Mocky Habeeb

A Developer's Guide to Amazon SimpleDB

Developer's Library

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Habeeb, Mocky, 1971-
A Developer's Guide to Amazon SimpleDB / Mocky Habeeb.
p. cm.
ISBN 978-0-321-62363-8 (pbk. : alk. paper) 1. Web services. 2. Amazon SimpleDB
(Electronic resource) 3. Cloud computing. 4. Database management. I. Title.
TK5105.88813.H32 2010
006.7'8—dc22
2010016954

Pearson Education, Inc
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671 3447

ISBN-13: 978-0-321-62363-8
ISBN-10: 0-321-62363-0

Text printed in the United States on recycled paper at RR Donnelley Crawfordsville in Crawfordsville, Indiana.

First printing, July 2010

Contents at a Glance

Contents

Preface

This book is a detailed guide for using Amazon SimpleDB. Over the years that I have been using this web service, I have always tried to contribute back to the developer community. This primarily involved answering questions on the SimpleDB forums and on stackoverflow.com. What I saw over time was a general lack of resources and understanding about the practical, day-to-day use of the service. As a result, the same types of questions were being asked repeatedly, and the same misconceptions seemed to be held by many people.

At the time of this writing, there are no SimpleDB books available. My purpose in writing this book is to offer my experience and my opinion about getting the most from SimpleDB in a more structured and thorough format than online forums. I have made every attempt to avoid rehashing information that is available elsewhere, opting instead for alternate perspectives and analysis.

About This Book

SimpleDB is a unique service because much of the value proposition has nothing to do with the actual web service calls. I am referring to the service qualities that include availability, scalability, and flexibility. These make great marketing bullet points, and not just for SimpleDB. You would not be surprised to hear those terms used in discussions of just about any server-side product. With SimpleDB, however, these qualities have a direct impact on how much benefit you get from the service. It is a service based on a specific set of tradeoffs; many features are specifically absent, and for good reason. In my experience, a proper understanding of these tradeoffs is essential to knowing if SimpleDB will be a good fit for your application.

This book is designed to provide a comprehensive discussion of all the important issues that come up when using SimpleDB. All of the available web service operations receive detailed coverage. This includes code samples, notes on how to solve common problems, and warnings about many pitfalls that are not immediately obvious.

Target Audience

This book is intended for software developers who want to use or evaluate SimpleDB. Certain chapters should also prove to be useful to managers, executives, or technologists who want to understand the value of SimpleDB and what problems it seeks to solve.

There is some difficulty in audience targeting that comes from the nature of the SimpleDB service. On the one hand, it is a web-based service that uses specific message formats over standard technologies like HTTP and XML. On the other hand, application developers, and probably most users, will never deal directly with the low-level wire protocol, opting instead for client software in his or her chosen programming language.

This creates (at least) two separate perspectives to use when discussing the service. The low-level viewpoint is needed for the framework designers and those writing a SimpleDB client, whereas a higher-level, abridged version is more suitable for application developers whose view of SimpleDB is strictly through the lens of the client software. In addition, the app developers are best served with a guide that uses a matching programming language and client.

The official Amazon documentation for SimpleDB is targeted squarely at the developers writing the clients. This is by necessity—SimpleDB is a web service, and the details need to be documented.

What I have tried to accomplish is the targeting of both groups. One of the most visible methods I used is splitting the detailed API coverage into two separate chapters.

Chapter 3, "A Code-Snippet Tour of the SimpleDB API," presents a detailed discussion of all the SimpleDB operations, including all parameters, error messages, and code examples in Java, C#, and PHP. This is fully suitable for both groups of developers, with the inclusion of practical advice and tips that apply to the operations themselves.

Chapter 10, "Writing a SimpleDB Client: A Language-Independent Guide," offers a guide and walkthrough for creating a SimpleDB client from scratch. This adds another layer to the discussion with much more detail about the low-level concerns and issues. This is intended for the developers of SimpleDB clients and those adding SimpleDB support to existing frameworks. Apart from Chapter 3, the remainder of the examples in the book are written in Java.

Code Examples

All of the code listings in this book are available for download at this book's website at http://www.simpledbbook.com/code.

1

Introducing Amazon SimpleDB

Amazon has been offering its customers computing infrastructure via Amazon Web Services (AWS) since 2006. AWS aims to use its own infrastructure to provide the building blocks for other organizations to use. The Elastic Compute Cloud (EC2) is an AWS offering that enables you to spin up virtual servers as you need the computing power and shut them off when you are done. Amazon Simple Storage Service (S3) provides fast and unlimited file storage for the web. Amazon SimpleDB is a service designed to complement EC2 and S3, but the concept is not as easy to grasp as "extra servers" and "extra storage." This chapter will cover the concepts behind SimpleDB and discuss how it compares to other services.

What Is SimpleDB?

SimpleDB is a web service providing structured data storage in the cloud and backed by clusters of Amazon-managed database servers. The data requires no schema and is stored securely in the cloud. There is a query function, and all the data values you store are fully indexed. In keeping with Amazon's other web services, there is no minimum charge, and you are only billed for your actual usage.

What SimpleDB Is Not

The name "SimpleDB" might lead you to believe that it is just like relational database management systems (RDBMS), only simpler to use. In some respects, this is true, but it is not just about making simplistic database usage simpler. SimpleDB aims to simplify the much harder task of creating and managing a database cluster that is fault-tolerant in the face of multiple failures, replicated across data centers, and delivers high levels of availability.

