Topic 3: Performance Modelling and Database Transactions

Introduction

This topic presents the learner with an understanding on basic implementation of performance on a database and management of transactions. Database performance being critical in its operations need to be ensured during the creation of a database which is the main focus of this topic. The topic further looks at the various areas of concern during performance modelling and the expected activities to be undertaken during the process. The topic also highlights on the steps involved in ensuring performance of a database is enforced including the guidelines be followed.

Objectives

By the end of this topic the learner should be able to:

i. Describe the process of performance modelling
ii. Explain performance modelling activities
iii. Explain the guidelines necessary in ensuring performance in a database
iv. Describe the steps to be followed in performance modelling

Learning activities

Learning Activity 3.1: Reading

Read the provided topic notes on performance modelling scenarios.

Learning Activity 3.2 Discussion

Participate in a discussion on the challenges likely to be faced during performance modelling.

Assessment

Participation on the discussion in learning activity 3.2 will be graded.

Topic Resources

1. S. K. Singh, (2011): Database Systems Concepts, Designs and Application (2nd ed), Pearson Education.
2. P. Rob and C. Coronel (2014): Database Systems Design, Implementation and Management (7th ed), Thomson Learning-Course Technology
3. P. O’Neil and E. O’Neil (2010): Database Principles, Programming and Performance (1st ed), Harcourt Asia Pte. Ltd.

https://www.tutorialspoint.com/dbms/dbms_transaction.htm
https://www.guru99.com/dbms-transaction-management.html

Topic 3 Notes

3.1 Performance Modeling

Performance modeling is a structured and repeatable approach to modeling the performance of your software. It begins during the early phases of your application design and continues throughout the application life cycle.

Performance is generally ignored until there is a problem. There are several problems with this reactive approach:

Performance problems are frequently introduced early in the design.

Design issues cannot always be fixed through tuning or more efficient coding.

Fixing architectural or design issues later in the cycle is not always possible. At best, it is inefficient, and is usually very expensive.

When you create performance models, you identify application scenarios and your performance objectives. Your performance objectives are your measurable criteria, such as response time, throughput (how much work in how much time), and resource utilization (CPU, memory, disk I/O, and network I/O). You break down your performance scenarios into steps and assign performance budgets. Your budget defines the resources and constraints across your performance objectives.

Performance modeling provides several important benefits:

Performance becomes part of your design.

Modeling helps answer the question "Will your design support your performance objectives?" By building and analyzing models, you can evaluate tradeoffs before you actually build the solution.

You know explicitly what design decisions are influenced by performance and the constraints performance puts on future design decisions. Frequently, these decisions are not captured and can lead to maintenance efforts that work against your original goals.

You avoid surprises in terms of performance when your application is released into production.

You end up with a document of itemized scenarios that help you quickly see what is important. That translates to where to instrument, what to test for, and how to know whether you are on or off track for meeting your performance goals.

Upfront performance modeling is not a replacement for scenario-based load testing or prototyping to validate your design. In fact, you have to prototype and test to determine what things cost and to see if your plan makes sense. Data from your prototypes can help you evaluate early design decisions before implementing a design that will not allow you to meet your performance goals.

Why Model Performance?

A performance model provides a path to discover what you do not know. The benefits of performance modeling include the following:

Performance becomes a feature of your development process and not an afterthought.

You evaluate your tradeoffs earlier in the life cycle based on measurements.

Test cases show you whether you are trending toward or away from the performance objectives throughout your application life cycle.

Modeling allows you to evaluate your design before investing time and resources to implement a flawed design. Having the processing steps for your performance scenarios laid out enables you to understand the nature of your application's work. By knowing the nature of this work and the constraints affecting that work, you can make more informed decisions.

Your model can reveal the following about your application:

i. What are the relevant code paths and how do they affect performance?

ii. Where do the use of resources or computations affect performance?
iii. Which are the most frequently executed code paths? This helps you identify where to spend time tuning.
iv. What are the key steps that access resources and lead to contention?
v. Where is your code in relation to resources (local, remote)?
vi. What tradeoffs have you made for performance?
vii. Which components have relationships to other components or resources?
viii. Where are your synchronous and asynchronous calls?
ix. What is your I/O-bound work and what is your CPU-bound work?

