MONASH University

MONASH
INFORMATION
TECHNOLOGY

Week 8 -
DML - Update, Delete +
Transaction Management

FIT2094 - FIT3171 Databases

Clayton Campus S2 2019.

GROUP OF EIGHT AUSTRALIA

Overview

- **Hour 1**
  - –SQL UPDATE
  - –SQL DELETE
  - –We will move on to some theory in between our SQL discussions -- introducing Transaction Management

… then COFFEE BREAK!

- **Hour 2**
  - –Transaction Management continued
    - •advanced stuff

[CLAYTON ONLY] Week 8 chat...

ATTENTION: Week 9 and Grand Final holiday might affect your tutorial NEXT week.

IF YOU HAVE A FRIDAY TUTE:

- **In MSB: more open consultation sessions if you need to catch up with a tutor for help etc.**
  a. **and you may get marked for participation in the open consultation in lieu of Week 9 Tute**
- **Other options: do self study at home, and email your tutor (or share on Google Drive) your answers to the Week 9 tute questions for participation marks.**
  a. **Note that you need to get most of the answers correct and demonstrate a decent attempt at answering them.**

NB: Following week is the MSB :-)

UPDATE / DELETE...
(the DML part of SQL)

Img src: @samuelzeller at Unsplash / icons8

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 ……
```

```
-- can also nest SELECTs! --
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';
```

[Clayton] Live demo

Remember: some of the SQL statement syntaxes in the unit are ORACLE SPECIFIC.

After a deferred exam, **update enrolment** for **Fred Blogs's FIT1002 S1/2013 to 50 P.** Assume we **don't know** his Student ID. (hint: nested 'SELECT stu_nbr' via first and last name)

Fred Blogs has **dropped out of FIT1004 S1/2013.**
Perform the necessary changes given his Student ID **11111111**.

Hint:
be specific about which enrolment 'row'

STU_NBR	STU_LNAME	STU_FNAME	STU_DOB
