Guarantees of Atomicity and Isolation

- Serializability
  - conflict, view
- Concurrency Control
  - locking
    - 2-phase locking, strict, rigorous
    - granularity, intention locks
- Other approaches to concurrency control
  - snapshot isolation
  - timestamp-based
  - optimistic
- Quiz 8
- Weak levels of concurrency
- Recovery

- Extensions:
  - Assign 7 by two days
  - Assign 8 by a week

quiz8

Q3
2 Points

Say we have a primary B+-tree of height 4 on attribute "zipcode" (not a candidate key) in a "person" table. So there will be many records with the same zipcode, but they are consecutive because the relation is sorted by zipcode.

Consider a query to find all people in a specific zipcode, and let's say there are 100 records with that zipcode. Further, let's say a single relation block can hold 10 records. Estimate the cost of executing this query. Assume $t_S = 4\text{ms}$, and $t_T = 0.1\text{ms}$.

How many milliseconds would this take? Enter nothing but the number, e.g., "49.3".

\[
= (4 + 1) \times 4.1 + 9 \times 0.1 = 21.4 \text{ msec}
\]
**Q4**  
2 Points

Do the same but with the assumption that the index is a secondary index (i.e., the relation data is not sorted by zipcode). Assume the order of the B+-tree is 500, i.e, 500 search keys/pointers can fit on a single B+-tree node.

\[
= (4 + 100) \times 4.1 = 104 \times 4.1 = 416 + 10.4 = 426.4
\]

---

**Q5**  
3 Points

Consider a query with a conjunctive predicate:

\[
\text{select * from R where a = 10 and b = 20.}
\]

- R occupies 1 million blocks on disk, and
- there are secondary indexes of height 4 on both R.a and R.b.
- Assume number of tuples in R with R.a = 10 is 1000, with R.b = 20 is 3000, and with both R.a = 10 & R.b = 20 is 200.

For all the indexes, assume the number of pointers in each leaf (to the actual records) is 500, and number of records of R per block is 100.

\[
= 4 \text{ (first leaf)} + 1 \text{ (second leaf)} + 1000 \text{ (data)} = 1005
\]
5.2: The same as the above but using the index on R.b

Q5
3 Points
Consider a query with a conjunctive predicate:

select * from R where a = 10 and b = 20.

- R occupies 1 million blocks on disk, and
- there are secondary indexes of height 4 on both R.a and R.b.
- Assume number of tuples in R with R.a = 10 is 1000, with R.b = 20 is 3000, and with both R.a = 10 & R.b = 20 is 200.

For all the indexes, assume the number of pointers in each leaf (to the actual records) is 500, and number of records of R per block is 100.

= blocks = 4 (first leaf) + 5 (extra leaf) + 3000 (data) = 3009

5.3: Same as above, but instead using "index-anding" to identify tuples w/o reading them, and then reading only those that match the entire predicate:

Q5
3 Points
Consider a query with a conjunctive predicate:

select * from R where a = 10 and b = 20.

- R occupies 1 million blocks on disk, and
- there are secondary indexes of height 4 on both R.a and R.b.
- Assume number of tuples in R with R.a = 10 is 1000, with R.b = 20 is 3000, and with both R.a = 10 & R.b = 20 is 200.

For all the indexes, assume the number of pointers in each leaf (to the actual records) is 500, and number of records of R per block is 100.

= 4 (first leaf a) + 1 (extra leaf a) + 4 (first leaf b) + 5 (extra leaf b) + 200 (data) = 214
6.1: Specify the minimal number of disk I/Os (i.e., # blocks transferred) required to identify all matching tuples

Consider a query with a disjunctive predicate:

```sql
select * from R where a = 10 OR b = 20
```

- R occupies 1 million blocks on disk
- secondary indexes of height 4 on both R.a and R.b
- 1000 tuples match R.a = 10, 1500 match R.b = 20,
  2000 tuples match the entire predicate.

What are the total number of disk I/Os (i.e., # blocks transferred) for each of the following options?

For all the indexes, assume the number of pointers on the leaf level (to the actual records) is 500 per block, and number of records of R per block is 100.

\[
4 \text{ (first leaf } a) + 1 \text{ (extra leaf } a) + 4 \text{ (first leaf } b) + 2 \text{ (extra leaf } b) = 11
\]

6.2: Using this approach (OR-ing the ptr sets), what would then be the total number of disk accesses to identify and load all matching tuples?

