Database Management and Performance Tuning

Concurrency Tuning

Pei Li

University of Zurich
Institute of Informatics

Unit 7

Acknowledgements: The slides are provided by Nikolaus Augsten
and adapted from "Database Tuning" by Dennis Shasha and Philippe Bonnet.

Outline

1. Index Tuning
   - Index Tuning Examples

2. Concurrency Tuning
   - Introduction to Transactions

Index Tuning Examples

- The examples use the following tables:
  - `Employee(ssnum,name,dept,manager,salary)`
  - `Student(ssnum,name,course,grade,stipend,evaluation)`

Exercise 1 – Query for Student by Name

- `Student` was created with non-clustering index on `name`.
- Query:

```
SELECT *
FROM Student
WHERE name='Bayer'
```

- Problem: Query does not use index on `name`.
- Solution: Try updating the catalog statistics.
  - Oracle, Postgres: `ANALYZE`
  - SQL Server: sp_createstats
  - DB2: `RUNSTATS`

Exercise 2 – Query for Salary I

- Non-clustering index on `salary`.
- Catalog statistics are up-to-date.
- Query:

```
SELECT *
FROM Emplyee
WHERE salary/12 = 4000
```

- Problem: Query is too slow.
- Solution: Index not used because of the arithmetic expression. Two Options:
  - Rewrite query:

    ```
    SELECT *
    FROM Emplyee
    WHERE salary = 48000
    ```

  - Use function based index.

Exercise 3 – Query for Salary II

- Non-clustering index on `salary`.
- Catalog statistics are up-to-date.
- Query:

```
SELECT *
FROM Emplyee
WHERE salary = 48000
```

- Problem: Query still does not use index. What could be the reason?
- Solution: The index is non-clustering. Many employees have a salary of 48000, thus the index may not help. It may still help for other, less frequent, salaries!

Exercise 4 – Clustering Index and Overflows

- Clustering index on `Student.ssnum`
- Page size: $2kB$
- Record size in `Student` table: $1KB$ (`evaluation` is a long text)
- Problem: Overflow when new evaluations are added.
- Solution: Clustering index does not help much due to large record size. A non-clustering index avoids overflows.

Exercise 5 – Non-clustering Index I

- `Employee` table:
  - 30 employee records per page
  - each employee belongs to one of 50 departments (`dept`)
  - the departments are of similar size
- Query:

```
SELECT ssnum
FROM Emplyee
WHERE dept = 'IT'
```

- Problem: Does a non-clustering index on `Employee.dept` help?
- Solution: Only if the index covers the query.
  - 30/50=60% of the pages will have a record with `dept = 'IT'`
  - table scan is faster than accessing 3/5 of the pages in random order

Exercise 6 – Non-clustering Index II

- `Employee` table:
  - 30 employee records per page
  - each employee belongs to one of 5000 departments (`dept`)
  - the departments are of similar size
- Query:

```
SELECT ssnum
FROM Emplyee
WHERE dept = 'IT'
```

- Problem: Does a non-clustering index on `Employee.dept` help?
- Solution: Only if the index covers the query.
  - only 30/5000=0.06% of the pages will have a record with `dept='IT'`
  - table scan is slower

Exercise 7 – Statistical Analysis

- Auditors run a statistical analysis on a copy of `Emplyee`.
- Queries:
  - count employees with a certain salary (frequent)
  - find employees with maximum or minimum salary within a particular department (frequent)
  - find an employee by its social security number (rare)
- Problem: Which indexes to create?
- Solution:
  - non-clustering index on `salary` (covers the query)
  - clustering composite index on `(dept, salary)` using a B^+-tree (all employees with the maximum salary are on consecutive pages)
  - non-clustering hash index on `ssnum`

Exercise 8 – Algebraic Expressions

- Student stipends are monthly, employee salaries are yearly.
- Query: Which employee is paid as much as which student?
- There are two options to write the query:

```
SELECT *                      SELECT *
FROM Employee, Student        FROM Employee, Student
WHERE salary = 12*stipend     WHERE salary/12 = stipend
```

- Index on a table with an algebraic expression not used.
- Problem: Which query is better?