One misconception that seems to be very common among people just learning about SimpleDB is the idea that migrating from an RDBMS to SimpleDB will automatically solve your database performance problems. Performance certainly is an important part of the equation when you seek to evaluate databases. Unfortunately, for some people, speed is the beginning and the end of the thought process. It can be tempting to view any of the new hosted database services as a silver bullet when offered by a mega-company like Microsoft, Amazon, or Google. But the fact is that SimpleDB is not going to solve your existing speed issues. The service exists to solve an entirely different set of problems. Reads and writes are not blazingly fast. They are meant to be "fast enough." It is entirely possible that AWS may increase performance of the service over time, based on user feedback. But SimpleDB is never going to be as speedy as a standalone database running on fast hardware. SimpleDB has a different purpose.

Robust database clusters replicating data across multiple data centers is not a data storage solution that is typically easy to throw together. It is a time consuming and costly undertaking. Even in organizations that have the database administrator (DBA) expertise and are using multiple data centers, it is still time consuming. It is costly enough that you would not do it unless there was a quantifiable business need for it. SimpleDB offers data storage with these features on a pay-as-you-go basis.

Of course, taking advantage of these features is not without a downside. SimpleDB is a moderately restrictive environment, and it is not suitable for many types of applications. There are various restrictions and limitations on how much data can be stored and transferred and how much network bandwidth you can consume.

Schema-Less Data

SimpleDB differs from relational databases where you must define a schema for each database table before you can use it and where you must explicitly change that schema before you can store your data differently. In SimpleDB, there is no schema requirement. Although you still have to consider the format of your data, this approach has the benefit of freeing you from the time it takes to manage schema modifications.

The lack of schema means that there are no data types; all data values are treated as variable length character data. As a result, there is literally nothing extra to do if you want to add a new field to an existing database. You just add the new field to whichever data items require it. There is no rule that forces every data item to have the same fields.

The drawbacks of a schema-less database include the lack of automatic integrity checking in the database and an increased burden on the application to handle formatting and type conversions. Detailed coverage of the impact of schema-less data on queries appears in Chapter 4, "A Closer Look at Select," along with a discussion of the formatting issues.

Stored Securely in the Cloud

The data that you store in SimpleDB is available both from the Internet and (with less latency) from EC2. The security of that data is of great importance for many applications, while the security of the underlying web services account should be important to all users.

To protect that data, all access to SimpleDB, whether read or write, is protected by your account credentials. Every request must bear the correct and authorized digital signature or else it is rejected with an error code. Security of the account, data transmission, and data storage is the subject of Chapter 8,"Security in SimpleDB-Based Applications."

Billed Only for Actual Usage

In keeping with the AWS philosophy of pay-as-you-go, SimpleDB has a pricing structure that includes charges for data storage, data transfer, and processor usage.There are no base fees and there are no minimums.At the time of this writing,Amazon's monthly billing for SimpleDB has a free usage tier that covers the first gigabyte (GB) of data storage, the first GB of data transfer, and the first 25 hours of processor usage each month. Data transfer costs beyond the free tier have historically been on par with S3 pricing, whereas storage costs have always been somewhat higher. Consult the AWS website at https://aws.amazon.com/simpledb/ for current pricing information.

Domains, Items, and Attribute Pairs

The top level of data storage in SimpleDB is the domain.A domain is roughly analogous to a database table.You can create and delete domains as needed.There are no configuration options to set on a domain; the only parameter you can set is the name of the domain.

All the data stored in a SimpleDB domain takes the form of name-value attribute pairs. Each attribute pair is associated with an item, which plays the role of a table row. The attribute name is similar to a database column name but unlike database rows that must all have identical columns, SimpleDB items can each contain different attribute names.This gives you the freedom to store different data in some items without changing the layout of other items that do not have that data. It also allows the painless addition of new data fields in the future.

Multi-Valued Attributes

It is possible for each attribute to have not just one value, but an array of values. For example, an application that allows user tagging can use a single attribute named "tags" to hold as many or as few tags as needed for each item.You do not need to change a schema definition to enable multi-valued attributes.All you need to do is add another attribute to an item and use the same attribute name with a different value.This provides you with flexibility in how you store your data.

Queries

SimpleDB is primarily a key-value store, but it also has useful query functionality. A SQL-style query language is used to issue queries over the scope of a single domain. A subset of the SQL select syntax is recognized. The following is an example SimpleDB select statement:

```
SELECT * FROM products WHERE rating > '03' ORDER BY rating LIMIT 10
```

You put a domain name—in this case, `products`—in the `FROM` clause where a table name would normally be. The `WHERE` clause recognizes a dozen or so comparison operators, but an attribute name must always be on the left side of the operator and a literal value must always be on the right. There is no relational comparison between attributes allowed here. So, the following is not valid:

```
SELECT * FROM users WHERE creation-date = last-activity-date
```

All the data stored in SimpleDB is treated as plain string data. There are no explicit indexes to maintain; each value is automatically indexed as you add it.

High Availability

High availability is an important benefit of using SimpleDB. There are many types of failures that can occur with a database solution that will affect the availability of your application. When you run your own database servers, there is a spectrum of different configurations you can employ.

To help quantify the availability benefits that you get automatically with SimpleDB, let's consider how you might achieve the same results using replication for your own database servers. At the easier end of the spectrum is a master-slave database replication scheme, where the master database accepts client updates and a second database acts as a slave and pulls all the updates from the master. This eliminates the single point of failure. If the master goes down, the slave can take over. Managing these failures (when not using SimpleDB) requires some additional work for swapping IP addresses or domain name entries, but it is not very difficult.

Moving toward the more difficult end of the self-managed replication spectrum allows you to maintain availability during failure that involves more than a single server. There is more work to be done if you are going to handle two servers going down in a short period, or a server problem and a network outage, or a problem that affects the whole data center.

Creating a database solution that maintains uptime during these more severe failures requires a certain level of expertise. It can be simplified with cloud computing services like EC2 that make it easy to start and manage servers in different geographical locations. However, when there are many moving parts, the task remains time consuming. It can also be expensive.

