

The Design of the Force.com Multitenant Internet Application Development Platform

Craig D Weissman

CTO, Salesforce.com
San Francisco, CA 94105

cweissman@salesforce.com

Steve Bobrowski

Technical Marketing Consultant, Salesforce.com

sbobrows001@gmail.com

ABSTRACT

Force.com is the preeminent on-demand application development platform in use today, supporting some 55,000+ organizations. Individual enterprises and commercial software-as-a-service (SaaS) vendors trust the platform to deliver robust, reliable, Internet-scale applications. To meet the extreme demands of its large user population, Force.com's foundation is a metadata-driven software architecture that enables multitenant applications.

The focus of this paper is multitenancy, a fundamental design approach that can dramatically improve SaaS application management. This paper defines multitenancy, explains its benefits, and demonstrates why metadata-driven architectures are the premier choice for implementing multitenancy.

Categories and Subject Descriptors

H.2.1: Database management – *Logical design*.

D.2.11: Software Engineering – *Software architecture, Data abstractions*.

General Terms

Algorithms, Performance, Design.

Keywords

Multi-tenancy, Query Optimization, Flex Schema, Domain Specific Language, Object-relational Mapping.

1. INTRODUCTION

History has shown that every so often, incremental advances in technology and changes in business models create major paradigm shifts in the way software applications are designed, built, and delivered to end users. Today, reliable broadband Internet access, service-oriented architectures (SOAs), and the cost inefficiencies of managing dedicated on-premise applications are driving a transition toward the delivery of managed, shared, Web-based services called software as a service (SaaS).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD '09, June 29–July 2, 2009, Providence, Rhode Island, USA.

Copyright 2009 ACM 978-1-60558-551-2/09/06...\$5.00.

2. MULTITENANT APPLICATIONS

To decrease the cost of delivering the same application to many different sets of users, an increasing number of applications are multitenant. Whereas a traditional single-tenant application requires a dedicated set of resources to fulfill the needs of just one organization, a multitenant application can satisfy the needs of multiple tenants (companies or departments within a company, etc.) using the hardware resources and staff needed to manage just a single software instance (Figure 1).

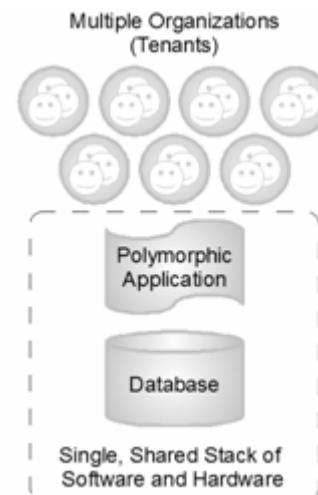


Figure 1. A multitenant application shares a single stack of resources to satisfy the needs of multiple organizations

Tenants using a multitenant service operate in virtual isolation from one another. Organizations can use and customize an application as though they each have a separate instance, yet their data and customizations remain secure and insulated from the activity of all other tenants. The single application instance effectively morphs at runtime for any particular tenant at any given time.

Multitenancy is an architectural approach that pays dividends to both application providers and users. Operating just one application instance for multiple organizations yields tremendous economy of scale for the provider. Only one set of hardware resources is necessary to meet the needs of all users, a relatively small, experienced administrative staff can efficiently manage only one stack of software and hardware, and developers can build and support a single code base on just one platform (operating system, database, etc.) rather than many. The economics afforded by multitenancy allow the application provider to, in turn, offer the service at a lower cost to customers.

Some interesting side benefits of multitenancy are improved quality, user satisfaction, and customer retention. Unlike single-tenant applications, which are isolated silos deployed outside the reach of the application provider, a multitenant application is one large community that is hosted by the provider itself. This design shift lets the provider gather operational information from the collective user population (which queries respond slowly, what errors happen, etc.) and make frequent, incremental improvements to the service that benefit the entire user community at once.

Two additional benefits of a multitenant platform-based approach are collaboration and integration. Because all users run all applications in one space, it is easy to allow any user of any application varied access to specific sets of data. This capability greatly simplifies the effort necessary to integrate related applications and the data they manage.

3. COMPARING CLOUD COMPUTING AND PAAS

Raw computing clouds are machine-centric services that provide on-demand infrastructure as a service (IaaS) for the deployment of applications. Such clouds provide little more than the computing power and storage capacity needed to execute virtual servers that comprise an application. Some SaaS vendors looking for a quick go-to-market strategy avoid the challenges of developing a true multitenant solution and deliver single-tenant instances via IaaS.

