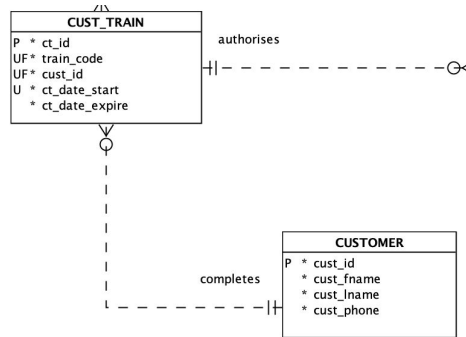
All rows fetched: 1 in 0.038 seconds

	MANUF_ID	MANUF_NAME
1	20	Parrot

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);
```

Note the use of a sub select here

TRANSACTIONS

Transactions

- Consider the following situation.

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

- *RENTAL: the return date is recorded, the employee who checked the drone in is recorded, the return status is recorded, and*
- *DRONE: the drone flight time is updated to reflect the time the customer has flown*

Assume that rental number 23 was returned undamaged on the 27th Sep 2022 at 10 AM, checked in by employee 10, after having been flown for 120.3 minutes as part of this rental. The SQL involved is:

T
R
A
N
S
A
C
T
I
O
N

```
UPDATE rental
SET rent_in_dt = to_date('27-Sep-2022 10:00','dd-Mon-yyyy hh24:mi'),
    rent_returned_damaged = 'N',
    emp_no_in = 10
WHERE rent_no = 23;

UPDATE DRONE
SET drone_flight_time = drone_flight_time + 120.3
WHERE drone_id = (SELECT drone_id FROM rental
                  WHERE rent_no = 23);

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_dt = to_date('27-Sep-2022','dd-Mon-yyyy'),  
    rent_returned_damaged = 'N',  
    emp_no_in = 10,  
WHERE rent_no = 23;
```

UPDATE DRONE

