COMP207 Database Development

Tutorial-1a

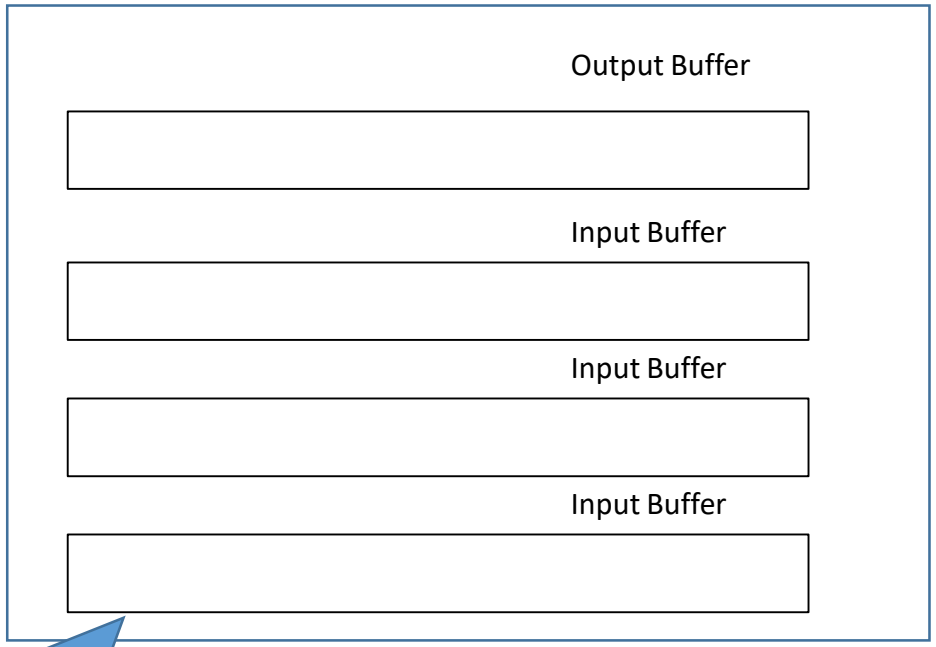
Transaction Processing

Database Organisation Concepts

- A database on disk is organised as files - each file consists of records (‘entities’, ‘tuples’) - each record consists of fields (‘attributes’)
- One tuple maps to one record in the file that holds the relation
- When a user requests a tuple (a logical record) for retrieval, the DBMS maps this onto secondary storage (disk) as a physical record in a ‘page’ (or block)
- A physical record (page) is the unit of transfer between secondary store and buffers in primary storage (and vice-versa) via operating-system file access routines

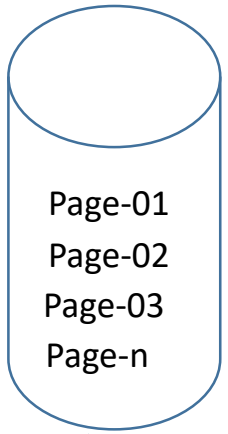
Database Organisation Concepts

Primary Storage: Main Memory (buffers)



Buffer size determines the size of a page on disk. When a record is retrieved, the page it is on is copied into buffer – carrying with it the other records on that page

Secondary Storage: Database as a file



The database is physically stored in pages (blocks). Each page can hold several records. Page size is determined by buffer size in main memory

Database Organisation Concepts

- A page consists of several logical records
  - Can be just one (large) logical record that fills the page
  - Very large logical records can span more than one physical record, i.e. a single record can be stored across several pages

Database Organisation Concepts

staffNo	lastName	position	branchNo	Page or Block #
21	White	Manager	B005	01
37	Brown	Assistant	B003	01
14	Green	Supervisor	B003	01
9	Brown	Assistant	B007	02
5	Grey	Manager	B003	02
41	Black	Assistant	B005	02