Exercise 8 – Solution

- If index on only one table, it should be used.
- Index on both tables, clustering on larger table: use it.
- Index on both tables, non-clustering on larger table:
  - small table has (much) less records than large table has pages: use index on large table
  - otherwise: use index on small table

Exercise 9 – Purchasing Department

- Purchasing department maintains table `Onorder(supplier,part,quantity,price)`.
- The table is heavily used during the opening hours, but not over night.
- Queries:
  - Q1: add a record, all fields specified (very frequent)
  - Q2: delete a record, `supplier` and `part` specified (very frequent)
  - Q3: find total quantity of a given part on order (frequent)
  - Q4: find the total value on order to a given supplier (rare)
- Problem: Which indexes should be used?

Exercise 9 – Solution

- Queries:
  - Q1: add a record, all fields specified (very frequent)
  - Q2: delete a record, `supplier` and `part` specified (very frequent)
  - Q3: find total quantity of a given part on order (frequent)
  - Q4: find the total value on order to a given supplier (rare)
- Solution: Clustering composite B^+-tree index on `(part,supplier)`.
  - eliminate overflows over night
  - attribute order important to support query Q3
  - hash index will not work for query Q3 (prefix match query)
- Discussion: Non-clustering index on `supplier` to answer query Q4?
  - index must be maintained and will hurt the performance of much more frequent queries Q1 and Q2
  - index does not help much if there are only few different suppliers

Exercise 10 – Point Query Too Slow

- `Employee` has a clustering B^+-tree index on `ssnum`.
- Queries:
  - retrieve employee by social security number (`ssnum`)
  - update employee with a specific social security number
- Problem: Throughput is still not enough.
- Solution: Use hash index instead of B^+-tree (faster for point queries).

Exercise 11 – Historical Immigrants Database

- Digitalized database of US immigrants between 1800 and 1900:
  - 17M records
  - each record has approx. 200 fields
    e.g., last name, first name, city of origin, ship taken, etc.
- Queries retrieve immigrants:
  - by last name and at least one other attribute
  - second attribute is often first name (most frequent) or year
- Problem: Efficiently serve 2M descendants of the immigrants. . .

Exercise 11 – Solution

- Clustering B^+-tree index on (`lastname,firstname`):
  - no overflow since database does not have updates
  - use high fill factor to increase space utilization
  - key compression should be used (long key, no update)
  - index useful also for prefix queries on `lastname`
- Composite non-clustering index on (`lastname,year`):
  - no maintenance cost (no updates)
  - attributes probably selective enough
- Non-clustering indexes on all frequent attribute combinations?
  - no maintenance, thus only limitation is space overhead
  - useful only if selective enough

Exercise 12 – Flight Reservation System

- An airline manages 1000 flights and uses the tables:
  - `Flight(flightID, seatID, passanger-name)`
  - `Totals(flightID, number-of-passangers)`
- Query: Each reservation
  - adds a record to `Flight`
  - increments `Totals.number-of-passangers`
- Queries are separate transactions.
- Problem: Lock contention on `Totals`.
- Solution:
  - Totals is a small table (1000 small records) and fits on few pages.
  - Without index, update scans table and scanned records are locked.
  - Clustering index on `flightID` avoids table scan and thus lock contention (row locking assumed).

Outline

1. Index Tuning
   - Index Tuning Examples
2. Concurrency Tuning
   - Introduction to Transactions

What is a Transaction?[1]

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Two main issues:
  1. concurrent execution of multiple transactions
  2. failures of various kind (e.g., hardware failure, system crash)

[1] Slides of section "Introduction to Transactions" are adapted from the slides "Database System Concepts", 6^{th} Ed., Silberschatz, Korth, and Sudarshan

ACID Properties

- Database system must guarantee ACID for transactions:
  - Atomicity: either all operations of the transaction are executed or none
  - Consistency: execution of a transaction in isolation preserves the consistency of the database
  - Isolation: although multiple transactions may execute concurrently, each transaction must be unaware of the other concurrent transactions.
  - Durability: After a transaction completes successfully, changes to the database persist even in case of system failure.

Atomicity

- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- What if failure (hardware or software) after step 3?
  - money is lost
  - database is inconsistent
- Atomicity:
  - either all operations or none
  - updates of partially executed transactions not reflected in database

Consistency

- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Consistency in example: sum $A + B$ must be unchanged
- Consistency in general:
  - explicit integrity constraints (e.g., foreign key)
  - implicit integrity constraints (e.g., sum of all account balances of a bank branch must be equal to branch balance)
- Transaction:
  - must see consistent database
  - during transaction inconsistent state allowed
  - after completion database must be consistent again

Isolation – Motivating Example

- Example: transfer $50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Imagine second transaction T_2:
  - $T_2 : R(A), R(B), print(A + B)$
  - T_2 is executed between steps 3 and 4
  - T_2 sees an inconsistent database and gives wrong result