```
SET drone_flight_time = drone_flight_time + 120.3  
WHERE drone_id = (SELECT drone_id FROM rental WHERE rent_no = 23);
```

COMMIT;

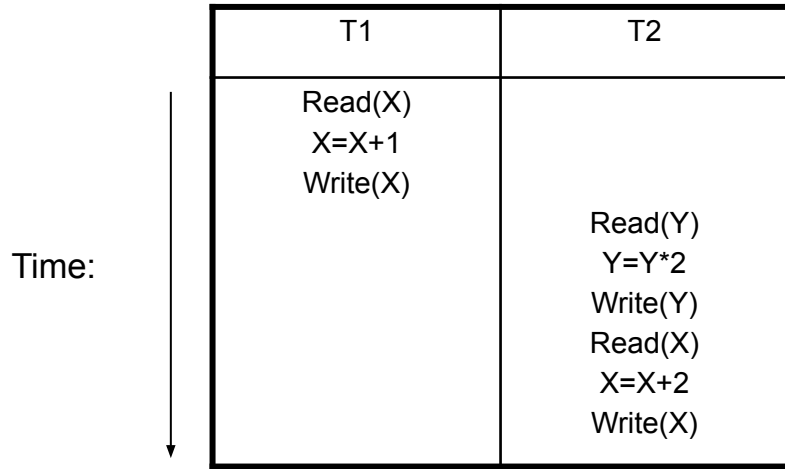
If the power for the database is lost after the first update to the RENTAL table, this (multiple answers possible):

- 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

Transaction Management

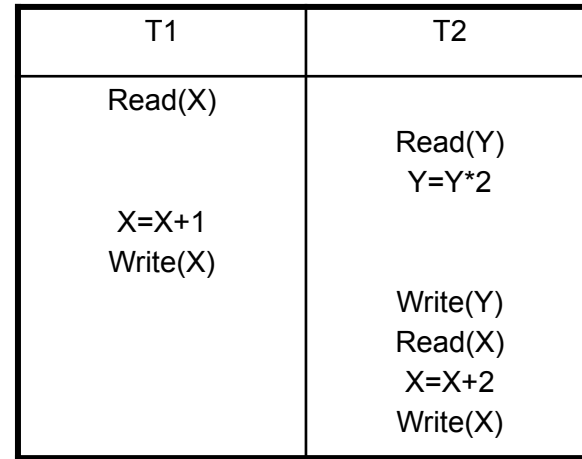
- Follows the ACID properties.
- Transaction boundaries
 - Start
 - first SQL statement is executed (eg. Oracle)
 - Some systems have a BEGIN WORK type command
 - End
 - COMMIT or ROLLBACK
- Concurrency Management
- Restart and Recovery.

Concurrency



Serial - limited throughput

T1 R(X), T1 W(X), T2 R(Y), T2 W(Y), T2 R(X), T2 W(X)
 OR alternative
 T2 R(Y), T2 W(Y), T2 R(X), T2 W(X), T1 R(X), T1 W(X),



Interleaved (non Serial)

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

Serializable outcome => equal to the outcome if the transactions executed serially

Q2. 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 (update which is overwritten - see text)
- 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.

Q3. 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

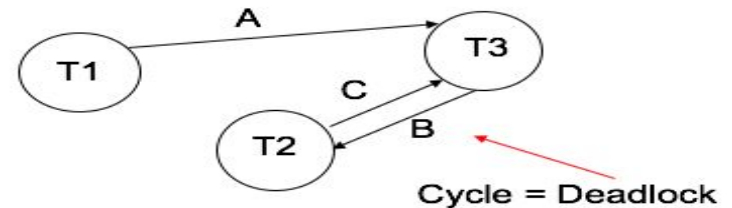
- Shared lock. Multiple processes can simultaneously hold shared locks, to enable them to read without updating.
 - if a transaction T_i has obtained a shared lock (denoted by **S**) on data item **Q**, then T_i can **read** this item but not **write** to this item
- Exclusive lock. A process that needs 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**

Lock Example – what happens?

Time	Tx	Access	A	B	C
0	(T1)	READ A			
1	(T2)	READ B			
2	(T3)	READ A			
3	(T1)	UPDATE A			
4	(T3)	READ C			
5	(T2)	READ C			
6	(T2)	UPDATE B			
7	(T2)	READ A			
8	(T2)	UPDATE C			
9	(T3)	READ B			

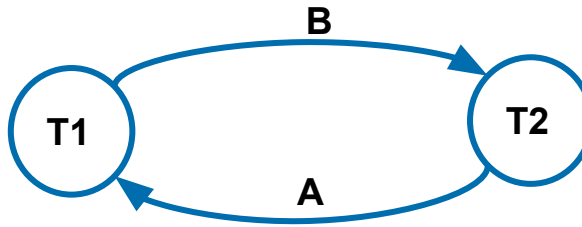
Wait-For-Graph (WFG)

TIME	TX	ACCESS	A	B	C
0	(T1)	READ A	S(T1)		
1	(T2)	READ B		S(T2)	
2	(T3)	READ A	S(T3)		
3	(T1)	UPDATE A	T1 Wait for T3		
4	(T3)	READ C			S(T3)
5	(T2)	READ C			S(T2)
6	(T2)	UPDATE B		X(T2)	
7	(T2)	READ A	S(T2)		
8	(T2)	UPDATE C			T2 Wait for T3
9	(T3)	READ B		T3 Wait for T2	