The records are held in the same order as the relation;
Here, there are two pages (‘blocks’ 01 and 02);
Each page here can hold three records;
If record-5 was required (on page-02) , a copy of the entire page-02 is transferred from disk to an input buffer in primary storage

File Organisation

- The Physical Arrangement of Data into Records and Pages ('Blocks') on Secondary Storage
- Files are organised as
  1. Heap (Unordered) Files
  2. Sequential (Ordered) Files
     - With an index it becomes an 'Indexed-Sequential File'
  3. Hash (Random or Direct) Files
- Other types of organisation are B^+-tree and Clustered Files - used under certain conditions

File Organisation
Heap File (Unordered)

- Simplest type of organisation
- Inserting records:
  - Inserted into last page of file so insertion is very efficient because there is no need to put a record in its 'correct place' if the file were ordered by an attribute (usually a key attribute – the 'key field' or 'primary key')
- Searching Records:
  - Done linearly - slow and inefficient (but is okay if done on a small file)
- Deleting Records:
  - Retrieve the page/Delete the record/Write page back to disk. Many deletions cause deterioration in performance and file has to be re-organised
- Good For:
  - bulk-loading data into the table as there is no need to find exact page where it has to go
  - Saves space if used with an index (no need to insert/delete into the correct place)

File Organisation
Sequential File (Ordered)

- Not usually used for database storage unless a primary index is added to give an ‘Indexed-Sequential File’
- The order is based on key-sequence order which is based on an ordering field. If this field is the primary key, it is the ‘ordering key’
- Can use binary search to retrieve records where the binary search works better for primary keys rather than secondary keys which may contain duplicates
  - In general, binary search is applied more frequently to data in primary storage than secondary storage

File Organisation
Sequential File (Ordered)

- Good For:
  - Exact key matching, Pattern matching, Range Matching, Part-key specification
- But:
  - The index is static (it is created when the file is created). Hence performance of an ISAM file deteriorates as the relation is updated
  - Frequent updates cause the file to lose key-access sequence so retrievals slow down
  - Overcome this problem with B^+-trees

File Organisation
Hash File (Random or Direct)

- Can use Static Hashing (hash address space is fixed at file creation) or use Dynamic Hashing (lets the file size change as the database grows and shrinks)
- Both use hashing to generate an address where record is to be stored by hashing on the value of an attribute (field). Hence the file appears to be in random order
- The base field that is hashed is the 'hash field' and if it is the primary key it is the 'hash key'
- No need to write records sequentially – the hash function calculates the address of the page where the record is to be stored. Ideally we want a hash function to spread the records evenly across the pages
- Popular to use a MOD function ('division-remainder' – takes the field value, divides it by a predetermined integer and use the remainder as the disk address) to generate an address to store the record – the address will be part of a page (which can hold several records)

File Organisation
Hash File (Random or Direct)

- Good For:
  - Searching on an exact match
- But:
  - Poor for retrieving pattern matching or range of values
  - Poor for retrieval on a field other than the hash field
    - Will need to do a linear search or add the non-hash field as a secondary index
  - Poor if updating the hash function – affected tuples need re-instating with the new hash address

Transaction Control

State transition diagram illustrating the states for transaction execution

At this stage, the buffer (main memory) holds the value of any updated variables. The disk (secondary storage) holds the original values and is not updated until the 'commit' operation is executed

