

FIT2094-FIT3171 2019 S1 -- Week 8 eBook

Credits and Copyrights:

Authors: FIT3171 2018 S2 UG Databases

FIT Database Teaching Team

Maria Indrawan, Manoj Kathpalia, Lindsay Smith, et al

Copyright © Monash University, unless otherwise stated. All Rights Reserved, except for individual components (or items) marked with their own licence restrictions

Change Log:

- Formatted and Imported from Alexandria APRIL 2019.

FIT2094-FIT3171 2019 S1 -- Week 8 eBook	1
8.0. SQL Part I – SQL Basic	2
8.0.1. Introduction to the Case Study	2
8.0.2. Recap: Oracle DATE format	4
8.0.3. Part A. Retrieving data from a single table	5
8.0.3. Part B. Retrieving data from multiple tables	6
8.1. Pre-Lecture Notes	8

8.0. SQL Part I – SQL Basic

8.0.1. Introduction to the Case Study

The following exercises will allow you to be familiar with writing basic SQL statements.

You will need to know this well, as SQL is an important skill in your future careers!

Use the **UNIVERSITY** database (set of tables) to complete the exercises. Figure 1.0 depicts the data model for the **UNIVERSITY** database. [Click here for a zoomed-in version hosted on Monash Alexandria.](#)