11111111	Blogs	Fred	01-JAN-90
11111112	Nice	Nick	10-OCT-94
11111113	Wheat	Wendy	05-MAY-90

STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	ENROL_MARK	ENROL_GRADE
11111111	FIT1001	2012	1	78	D
11111111	FIT1002	2013	1		
11111111	FIT1004	2013	1		

UNIT_CODE	UNIT_NAME
FIT1001	Computer Systems
FIT1002	Computer Programming
FIT1004	Database

Transaction Management: Theory

Img src: @austindistel at Unsplash

Transactions

- Databases are used widely to store financial info
- 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.

Sam's account should be reduced by 100.

Q1. Which of the following SQL statements is correct for the above operation? Assume Sam's account number is '123'.

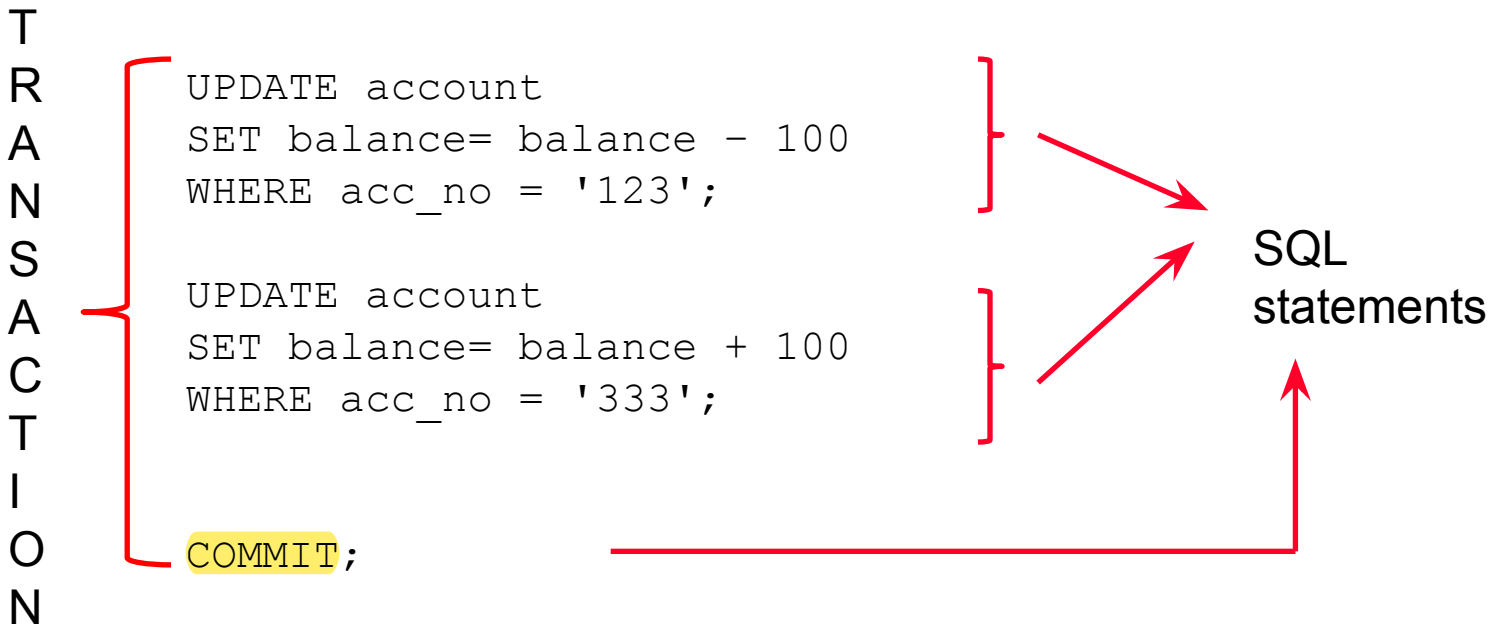
A.
```
UPDATE account
  SET balance = balance – 100;
```

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

C.
```
UPDATE account
  SET acc_no = balance + 100;
```

D.
```
UPDATE account
  SET balance = balance + 100
  WHERE acc_no = '123';
```

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:



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

[Clayton] Q&A - Why? What happens if it is divisible?

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

Q2. According to the *atomicity* property, the transaction below is complete when statement number _____ is completed.

1
```
UPDATE account
SET balance= balance – 100
WHERE acc_no = '123';
```

2
```
UPDATE account
SET balance= balance + 100
WHERE acc_no = '333';
```

3
```
COMMIT;
```

A. 1
B. 2
C. 3
D. None of the above.

[Clayton] Hint: “Atomicity: all database operations (SQL requests) of a transaction must be entirely completed or entirely aborted”

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.*