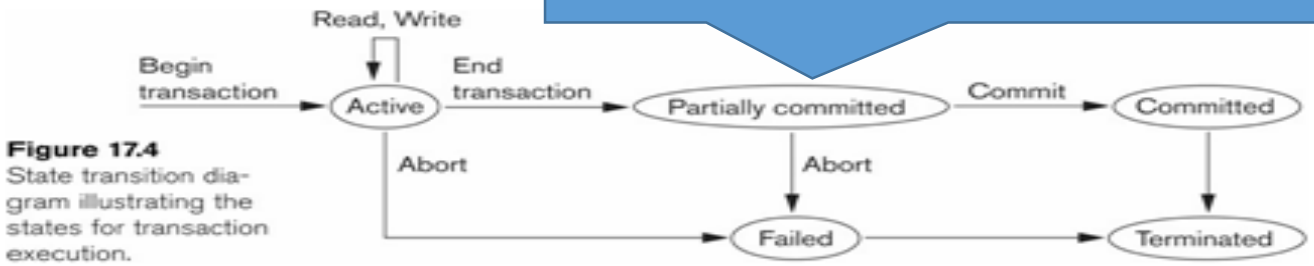
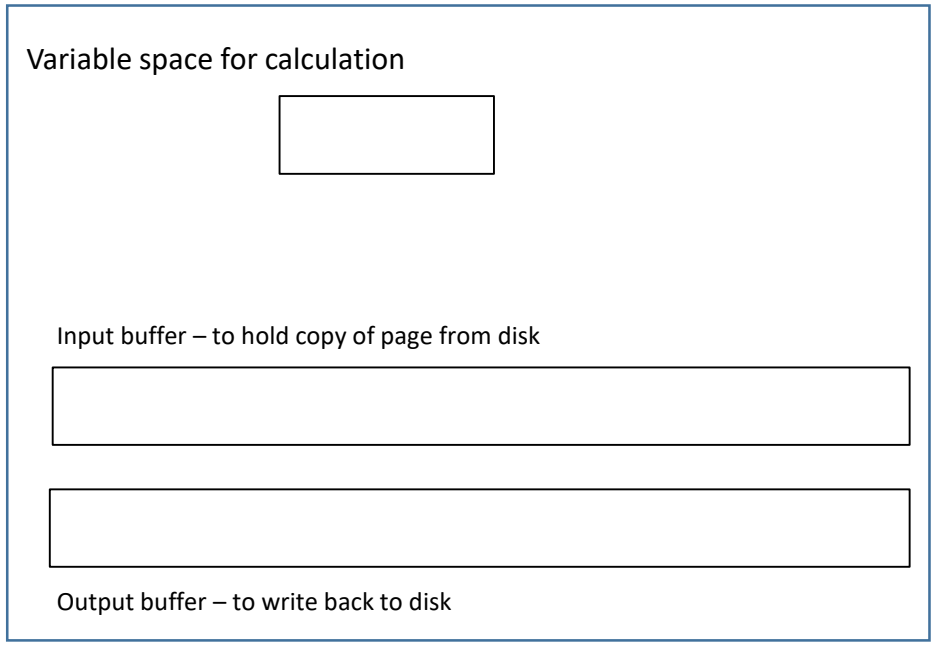


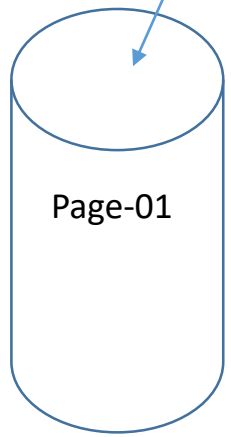
Figure 17.4
State transition diagram illustrating the states for transaction execution.