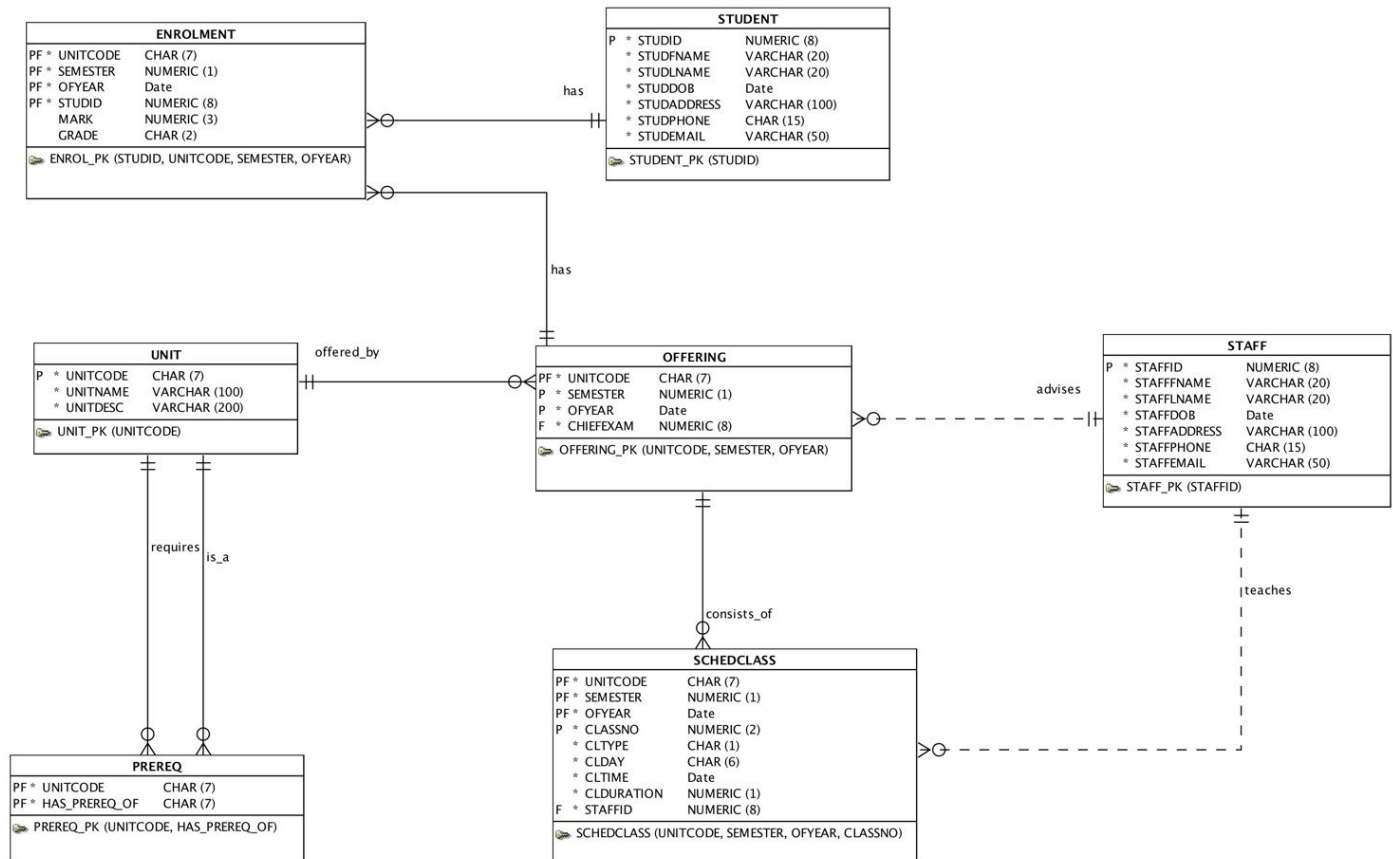


Figure 1.0. UNIVERSITY Database Model

In the Monash Oracle database, this **UNIVERSITY** set of tables has been created under the user **SHAREDSAMPLES**.

IMPORTANT: As this is a production server, and many of your classmates may be using the database at the same time, please be patient while waiting for your SQL query to run. Please respect others by not bombarding the server with many requests if your previous SELECT statement fails - wait a couple seconds before retrying. Consult your tutor if there are any technical issues.

To use these tables you need to add the prefix **SHAREDSAMPLES** to the table names that you use in an SQL statement. For example, if you want to retrieve data from **UNIT** table you need to write:

```
SELECT unitcode, unitname  
FROM SHAREDSAMPLES.unit;
```

instead of

```
-- wrong! --  
SELECT unitcode, unitname  
FROM unit;
```

8.0.2. Recap: Oracle DATE format

This week we make use of Oracle dates – to use these correctly you should note the following:

- The Oracle **date** data type contains both date and time, however, you can choose to use just a date, just a time, both or parts of a date depending on the format strings used
- **to_date**: converts from a string to a date according to a format string
- **to_char**: converts from a date to a string according to a format string

The Oracle documentation links are:

- [Format models](#)
- [to_date](#)
- [to_char](#)

8.0.3. Part A. Retrieving data from a single table

Construct SELECT statements to do the following.

1. List all students and their details.
2. List all units and their details.
3. List all students who have the surname 'Smith'.
4. List the student's details for those students who have surname starts with the letter "S". In the display, rename the columns **studfname** and **studlname** to **firstname** and **lastname**.
5. List the student's surname, firstname and address for those students who have surname starts with the letter "S" and firstname contains the letter "i".
6. List the unit code and semester of all units that are offered in the year 2014.

To complete this question you need to use the Oracle function **to_char** to convert the data type for the year component of the offering date into text. For example, **to_char(ofyear, 'yyyy')** – here we are only using the year part of the date.

7. List the unit code of all units that are offered in semester 1 of 2014.
8. Assuming that a unit code is created based on the following rules:
 - The first three letters represent faculty abbreviation, eg **FIT** for the Faculty of Information Technology.
 - The first digit of the number following the letter represents the year level.
 - Therefore, list the **unit details of all first year units in the Faculty of Information Technology**.

9. List the unit code and semester of all units that were offered in either semester 1 or summer of 2013. Note: summer semester is recorded as semester 3.
10. List the student number, mark, unit code and semester for those students who have passed any unit in semester 1 of 2013.

8.0.3. Part B. Retrieving data from multiple tables

Note: remember to use the foreign key and the primary key when joining two or more tables.

1. List the name of all students who have marks in the range of 60 to 70.
2. List all the unit codes, semester and name of the chief examiner for all the units that are offered in 2014.
3. List the name (firstname and surname), unit names, the year and semester of enrolment of all units taken so far.
4. List all the unit codes and the unit names and their year and semester offerings.
To display the date correctly in Oracle, you need to use the **to_char** function. For example, **to_char(ofyear, 'YYYY')**
5. List the unit code, semester, class type (lecture or tutorial), day and time for all units taught by Albus Dumbledore in 2013. Sort the list according to the unit code.
6. Create a study statement for Mary Smith. A study statement contains unit code, unit name, semester and year study was attempted, the mark and grade.

7. List the unit code, unit name and the unit code and unit name of the pre-requisite units of all units in the database.
 8. List the unit code and unit name of the pre-requisite units of 'Advanced Data Management' unit.
 9. Find all students (list their id, firstname and surname) who have a failed unit in the year 2013.
 10. List the student name, unit code, semester and year for those students who do not have marks recorded.
-

Copyright © Monash University, unless otherwise stated. All Rights Reserved, except for individual components (or items) marked with their own licence restrictions.