When you use SimpleDB, you get high availability with your data replicated to different geographic locations automatically. You do not need to do any extra work or become an expert on high availability or the specifics of replication techniques for one vendor's database product. This is a huge benefit not because that level of expertise is not worth attaining, but because there is a whole class of applications that previously could not justify that effort.

Database Consistency

One of the consequences of replicating database updates across multiple servers and data centers is the need to decide what kind of consistency guarantees will be maintained. A database running on a single server can easily maintain strong consistency. With strong consistency, after an update occurs, every subsequent database access by every client reflects the change and the previous state of the database is never seen.

This can be a problem for a database cluster if the purpose of the cluster is to improve availability. If there is a master database replicating updates to slave databases, strong consistency requires the slaves to accept the update at the same time as the master. All access to the database would then be strongly consistent. However, in the case of a problem preventing communication between the master and a slave, the master would be unable to accept updates because doing so out of sync with a slave would break the consistency guarantee. If the database rejects updates during even simple problem scenarios, it defeats the availability. In practice, replication is often not done this way. A common solution to this problem is to allow only the master database to accept updates and do so without direct contact with any slave databases. After the master commits each transaction, slaves are sent the update in near real-time. This amounts to a relaxing of the consistency guarantee. If clients only connect to the slave when the master goes down, then the weakened consistency only applies to this scenario.

SimpleDB sports the option of either eventual consistency or strong consistency for each read request. With eventual consistency, when you submit an update to SimpleDB, the database server handling your request will forward the update to the other database servers where that domain is replicated. The full update of all replicas does not happen before your update request returns. The replication continues in the background while other requests are handled. The period of time it takes for all replicas to be updated is called the eventual consistency window. The eventual consistency window is usually small. AWS does not offer any guarantees about this window, but it is frequently less than one second.

A couple things can make the consistency window larger. One is a high request load. If the servers hosting a given SimpleDB domain are under heavy load, the time it takes for full replication is increased. Additionally a network or server failure can block replication until it is resolved. Consider a network outage between data centers hosting your data. If the SimpleDB load-balancer is able to successfully route your requests to both data centers, your updates will be accepted at both locations. However, replication will fail between the two locations. The data you fetch from one will not be consistent with updates you have applied to the other. Once the problem is fixed, SimpleDB will complete the replication automatically.

Using a consistent read eliminates the consistency window for that request. The results of a consistent read will reflect all previous writes. In the normal case, a consistent read is no slower than an eventually consistent read. However, it is possible for consistent read requests to display higher latency and lower bandwidth on occasion.

Sizing Up the SimpleDB Feature Set

The SimpleDB API exposes a limited set of features. Here is a list of what you get:

- You can create named domains within your account. At the time of this writing, the initial allocation allows you to create up to 100 domains. You can request a larger allocation on the AWS website.
- You can delete an existing domain at any time without first deleting the data stored in it.
- You can store a data item for the first time or for subsequent updates using a call to `PutAttributes`. When you issue an update, you do not need to pass the full item; you can pass just the attributes that have changed.
- There is a batch call that allows you to put up to 25 items at once.
- You can retrieve the data with a call to `GetAttributes`.
- You can query for items based on criteria on multiple attributes of an item.
- You can store any type of data. SimpleDB treats it all as string data, and you are free to format it as you choose.
- You can store different types of items in the same domain, and items of the same type can vary in which attributes have values.

Benefits of Using SimpleDB

When you use SimpleDB, you give up some features you might otherwise have, but as a trade-off, you gain some important benefits, as follows:

- **Availability—** When you store your data in SimpleDB, it is automatically replicated across multiple storage nodes and across multiple data centers in the same region.
- **Simplicity—** There are not a lot of knobs or dials, and there are not any configuration parameters. This makes it a lot harder to shoot yourself in the foot.
- **Scalability—** The service is designed for scalability and concurrent access.
- **Flexibility—** Store the data you need to store now, and if the requirements change, store it differently without changing the database.
- **Low latency within the same region—** Access to SimpleDB from an EC2 instance in the same region has the latency of a typical LAN.
- **Low maintenance—** Most of the administrative burden is transferred to Amazon. They maintain the hardware and the database software.

Database Features SimpleDB Doesn't Have

There are a number of common database features noticeably absent from Amazon SimpleDB. Programs based on relational database products typically rely on these features.You should be aware of what you will not find in SimpleDB, as follows:

- **Full SQL support—** A query language similar to SQL is supported for queries only. However, it only applies to "select" statements, and there are some syntax differences and other limitations.
- **Joins—** You can issue queries, but there are no foreign keys and no joins.
- **Auto-incrementing primary keys—** You have to create your own primary keys in the form of an item name.
- **Transactions—** There are no explicit transaction boundaries that you can mark or isolation levels that you can define.There is no notion of a commit or a rollback. There is some implicit support for atomic writes, but it only applies within the scope of each individual item being written.

Higher-Level Framework Functionality

This simplicity of what SimpleDB offers on the server side is matched by the simplicity of what AWS provides in officially supported SimpleDB clients.There is a one-to-one mapping of service features to client calls.There is a lot of functionality that can be built atop the basic SimpleDB primitives. In addition, the inclusion of these advance features has already begun with a number of third-party SimpleDB clients. Popular persistence frameworks used as an abstraction layer above relational databases are prime candidates for this.

Some features normally included within the database server can be written into SimpleDB clients for automatic handling.Third-party client software is constantly improving, and some of the following features may be present already or you may have to write it for yourself:

- **Data formatting—** Integers, floats, and dates require special formatting in some cases.
- **Object mapping—** It can be convenient to map programming language objects to SimpleDB attributes.
- **Sharding—** The domain is the basic unit of horizontal scalability in SimpleDB. However, there is no explicit support for automatically distributing data across domains.
- **Cache integration—** Caching is an important aspect of many applications, and caching popular data objects is a well-understood optimization. Configurable caching that is well integrated with a SimpleDB client is an important feature.