Transaction Control Example

Main Memory - holds pages from the database



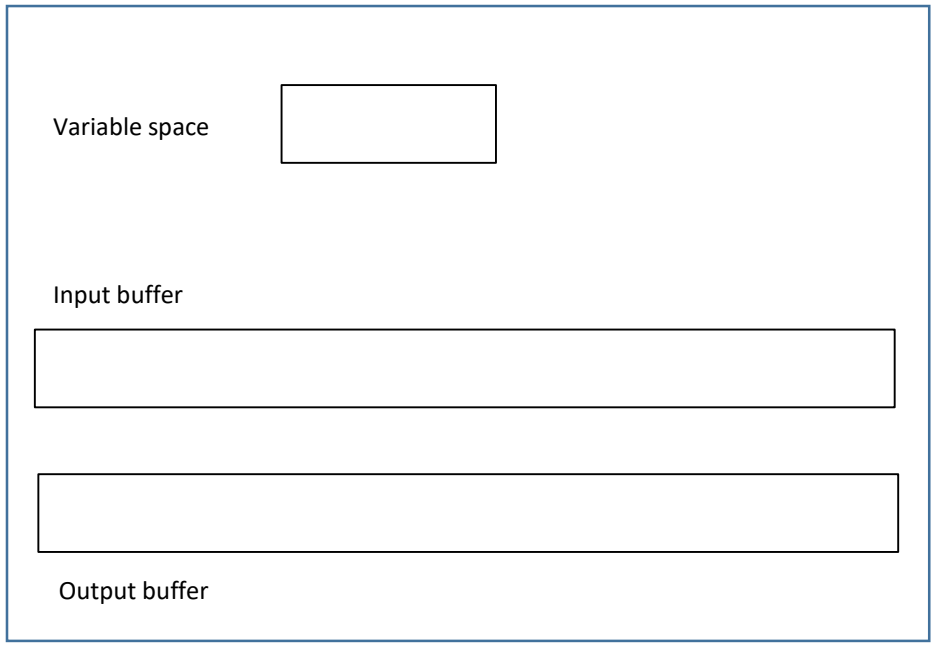
Secondary Storage (disk)
holds the database on pages



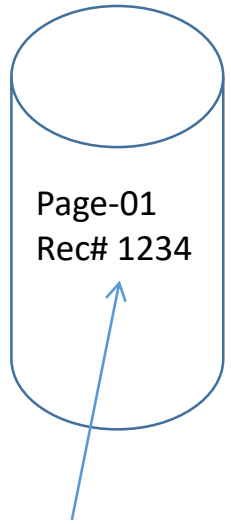
Transaction Control Example

Give staff number 1234 a 10% pay rise

UPDATE Staff SET salary = salary*1.1 WHERE staffNo = 1234;



Secondary Storage holds the STAFF database



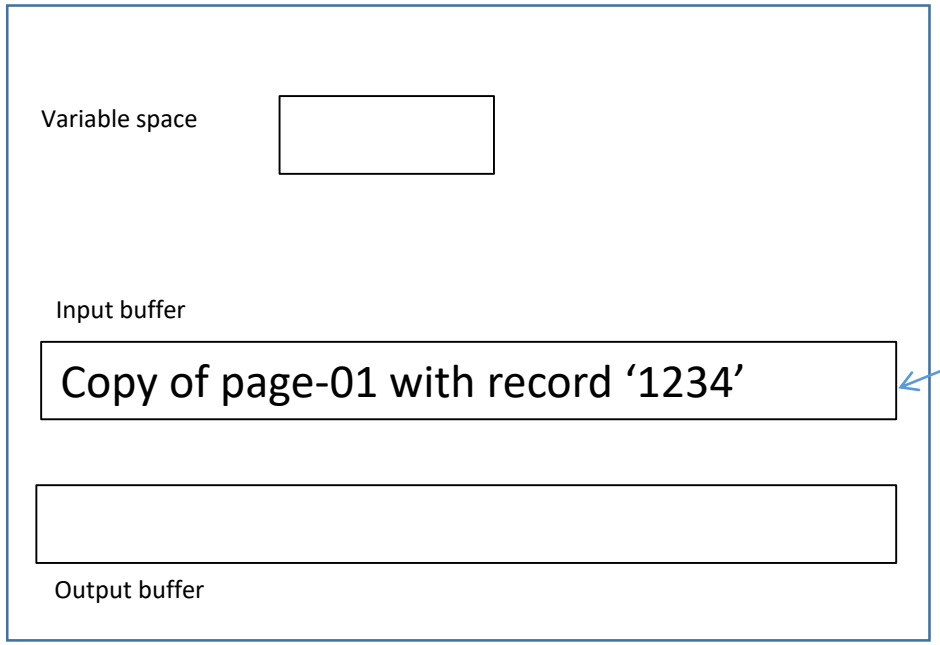
Record ‘1234’ is held on a page on disk. This page may hold other records as well