Q3. Which transaction property is violated when a transaction T2 (Jim checking the account balance) is allowed to read the balance of Jim's account while the transaction T1 (the money transfer from Sam's to Jim's) has not been completed?

A. Atomicity.
B. Isolation.
C. Consistency.
D. Durability.

[Clayton] Hint: Assume the transactions are running on a very slooooooooow computer, does 1 transaction every 2 seconds and allows limited parallel processing. Jim will find that he's short of $!

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.

[Clayton] Marc's anecdote

Related content:

Many of you who studied Operating Systems and Parallel Processing (and similar units) will notice similarities with operating system design issues/theory.

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)

[Clayton] Why interleaving? Same question as computer job scheduling...

Transaction Management: Schedules, Serializability

Img src: @austindistel at Unsplash

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

Writing schedules

- In Transaction Management, only the following operations are considered: Read, Write, Commit, and Abort.
- Therefore the Serial Schedule below can be written as:

  S1: r0(X); w0(X); c0; r1(Y); w1(Y); r1(X); w1(X); c1;

- Note that **r** means read, **w** means write, and **c** means commit. Other operations are omitted. 0 and 1 indicate the transaction number.

Time:

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

Writing schedules

If transactions cannot be interleaved, there are two possible correct schedules: one that has all operations in T0 before all operations in T1, and a second schedule that has all operations in T1 before all operations in T0 (these are known as ***serial schedules***).

Serial Schedule 1: r0(X); w0(X); c0; r1(Y); w1(Y); r1(X); w1(X); c1;

Serial Schedule 2: r1(Y); w1(Y); r1(X); w1(X); c1; r0(X); w0(X); c0;