Consider a query with a disjunctive predicate:

```sql
select * from R where a = 10 OR b = 20
```

- R occupies 1 million blocks on disk
- secondary indexes of height 4 on both R.a and R.b
- 1000 tuples match R.a = 10, 1500 match R.b = 20,
  2000 tuples match the entire predicate.

What are the total number of disk I/Os (i.e., # blocks transferred) for each of the following options?

For all the indexes, assume the number of pointers on the leaf level (to the actual records) is 500 per block, and number of records of R per block is 100.

\[
4 \text{ (first leaf } a) + 1 \text{ (extra leaf } a) + 4 \text{ (first leaf } b) + 2 \text{ (extra leaf } b) + 2000 = 2011
\]
**Q4.1**
1 Point

How many runs are created?

- \( \text{br} = 400 \text{ M} = 100 \) \( \rightarrow \) 4 runs
- each run the size of memory, so 100 blocks

**Q4.2**
1 Point

How many seeks for run creation?

- for each run need to seek to beginning of space in table to read, and then again to write. = \( 4 \times 2 \times 8 \)
- total blocks is just \( 2 \times \text{br} = 800 \)
quiz 9

Q4 External sorting
3 Points

Assume you need to sort 2000 tuples. 5 tuples fit into a block, and the system has 100 blocks of memory available. Count all reads and writes.

Q4.3
1 Point

How many merge steps (phases) are needed?

Total number of disk transfers for external sorting:

\[ b_r \cdot (2 \cdot \lceil \log_{b_r} (b_r/M) \rceil + 1) \]

Seeks:

\[ 2 \cdot \lceil b_r/M \rceil + \lceil b_r/b_r \rceil \cdot \lceil \log_{b_r} (b_r/M) \rceil - 1 \]

\[ = \text{ceiling}(\log_{99}(b_r/M)) = 1 \]

quiz 9

Q4 External sorting
3 Points

Assume you need to sort 2000 tuples. 5 tuples fit into a block, and the system has 100 blocks of memory available. Count all reads and writes.

Q4.4
0 Points

Assume 60 blocks are used for the output buffer.

How many seeks for merging?

Total number of disk transfers for external sorting:

\[ b_r \cdot (2 \cdot \lceil \log_{b_r} (b_r/M) \rceil + 1) \]

Seeks:

\[ 2 \cdot \lceil b_r/M \rceil + \lceil b_r/b_r \rceil \cdot \lceil \log_{b_r} (b_r/M) \rceil - 1 \]

\[ b_b \text{ is size of input and output buffers assume } = 1 \]

• seeks:
  • 60 blocks output buffer.
    • has to be emptied \( \text{ceiling}(400 / 60) = 7 \) times
  • \((100 - 60) / 4 = 10\) blocks for each input buffer
    • has to be refilled \( \text{ceiling}(100 / 10) = 10 \) times per run = 10 * 4 = 40
  • total is 7 + 40 = 47 seeks
  • block transfers = 2 * entire relation = 2 * 400 = 800
Consider an equi-join on attribute B of relations R and S.

Assume:
- \( b_R \) and \( b_S \) are 1000 and 2000, respectively
- we have an B+tree index on relation S on attribute B, of height 3.
- leaf nodes hold ptrs to 500 records
- B is a key in R, but not in S.
- 100 tuples in R each have 4 matches in S
- Each block of R or S holds 50 tuples.
- The below totals should reflect both identifying the matches, and returning the corresponding tuples.