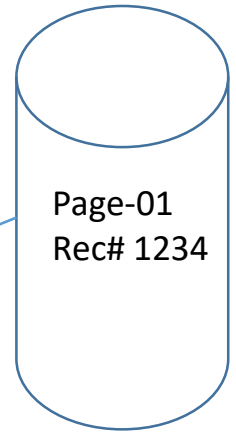
Transaction Control Example

Give staff number ‘1234’ a 10% pay rise

UPDATE Staff SET salary = salary*1.1 WHERE staffNo = 1234;

Secondary Storage holds the STAFF database





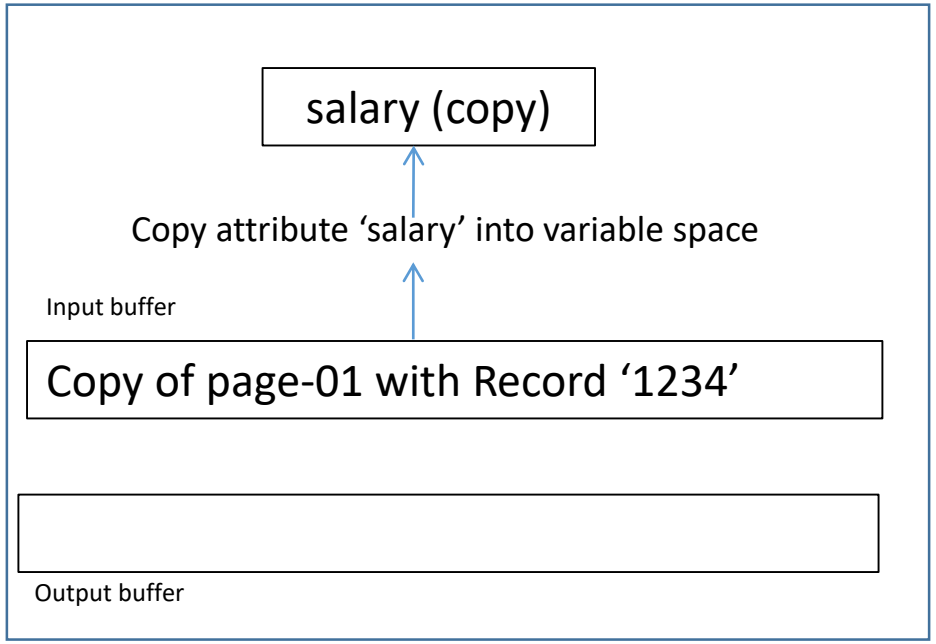
Use staffNo (primary key) to find the page address holding record ‘1234’ and copy the page to input buffer

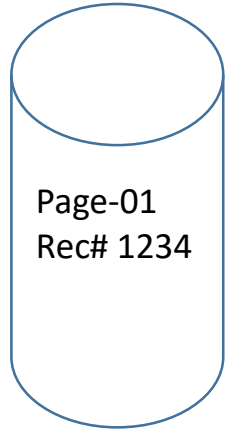
Transaction Control Example -1

Give staff number ‘1234’ a 10% pay rise

UPDATE Staff SET salary = salary*1.1 WHERE staffNo = 1234;