Isolation

- Trivial isolation: run transactions serially
- Isolation for concurrent transactions: For every pair of transactions T_i and T_j, it appears to T_i as if either T_j finished execution before T_i started or T_j started execution after T_i finished.
- Schedule:
  - specifies the chronological order of a sequence of instructions from various transactions
  - equivalent schedules result in identical databases if they start with identical databases
- Serializable schedule:
  - equivalent to some serial schedule
  - serializable schedule of $T1$ and $T2$ is either equivalent to $T1, T2$ or $T2, T1$

Durability

- When a transaction is done it commits.
- Example: transaction commits too early
  - transaction writes A, then commits
  - A is written to the disk buffer
  - then system crashes
  - value of A is lost
- Durability: After a transaction has committed, the changes to the database persist even in case of system failure.
- Commit only after all changes are permanent:
  - either written to log file or directly to database
  - database must recover in case of a crash

Locks

- A lock is a mechanism to control concurrency on a data item.
- Two types of locks on a data item A:
  - exclusive – $xL(A)$: data item A can be both read and written
  - shared – $sL(A)$: data item A can only be read.
- Lock request are made to concurrency control manager.
- Transaction is blocked until lock is granted.
- Unlock A – $uL(A)$: release the lock on a data item A

Lock Compatibility

- Lock compatibility matrix:

$T_1 \downarrow \quad T_2 \rightarrow$	shared	exclusive
shared	true	false
exclusive	false	false

- T_1 holds shared lock on A:
  - shared lock is granted to T_2
  - exclusive lock is not granted to T_2
- T_2 holds exclusive lock on A:
  - shared lock is not granted to T_2
  - exclusive lock is not granted to T_2
- Shared locks can be shared by any number of transactions.

Locking Protocol

- Example transaction T_2 with locking:
  1. $sL(A)$, $R(A)$, $uL(A)$
  2. $sL(B)$, $R(B)$, $uL(B)$
  3. $print(A + B)$
- T_2 uses locking, but is not serializable
  - A and/or B could be updated between steps 1 and 2
  - printed sum may be wrong
- Locking protocol:
  - set of rules followed by all transactions while requesting/releasing locks
  - locking protocol restricts the set of possible schedules

Pitfalls of Locking Protocols – Deadlock

- Example: two concurrent money transfers
  - T_1: $R(A), A \leftarrow A + 10, R(B), B \leftarrow B - 10, W(A), W(B)$
  - T_2: $R(B), B \leftarrow B + 50, R(A), A \leftarrow A - 50, W(A), W(B)$
  - possible concurrent scenario with locks: $T_1.xL(A)$, $T_1.R(A)$, $T_2.xL(B)$, $T_2.R(B)$, $T_2.xL(A)$, $T_1.xL(B)$, $\ldots$
  - T_1 and T_2 block each other – no progress possible
- Deadlock: situation when transactions block each other
- Handling deadlocks:
  - one of the transactions must be rolled back (i.e., undone)
  - rolled back transaction releases locks

Pitfalls of Locking Protocols – Starvation

- Starvation: transaction continues to wait for lock
- Examples:
  - the same transaction is repeatedly rolled back due to deadlocks
  - a transaction continues to wait for an exclusive lock on an item while a sequence of other transactions are granted shared locks
- Well-designed concurrency manager avoids starvation.

Two-Phase Locking

- Protocol that guarantees serializability.
- Phase 1: growing phase
  - transaction may obtain locks
  - transaction may not release locks
- Phase 2: shrinking phase
  - transaction may release locks
  - transaction may not obtain locks

Two-Phase Locking – Example

- Example: two concurrent money transfers
  - T_1: $R(A), A \leftarrow A + 10, R(B), B \leftarrow B - 10, W(A), W(B)$
  - T_2: $R(A), A \leftarrow A - 50, R(B), B \leftarrow B + 50, W(A), W(B)$
- Possible two-phase locking schedule:
  1. $T_1 : xL(A), xL(B), R(A), R(B), W(A \leftarrow A + 10), uL(A)$
  2. $T_2 : xL(A), R(A), xL(B)$ *(wait)*
  3. $T_1 : W(B \leftarrow B - 10), uL(B)$
  4. $T_2 : R(B), W(A \leftarrow A - 50), W(B \leftarrow B + 50), uL(A), uL(B)$
- Equivalent serial schedule: T_1, T_2