The model can reveal the following about your goals:

i. What is the priority and achievability of different performance goals?
ii. Where have your performance goals affected design?

Risk Management

The time, effort, and money you invest up front in performance modeling should be proportional to project risk. For a project with significant risk, where performance is critical, you may spend more time and energy up front developing your model. For a project where performance is less of a concern, your modeling approach might be as simple as white-boarding your performance scenarios.

Budget

Performance modeling is essentially a "budgeting" exercise. Budget represents your constraints and enables you to specify how much you can spend (resource-wise) and how you plan to spend it. Constraints govern your total spending, and then you can decide where to spend to get to the total. You assign budget in terms of response time, throughput, latency, and resource utilization.

Performance modeling does not need to involve a lot of up-front work. In fact, it should be part of the work you already do. To get started, you can even use a whiteboard to quickly capture the key scenarios and break them down into component steps.

If you know your goals, you can quickly assess if your scenarios and steps are within range, or if you need to change your design to accommodate the budget. If you do not know your goals (particularly resource utilization), you need to define your baselines. Either way, it is not long before you can start prototyping and measuring to get some data to work with.

3.1.1 What You Must Know

Performance models are created in document form by using the tool of your choice (a simple Word document works well). The document becomes a communication point for other team members. The performance model contains a lot of key information, including goals, budgets (time and resource utilization), scenarios, and workloads. Use the performance model to play out possibilities and evaluate alternatives, before committing to a design or implementation decision. You need to measure to know the cost of your tools. For example, how much does a certain API cost you?

3.1.2 Best Practices

Consider the following best practices when creating performance models:

Determine response time and resource utilization budgets for your design.

Identify your target deployment environment.

Do not replace scenario-based load testing with performance modeling, for the following reasons:

Performance modeling suggests which areas should be worked on but cannot predict the improvement caused by a change.
Performance modeling informs the scenario-based load testing by providing goals and useful measurements.
Modeled performance may ignore many scenario-based load conditions that can have an enormous impact on overall performance.

3.1.3 Information in the Performance Model

The information in the performance model is divided into different areas. Each area focuses on capturing one perspective. Each area has important attributes that help you execute the process.

Category	Description
Application Description	The design of the application in terms of its layers and its target infrastructure.
Scenarios	Critical and significant use cases, sequence diagrams, and user stories relevant to performance.
Performance Objectives	Response time, throughput, resource utilization.
Budgets	Constraints you set on the execution of use cases, such as maximum execution time and resource utilization levels, including CPU, memory, disk I/O, and network I/O.
Measurements	Actual performance metrics from running tests, in terms of resource costs and performance issues.
Workload Goals	Goals for the number of users, concurrent users, data volumes, and information about the desired use of the application.
Baseline Hardware	Description of the hardware on which tests will be run — in terms of network topology, bandwidth, CPU, memory, disk, and so on.

Inputs
A number of inputs are required for the performance modeling process. These include initial (maybe even tentative) information about the following:
Scenarios and design documentation about critical and significant use cases.
Application design and target infrastructure and any constraints imposed by the infrastructure.
QoS requirements and infrastructure constraints, including service level agreements (SLAs).
Workload requirements derived from marketing data on prospective customers.
Outputs
The output from performance modeling is the following:

Performance Model Document
The performance model document may contain the following:

i. Performance objectives.
ii. Budgets.
iii. Workloads.
iv. Itemized scenarios with goals.
v. Test cases with goals.

An itemized scenario is a scenario that you have broken down into processing steps. For example, an order scenario might include authentication, order input validation, business rules validation, and orders being committed to the database. The itemized scenarios include assigned budgets and performance objectives for each step in the scenario.

Test Cases with Goals
You use test cases to generate performance metrics. They validate your application against performance objectives. Test cases help you determine whether you are trending toward or away from your performance objectives.