Secondary Storage holds the STAFF database



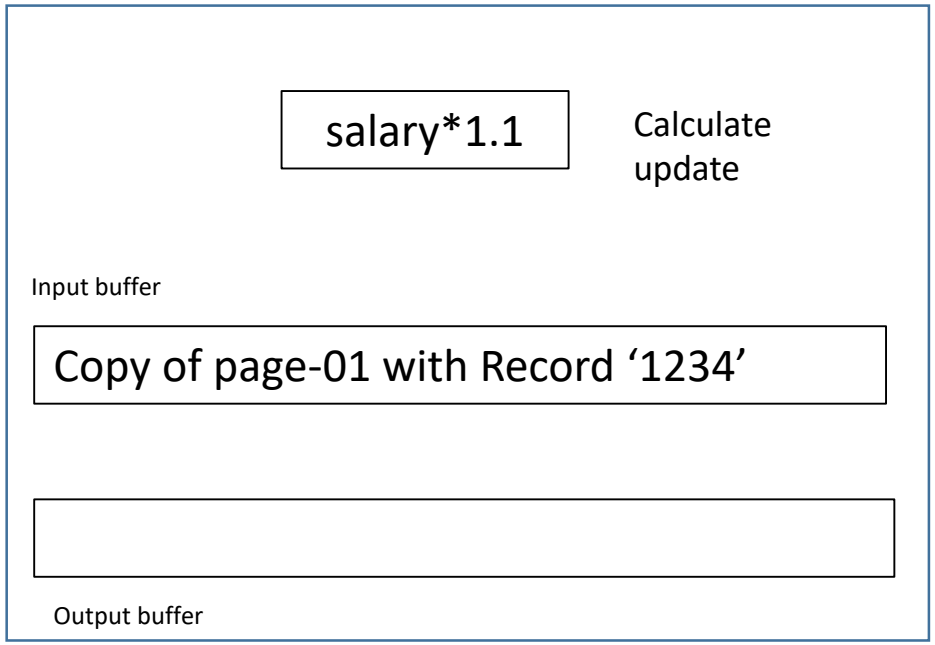


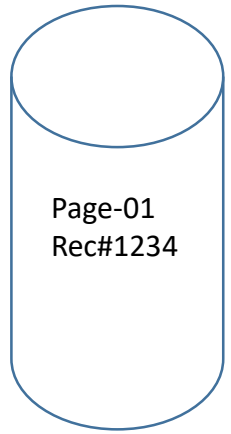
Transaction Control Example -1

Give staff number ‘1234’ a 10% pay rise

UPDATE Staff SET salary = salary*1.1 WHERE staffNo = 1234;

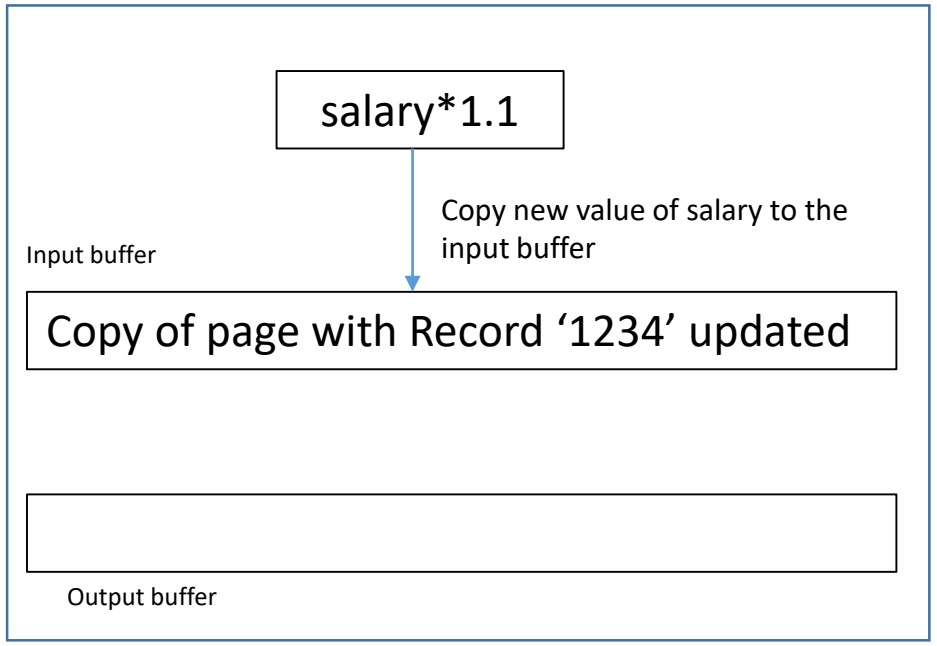
Secondary Storage holds the STAFF database