Platform as a service (PaaS), such as Force.com, is an application-centric approach that abstracts the concept of servers altogether. PaaS lets developers focus on core application development from day one and to deploy an application with the push of a button. The provider never needs to worry about multitenancy, high-availability, load-balancing, scalability, system backups, patches and security, and other infrastructure-related concerns.

4. METADATA-DRIVEN ARCHITECTURES

Multitenancy is practical only when it can support applications that are reliable, customizable, upgradeable, secure, and fast. But how can a multitenant application allow each tenant to create custom extensions to standard data objects and entirely new custom data objects? How will tenant-specific data be kept secure in a shared database? How can one tenant customize the application's interface and business logic without affecting the functionality or availability of the application for all other tenants? How can the application's code base be patched or upgraded without breaking tenant-specific customizations?

It's difficult to create a statically compiled application executable that can meet these and other unique challenges of multitenancy. Inherently, a multitenant application must be dynamic in nature, or polymorphic, to fulfill the individual expectations of various tenants and their users.

For these reasons, multitenant application designs have evolved to use a runtime engine that generates application components from metadata—data about the application itself. In a well-defined metadata-driven architecture (Figure 2), there is a clear separation of the compiled runtime engine (kernel), application data, the metadata that describes the base functionality of an application, and the metadata that corresponds to each tenant's data and customizations. These distinct boundaries make it possible to independently update the system kernel, modify the core application, or customize tenant-

specific components, with virtually no risk of one affecting the others.

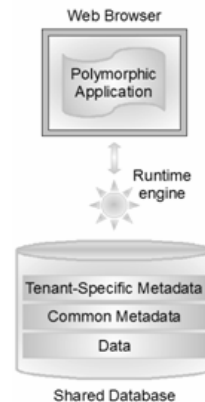


Figure 2. Well-defined metadata-driven architecture

5. ARCHITECTURE OVERVIEW

Force.com's optimized metadata-driven architecture delivers extraordinary performance, scalability, and customization for on-demand, multitenant applications.

In Force.com, everything exposed to developers and application users is internally represented as metadata. Forms, reports, work flows, user access privileges, tenant-specific customizations and business logic, even the definitions of underlying data tables and indexes, are all abstract constructs that exist merely as metadata in Force.com's Universal Data Dictionary (UDD). For example, when a developer is building a new custom application and defines a custom table, lays out a form, or writes some procedural code, Force.com does not create an "actual" table in a database or compile any code. Instead, Force.com simply stores metadata that the platform's engine can use to generate the "virtual" application components at runtime. When someone wants to modify or customize something about the application, all that's required is a simple non-blocking update to the corresponding metadata.

Because metadata is a key ingredient of Force.com applications, the platform's runtime engine must optimize access to metadata; otherwise, frequent metadata access would prevent the platform from scaling. With this potential bottleneck in mind, Force.com uses metadata caches to maintain the most recently used metadata in memory, avoid performance sapping disk I/O and code recompilations, and improve application response times.

Force.com stores the application data for all virtual tables in a few large database tables that serve as heap storage. The platform's engine then materializes virtual table data at runtime by considering corresponding metadata.

To optimize access to data in the system's large tables, Force.com's engine relies on a set of specialized pivot tables that maintain denormalized data for various purposes such as indexing, uniqueness, relationships, etc.

Force.com's data processing engine helps streamline the overhead of large data loads and online transaction processing applications by transparently performing data modification operations in bulk. The engine has built-in fault recovery mechanisms that automatically retry bulk save operations after factoring out records that cause errors.

To further hone application response times, the platform employs an external search service that optimizes full-text indexing and searches. As applications update data, the search service’s background processes asynchronously update tenant- and user-specific indexes in near real time. This separation of duties between the application engine and the search service lets platform applications efficiently process transactions without the overhead of text index updates, and at the same time quickly provide users with accurate search results.

As Force.com’s runtime application generator dynamically builds applications in response to specific user requests, the engine relies heavily on its “multitenant-aware” query optimizer to execute internal operations as efficiently as possible. The query optimizer considers which user is executing a given application function, and then, using related tenant-specific metadata maintained in the UDD along with internal system pivot tables, builds and executes data access operations as optimized database queries.