8.1. Pre-Lecture Notes

These are notes you may find useful to read through before the lecture, adapted from Lindsay Smith's lecture material. NOTE: THESE ARE NOT THE FINAL LECTURE SLIDES.

Read up the pre-lecture notes below.

IMPORTANT: AS THE LECTURE FOCUSES ON THE 'FLIPPED CLASSROOM' APPROACH - I.E. MORE INTERACTIVE DISCUSSION AND LESS ON DISCOVERING NEW MATERIAL - THE THEORY SLIDES BELOW ARE PROVIDED FOR YOUR READING CONVENIENCE BEFORE THE LECTURE.

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';
```

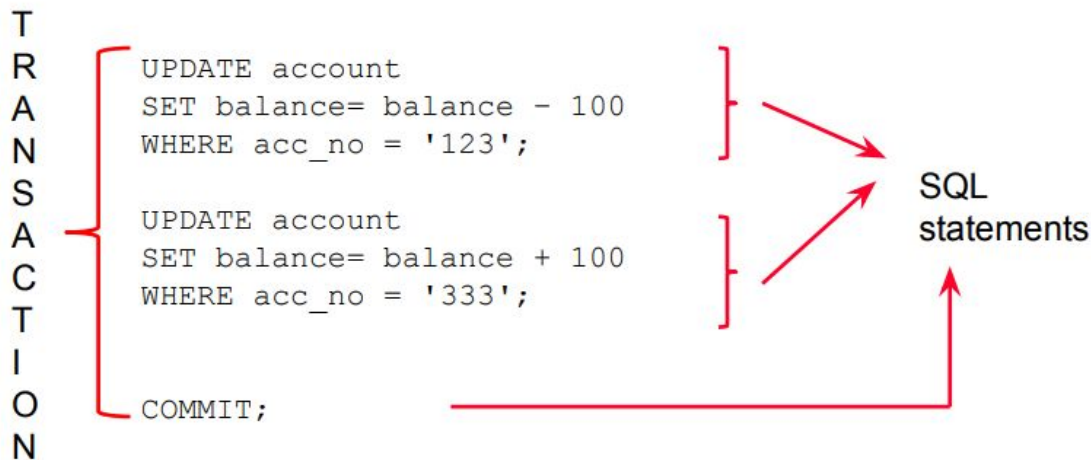
Transactions

- 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:



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

Transaction Properties

- A transaction must have the following properties:
 - **A**tomicity
 - all database operations (SQL requests) of a transaction must be entirely completed or entirely aborted
 - **C**onsistency
 - it must take the database from one consistent state to another
 - **I**solation
 - 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
 - **D**urability
 - 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*.

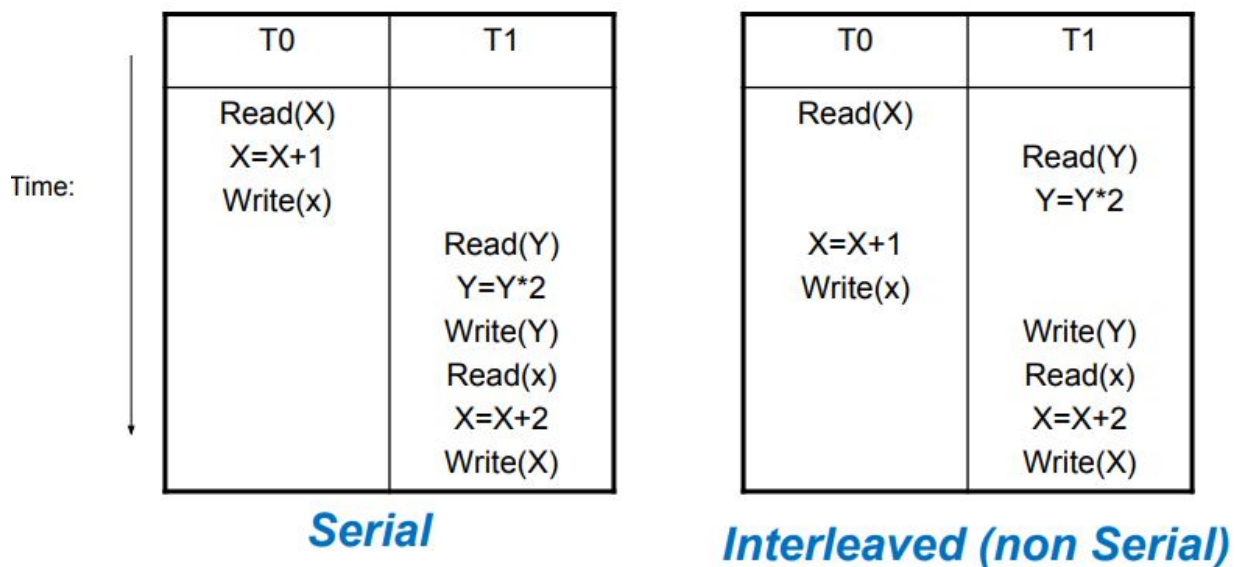