Service Limits

There are quite a few limitations on what you are allowed to do with SimpleDB. Most of these are size and quantity restrictions. There is an underlying philosophy that small and quickly serviced units of work provide the greatest opportunity for load balancing and maintaining uniform service levels. AWS maintains a current listing of the service limitations within the latest online SimpleDB Developer Guide at the AWS website. At the time of this writing, the limits are as follows:

- Max storage per domain: 10GB
- Max attribute values per domain: 1 billion
- Initial max domains per account: 100
- Max attribute values per item: 256
- Max length of item name, attribute name, or value: 1024 bytes
- Max query execution time: 5 seconds
- Max query results: 2500
- Max query response size: 1MB
- Max comparisons per query: 20

These limits may seem restrictive when compared to the unlimited nature of data sizes you can store in other database offerings. However, there are two things to keep in mind about these limits. First, SimpleDB is not a general-purpose data store suitable for everything. It is specifically designed for storing small chunks of data. For larger data objects that you want to store in the cloud, you are advised to use Amazon S3. Secondly, consider the steps that need to be taken with a relational database at higher loads when performance begins to degrade. Typical recommendations often include offloading processing from the database, reducing long-running queries, and applying selective de-normalization of the data. These limits are what help enable efficient and automatic background replication and high concurrency and availability. Some of these limits can be worked around to a degree, but no workarounds exist for you to make SimpleDB universally appropriate for all data storage needs.

Abandoning the Relational Model?

There have been many recent products and services offering data storage but rejecting the relational model. This trend has been dubbed by some as the NoSQL movement. There is a fair amount of enthusiasm both for and against this trend. A few of those in the "against" column argue that databases without schemas, type checking, normalization, and so on are throwing away 40 years of database progress. Likewise, some proponents are quick to dispense the hype about how a given NoSQL solution will solve your problems. The aim of this section is to present a case for the value of a service like SimpleDB that addresses legitimate criticism and avoids hype and exaggeration.

A Database Without a Schema

One of the primary areas of contention around SimpleDB and other NoSQL solutions centers on the lack of a database schema. Database schemas turn out to be very important in the relational model. The formalism of predefining your data model into a schema provides a number of specific benefits, but it also imposes restrictions.

SimpleDB has no notion of a schema at all. Many of the structures defined in a typical database schema do not even exist in SimpleDB. This includes things such as stored procedures, triggers, relationships, and views. Other elements of a database schema like fields and types do exist in SimpleDB but are flexible and are not enforced on the server. Still other features, like indexes, require no formal definition because the SimpleDB service creates and manages them behind the scenes.

However, the lack of a schema requirement in SimpleDB does not prevent you from gaining the benefits of a schema. You can create your own schema for whatever portion of your data model that is appropriate. This allows you to cherry-pick the benefits that are helpful to your application without the unneeded restrictions.

One of the most important things you gain from codifying your data layout is a separation between it and the application. This is an enabling feature for tools and application plug-ins. Third-party tools can query your data, convert your data from one format to another, and analyze and report on your data based solely on the schema definition. The alternative is less attractive. Tools and extensions are more limited in what they can do without knowledge of the formats. For example, you cannot compute the sum of values in a numeric column if you do not know the format of that column. In the degenerate case, developers must search through your source code to infer data types.

In SimpleDB, many of the most common database features are not available. Query, however, is one important feature that is present and has some bearing on your data formatting. Because all the data you store in SimpleDB is variable length character data, you must apply padding to numeric data in order for queries to work properly. For example, if you want to store an attribute named "price" with a value of "269.94," you must first add leading zeros to make it "00000269.94." This is required because greater-than and less-than comparisons within SimpleDB compare each character from left to right. Padding with zeros allows you to line up the decimal point so the comparisons will be correct for all possible values of that attribute. Relational database products handle this for you behind the scenes when you declare a column type is a numeric type like int.

This is a case in SimpleDB where a schema is beneficial. The code that initially imports records into SimpleDB, the code that writes records as your app runs, and any code that uses a numeric attribute in a query all need to use the exact same format. Explicitly storing the schema externally is a much less error-prone approach than implicitly defining the format in duplicated code across various modules.

Another benefit of the predefined schema in the relational model is that it forces you to think through the data relationships and make unambiguous decisions about your data layout. Sometimes, however, the data is simple, there are no relationships, and creating a data model is overkill. Sometimes you may still be in the process of defining the data model. SimpleDB can be used as part of the prototyping process, enabling you to evolve your schema dynamically as issues surface that may not otherwise have become known so quickly.You may be migrating from a different database with an existing data model.The important thing to remember is that SimpleDB is simple by design. It can be useful in a variety of situations and does not prevent you from creating your own schema external to SimpleDB.

Areas Where Relational Databases Struggle

Relational databases have been around for some time.There are many robust and mature products available. Modern database products offer a multitude of features and a host of configuration options.

One area where difficulty arises is with database features that you do not need or that you should not use for a particular application. Applications that have simple data storage requirements do not benefit from the myriad of available options. In fact, it can be detrimental in a couple different ways. If you need to learn the intricacies of a particular database product before you can make good use of it, the time spent learning takes away from time you could have spent on your application. Knowledge of how database products work is good to have. It would be hard to argue that you wasted your time by learning it because that information could serve you well far into the future. Similarly, if there is a much simpler solution that meets your needs, you could choose that instead. If you had no immediate requirement to gain product specific database expertise, it would be hard to insist that you made the wrong choice. It is a tough sell to argue that the more time-consuming, yet educational, route is always better than the simple and direct route.This is a challenge faced by databases today, when the simple problems are not met with simple solutions.

