THE UNIVERSITY OF SYDNEY

School of Information Technologies

COMP9120 Relational Database Systems **Semester 2, 2016**

Example Solutions: Practice Questions Part 2

Question 1: Transactions [23 points]

a) Give a brief explanation for each of the four ACID properties that should ideally be guaranteed by database transactions. **[8 marks]**

Example Solution

ACID = Atomic, Consistent, Isolated, Durable

Atomic – Transactions happen in their entirety or not at all. The DBMS allows a completed transaction to commit its changes to the system, or roll-back to its original state.

Consistent – A transaction must preserve the consistency of the database. That is, if integrity constraints hold before a transaction starts, integrity constraints should also hold after the transaction terminates.

Isolated – Concurrent transactions should operate in isolation, unaffected by the operations of other transactions. This is achieved with a concurrency control mechanism such as Strict 2-Phase Locking.

Durable – The effects of committed transactions should survive a system failure, allowing the system to be recovered to the state at the point of the failure. A recovery manager uses techniques such as write-ahead logging to keep a record of changes made by transactions so that in the event of a crash the database can recover all committed changes.

b) Determine for each of the following schedules whether they are conflict-serializable or not.

(i) $r_1(X)$ $r_2(X)$ $w_1(X)$ $w_2(X)$ **[2 marks]**

Solution

Not conflict serializable: conflict pair $r_2(X)$ $w_1(X)$ requires part of T2 to happen before T1, but conflict pairs $r_1(X)$ $w_2(X)$ and $w_1(X)$ $w_2(X)$ require parts of T1 to happen before T2.

(ii) $w_1(X)\ r_2(Y)\ r_1(Y)\ r_2(X)$ **[2 marks]**

Solution

Conflict serializable: only conflict pair is $w_1(X) r_2(X)$, so execution schedule is equivalent to T1T2 (but not T2T1).

(iii) $r_1(X)\ r_1(Y)\ w_1(X)\ r_2(Y)\ w_3(Y)\ w_1(X)\ r_2(Y)$ **[2 marks]**

Solution

Not conflict serializable: conflict pairs $r_2(Y)\ w_3(Y)$ and $w_3(Y)\ r_2(Y)$ require parts of T3 to occur before parts of T2 and vice versa.

c) Briefly explain Strict Two Phase Locking (Strict 2PL), and show whether the above schedules can be produced by the protocol. **[9 marks]**

Example Solution

In 2PL transactions must lock the data they wish to read or write in advance of doing go. If a transaction cannot acquire a lock on an object, it must wait until the existing locks are released

Data that is only read can be acquired with a shared lock, while data that is to be written to must be acquired with an exclusive lock.

Shared locks can be held by multiple transactions on the same object, but an exclusive lock cannot be acquired on an object already shared-locked, and no other locks can be acquired on an object that is already exclusively locked.

In Strict 2PL acquired locks are held by the transaction until it commits or rolls back.

With shared (S) and exclusive (X) locks before corresponding operations, the schedules would become:

i) $S_1(X)\ r_1(X)\ S_2(X)\ r_2(X)\ X_1(X)\ w_1(X)\ X_2(X)\ w_2(X)$

Not allowed: $X_1(X)$ cannot be acquire since $S_2(X)$ already acquired.

Note: in this situation both transactions will be in DEADLOCK since both transactions need the other to release its locks.

ii) $X_1(X)\ w_1(X)\ S_2(Y)\ r_2(Y)\ S_1(Y)\ r_1(Y)\ S_2(X)\ r_2(X)$

Allowed, ***assuming $X_1(X)$ released after final operation*** of T1.

iii) $S_1(X)\ r_1(X)\ S_1(Y)\ r_1(Y)\ X_1(X)\ w_1(X)\ S_2(Y)\ r_2(Y)\ X_3(Y)\ w_3(Y)\ X_1(X)\ w_1(X)\ S_2(Y)\ r_2(Y)$

Not allowed: $X_3(Y)$ cannot be acquired because other transactions already have a lock on Y.

Question 2: Storage and Indexing [28 points]

A database contains the relation *Computers(<u>comp_id</u>, room_id, comp_desc)*. Each record is 250 bytes, with *comp_id* using 4 bytes and *room_id* using 16 bytes. The relation holds 30,000 entries. The relation is stored with a sparse integrated B+ tree index on *comp_id*. A secondary B+ tree index is also constructed on *room_id*. Entries in both indexes are of the form <*search_key*, *rowid*>, where *rowid* pointers take up 4 bytes. The page size for the DBMS is 4096 bytes, with 100 bytes reserved for header data.

a) Estimate the number of pages required to hold the data described (including indexes), stating any assumptions you have made in your calculations. **[12 marks]**

Example Solution

Available page size = 4096 – 100 = 3996 bytes

Record data

Total record size = 250 bytes

Max records per page = floor(3996/250) = 15 records

Assume average page fill=75%

Average records/page = 0.75 * 15 = 11.25

# pages for records = ceil(30000/11.25) = <u>2667 pages</u>

Primary Index:

Here we use the information given in the question that the index on comp_id takes the form <search_key, rowid>.

search key size is size of *comp_id* key = 4 bytes

<*search key*, *rowid*> entry size = 4 + 4 = 8 bytes

Max entries per page = floor(3996/8) = 499 entries

Assume average page fill of 75%

Fan out = 499 * 0.75 = 374.25

From lecture notes on Integrated Tree Index:

Lowest level of index = ceil(2667 record pages/ 374.25 fan-out) = 8 pages

Only one page (root) required for next level since 8 entries fits in one page.