If transactions can be interleaved, there may be many possible interleavings (***nonserial schedules***)

Interleaved Schedule 1:

r0(X); r1(Y); w0(X); w1(Y); r1(X); c0; w1(X); c1;

Interleaved Schedule 2:

r0(X); r1(Y); w1(Y); r1(X); w1(X); c1; w0(X); c0;

[Clayton] Audience Q&A

Let's learn to understand these schedules

Serial Schedule 1: r0(X); w0(X); c0; r1(Y); w1(Y); r1(X); w1(X); c1;
(NB: Colours for ease of recognising columns only…)

T0	T1
Read(X) ??? ???	 ??? ??? ??? ??? Commit.

[Clayton] Audience Q&A

Let's learn to understand these schedules

Interleaved Schedule 2: r0(X); r1(Y); w1(Y); r1(X); w1(X); c1; w0(X); c0;

T0	T1
Read(X)	
	???
	???
	???
	???
	???
???	
???	

Serializability (Note: US spelling)

"A given interleaved execution of some set of transactions is said to be ***serializable*** if and only if it produces the same result as some serial execution of those same transactions".

For interleaved schedules, we need to determine whether the interleaved schedules are correct, i.e., are ***serializable***.

"Serializable schedules are equivalent to a serial schedule of the same transactions".

Serializability Theory: an important theory of concurrency control which attempts to determine which schedules are "correct" and which are not and to develop techniques that allow only correct schedules. To determine if a schedule is conflict serializable a *precedence graph* is used.

Serializability (Note: US spelling)

"A given interleaved execution of some set of transactions is said to be ***serializable*** **if and only if it produces the same result as some serial execution** of those same transactions".

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
	Write(Y)
X=X+1	
Write(X)	
	Read(X)
	X=X+2
	Write(X)

Interleaved

This is serializable