Another pain point with relational databases is horizontal scaling. It is easy to scale a database vertically by beefing up your server because memory and disk drives are inexpensive. However, scaling a database across multiple servers can be extremely difficult. There is a whole spectrum of options available for horizontal scaling that includes basic master-slave replication as well as complicated sharding strategies.These solutions each require a different, and sometimes considerable, amount of expertise. Nevertheless, they all have one thing in common when compared to vertical scaling solutions. On top of the implementation difficulty, each additional server results in an additional increase in ongoing maintenance responsibility. Moreover, it is not merely the additional server maintenance of having more servers. I am referring to the actual database administration tasks of managing additional replicas, backups, and log shipping. It also includes the tasks of rolling out schema changes and new indexes to all servers in the cluster.

If you are in a situation where you want a simple database solution or you want horizontal scaling, SimpleDB is definitely a service to consider. However, you may need to be prepared to defend your decision.

Scalability Isn't Your Problem

Around every corner, you can find people who will challenge your efforts to scale horizontally. Beyond the cost and difficulty, there is a degree of resistance to products and services that seek to solve these problems.

The typical, and now clichéd, advice tends to be that scalability is not your problem, and trying to solve scalability at the outset is a case of premature optimization. This is followed by a discussion of how many daily page views a single high-performance database server can support. Finally, it ends by noting that it is really just a problem for when you reach the scale of Google or Amazon.

The premise of the argument is actually solid, although not applicable to all situations. The premise is that when you are building a site or service that nobody has heard of yet, you are more concerned about handling loads of people than about making the site remarkable. It is good advice for these situations. Moreover, it is especially timely considering that there is a small but religious segment of Internet commentators who eagerly chime, "X doesn't scale," where X is any alternative to the solution the commenter uses. Among programmers, there is a general preoccupation with performance optimization that seems somewhat out of balance.

The fact is that for many projects, scalability really is not your problem, but availability can be. Distributing your data store across servers from the outset is not a premature optimization when you can quantify the cost of down time. If a couple hours of downtime will have an impact on your business, then availability is something worth thinking about. For the IT department delivering a mission-critical application, availability is important. Even if only 20 users will use it during normal business hours, when it provides a competitive advantage, it is important to maintain availability through expected outages. When you have a product launch, and your credibility is at stake as much as your revenue, you are not putting the cart before the horse when you protect yourself against hardware failures.

There are many situations where availability is an important system quality. Look at how common it is for a multi-server web cluster to host one website. Before you can add a second web server, you must first solve a small set of known problems. User sessions have to be managed properly; load balancing has to be in place and routing around unresponsive servers. However, web server clusters are useful for more than high-traffic load handling. They are also beneficial because we know that hardware will fail, and we want to maintain service during the failure. We can add another web server because it is neither costly nor difficult, and it improves the availability. With the advent of systems designed to provide higher database availability that are not costly nor hard, availability becomes worth pursuing for less-critical projects.

Avoiding the SimpleDB Hype

There are many different application scenarios where SimpleDB is an interesting option. That said, some people have overstated the benefits of using SimpleDB specifically and hosted NoSQL databases in general. The reasoning seems to be that services running on the infrastructure of companies like Amazon, Google, or Microsoft will undoubtedly have nearly unlimited automatic scalability. Although there is nothing wrong with enthusiasm for products and services that you like, it is good to base that enthusiasm on reality.

Do not be fooled into thinking that any of these new databases is going to be a panacea. Make sure you educate yourself about the pros and cons of each solution as you evaluate it. The majority of services in this space have a free usage tier, and all the open-source alternatives are completely free to use. Take advantage of it, and try them out for yourself. We live in an amazing time in history where the quantity of information available at our fingertips is unprecedented. Access to web-based services and open-source projects is a huge opportunity. The tragedy is that in a time when it has never been easier to gain personal experience with new technology, all too often we are tempted to adopt the opinions of others instead of taking the time to form our own opinions. Do not believe the hype—find out for yourself.

Putting the DBA Out of Work

One of the stated goals of SimpleDB is allowing customers to outsource the time and effort associated with managing a web-scale database. Managing the database is traditionally the world of the DBA. Some people have assumed that advocating the use of SimpleDB amounts to advocating a world where the DBA diminishes in importance. However, this is not the case at all.

One of the things that have come about from the widespread popularity of EC2 has been a change in the role of system administrators. What we have found is that managing EC2 virtual instances is less work than managing a physical server instance. However, the result has not been a rash of system administrator firings. Instead, the result has been that system administrators are able to become more productive by managing larger numbers of servers than they otherwise could. The ease of acquisition and the low cost to acquire and release the computing power have led, in many cases, to a greater and more dynamic use of the servers. In other words, organizations are using more server instances because the various levels of the organization can handle it, from a cost, risk, and labor standpoint.

SimpleDB and its cohorts seem to facilitate a similar change but on a smaller scale. First, SimpleDB has less general applicability than EC2. It is a suitable solution for a much smaller set of problems. AWS fully advocates the use of existing relational database products. SimpleDB is an additional option, not a replacement. Moreover, SimpleDB finds good usage in some areas where a relational database might not normally be used, as in the case of storing web user session data. In addition, for those projects that choose to use SimpleDB instead of, or along with, a relational database, it does not mean that there is no role for the DBA. Some tasks remain similar to EC2, which can result in a greater capacity for IT departments to create solutions.

Dodging Copies of C.J. Date

There are database purists who wholeheartedly try to dissuade people from using any type of non-relational database on principle alone. Not only that, but they also go to great lengths to advocate the proper use of relational databases and lament the fact that no current database products correctly implement the relational model. Having found the one-true data storage paradigm, they believe that the relational model is "right" and is the only one that will last. The purists are not wrong in their appreciation for the relational model and for SQL. The relational model is the cornerstone of the database field, and more than that, an invaluable contribution to the world of computing. It is one of the two best things to come out of 1969. Invented by a mathematician and considered a branch of mathematics itself, there is a solid theoretical rigor that underlies its principles. Even though it is not a complete or finished branch, the work to date has been sound.