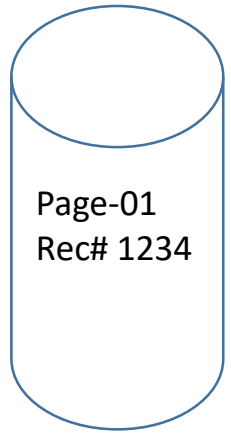


Transaction Control Example -1
Give staff number ‘1234’ a 10% pay rise

UPDATE Staff SET salary = salary*1.1 WHERE staffNo = 1234;



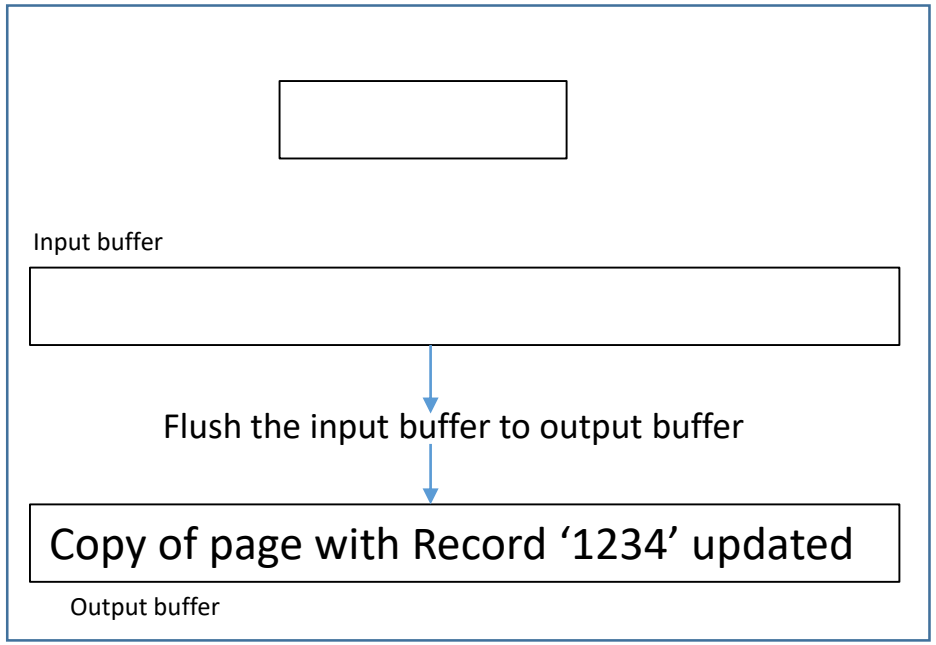
Secondary Storage holds the STAFF database



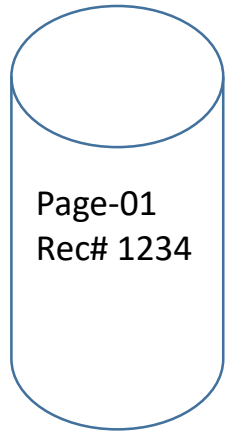
Original record ‘1234’ is still on a page on disk with the original value of salary. At this stage, the operation is ‘partially committed’ – the updated value of salary is in main memory, not on the disk

Transaction Control Example -1
Give staff number ‘1234’ a 10% pay rise

UPDATE Staff SET salary = salary*1.1 WHERE staffNo = 1234;