Total # index pages = 1 + 8 = <u>9 pages</u>

Secondary Index:

search key size is size of *room_id* key = 16 bytes

<*search key*, *rowid*> entry size = 16 + 4 = 20 bytes

Max entries per page = floor(3996/20) = 199

Assume average page fill of 75%

Fan out = 199 * 0.75 = 149.25

Dense, unclustered index

# pages in leaf level = ceil(30000/149.25) = 202

202 leaf pages indexed by ceil(202/fan-out) = 2

Only one page (root) required for next level

Total # index pages = 1 + 2 + 202 = 205 pages

Total number of pages = 205 + 9 + 2667 = 2881

b) The Computers relation is updated and now contains 50,000 records, distributed uniformly across 500 *room_id* values, stored in 4000 pages. The updated primary B+-tree index has 3 levels (including the root but excluding the leaf-level record pages). The secondary B+-tree index also has three levels and a fan-out of 150.

(i) Calculate the I/O cost to obtain the records in the following query via the most appropriate index: **[2 marks]**

SELECT * FROM Computers C WHERE C.comp_id =4;

Example Solution

Equality search (since on key), so one result. Obtain via primary index. Cost is that of traversing 3 pages from tree levels, plus retrieval of one records page so 4 page reads.

(ii) Explain why the cost of the following query, again using the most appropriate index, will be higher than for the previous query: **[3 marks]**

SELECT * FROM Computers C WHERE C.room _id>=150 AND C.room_id<155;

Example Solution

Search must use secondary tree. Records are clustered on integrated primary index, not secondary index. The main drawback here is that, matches will not be co-located, and in worst case will require a page retrieval for each matching record.

Multiple matching values (5, specifically), and non-unique search key, so multiple records to retrieve (in this case 5 values with 50000/500=100 records per value gives 500 records to retrieve).

c) A second relation *Users(emp_id, comp_id, acces_priv)* exists in the database, with *comp_id* a foreign key to the *Computers* relation. The relation consists of 100,000 records stored in 600 pages. It also has an unclustered index on *comp_id*, with three levels (again, including the root). *Users* and the updated *Computers* relation are used to find employees with access to computers in certain rooms with the following query:

SELECT U.emp_id

FROM Computers C, Users U

WHERE C.comp_id =U.comp_id AND C.room _id>=150 AND C.room_id<155;

(i) Estimate the I/O cost of evaluating the query with a block-nested join (using blocks of one page for each table) of the two relations, followed by in-memory selection and projection of the matching tuples. **[3 marks]**

Example Solution

Block-nested loop join with single-page blocking costs N+N*M page reads, where N and M are the numbers of pages in the outer and inner relations, respectively

Computers has 4000 pages, and Users has 600 pages, so make Users the outer relation to minimise the cost I/O cost is 600+600*4000 = 2400600 page reads

(ii) Consider an alternative plan in which a range search first finds records from *Computers* with matching *room_id* values, followed by an index nested loop join with *Users* and in-memory projection of the results. Explain what factors affect the I/O cost of this plan, and how these differ to the previous plan. **[8 marks]**

Example Solution

Range search on Computers makes use of secondary index for room_id and costs:

3 (traverse each level of *secondary* index tree) + 3 (get additional (linked) leaf pages containing data entries: (each entry fits on a page, but each page has a maximum number of index entries it can fit)) + 500 (get pages containing data records since secondary index is unclustered and not integrated like primary) = 506 page cost

Matching records materialise to fewer pages than whole Computers relation (to 500/15=34 pages) but there is a cost to write these back to disk (34 writes, assuming buffer frame can't hold these before index join)

Index-nested loop join costs N+r*Q page reads, where N is the number of pages in the outer relation, r is the number of records in the outer relation, and Q is the cost to find a matching record via the index

Users has 100,000 records, for 50,000 computers, so assume uniform distribution and so 2 records for each computer.

Index lookup on comp_id in Users will be 3 (tree levels), plus 2 pages for each match, assuming each record is in a different page.

Index join cost will be 34 + 500*(3+2) = 2534

Total cost is therefore 506 + 34 + 2534 = <u>3074 I/Os</u>

Pipelining would avoid materialisation costs, saving 34+34=68 I/Os