The world of mathematics and academic research is an interesting place. When you have spent large quantities of your life and career there, you are highly qualified to make authoritative comments on topics like correctness and provability. Nevertheless, being either a relational model expert or merely someone who holds them in high regard does not say anything about your ability to deliver value to users. It is clearly true that modeling your data "correctly" can provide measurable benefits and that making mistakes in your model can lead to certain classes of problems. However, you can still provide significant user value with a flawed model, and correctness is no guarantee of success.

It is like perfectly generated XHTML that always validates. It is like programming with a functional style (in any programming language) that lets you prove your programs are correct. It is like maintaining unit tests that provide 100% test coverage for every line of code you write. There is nothing inherently bad you can say about these things. In fact, there are plenty of good things to say about them. The problem is not a technical problem—it is a people problem. The problem is when people become hyper-focused on narrow technological aspects to the exclusion of the broader issues of the application's purpose.

The people conducting database research and the ones who take the time to help educate the computing industry deserve our respect. If you have a degree in computer science, chances are you studied C.J. Date's work in your database class. Among professional programmers, there is no good excuse for not knowing data and relational fundamentals. However, the person in the next row of cubicles who is only contributing condescending criticism to your project is no C.J. Date. In addition, the user with 50 times your stackoverflow.com reputation who ridicules the premise of your questions without providing useful suggestions is no E.F. Codd. Understanding the theory is of great importance. Knowing how to deliver value to your users is of greater importance. In the end, avoid vociferous ignorance and don't let anyone kick copies of C.J. Date in your face.

Other Pieces of the Puzzle

In the world of cloud computing, there are a growing number of companies and services from which to choose. Each service provider seeks to align its offerings with a broader strategy. With Amazon, that strategy includes providing very basic infrastructure building blocks for users to assemble customized solutions. AWS tries to get you to use more than one service offering by making the different services useful with each other and by offering fast and free data transfer between services in the same region. This section describes three other Amazon Web Services, along with some ways you might find them to be useful in conjunction with SimpleDB.

Adding Compute Power with Amazon EC2

AWS sells computing power by the hour via the Amazon Elastic Compute Cloud (Amazon EC2). This computing power takes the form of virtual server instances running on top of physical servers within Amazon data centers. These server instances come in varying amounts of processor horsepower and memory, depending on your needs and budget. What makes this compute cloud elastic is the fact that users can start up, and shut down, dozens of virtual instances at a moment's notice.

These general-purpose servers can fulfill the role of just about any server. Some of the popular choices include web server, database server, batch-processing server, and media server. The use of EC2 can result in a large reduction in ongoing infrastructure maintenance when compared to managing private in-house servers. Another big benefit is the elimination of up-front capital expenditures on hardware in favor of paying for only the compute power that is used.

The sweet spot between SimpleDB and EC2 comes for high-data bandwidth applications. For those apps that need fast access to high volumes of data in SimpleDB, EC2 is the platform of choice. The free same region data transfer can add up to a sizable cost savings for large data sets, but the biggest win comes from the consistently low latency. AWS does not guarantee any particular latency numbers but typically, round-tripping times are in the neighborhood of 2 to 7 milliseconds between EC2 instances and SimpleDB in the same region. These numbers are on par with the latencies others have reported between EC2 instances. For contrast, additional latencies of 50 to 200 milliseconds or more are common when using SimpleDB across the open Internet. When you need fast SimpleDB, EC2 has a lot to offer.

Storing Large Objects with Amazon S3

Amazon Simple Storage Service (Amazon S3) is a web service that enables you to store an unlimited number of files and charges you (low) fees for the actual storage space you use and the data transfer you use. As you might expect, data transfer between S3 and other Amazon Web Services is fast and free. S3 is easy to understand, easy to use, and has a multitude of great uses. You can keep the files you store in S3 private, but you can also make them publicly available from the web. Many websites are using S3 as a media-hosting service to reduce the load on web servers.

EC2 virtual machine images are stored and loaded from S3. EC2 copies storage volumes to and loads storage volumes from S3. The Amazon CloudFront content delivery network can serve frequently accessed web files in S3. The Amazon Elastic MapReduce service runs MapReduce jobs stored in S3. Publicly visible files in S3 can be served up via the BitTorrent peer-to-peer protocol. The list of uses goes on and on.... S3 is really a common denominator cloud service.

SimpleDB users can also find good uses for S3. Because of the high speed within the Amazon cloud, S3 is an obvious storage location choice for SimpleDB import and export data. It is also a solid location to place SimpleDB backup files.

Queuing Up Tasks with Amazon SQS

Amazon Simple Queue Service (Amazon SQS) is a web service that reliably stores messages between distributed computers. Placing a robust queue between the computers allows them to work independently. It also opens the door to dynamically scaling the number of machines that push messages and the number that retrieve messages.

Although there is no direct connection between SQS and SimpleDB, SQS does have some complementary features that can be useful in SimpleDB-based applications. The semantics of reliable messaging can make it easier to coordinate multiple concurrent clients than when using SimpleDB alone. In cases where there are multiple SimpleDB clients, you can coordinate clients using a reliable SQS queue. For example, you might have multiple servers that are encoding video files and storing information about those videos in SimpleDB. SimpleDB makes a great place to store that data, but it could be cumbersome for use in telling each server which file to process next. The reliable message delivery of SQS would be much more appropriate for that task.

Comparing SimpleDB to Other Products and Services

Numerous new types of products and services are now available or will soon be available in the database/data service space. Some of these are similar to SimpleDB, and others are tangential. A few of them are listed here, along with a brief description and comparison to SimpleDB.

Windows Azure Platform

The Windows Azure Platform is Microsoft's entry into the cloud-computing fray. Azure defines a raft of service offerings that includes virtual computing, cloud storage, and reliable message queuing. Most of these services are counterparts to Amazon services. At the time of this writing, the Azure services are available as a Community Technology Preview. To date, Microsoft has been struggling to gain its footing in the cloud services arena.