Process
The performance modeling process model is summarized below

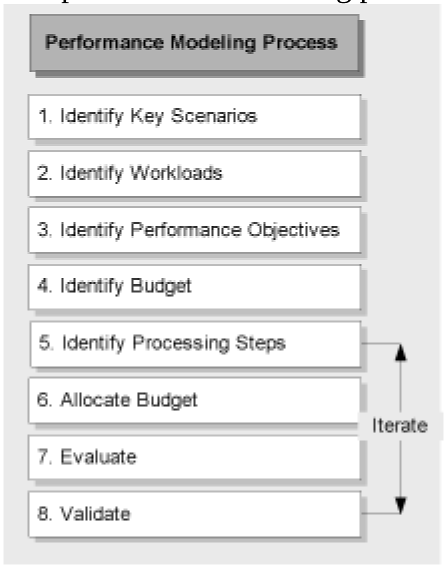
Performance Modeling Process
1. Identify Key Scenarios
2. Identify Workloads
3. Identify Performance Objectives
4. Identify Budget
5. Identify Processing Steps
6. Allocate Budget
7. Evaluate
8. Validate
Iterate

Fig 3.1 Performance modelling process

The performance modeling process involves the following steps:
Step 1 – Identify Key Scenarios
Identify your application scenarios that are important from a performance perspective. If you have documented use cases or user stories, use them to help you define your scenarios. Key scenarios include the following:

i. Critical scenarios.
ii. Significant scenarios.
iii. Critical Scenarios

These are the scenarios that have specific performance expectations or requirements. Examples include scenarios covered by SLAs or those that have specific performance objectives.

Significant Scenarios
Significant scenarios do not have specific performance objectives such as a response time goal, but they may impact other critical scenarios.
To help identify significant scenarios, identify scenarios with the following characteristics:

i. Scenarios that run in parallel to a performance-critical scenario.
ii. Scenarios that are frequently executed.

iii. Scenarios that account for a high percentage of system use.
iv. Scenarios that consume significant system resources.

Do not ignore your significant scenarios. Your significant scenarios can influence whether your critical scenarios meet their performance objectives. Also, do not forget to consider how your system will behave if different significant or critical scenarios are being run concurrently by different users. This "parallel integration" often drives key decisions about your application's units of work. For example, to keep search response brisk, you might need to commit orders one line item at a time.

Step 2 – Identify Workload

Workload is usually derived from marketing data. It includes the following:

i. Total users.
ii. Concurrently active users.
iii. Data volumes.
iv. Transaction volumes and transaction mix.

For performance modeling, you need to identify how this workload applies to an individual scenario. The following are example requirements:

You might need to support 100 concurrent users browsing.

You might need to support 10 concurrent users placing orders.

Note Concurrent users are those users that hit a Web site at exactly the same moment. Simultaneous users are those users who have active connections to the same site.

Step 3 – Identify Performance Objectives

For each scenario identified in Step 1, write down the performance objectives. The performance objectives are determined by your business requirements.

Performance objectives usually include the following:

Response time. For example, the product catalog must be displayed in less than 3 seconds.

Throughput. For example, the system must support 100 transactions per second.

Resource utilization. A frequently overlooked aspect is how much resource your application is consuming, in terms of CPU, memory, disk I/O, and network I/O.

Consider the following when establishing your performance objectives:

i. Workload requirements.
ii. Service level agreements.
iii. Response times.
iv. Projected growth.
v. Lifetime of your application.

For projected growth, you need to consider whether your design will meet your needs in six months time, or one year from now. If the application has a lifetime of only six months, are you prepared to trade some extensibility for performance? If your application is likely to have a long lifetime, what performance are you willing to trade for maintainability?

Step 4 – Identify Budget

Budgets are your constraints. For example, what is the longest acceptable amount of time that an operation should take to complete, beyond which your application fails to meet its performance objectives.

Your budget is usually specified in terms of the following:

i. Execution time.
ii. Resource utilization.

Execution Time

Your execution time constraints determine the maximum amount of time that particular operations can take.

