The Force.com Multitenant Architecture

UPDATED March 2016

An Under the Hood Look at Force.com

Last updated on 3-31-2016

Force.com is the preeminent cloud application development platform in use today, supporting more than 100,000 organizations and 220,000 deployed apps. Individual enterprises and commercial *Software as a Service (SaaS)* vendors trust the platform to deliver robust, reliable, and Internet-scale applications. To meet the high demands of its large user population, Force.com's foundation is a metadata-driven software architecture that enables multitenant applications. This paper explains the technology that makes the Force.com platform fast, scalable, and secure for any type of application.

Cloud Computing

Since the turn of the millennium, *cloud computing* has revolutionized the landscape of the IT world because it provides enterprise-grade computing resources that are affordable and instantly available. Clouds provide straightforward access to IT resources–you just access as many resources as you need when you need them, and never have to deal with the complexities of managing all of the underlying mechanisms that provide those resources. Life is suddenly a lot simpler and easier with cloud computing.

Multitenancy

Multitenancy is the fundamental technology that clouds use to share IT resources cost-efficiently and securely. Just like a bank–in which many tenants cost-efficiently share a hidden, common infrastructure, yet utilize a defined set of highly secure services, with complete privacy from other tenants–a cloud uses multitenancy technology to share IT resources securely among multiple applications and *tenants* (businesses, organizations, etc.) that use the cloud. Some clouds use virtualization-based architectures to isolate tenants; others use custom software architectures to get the job done.

The multitenant design of a cloud service can have a dramatic impact on the application delivery and productivity of an IT organization, yet most CIOs, CTOs, system architects, and developers who use clouds don't give it a second thought because it's all magic that transparently happens behind the scenes. This paper presents an "under the hood" look at the unique underlying design of core Salesforce Platform technology, Force.com.

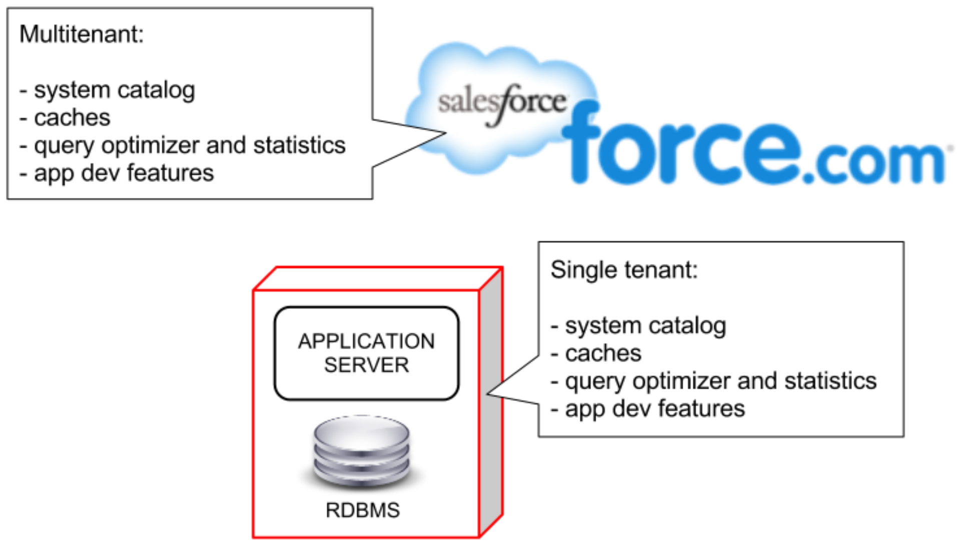
Force.com Architecture Overview

Force.com (http://www.force.com/) is the proven cloud application development platform that powers many popular salesforce.com cloud applications (http://www.salesforce.com/products/) (Sales Cloud, Service Cloud, etc.), as well as custom applications that customers build to satisfy their specific business requirements. The following sections provide you with an overview of key aspects of the platform's design.