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

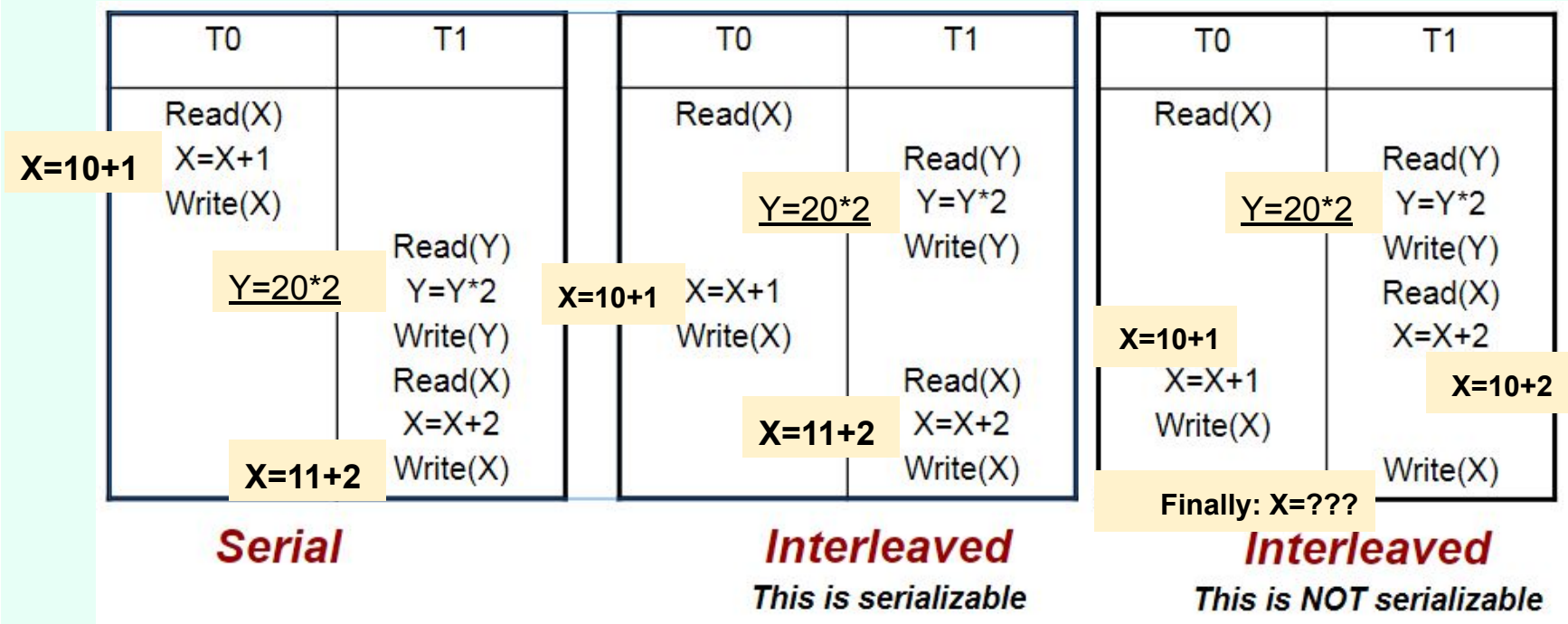
Interleaved

This is NOT serializable

[Clayton] Audience Q&A

Let's learn to understand serializability

Assume at the start: X = 10, Y = 20 (bold and underline used for clarity)



Coffee break - see you in 10 minutes.

Img src: Jazmin Quaynor @jazminantoinette at Unsplash

Transaction Management: Locks and Deadlock

Img src: @austindistel at Unsplash

Concurrency Management - Solution

- Locking mechanism
  - A mechanism to overcome the problems caused by non-serialized transactions (i.e. lost update, temporary update, and incorrect summary problems).
- 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
    ("diskpage ...section of a disk" -- Coronel & Morris)
  - 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

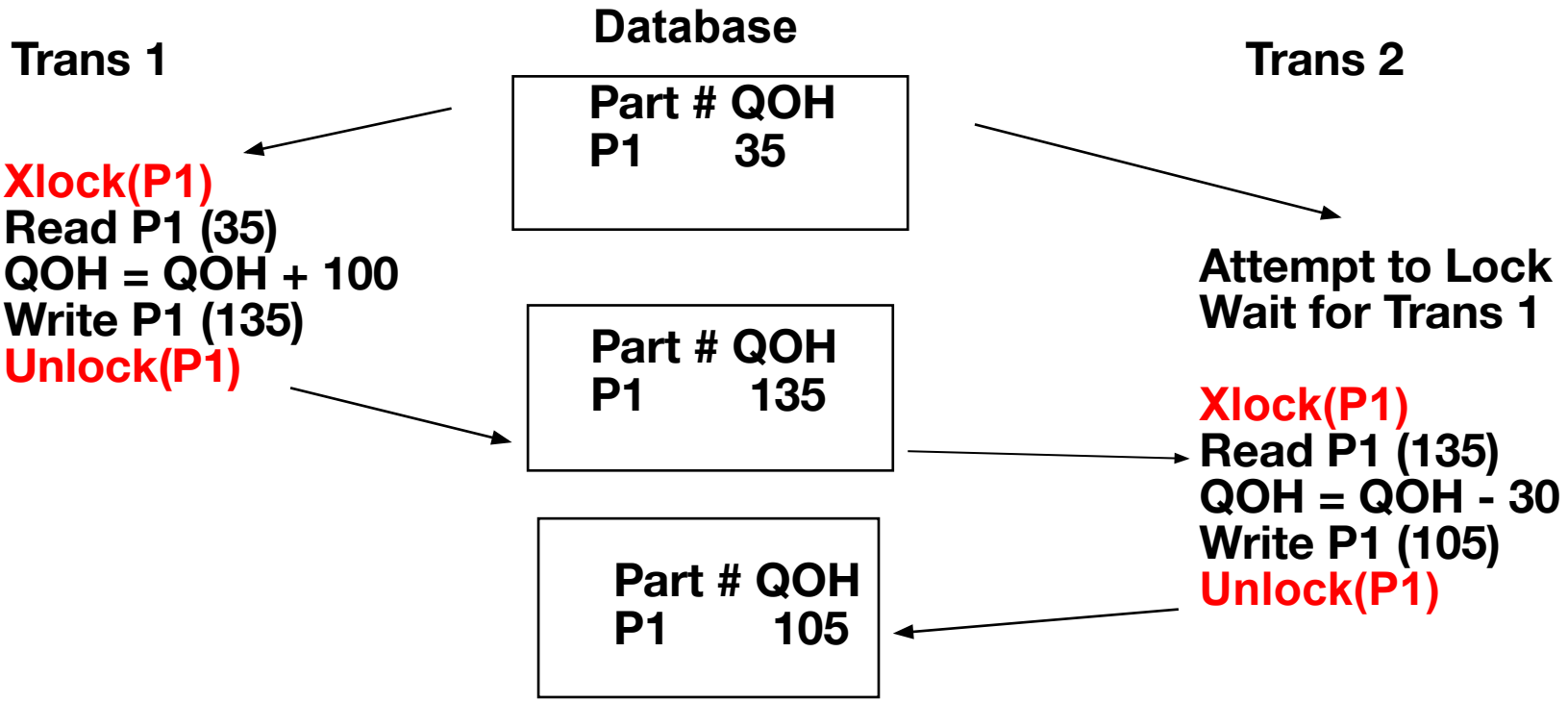
    Allows concurrent transactions to access the same row, as long as they require the use of different attributes within that row.

Lock Types

- Shared lock. Multiple processes can simultaneously hold shared locks, to enable them to read without updating.
  - if a transaction $\mathbf{T}_i$ has obtained a shared lock (denoted by **S**) on data item **Q**, then $\mathbf{T}_i$ 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 $\mathbf{T}_i$ has obtained an exclusive lock (denoted **X**) on data item **Q,** then $\mathbf{T}_i$ can both **read** and **write** to item **Q**

Exclusive Locks – Example 1

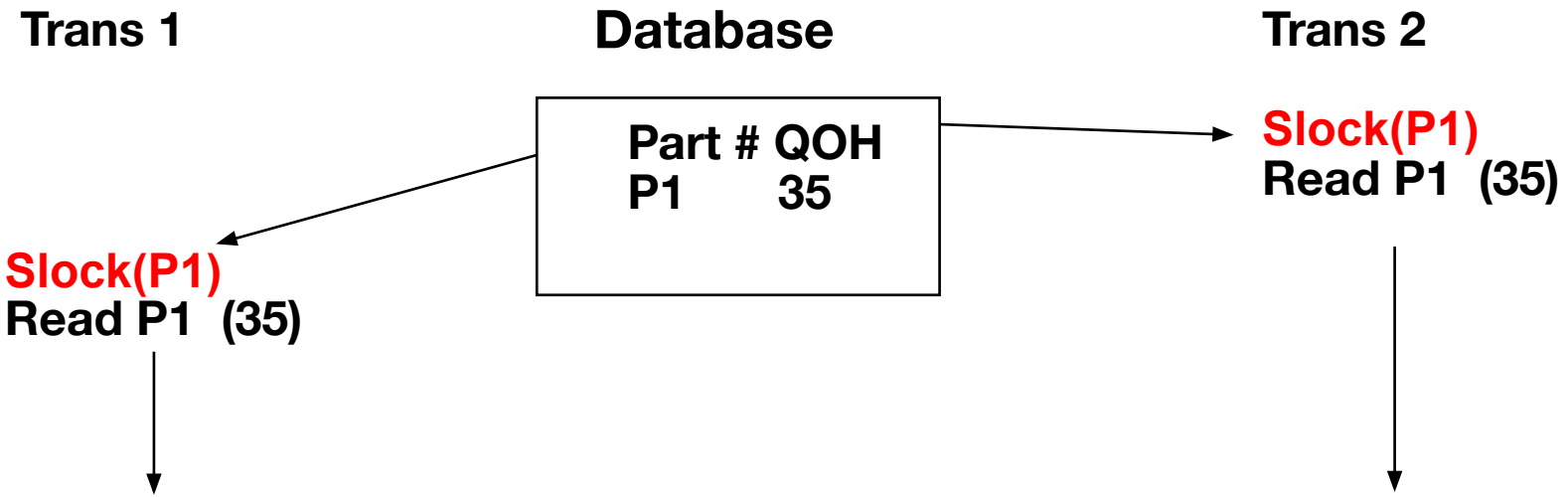
- Write-locked items
  - require an Exclusive Lock
  - a single transaction exclusively holds the lock on the item



NB: typo corrected - the correct text is **Write P1 (105)** in Transaction 2.

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 Example 3 – what happens?