Now that you have a general idea of the key architecture components that make up the underlying mechanisms of Force.com, the following sections explain the structure and purpose of various internal system elements in more detail.

6. DATA DEFINITION AND STORAGE

Rather than attempting to manage a vast, ever-changing set of actual database structures on behalf of each application and tenant, the Force.com storage model manages “virtual” database structures using a set of metadata, data, and pivot tables, as illustrated in Figure 3.

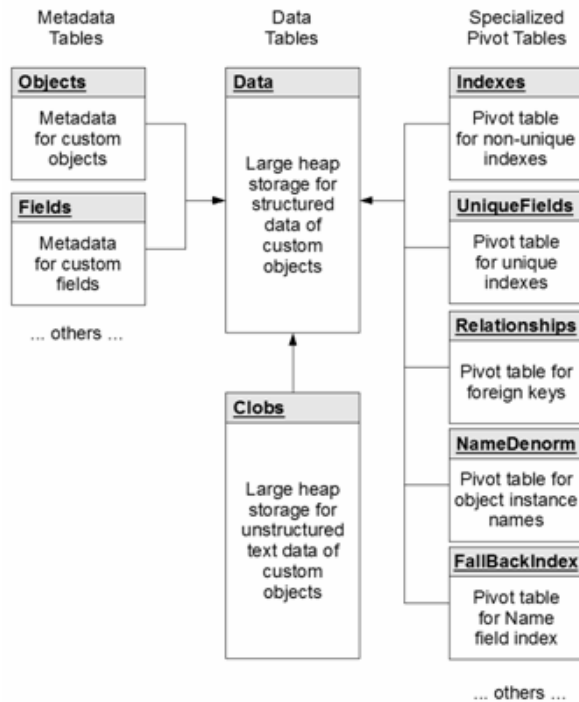


Figure 3. Force.com’s data definition and storage model

When organizations create custom application objects (i.e., custom tables), the UDD keeps track of metadata concerning the objects, their fields, relationships, and other object definition characteristics.

Meanwhile, a few large database tables store the structured and unstructured data for all virtual tables, and a set of related, specialized pivot tables maintain denormalized data that makes the combined data set extremely functional.

6.1 The Objects Metadata Table

The Objects metadata table stores information about the custom objects (a.k.a. tables or entities) that an organization defines for an application, including a unique identifier for an object (ObjID), the organization (OrgID) that owns the object, and the name given to the object (ObjName).

6.2 The Fields Metadata Table

The Fields metadata table stores information about the custom fields (a.k.a. columns or attributes) that an organization defines for custom objects, including a unique identifier for a field (FieldID), the organization (OrgID) that owns the encompassing object, the object that contains the field (ObjID), the name of the field (FieldName), the field’s datatype, a Boolean value to indicate if the field requires indexing (IsIndexed), and the position of the field in the object relative to other fields (FieldNum).

6.3 The Data Table

The Data table stores the application-accessible data that maps to all custom objects and their fields, as defined by metadata in Objects and Fields. Each row includes identifying fields such as a global unique identifier (GUID), the organization that owns the row (OrgID), and the encompassing object identifier (ObjID). Each row in the Data table also has a Name field that stores a “natural name” for corresponding object instances; for example, an Account object might use “Account Name,” a Case object might use “Case Number,” etc. The Value0 ... Value500 columns store application data that maps to the objects and fields declared in the Objects and Fields tables, respectively; all “flex” columns use a variable-length string datatype so they can store any structured type of application data (strings, numbers, dates, etc.).

Custom fields can use any one of a number of standard structured datatypes such as text, number, date, and date/time as well as special use structured datatypes such as picklist (enumerated field), autonumber, formula, master-detail relationship (foreign key), checkbox (Boolean), email, URL, and others. Custom fields can also be required and have custom validation rules (for example, one field must be greater than another field), both of which are enforced by the platform’s application server.

When an organization declares or modifies a custom application object, Force.com manages a row of metadata in the Objects table that defines the object. Likewise, for each custom field, Force.com manages a row in the Fields table, including metadata that maps the field to a specific flex column in the Data table for the storage of corresponding field data. Because Force.com manages object and field definitions as metadata rather than actual database structures, the platform can tolerate multitenant application schema maintenance activities without blocking the concurrent activity of other tenants and users.