Resource Utilization

Resource utilization requirements define the threshold utilization levels for available resources. For example, you might have a peak processor utilization limit of 75 percent and your memory consumption must not exceed 50 MB.

Common resources to consider include the following:

i. CPU.
ii. Memory.
iii. Network I/O.
iv. Disk I/O.

Step 5 – Identify Processing Steps

Itemize your scenarios and divide them into separate processing steps, such as those shown in below. If you are familiar with UML, use cases and sequence diagrams can be used as input. Similarly, Extreme Programming user stories can provide useful input to this step.

Processing Steps

1. An order is submitted by client.
2. The client authentication token is validated.
3. Order input is validated.
4. Business rules validate the order.
5. The order is sent to a database server.
6. The order is processed.
7. A response is sent to the client.

An added benefit of identifying processing steps is that they help you identify those points within your application where you should consider adding custom instrumentation. Instrumentation helps you to provide actual costs and timings when you begin testing your application.

Step 6 – Allocate Budget

Spread your budget (determined in Step 4, "Identify Budget") across your processing steps (determined in Step 5, "Identify Processing Steps") to meet your performance objectives. You need to consider execution time and resource utilization. Some of the budget may apply to only one processing step. Some of the budget may apply to the scenario and some of it may apply across scenarios.

Assigning Execution Time to Steps

When assigning time to processing steps, if you do not know how much time to assign, simply divide the total time equally between the steps. At this point, it is not important for the values to be precise because the budget will be reassessed after measuring actual time, but it is important to have an idea of the values. Do not insist on perfection, but aim for a reasonable degree of confidence that you are on track.

You do not want to get stuck, but, at the same time, you do not want to wait until your application is built and instrumented to get real numbers. Where you do not know execution times, you need to try spreading the time evenly, see where there might be problems or where there is tension.

If dividing the budget shows that each step has ample time, there is no need to examine these

further. However, for the ones that look risky, conduct some experiments (for example, with prototypes) to verify that what you will need to do is possible, and then proceed.
Note that one or more of your steps may have a fixed time. For example, you may make a database call that you know will not complete in less than 3 seconds. Other times are variable. The fixed and variable costs must be less than or equal to the allocated budget for the scenario.

Assigning Resource Utilization Requirements
When assigning resources to processing steps, consider the following:
Know the cost of your materials. For example, what does technology *x* cost in comparison to technology *y*.

i. Know the budget allocated for hardware. This defines the total resources available at your disposal.
ii. Know the hardware systems already in place.
iii. Know your application functionality. For example, heavy XML document processing may require more CPU, chatty database access or Web service communication may require more network bandwidth, or large file uploads may require more disk I/O.

Step 7 – Evaluate
Evaluate the feasibility and effectiveness of the budget before time and effort is spent on prototyping and testing. Review the performance objectives and consider the following questions:

i. Does the budget meet the objectives?
ii. Is the budget realistic? It is during the first evaluation that you identify new experiments you should do to get more accurate budget numbers.
iii. Does the model identify a resource hot spot?
iv. Are there more efficient alternatives?
v. Can the design or features be reduced or modified to meet the objectives?
vi. Can you improve efficiency in terms of resource consumption or time?
vii. Would an alternative pattern, design, or deployment topology provide a better solution?
viii. What are you trading off? Are you trading productivity, scalability, maintainability, or security for performance?
Consider the following actions:

i. Modify your design.
ii. Reevaluate requirements.
iii. Change the way you allocate budget.

Step 8 – Validate
Validate your model and estimates. Continue to create prototypes and measure the performance of the use cases by capturing metrics. This is an ongoing activity that includes prototyping and measuring. Continue to perform validation checks until your performance goals are met.
The further you are in your project's life cycle, the greater the accuracy of the validation. Early on, validation is based on available benchmarks and prototype code, or just proof-of-concept code. Later, you can measure the actual code as your application develops.

Revision questions

1. What is performance modelling
2. When is performance modelling done during database design?
3. Why should designers model performance of a database?

4. Why is budget allocation necessary in performance modelling?

What steps need to be followed during performance modeling?