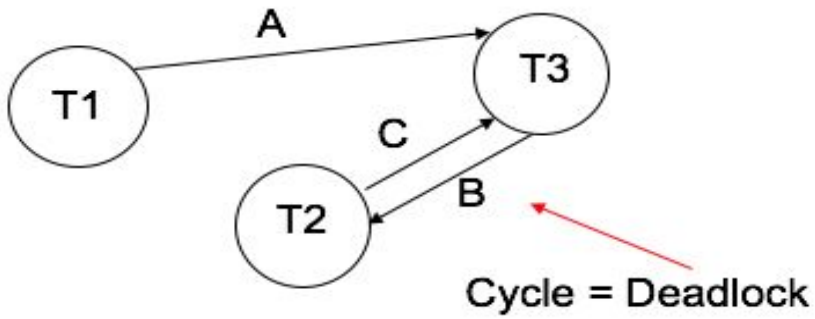
Time	Tx	Access	A	B	C
0	(T1)	READ A			
1	(T2)	READ B			
2	(T3)	READ A			
3	(T1)	UPDATE A			
4	(T3)	READ C			
5	(T2)	READ C			
6	(T2)	UPDATE B			
7	(T2)	READ A			
8	(T2)	UPDATE C			
9	(T3)	READ B			

*Build a **Wait For Graph** by showing each transaction and then drawing a directed edge (line with an arrow) from the waiting transaction to the waiting for transaction, label with the resource being waited for*

[Clayton] NB: different format from earlier schedules. Column headers are data items!

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	Wait(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			Wait(T3)
9	(T3)	READ B		Wait(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.

[Clayton] Same concept as the OS definition of Deadlock.
[Clayton] Anecdote courtesy John Hurst (OS class of 2011-12)
Img src: modified from Wikipedia

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.

Transaction Management: Restart / Recover, Transaction Log

Img src: @austindistel at Unsplash

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				

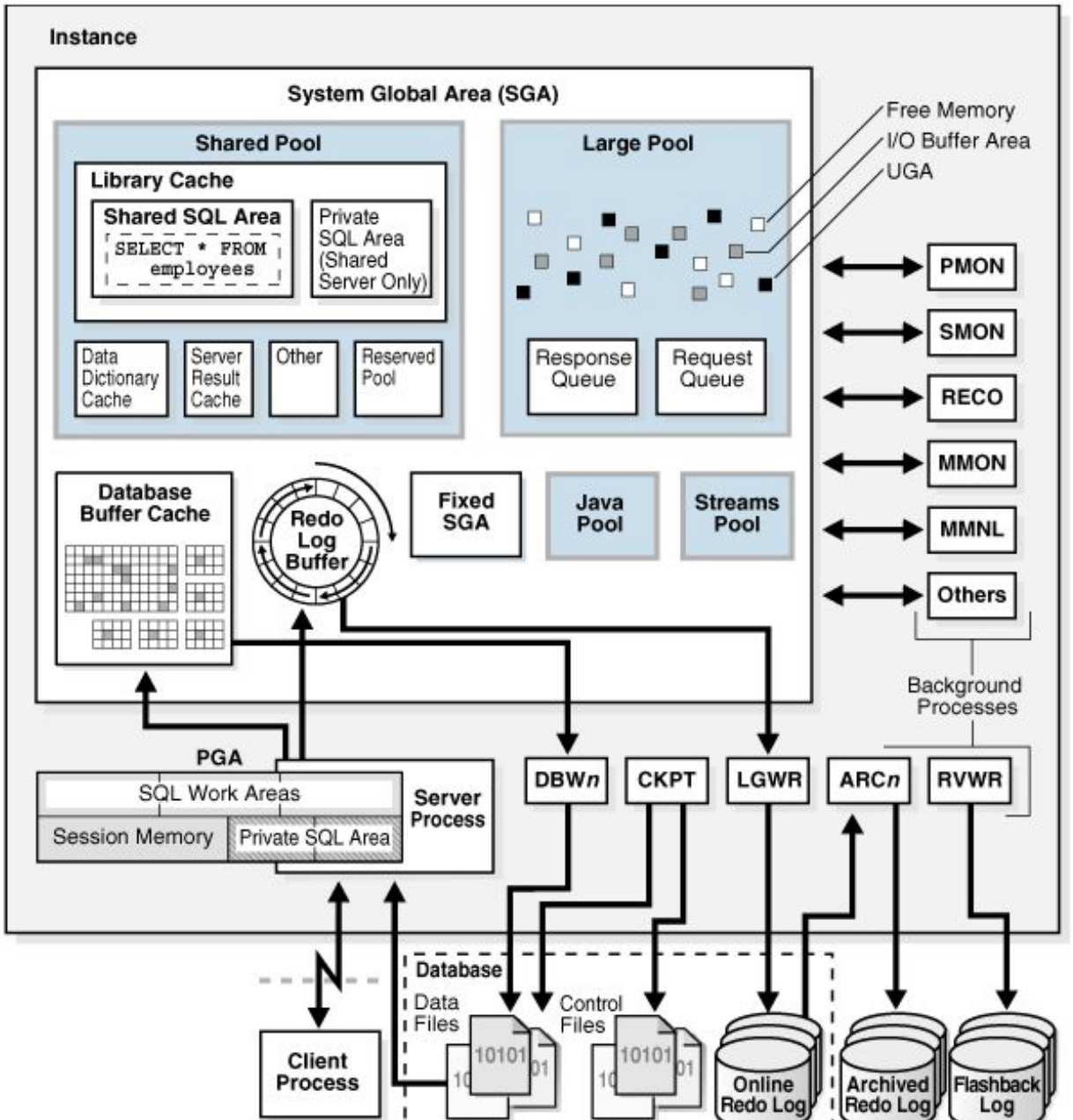
TRL_ID = Transaction log record ID
TRX_NUM = Transaction number
(Note: The transaction number is automatically assigned by the DBMS.)

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
  - Logging still happens!

- 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's 'before' values
  - Example →
  - (portions of log omitted for clarity)

ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
1558-QW1	PROD_QOH	25	23
10011	CUST_BALANCE	525.75	615.73

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.

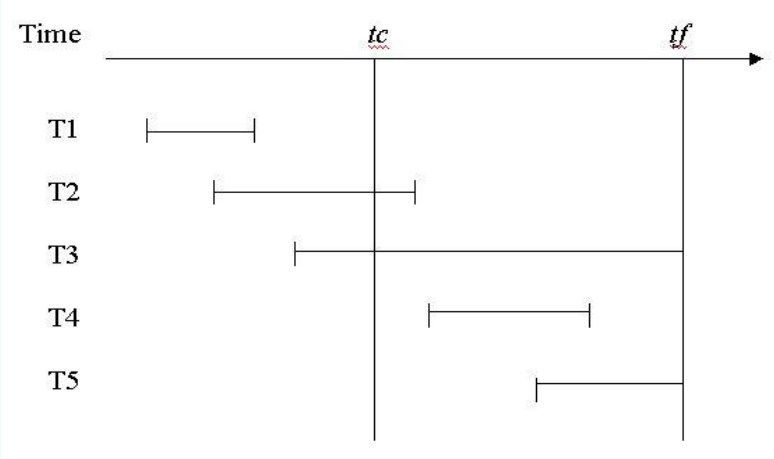
[Clayton] Audience Q&A

Let's learn to understand the write through policy - rolling back?