No two fields of the same object can map to the same flex column (slot) in the Data table for storage; however, a single flex column can manage the information of multiple fields, as long as each field stems from a different object.

| Data | | | | |
|---------|-------|-------|-----|----------|
| GUID | OrgID | ObjID | ... | Val0 |
| a01...1 | org1 | a01 | ... | Up |
| a01...2 | org1 | a01 | ... | Flat |
| a02...1 | org1 | a02 | ... | 20080129 |
| a02...2 | org1 | a02 | ... | 20080214 |
| a03...1 | org1 | a03 | ... | 41.23 |
| a03...2 | org1 | a03 | ... | -10.3 |

Figure 4. Example of single flex column

As the simplified representation of the Data table in Figure 4 shows, flex columns are of a universal datatype (variable-length string), which permits Force.com to share a single flex column among multiple fields that use various structured datatypes (strings, numbers, dates, etc.).

Force.com stores all flex column data using a canonical format and uses underlying database system datatype-conversion functions (e.g., TO_NUMBER, TO_DATE, TO_CHAR), as necessary, when applications read and write data to flex columns.

6.4 The CLOBs Table

Force.com supports the declaration of fields as character large objects (CLOBs) to permit the storage of long text fields up to 32,000 characters. For each row in the Data table that has a CLOB, Force.com stores the CLOB out-of-line in a pivot table called CLOBs, which the system can join with corresponding rows in the Data table as necessary.

6.5 The Indexes Pivot Table

Traditional database systems rely on indexes to quickly locate specific rows in a database table that have fields matching a specific condition. However, it is not practical to create native database indexes for the flex columns of the Data table because Force.com is likely using a single flex column to store the data of many fields that have varying structured datatypes. Instead, Force.com manages an index of the Data table by synchronously copying field data marked for indexing to an appropriate column in a pivot table called Indexes.

The Indexes table contains strongly typed, indexed columns such as StringValue, NumValue, and DateValue that Force.com uses to locate field data of the corresponding datatype. For example, Force.com would copy a string value in a Data table flex column to the StringValue field in Indexes, a date value to the DateValue field, etc. The underlying indexes of the Indexes table are standard non-unique database indexes. When an internal system query includes a search parameter that references a structured field in a custom object, the platform's query optimizer uses the Indexes table to help optimize associated data access operations.

Note: Force.com can handle searches across multiple languages because the platform's application servers use a case-folding algorithm that converts string values to a universal, case-insensitive format. The StringValue column of the Indexes table stores string values in this format. At runtime, the query optimizer automatically builds data access operations so that the optimized SQL statement filters on the corresponding case-folded StringValue that corresponds to the literal provided in the search request.

6.6 The UniqueFields Pivot Table

Force.com lets an organization indicate when a field in an object must contain unique values (case-sensitive or case-insensitive).

Considering the arrangement of the Data table and shared usage of the Value columns for custom field data, it is not practical to create unique database indexes for the table.

To support uniqueness for custom fields, Force.com uses the pivot table called UniqueFields; this table is very similar to the Indexes pivot table except that the UniqueFields table's underlying database indexes enforce uniqueness. When an application attempts to insert a duplicate value into a field that requires uniqueness, or an administrator attempts to enforce uniqueness on an existing field that contains duplicate values, Force.com relays an appropriate error message to the application.

6.7 The Relationships Pivot Table

Force.com provides "relationship" datatypes that an organization can use to declare relationships (referential integrity) among application objects. When an organization declares an object's field with a relationship type, the platform maps the field to a Value field in the Data table, and then uses this field to store the ObjID of a related object.

To optimize join operations, Force.com maintains a pivot table called Relationships. The Relationships index table has two underlying database unique composite indexes that allow for efficient object traversals in either direction, as necessary.

6.8 Additional Pivot Tables

The following pivot tables also provide important functionality to Force.com:

- FallbackIndex Table—Records the Name of all objects, allowing the application server to perform a fall-back search on the Name field and return reasonable search results in the event the platform's external search engine becomes overloaded.
- NameDenorm Table—Stores the ObjID and Name of each object instance, which allows Force.com to execute a simple query that retrieves the Name of each referenced object instance for display as part of a hyperlink.
- History Tracking Table—Stores information about all field changes, which allows Force.com to provide history tracking used in audit trails.