• R blocks? No index, so read through the whole thing and probe S’s index: 1000
  • index blocks?
    • 50 * 1000 tuples, each w/ a cost 3 probe = 150,000
  • S blocks?
    • 100 values of S match
      • they are consecutive
      • assume each back of 4 are on the same page
      • = 100 * 1 = 100

• R blocks? No index, so read through the whole thing and probe S’s index: 1000. (same)
  • index blocks?
    • 50 * 1000 tuples, each w/ a cost 3 probe = 150,000 (same)
  • S blocks?
    • 100 values of S match
      • they are consecutive
      • assume each of the four matching R value are on different pages
      • = 100 * 4 = 400 (different)
Midterm 2

- quizzes 7, 8, 9
- assignments 5, 6, 7

non-complete Set of Topics:
- mapping relations to files
- all in one
- 1-to-1
- co-locating related relations
- mapping tuples to files
- no ordering (heap organization)
- ordered
- hash (a bucket per page, for example)
- insertions and deletions
- indexes
- primary vs secondary
- dense vs sparse
- B+-trees
- query processing
- cost estimation
- selectivity
- histograms
- etc
- query costs
- sorting
- sort-merge
- joins
- nested loop
- block nested loop
- indexed nested-loop join
- merge join
- hash join

Optimistic Concurrency Control

- Based on validation at transaction end (section 18.6)
- Intuition
  - Let the transactions execute as they wish
  - At the very end when they are about to commit, check if there might be any problems/conflicts etc
    - If no, let it commit
    - If yes, abort and restart
- Upshot:
  - Hope not too many problems/aborts
  - Very fast for read transactions, or when low contention
Optimistic Concurrency Control

- Each transaction $T_i$ has 3 timestamps
  - Start($T_i$) : the time when $T_i$ started its execution
  - Validation($T_i$): the time when $T_i$ enters its validation phase
  - Finish($T_i$) : the time when $T_i$ finished its write phase
- Serializability order is validation order
  - $TS(T_i) = Validation(T_i)$
  - increases concurrency.
- Higher degree of concurrency if conflicts low.
  - because the serializability order is not pre-decided, and
  - relatively few transactions will have to be rolled back.

Optimistic Concurrency Control

- If for all $T_i$ with $TS(T_i) < TS(T_k)$ either one of the following condition holds:
  - $finish(T_i) < start(T_k)$ or
  - $start(T_k) < finish(T_i) < validation(T_k)$ and the set of data items written by $T_i$ does not intersect with the set of data items read by $T_k$.
  - then validation succeeds and $T_k$ can be committed. Otherwise, validation fails and $T_k$ is aborted.
- Justification: Either the first condition is satisfied, and there is no overlapped execution, or the second condition is satisfied and
  - the writes of $T_k$ do not affect reads of $T_i$ since they occur after $T_i$ has finished its reads.
  - the writes of $T_i$ do not affect reads of $T_k$ since $T_k$ does not read any item written by $T_i$. 
Optimistic Concurrency Control

- $T_{25} < T_{26}$
- $T_{25}$ validates because first
- $T_{26}$ validates because $T_{25}$ had no writes

<table>
<thead>
<tr>
<th>$T_{25}$</th>
<th>$T_{26}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>read($B$)</td>
<td>read($B$)</td>
</tr>
<tr>
<td>$B := B - 50$</td>
<td></td>
</tr>
<tr>
<td>read($A$)</td>
<td>read($A$)</td>
</tr>
<tr>
<td>$A := A + 50$</td>
<td></td>
</tr>
<tr>
<td>&lt;validate&gt; display($A + B$)</td>
<td>&lt;validate&gt; write($B$)</td>
</tr>
<tr>
<td></td>
<td>&lt;validate&gt; write($A$)</td>
</tr>
</tbody>
</table>

---

Optimistic Concurrency Control

\[ x = 0 \]

<table>
<thead>
<tr>
<th></th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;start&gt; &lt;validate&gt; &lt;commit&gt;</td>
<td>&lt;start&gt; &lt;validate&gt;</td>
<td>&lt;start&gt; &lt;validate&gt;</td>
<td>&lt;start&gt; &lt;validate&gt; &lt;commit&gt;</td>
</tr>
<tr>
<td>write($X$) = 1</td>
<td>read($X$) = 0</td>
<td>read($X$) = 1</td>
<td>read($X$) = 1</td>
</tr>
<tr>
<td>&lt;validate&gt; &lt;commit&gt;</td>
<td>&lt;validate&gt;</td>
<td>&lt;validate&gt;</td>
<td>&lt;validate&gt;</td>
</tr>
<tr>
<td>&lt;validate&gt; &lt;commit&gt;</td>
<td>&lt;validate&gt;</td>
<td>&lt;validate&gt;</td>
<td>&lt;commit&gt;</td>
</tr>
</tbody>
</table>