There have been numerous, somewhat confusing, changes in product direction and product naming. Although Microsoft's cloud platform has been lagging behind AWS a bit, it seems that customer feedback is driving the recent Azure changes. There is every reason to suspect that once Azure becomes generally available, it will be a solid alternative to AWS.

Among the services falling under the Azure umbrella, there is one (currently) named Windows Azure Table. Azure Table is a distributed key-value store with explicit support for partitioning across storage nodes. It is designed for scalability and is in many ways similar to SimpleDB. The following is a list of similarities between Azure Table and SimpleDB:

- All access to the service is in the form of web requests. As a result, any programming language can be used.
- Requests are authenticated with encrypted signatures.
- Consistency is loosened to some degree.
- Unique primary keys are required for each data entity.
- Data within each entity is stored as a set of properties, each of which is a name-value pair.
- There is a limit of 256 properties per entity.
- A flexible schema allows different entities to have different properties.
- There is a limit on how much data can be stored in each entity.
- The number of entities you can get back from a query is limited and a query continuation token must be used to get the next page of results.
- Service versioning is in place so older versions of the service API can still be used after new versions are rolled out.
- Scalability is achieved through the horizontal partitioning of data.

There are also differences between the services, as listed here:

- Azure Table uses a composite key comprised of a partition key followed by a row key, whereas SimpleDB uses a single item name.
- Azure Table keeps all data with the same partition key on a single storage node. Entities with different partition keys may be automatically spread across hundreds of storage nodes to achieve scalability. With SimpleDB, items must be explicitly placed into multiple domains to get horizontal scaling.
- The only index in Azure Table is based on the composite key. Any properties you want to query or sort must be included as part of the partition key or row key. In contrast, SimpleDB creates an index for each attribute name, and a SQL-like query language allows query and sort on any attribute.
- To resolve conflicts resulting from concurrent updates with Azure Table, you have a choice of either last-write-wins or resolving on the client. With SimpleDB, last-write-wins is the only option.

- Transactions are supported in Azure Table at the entity level as well as for entity groups with the same partition key. SimpleDB applies updates atomically only within the scope of a single item.

Windows Azure Table overall is very SimpleDB-like, with some significant differences in the scalability approach. Neither service has reached maturity yet, so we may still see enhancements aimed at easing the transition from relational databases.

It is worth noting that Microsoft also has another database service in the Windows Azure fold. Microsoft SQL Azure is a cloud database service with full replication across physical servers, transparent automated backups, and support for the full relational data model. This technology is based on SQL Server, and it includes support for T-SQL, stored procedures, views, and indexes. This service is intended to enable direct porting of existing SQL-based applications to the Microsoft cloud.

Google App Engine

App Engine is a service offered by Google that lets you run web applications, written in Java or Python, on Google's infrastructure. As an application-hosting platform, App Engine includes many non-database functions, but the App Engine data store has similarities to SimpleDB. The non-database functions include a number of different services, all of which are available via API calls. The APIs include service calls to Memcached, email, XMPP, and URL fetching.

App Engine includes an API for data storage based on Google Big Table and in some ways is comparable to SimpleDB. Although Big Table is not directly accessible to App Engine applications, there is support in the data store API for a number of features not available in SimpleDB. These features include data relations, object mapping, transactions, and a user-defined index for each query.

App Engine also has a number of restrictions, some of which are similar to SimpleDB restrictions, like query run time. By default, the App Engine data store is strongly consistent. Once a transaction commits, all subsequent reads will reflect the changes in that transaction. It also means that if the primary storage node you are using goes down, App Engine will fail any update attempts you make until a suitable replacement takes over. To alleviate this issue, App Engine has recently added support for the same type of eventual consistency that SimpleDB has had all along. This move in the direction of SimpleDB gives App Engine apps the same ability as SimpleDB apps to run with strong consistency with option to fall back on eventual consistency to continue with a degraded level of service.

Apache CouchDB

Apache CouchDB is a document database where a self-contained document with metadata is the basic unit of data. CouchDB documents, like SimpleDB items, consist of a group of named fields. Each document has a unique ID in the same way that each SimpleDB item has a unique item name. CouchDB does not use a schema to define or validate documents. Different types of documents can be stored in the same database. For querying, CouchDB uses a system of JavaScript views and map-reduce. The loosely structured data in CouchDB documents is similar to SimpleDB data but does not place limits on the amount of data you can store in each document or on the size of the data fields.

CouchDB is an open-source product that you install and manage yourself. It allows distributed replication among peer servers and has full support for robust clustering. CouchDB was designed from the start to handle high levels of concurrency and to maintain high levels of availability. It seeks to solve many of the same problems as SimpleDB, but from the standpoint of an open-source product offering rather than a pay-as-you-go service.

Dynamo-Like Products

Amazon Dynamo is a data store used internally within Amazon that is not available to the public. Amazon has published information about Dynamo that includes design goals, runtime characteristics, and examples of how it is used. From the published information, we know that SimpleDB has some things in common with Dynamo, most notably the eventual consistency.

Since the publication of Dynamo information, a number of distributed key-value stores have been developed that are in the same vein as Dynamo. Three open-source products that fit into this category are Project Voldemort, Dynomite, and Cassandra. Each of these projects takes a different approach to the technology, but when you compare them to SimpleDB, they generally fall into the same category. They give you a chance to have highly available key-value access distributed across machines. You get more control over the servers and the implementation that comes with the maintenance cost of managing the setup and the machines. If you are looking for something in this class of data storage, SimpleDB is a likely touch-free hosted option, and these projects are hands-on self-hosted alternatives.

Compelling Use Cases for SimpleDB

SimpleDB is not a replacement for relational databases. You need to give careful consideration to the type of data storage solution that is appropriate for a given application. This section includes a discussion of some of the use cases that match up well with SimpleDB.