6.9 Partitioning of Data and Metadata

All Force.com data, metadata, and pivot table structures, including underlying database indexes, are physically partitioned by OrgID (by tenant) using native database partitioning mechanisms. Data partitioning is a proven technique that database systems provide to physically divide large logical data structures into smaller, more manageable pieces. Partitioning can also help to improve the performance, scalability, and availability of a large database system such as a multitenant environment. For example, by definition, every Force.com application query targets a specific tenant's information, so the query optimizer need only consider accessing data partitions that contain a tenant's data rather than an entire table or index—this common optimization is sometimes referred to as "partition pruning."

7. APPLICATION DEVELOPMENT, LOGIC, AND PROCESSING

Force.com supports two different ways to create custom applications and their individual components: declaratively by using the native platform application framework and programmatically by

using application programming interfaces (APIs). The following sections explain more about each approach and related application development topics.

7.1 The Application Framework

Developers can declaratively build custom Force.com applications using the Force.com application framework. The platform's native point-and-click interface supports all facets of the application development process, including the creation of an application's data model (custom objects and their fields, relationships, etc.), security and sharing model (users, organization hierarchies, profiles, etc.), user interface (screen layouts, data entry forms, reports, etc.), as well as logic and work flow.

Force.com application framework user interfaces are easy to build because there's no coding involved. Behind the scenes, they support all the usual data access operations, including queries, inserts, updates, and deletes.

Force.com's native integrated development environment (IDE) provides easy access to many built-in platform features that make it easy to implement common application functionality without writing complicated and error-prone code. These features include:

- Declarative workflows—Predefined actions triggered by the insert or update of an object instance (row). A workflow can trigger a task, email alert, update a data field, or send a message.
- Encrypted/masked fields—Text fields that developers configure to encrypt the corresponding data and use input masks to hide screen information.
- Validation rules—Rules that enforce a domain integrity rule without any programming.
- Formula fields—A declarative feature of the Force.com application framework that makes it easy to add a calculated field to an object.
- Roll-up summary fields—A cross-object field that makes it easy to aggregate child field information in a parent object.

7.2 Metadata and Web Services APIs

Force.com also provides programmatic APIs for building applications. These APIs are compatible with SOAP-based development environments, including Visual Studio .NET (C#) and Apache Axis (Java and C++).

The Force.com Metadata API is useful for managing application components—to create and modify the metadata that corresponds to custom object definitions, page layouts, work flows, etc. To create, retrieve, update, or delete object instances (rows of data), applications can use the Force.com Web Services API.

To access the Force.com Web service, developers first download a Web Service Description Language (WSDL) file. The development platform then uses the WSDL file to generate an API to access the organization's corresponding Force.com Web service (data model).

There are two types of Force.com WSDL files. An Enterprise WSDL file is for developers who are building organization-specific applications. An Enterprise WSDL file is a strongly typed representation of an organization's data model. It provides information about the organization's schema, data types, and fields to the development environment, allowing for a tighter integration between it and the Force.com Web service. An Enterprise WSDL changes if custom fields or custom objects are added to, renamed, or

removed from an organization's application schema. In contrast, a Partner WSDL file is for salesforce.com partners that are developing client applications for multiple organizations. As a loosely typed representation of the Force.com object model, a Partner WSDL provides an API that is useful for accessing data within any organization.

7.3 Bulk Processing with API Calls

Transaction-intensive applications generate less overhead and perform much better when they combine and execute repetitive operations in bulk. For example, contrast two ways an application can load many new instances of an object. An inefficient approach is to use a routine with loop that inserts individual object instances, making one API call for each insert operation. A much more efficient approach is to create an array of object instances and have the routine insert all of them with a single API call.

Applicable Force.com Web Services API calls such as create(), update(), and delete() support bulk operations. For maximum efficiency, the platform implicitly bulk-processes all internal steps related to an explicit bulk operation, as illustrated in Figure 5.