Multitenant Kernel

Force.com is a modern *Platform as a Service (PaaS)* that's built for cloud computing, with multitenancy inherent in its design. A quick way to understand what makes Force.com unique is to consider the following figure that compares a traditional application development platform with Force.com's multitenant approach.

At the heart of all conventional application development platforms beats a *relational database management system (RDBMS),* most of which were designed in the 1970s and 1980s to support individual organizations' on-premises deployments. All the core mechanisms in an RDBMS–such as its system catalog, caching mechanisms, query optimizer, and application development features–are built to support single-tenant applications and be run directly on top of a specifically tuned host operating system and raw hardware. Without significant development efforts, multitenant cloud database services built with a standard RDBMS are only possible with the help of virtualization. Unfortunately, the extra overhead of a hypervisor typically hurts the performance of an RDBMS.



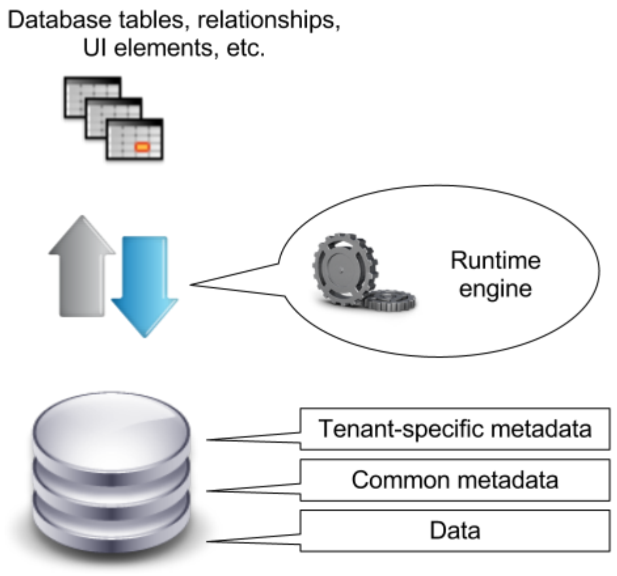
In contrast, Force.com combines several different persistence technologies, including a custom-designed relational database schema, which are innately designed for clouds and multitenancy–no virtualization required. The benefits of Force.com's unique architecture are extraordinary. Force.com is a proven, reliable, and secure cloud application development offering today that serves 100,000+ businesses (http://www.salesforce.com/customers/), millions of users, all with exceptional performance and reliability (http://trust.salesforce.com/trust/).

Metadata-Driven Kernel

An inherently multitenant platform such as Force.com, which is designed specifically to service a cloud, is difficult to build because it must be reliable, scalable, customizable by tenants, upgradeable without downtime, secure, and fast. But how can you keep tenant-specific data secure in a shared database so that one tenant can't see another tenant's data? How can one tenant customize various schema objects and an application's user interface in real time without affecting the functionality or availability of the system for all other tenants? How can the system's code base be patched or upgraded without breaking tenant-specific schemas? And how will the system's response time scale as tens of thousands of tenants use the service?

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

For these reasons, Force.com's core technology uses a runtime engine that materializes all application data from *metadata*–data about the data itself. In Force.com's well-defined metadata-driven architecture, there is a clear separation of the compiled runtime database engine (kernel), tenant data, and the metadata that describes each application. These distinct boundaries make it possible to independently update the system kernel and tenant-specific applications and schemas, with virtually no risk of one affecting the others.