Secondary Storage holds the STAFF database

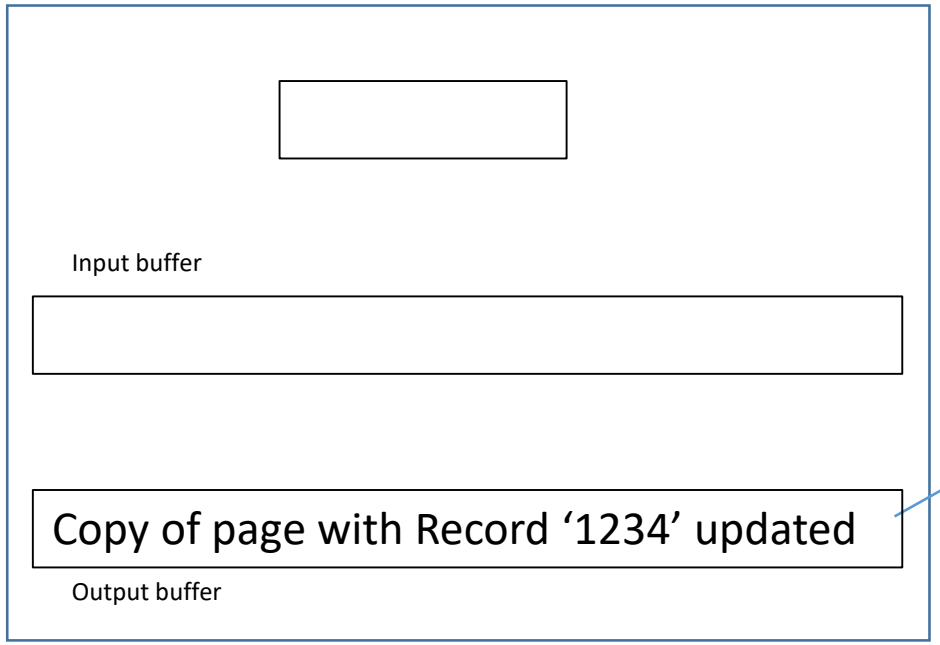


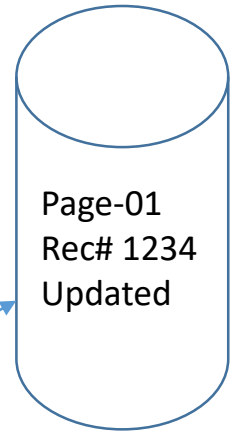
Original record ‘1234’ is still on a page on disk with the original value of salary. At this stage, the operation is ‘partially committed’ – the updated value of salary is in main memory, not on the disk

Transaction Control Example -1
Give staff number ‘1234’ a 10% pay rise

UPDATE Staff SET salary = salary*1.1 WHERE staffNo = 1234;

Secondary Storage holds the STAFF database





On commit:
Replace record ’1234’ on disk with update. Transaction is now committed

Buffer Management

- Buffer Manager is responsible for finding suitable buffer space for transaction execution
- Buffers are replaced using various strategies when they become full or need to be flushed back to disk
  - First-in-First-out (FIFO), Least-recently-Used (LRU)
  - Can use ‘Force Writing’
  - Can prevent re-reading from disk if data already in buffer

Buffer Management

- Two variables control buffer movement
  - Pin Count variable and Dirty Bit variable
    - Initially set to 0
- To select buffers for replacement, the buffer manager reads the variables such that:
  - If Pin Count = 1 then buffer is ‘pinned’ to memory and cannot be selected for replacement – find another buffer with Pin Count = 0
  - If Dirty Bit = 1 then buffer has been modified and can be flushed to disk

Buffer Management

- Steal, No Force policy
  - Used by most databases to control buffers
- Steal
  - Buffer can be flushed to disk before the commit operation (Pin Count set to 0)
  - Avoids using large amounts of buffer space by holding on to pages until commit operation
- No Force
  - Allows frequently-used pages to remain in memory