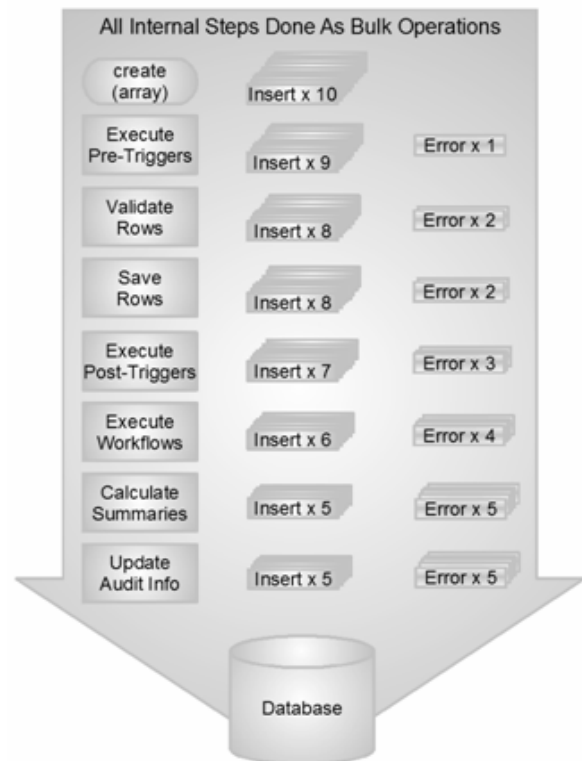


Figure 5. Force.com's bulk processing engine

Figure 5 also shows how Force.com's bulk processing engine can account for isolated faults encountered during any step along the way. When a bulk operation starts in partial save mode, the engine identifies a known start state and then attempts to execute each step in the process (bulk validate field data, bulk fire pre-triggers, bulk save records, etc.). If the engine detects errors during any step, the engine rolls back offending operations and all side effects, removes the rows that are responsible for the faults, and continues, attempting to bulk process the remaining subset of rows. This process iterates through each stage of the process until the engine

can commit a subset of rows without any errors. The application can examine a return object to identify which rows failed and what exceptions they raised.

Note: At the discretion of the application, an all-or-nothing mode is also available for bulk operations. Also, the execution of triggers during a bulk operation is subject to internal governors that restrict the amount of work.

7.4 Deletes, Undeletes, and the Recycle Bin

When someone deletes an individual object instance (record) from a custom object, Force.com simply marks the object instance for deletion by modifying the object instance's `IsDeleted` field (in the `Data` table). This effectively places the object in what is known as the platform's Recycle Bin. Force.com lets users view and restore selected object instances from the Recycle Bin for up to 30 days before permanently removing them from the internal `Data` table. The platform limits the total number of records it maintains for an organization based on the total number of user licenses for the organization.

When someone deletes a parent record involved in a master-detail relationship, Force.com automatically deletes all related child records, provided that doing so would not break any referential integrity rules in place. For example, when a user deletes a `SalesOrder`, Force.com automatically cascades the delete to dependent `LineItems`. Should someone subsequently restore a parent record from the Recycle Bin, the platform automatically restores all child object instances as well.

In contrast, when someone deletes a referenced parent record involved in a lookup relationship, Force.com automatically sets all dependent keys to null. If someone subsequently restores the parent record, Force.com automatically restores the previously nulled lookup relationships except for the relationships that were reassigned between the delete and restore operations.

The Recycle Bin also stores dropped fields and their data until an organization permanently deletes them or 45 days has elapsed, whichever happens first. Until that time, the entire field and all its data is available for restoration.

7.5 Data Definition Processing

Certain types of modifications to the definition of an object require more than simple UDD metadata updates. In such cases, Force.com uses efficient mechanisms that help reduce the overall performance impact on the platform's multitenant applications.

For example, consider what happens behind the scenes when someone modifies a column's datatype from picklist to text. Force.com first allocates a new slot for the column's data, bulk copies the picklist labels associated with current values, and then updates the column's metadata so that it points to the new slot. While all of this happens, access to data is normal and applications continue to function without any noticeable impact.

As another example, consider what happens when someone adds a roll-up summary field to a table. In this case, Force.com asynchronously calculates initial summaries in the background using an efficient bulk operation. While the background calculation is happening, users that view the new field receive an indication that Force.com is calculating the field's value.

7.6 APEX

Apex is a strongly typed, object-oriented procedural programming language that developers can use to declare program variables and constants and execute traditional flow control statements (if-else, loops, etc.), data manipulation operations (insert, update, upsert, delete), and transaction control operations (setSavepoint, rollback) on behalf of Force.com applications. Developers can build Apex routines that add custom business logic to most application events, including button clicks, updates to data, Web service requests, custom batch services, and others.

Apex is similar to Java and is an integral Force.com component that helps the platform deliver reliable multitenant applications. For example, Force.com automatically validates all embedded Sforce Object Query Language (SOQL) and Sforce Object Search Language (SOSL) statements within an Apex class to prevent code that would otherwise fail at runtime. The platform then maintains corresponding object dependency information for valid Apex classes and uses this information to prevent changes to metadata that would otherwise break dependent applications.