Durability - Example

- 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.

Serial and *Interleaved* transactions.



The impact of interleaved transactions

TABLE 10.2 Normal Execution of Two Transactions

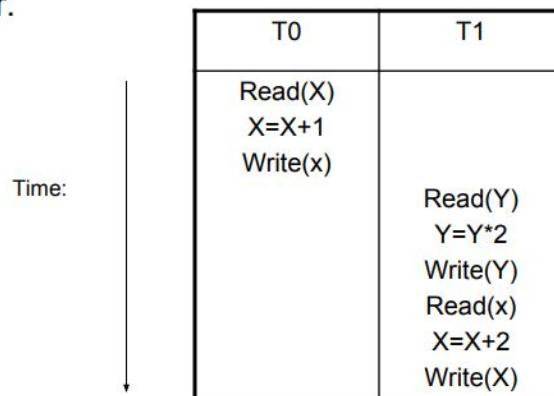
TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH = 135 - 30	
6	T2	Write PROD_QOH	105

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.



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 (**non-serial 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;

“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	T0	T1	T0	T1
Read(X) X=X+1 Write(X)	Read(Y) Y=Y*2 Write(Y) Read(X) X=X+2 Write(X)	Read(X) X=X+1 Write(X)	Read(Y) Y=Y*2 Write(Y) Read(X) X=X+2 Write(X)	Read(X) X=X+1 Write(X)	Read(Y) Y=Y*2 Write(Y) Read(X) X=X+2 Write(X)
Serial		Interleaved <i>This is serializable</i>		Interleaved <i>This is NOT serializable</i>	

“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.

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
- 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

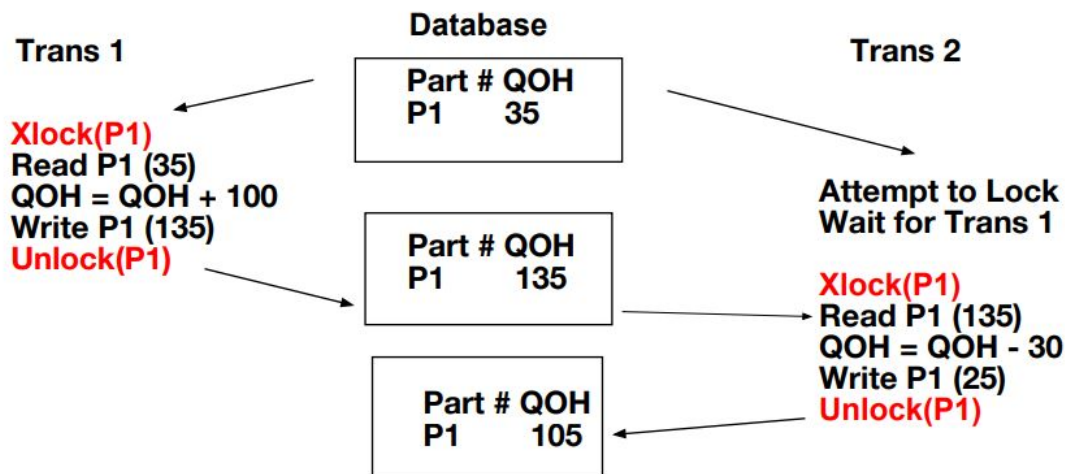