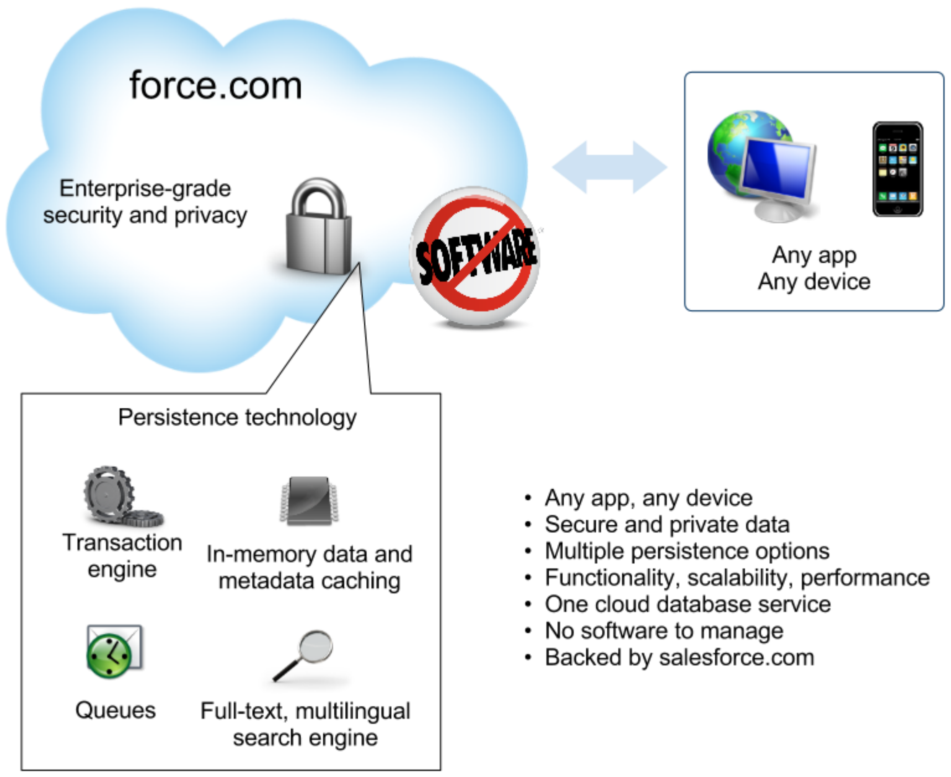


Every logical database object that Force.com exposes is internally managed using metadata. Objects, (*tables* in traditional relational database parlance), fields, stored procedures, and database triggers are all abstract constructs that exist merely as metadata in Force.com's *Universal Data Dictionary (UDD)*. For example, when you define a new application object or write 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 system's engine can use to generate the virtual application components at runtime. When you need to modify or customize something about the application schema, like modify an existing field in an object, 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 system's runtime engine must optimize access to metadata; otherwise, frequent metadata access would prevent the service from scaling. With this potential bottleneck in mind, Force.com uses massive and sophisticated *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.

Polyglot Persistence

Force.com integrates and optimizes several different data persistence technologies to deliver transparent polyglot persistence for all your applications and devices. With Force.com, you don't have to deal with the complexity of trying to integrate, manage, test, and support several systems, and you only have to code to a single API, no matter which type of persistence is optimal for a given situation. The following figure is an overview of a sampling of Force.com's persistence technology.


force.com
Enterprise-grade security and privacy
SOFTWARE
Any app
Any device
Persistence technology
Transaction engine
In-memory data and metadata caching
Queues
Full-text, multilingual search engine
• Any app, any device
• Secure and private data
• Multiple persistence options
• Functionality, scalability, performance
• One cloud database service
• No software to manage
• Backed by salesforce.com