To prevent malicious or unintentional monopolization of shared, multitenant platform resources, Force.com has an extensive set of governors and resource limits associated with Apex code execution. For example, Force.com closely monitors the execution of an Apex script and limits how much CPU time it can use, how much memory it can consume, how many queries and DML statements it can execute, and much more. Individual queries that the platform's optimizer regards as too expensive to execute throw a runtime exception to the caller. Although such limits might sound somewhat restrictive, they are necessary to protect the overall scalability and performance of the shared platform for all concerned applications. In the long term, these measures help to promote better coding techniques among platform developers and create a better experience for everyone. For example, a developer that initially tries to code a loop that inefficiently updates a thousand rows one row at a time will receive runtime exceptions due to resource limits and then begin using Force.com's efficient bulk processing API calls.

To further avoid potential platform problems introduced by poorly written applications, the deployment of a new production application is a process that is strictly managed. Before an organization can transition a new custom application from development to production status, salesforce.com requires unit tests that validate the functionality of the application's Apex routines. Submitted unit tests must cover no less than 75 percent of the application's source code. Salesforce.com executes submitted unit tests in the Force.com Sandbox environment to ascertain if the application will adversely affect the performance and scalability of the multitenant population at large. The results of an individual unit test indicate basic information such as the total number of lines executed as well as specific information about the code that was not executed by the test.

Once salesforce.com certifies an application for production, the deployment process consists of a single transaction that copies all the application's metadata into a production Force.com instance and reruns the corresponding unit tests. If any part of the process fails, Force.com simply rolls back the transaction and returns exceptions to help troubleshoot the problem.

After a production application is live, Force.com's built-in performance profiler automatically analyzes and provides associated feedback to administrators. Performance analysis reports include

information about slow queries, data manipulations, and sub-routines that developers can review and use to tune application functionality. The platform also logs and returns information about runtime exceptions to administrators to help debug their applications.

8. INTERNAL QUERY OPTIMIZATIONS

Most modern database systems determine optimal query execution plans by employing a cost-based query optimizer that considers relevant statistics about target table and index data. However, conventional cost-based optimizer statistics are designed for single-tenant applications and fail to account for the data access characteristics of any given user executing a query in a multitenant environment. For example, a given query that targets an object (table) with a large volume of data would most likely execute more efficiently using different execution plans for users with high visibility (a manager that can see all object instances) versus users with low visibility (sales people that can only see rows related to themselves).

To provide sufficient statistics for determining optimal query execution plans in a multitenant platform, Force.com maintains a complete set of optimizer statistics (tenant-, group-, and user-level) for each virtual multitenant object. Statistics reflect the number of rows that a particular query can potentially access, carefully considering overall tenant-specific object statistics (total number of rows owned by the tenant as a whole, etc.) as well as more granular statistics (the number of rows that a specific privilege group or end user can potentially access, etc.).

Force.com also maintains other types of statistics that prove helpful with certain queries. For example, the platform maintains statistics for all custom indexes to reveal the total number of non-null and unique values in the corresponding field, and histograms for picklist fields that reveal the cardinality of each list value.

When existing statistics are not in place or are not considered helpful, Force.com’s optimizer has a few different strategies it uses to help build reasonably optimal queries. For example, when a query filters on the Name field of an object, the optimizer can use the FallbackIndex pivot table to efficiently find requested object instances. In other scenarios, the optimizer will dynamically generate missing statistics at runtime.

Used in tandem with optimizer statistics, Force.com’s optimizer also relies on internal security related tables (Groups, Members, GroupBlowout, and CustomShare) that maintain information about the security domains of platform users, including a given user’s group memberships and custom access rights for objects.

The flow diagram in Figure 6 illustrates what happens when Force.com intercepts a request for data that is in one of the large heap tables such as Data. The request might originate from any number of sources, such as a page request from an Application Framework application, a Web services API call, or an Apex script. First, the platform executes “pre-queries” that consider the multitenant-aware statistics. Then, considering the results returned by the pre-queries, the platform builds an optimal database query for execution in the specific setting.

As Table 1 shows, Force.com can execute the same query four different ways, depending on who submits the query and the selectivity of the query’s filter conditions.