Allows concurrent transactions to access the same row, as long as they require the use of different attributes within that row.

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**

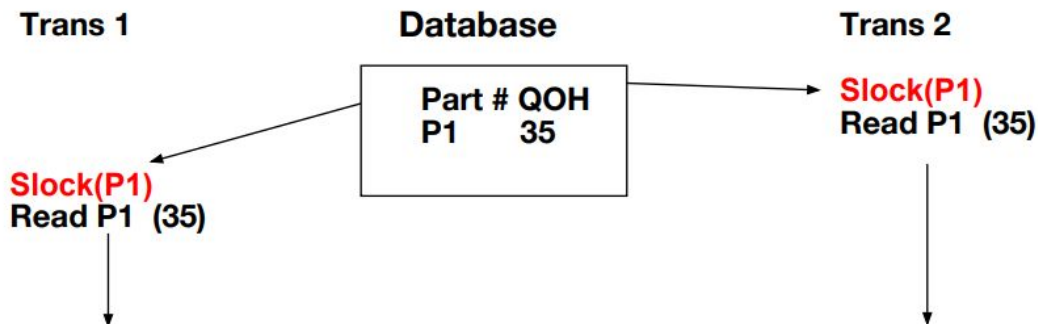
Exclusive Locks – Example 1

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



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 transaction it is waiting for, label with the resource being waited for

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 data base, 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				



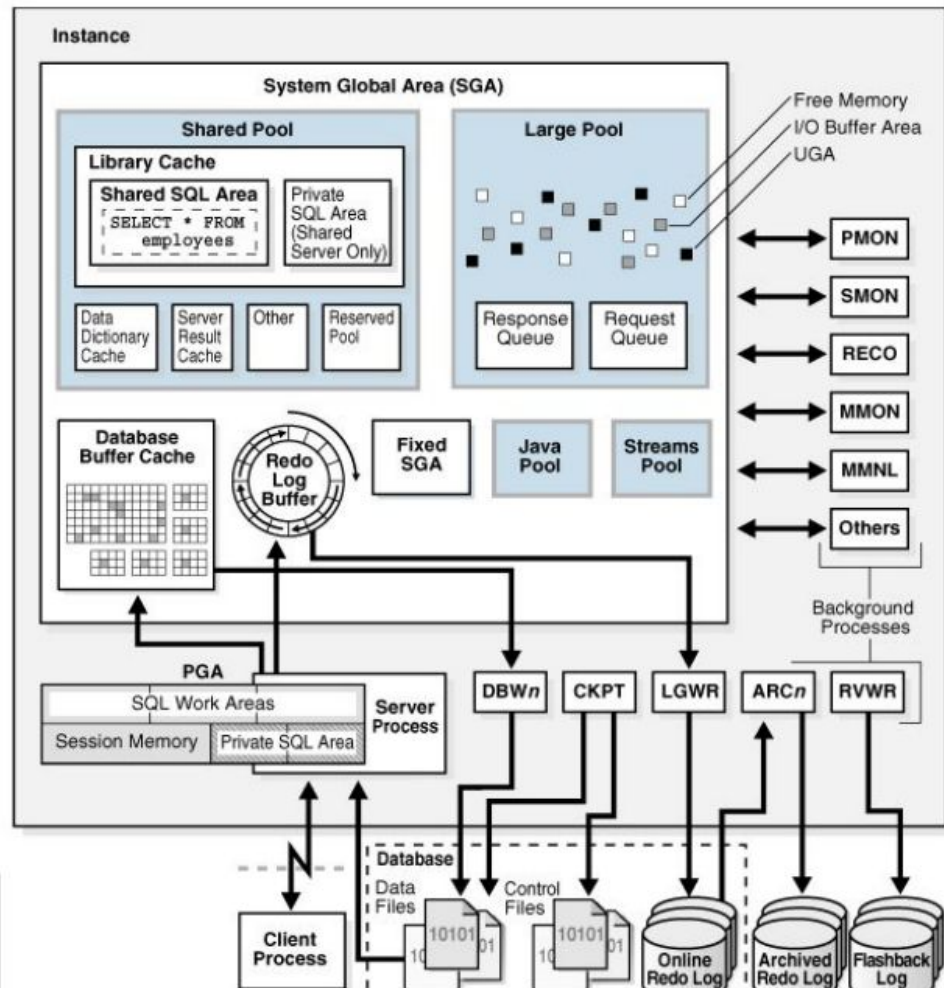
TRL_ID = Transaction log record ID
TRX_NUM = Transaction number
 (Note: The transaction number is automatically assigned by the DBMS.)

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 data base.

Backup

- A backup is a copy of the data base stored on a different device to the data base, and therefore less likely to be subjected to the same catastrophe that damages the data base. (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 data base from the most recent backup. This will restore the data base 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**

EOF.