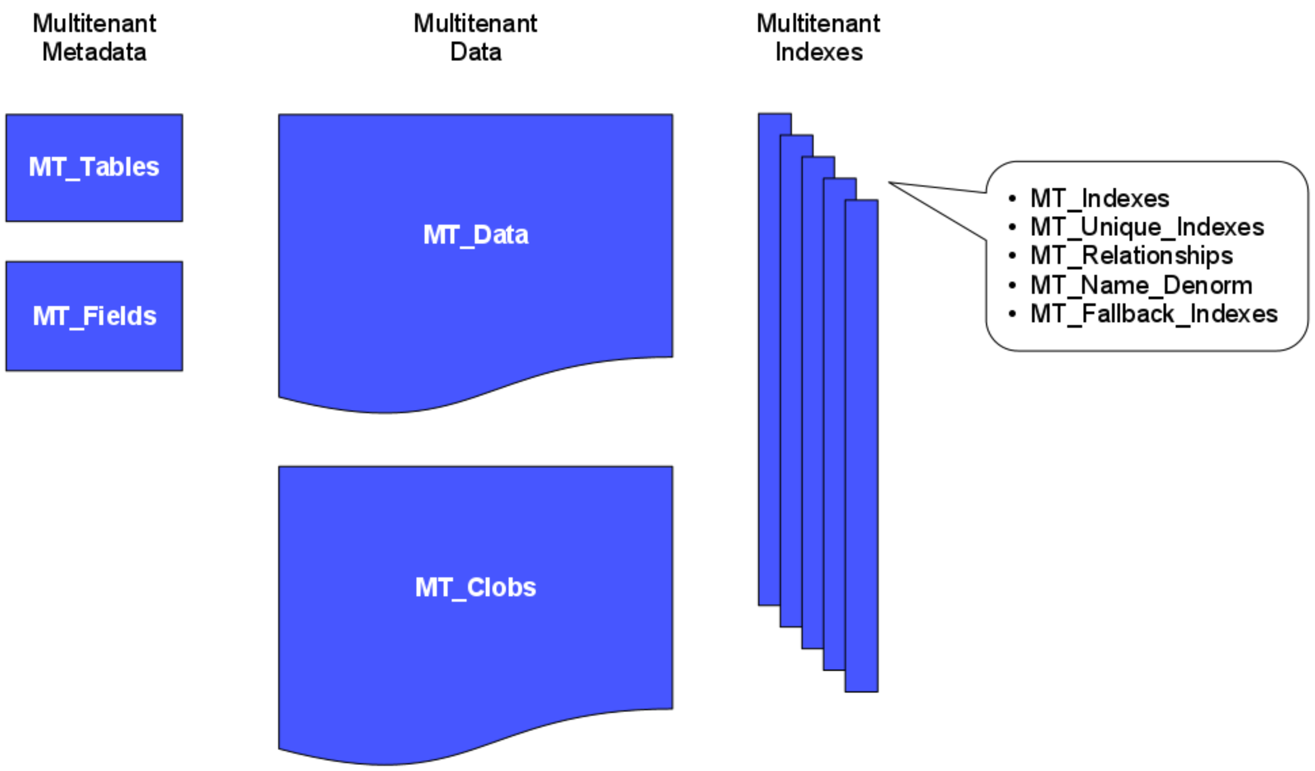
At the heart of Force.com is its transaction database engine. Force.com uses a relational database engine with a specialized data model that is optimal for multitenancy. The next section of this paper explains more about this schema.

Force.com also uses other persistence technologies to deliver fast, scalable performance for various operations. For example, to further hone application response times, Force.com employs a search engine (separate from the transaction engine) 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 transaction engine and the search service lets applications efficiently process transactions without the overhead of text index updates and at the same time quickly provide users with accurate search results.

Now that you have a general idea of the key architecture components that make up the underlying mechanisms of Force.com, it's time to learn more about the structure and purpose of various internal system elements that make up the data model that supports the transaction engine.

Multitenant Data Model

Building a cloud application development platform that attempts to manage a vast, ever-changing set of actual database structures on behalf of each application and tenant would be next to impossible as the service grows. Instead, the Force.com storage model manages virtual database structures using a set of metadata, data, and pivot tables, as illustrated in the following figure.



When you create application schemas, the UDD keeps track of metadata concerning the objects, their fields, their relationships, and other object attributes. Meanwhile, a few large database tables store the structured and unstructured data for all virtual tables. A set of related multitenant indexes, implemented as simple pivot tables with denormalized data, make the combined data set extremely functional. The following sections explain each type of component in more detail.