Web Services for Connected Systems

IT departments in the enterprise are tasked with delivering business value and support in an efficient way. In recent years, there has been movement toward both service orientation and cloud computing. One of the driving forces behind service orientation is a desire to make more effective use of existing applications. Simple Object Access Protocol (SOAP) has emerged as an important standard for message passing between these connected systems as a means of enabling forward compatibility. For new services deployed in the cloud, SimpleDB is a compelling data storage option.

Data transfer between EC2 instances and the SimpleDB endpoint in the same region is fast and free. The consistent speed and high availability of SimpleDB are helpful when defining a Service Level Agreement (SLA) between IT and business units. All this meshes with the ability of EC2 to scale out additional instances on demand.

Low-Usage Application

There are applications in the enterprise and on the open web that do not see a consistent heavy load. They can be low usage in general with periodic or seasonal spikes—for instance, at the end of the month or during the holidays. Sometimes there are few users at all times by design or simply by lack of popularity.

For these types of applications, it can be difficult to justify an entire database server for the one application. The typical answer in organizations with sufficient infrastructure is to host multiple databases on the same server. This can work well but may not be an option for small organizations or for individuals. Shared database hosting is available from hosting companies, but service levels are notoriously unpredictable. With SimpleDB, low-usage applications can run within the free tier of service while maintaining the ability to scale up to large request volumes when necessary. This can be an attractive option even when database-sharing options are available.

Clustered Databases Without the Time Sink

Clustering databases for scalability or for availability is no easy task. If you already have the heavy data access load or if you have the quantifiable need for uptime, it is obviously a task worth taking on. Moreover, if you already have the expertise to deploy and manage clusters of replicated databases, SimpleDB may not be something you need. However, if you do have the experience, you know many other things as well: you know the cost to roll the clusters into production, to roll out schema updates, and to handle outages. This information can actually make it easier to decide whether new applications will provide enough revenue or business value to merit the time and cost. You also have a great knowledge base to make comparisons between in-house solutions and SimpleDB for the features it provides.

You may have a real need for scalability or uptime but not the expertise. In this case, SimpleDB can enable you to outsource the potentially expensive ongoing database maintenance costs.

Dynamic Data Application

Rigid and highly structured data models serve as the foundation of many applications, while others need to be more dynamic. It is becoming much more important for new applications to include some sort of social component than it was in the past. Along with these social aspects, there are requirements to support various types of user input and customization, like tagging, voting, and sharing. Many types of social applications require community building, and can benefit from a platform, which allows data to be stored in new ways, without breaking the old data. Customer-facing applications, even those without a social component, need to be attentive to user feedback.

Whether it is dynamic data coming from users or dynamic changes made in response to user feedback, a flexible data store can enable faster innovation.

Amazon S3 Content Search

Amazon S3 has become a popular solution for storing web-accessible media files. Applications that deal with audio, video, or images can access the media files from EC2 with no transfer costs and allow end users to download or stream them on a large scale without needing to handle the additional load. When there are a large number of files in S3, and there is a need to search the content along various attributes, SimpleDB can be an excellent solution.

It is easy to store attributes in SimpleDB, along with pointers to where the media is stored in S3. SimpleDB creates an index for every attribute for quick searching. Different file types can have different attributes in the same SimpleDB domain. New file types or new attributes on existing file types can be added at any time without requiring existing records to be updated.

Empowering the Power Users

For a long time, databases have been just beyond the edge of what highly technical users can effectively reach. Many business analysts, managers, and information workers have technical aptitude but not the skills of a developer or DBA. These power users make use of tools like spreadsheet software and desktop databases to solve problems. Unfortunately, these tools work best on a single workstation, and attempts at sharing or concurrent use frequently cause difficulty and frustration; enterprise-capable database software requires a level of expertise and time commitment beyond what these users are willing to spend.

The flexibility and scalability of SimpleDB can be a great boon to a new class of applications designed for power users. SimpleDB itself still requires programming on the client and is not itself directly usable by power users. However, the ability to store data directly without a predefined schema and create queries is an enabling feature. For applications that seek to empower the power users, by creating simple, open-ended applications with dynamic capabilities, SimpleDB can make a great back end.

Existing AWS Customers

This chapter pointed out earlier the benefits of using EC2 for high-bandwidth applications. However, if you are already using one or more of the Amazon Web Services, SimpleDB can be a strong candidate for queryable data storage across a wide range of applications. Of course, running a relational database on an EC2 instance is also a viable and popular choice. Moreover, you would do well to consider both options. SimpleDB requires you to make certain trade-offs, but if the choices provide a net benefit to your application, you will have gained some great features from AWS that are difficult and time consuming to develop on your own.

Summary

Amazon SimpleDB is a web service that enables you to store semi-structured data within Amazon's data centers. The service provides automatic, geographically diverse data replication and internal routing around failed storage nodes. It offers high availability and enables horizontal scalability. The service allows you to offload hardware maintenance and database management tasks.

You can use SimpleDB as a distributed key-value store using the `GetAttributes`, `PutAttributes`, and `DeleteAttributes` API calls. You also have the option to query for your data along any of its attributes using the Select API call. SimpleDB is not a relational database, so there are no joins, foreign keys, schema definitions, or relational constraints that you can specify. SimpleDB also has limited support for transactions, and updates propagate between replicas in the background. SimpleDB supports strong consistency, where read operations immediately reflect the results of all completed and eventual consistency, where storage nodes are updated asynchronously in the background.

The normal window of time for all storage nodes to reach consistency in the background is typically small. During a server or network failure, consistency may not be reached for longer periods of time, but eventually all updates will propagate. SimpleDB is best used by applications able to deal with eventual consistency and benefit from the ability to remain available in the midst of a failure.

Index