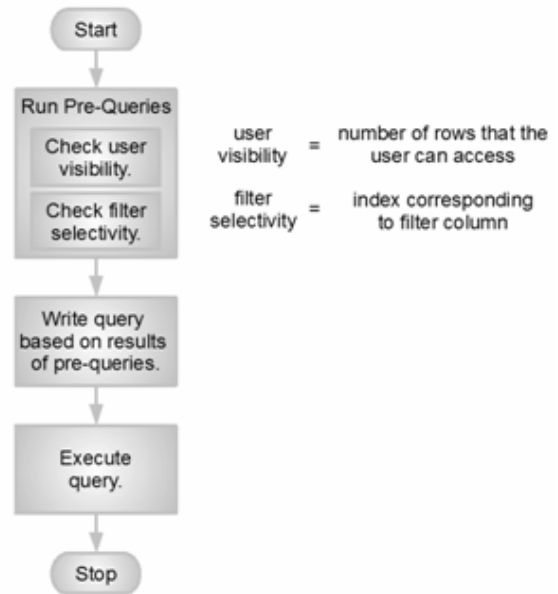


Figure 6. Force.com’s database query process

Table 1. Force.com builds a database query for a data access request based on the results of pre-queries

| Pre-Query Selectivity Measurements | | Write final database access query, forcing ... |
|------------------------------------|--------|--|
| User | Filter | |
| Low | Low | ... nested loops join; drive using view of rows that the user can see. |
| Low | High | ... use of index related to filter. |
| High | Low | ... ordered hash join; drive using Data table. |
| High | High | ... use of index related to filter. |

9. FULL-TEXT SEARCH ENGINE

Web-based application users have come to expect an interactive search capability to scan an application’s data and return up-to-date ranked results in sub-second response times. To provide such robust functionality for platform applications, Force.com uses an architecture based on an external search engine.

As applications update data in text fields (CLOBs, Name, etc.), a pool of indexing servers asynchronously update corresponding indexes, which the search engine maintains outside the core database. To optimize the indexing process, Force.com synchronously copies modified chunks of text data to an internal “to-be-indexed” table as transactions commit, thus providing a relatively small data source that minimizes the amount of data that indexing servers must read from disk. The search engine automatically maintains separate indexes for each tenant.

Depending on the current load and utilization of indexing servers, text index updates may noticeably lag behind actual transactions. To avoid unexpected search results originating from stale indexes,

Force.com also maintains an MRU cache of recently updated objects that the platform's application servers consider when materializing full-text search results. The platform maintains MRU caches on a per-user and per-organization basis to efficiently support possible search scopes.

Force.com optimizes the ranking of records within search results using several different methods. For example, the system considers the security domain of the user performing a search and weighs heavier those objects to which the current user has access. The system can rank more actively updated objects ahead of those that are relatively static. The user can choose to weight search results as desired, for example, placing more emphasis on recently modified objects.

10. CONCLUSION

Platform as a service (PaaS) and software as a service (SaaS) are contemporary software application development and delivery models that an increasing number of organizations are using to improve their time to market, reduce capital expenditures, and improve overall competitiveness in a challenging global economy. Internet-based, shared computing platforms are attractive because they let businesses quickly access hosted, managed software assets on demand and avoid the costs and complexity associated with the purchase, installation, configuration, and ongoing maintenance of an on-premises data center.

The most successful on-demand SaaS/PaaS company is salesforce.com. The company developed the Force.com platform whose metadata-driven architecture enables anyone to efficiently build and deliver sophisticated, customizable, mission-critical, Internet-scale multitenant applications. Using standards-based Web service APIs and native platform development tools, Force.com developers can easily build all components of a Web-based application, including the data model, user interface, business logic, integrations with other applications, and more.

Over the past 10 years, salesforce.com engineers have optimized all layers of the Force.com platform for multitenancy, with features that let the platform deliver unprecedented Internet scalability to the height of 170 million transactions daily. Platform features such as the bulk data processing API, the Apex programming language, an external full-text search engine, and its unique query optimizer help make multitenant platform applications highly efficient and scalable with little or no thought from developers.

Salesforce.com's managed approach for the deployment of production applications ensures top-notch performance, scalability, and reliability for all dependent applications. Additionally, salesforce.com continually monitors and gathers operational information from Force.com applications to help drive incremental improvements and new platform features that immediately benefit existing and new applications.