A very naive fictional example follows. Columns and many other details omitted for clarity.

	(comments)	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE	AFTER	(comments for playback)
2	User connects to their DB							
3	Transaction starts	START						
4	Makes new student	CREATE	STUDENT	548855				
5	and updates their details	UPDATE	STUDENT	548855	STUDENT_ID	NULL	1234	
6	**(oracle does its checkpointing)**	**CHECKPOINT**						
7		UPDATE	STUDENT	548855	STUDENT_NAME	NULL	Brendon Huynh	REDOLIST
8		UPDATE	STUDENT	548855	FEES_PAID	NULL	TRUE	REDOLIST
9		COMMIT						
10	Transaction starts	START						
11	Updates another student	UPDATE	STUDENT	456666	FEES_PAID	TRUE	FALSE	UNDOLIST
12		UPDATE	STUDENT	456666	STUDENT_POSTCODE	3000	2000	UNDOLIST
13	**(server crashes -- so we don't know how much data at this point were correctly 'sync'ed to disk... Assume it didn't write the last 2 chars of the postcode properly...)**							
14	(Brendon Huynh must be the final version. OK.)	UPDATE	STUDENT	548855	STUDENT_NAME	NULL	**Brendon Huynh**	REDOLIST
15	(TRUE must be the final version. OK.)	UPDATE	STUDENT	548855	FEES_PAID	NULL	**TRUE**	REDOLIST
16	(No COMMIT, rolling back to last known 'state' of TRUE)	UPDATE	STUDENT	456666	FEES_PAID	FALSE	**TRUE**	UNDOLIST
17	(No COMMIT, rolling back to last known 'state' of 3000)	UPDATE	STUDENT	456666	STUDENT_POSTCODE	20???	**3000**	UNDOLIST

Q4. What transaction will need to be REDONE (in the REDO list)?



tc =time of checkpoint
tf = time of failure

A. T1 and T2.
B. T2 and T4.
C. T2 and T5.
D. T1, T2 and T3.
E. None of the above.

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

- Re-build 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**

Further discussion

Many modern DBs are cloud-hosted.

How reliable are they?

Online backup/replication, scalability, privacy?