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 Recovery

- Recovery involves processes to return the contents of database to its last consistent state:
 - Soft crashes
 - loss of volatile storage, but no damage to disks.
 - Hard crashes
 - anything that makes the disk permanently unreadable.
- Requires a record of actions which have been taken
 - 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

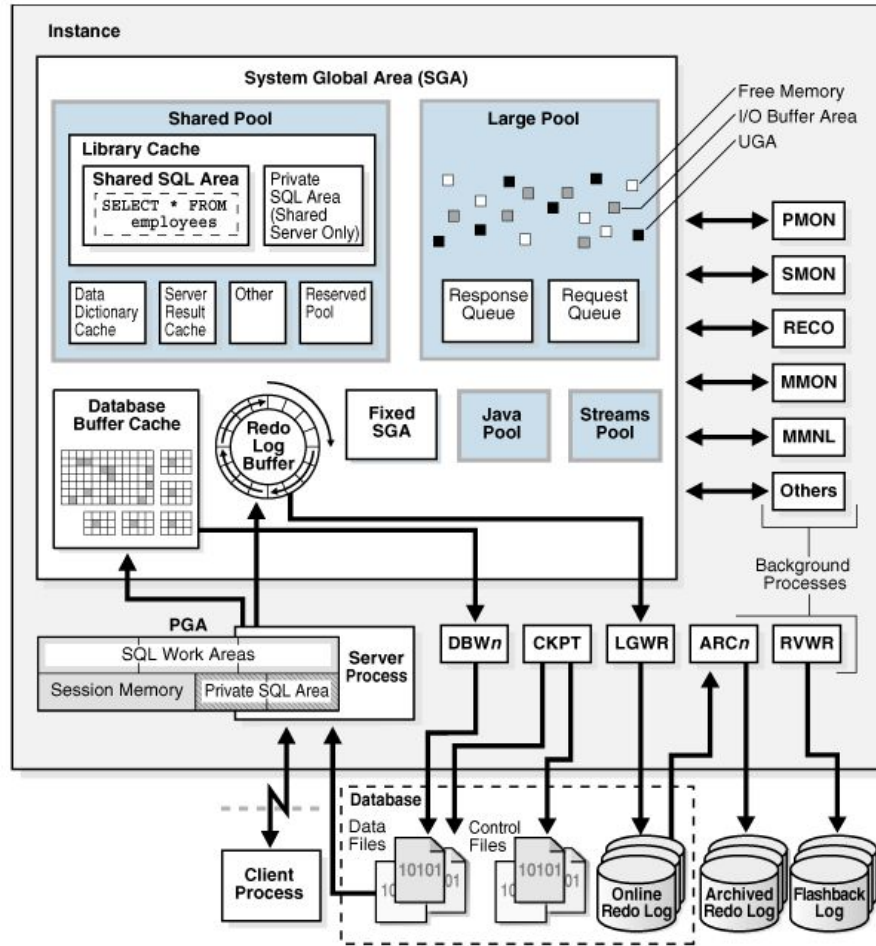
(Note: The transaction number is automatically assigned by the DBMS.)

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 – *general information only*

INSTANCE (memory resident)



DATABASE
(on disk)

Soft Crash Recovery - 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

Recovery Procedure for Write Through

Once the cause of the crash has been rectified, and the database is being restarted:

- **STEP 1: Using the log, compile REDO and UNDO lists**
 - The last checkpoint before the crash in the log file is identified. It is then read forward from, and two lists are constructed:
 - a REDO list containing the transaction-ids of transactions that were committed, and
 - a UNDO list containing the transaction-ids of transactions that never committed
- **STEP 2: UNDO incomplete or rolled back transactions** *starting from newest* (ROLLBACK using before images)
- **STEP 3: REDO committed transactions** *starting from oldest* (ROLLFORWARD using after images)

For each step the actions can be listed in the form:



— time →

with 1 ,2, 3 etc representing the transaction ids

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
---	---	---	---	---	---	---

Soft Crash Recovery - 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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

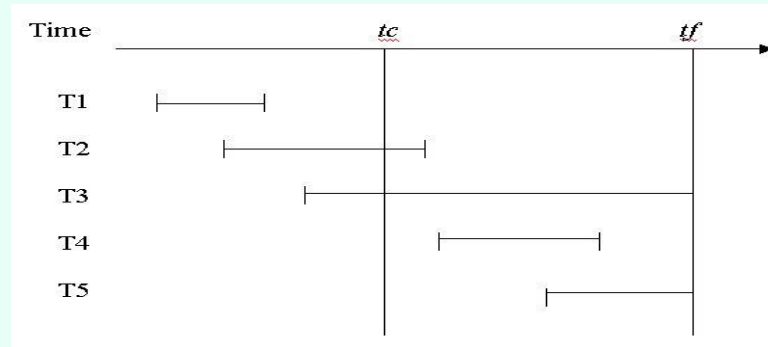
Step 1: Using the log compile REDO list

REDO list						
1	2	3	4	5	6	7

Step 2: REDO committed transactions starting from oldest

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Q4. 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

Hard Crash 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 Process

- 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***