Multitenant Metadata

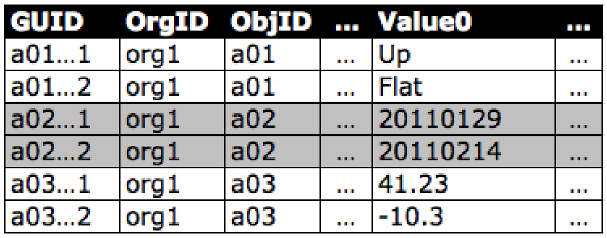
Force.com has two core internal tables that it uses to manage metadata that corresponds to a tenant's schema objects: MT_Objects and MT_Fields. (Please note that, for clarity, the actual names of Force.com system tables and columns are not necessarily cited in this paper.)

> The *MT_Objects* system table stores metadata about the tables 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).
> The *MT_Fields* system table stores metadata about the fields (columns) that an organization defines for each object, 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).

Multitenant Data

The *MT_Data* system table stores the application-accessible data that maps to all organization-specific tables and their fields, as defined by metadata in MT_Objects and MT_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 MT_Data table also has a Name field that stores a "natural name" for corresponding records; for example, an Account record might use "Account Name," a Case record might use "Case Number," and so on.

Value0 … Value500 *flex columns*, otherwise known as *slots*, store application data that maps to the tables and fields declared in MT_Objects and MT_Fields, respectively; all flex columns use a variable-length string datatype so that they can store any structured type of application data (strings, numbers, dates, etc.). As the following figure illustrates, no two fields of the same object can map to the same slot in MT_Data for storage; however, a single slot can manage the information of multiple fields, as long as each field stems from a different object.



GUID	OrgID	ObjID	...	Value0	...
a01...1	org1	a01	...	Up	...
a01...2	org1	a01	...	Flat	...
a02...1	org1	a02	...	20110129	...
a02...2	org1	a02	...	20110214	...
a03...1	org1	a03	...	41.23	...
a03...2	org1	a03	...	-10.3	...

MT_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, *rich-structured datatypes* such as picklist (enumerated field), auto-number (auto-incremented, system-generated sequence number), formula (read-only derived value), master-detail relationship (foreign key), checkbox (Boolean), email, URL, and others. MT_Fields can also be required (not null) and have custom validation rules (for example, one field must be greater than another field), both of which Force.com enforces.

When an organization declares or modifies an object, Force.com manages a row of metadata in MT_Objects that defines the object. Likewise, for each field, Force.com manages a row in MT_Fields, including metadata that maps the field to a specific flex column in MT_Data for the storage of corresponding field data. Because Force.com manages object and field definitions as metadata rather than actual database structures, the system can tolerate *online multitenant application schema maintenance* activities without blocking the concurrent activity of other tenants and users. By comparison, online table redefinition for traditional relational database systems typically requires laborious, complicated processes and scheduled application downtime.

As the simplified representation of MT_Data in the previous figure 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 data from and write data to flex columns.

Although not shown in the previous figure, MT_Data also contains other columns. For example, there are four columns to manage *auditing* data, including which user created a row and when that row was created, and which user last modified a row and when that row was last modified. MT_Data also contains an *IsDeleted column* that Force.com uses to indicate when a row has been deleted.

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

Note: Force.com also stores CLOBs in an indexed form outside of the database for fast text searches. See later in this paper for more information about Force.com's text search engine.

Multitenant Indexes

Force.com automatically indexes various types of fields to deliver scalable performance–without you ever having to think about it. This section explains more about the unique way that Force.com manages index data for multiple tenants.

Traditional database systems rely on native database 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 MT_Data because Force.com uses a single flex column to store the data of many fields with varying structured datatypes. Instead, Force.com manages an index of MT_Data by synchronously copying field data marked for indexing to an appropriate column in an *MT_Indexes* pivot table.

MT_Indexes 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 an MT_Data flex column to the StringValue field in MT_Indexes, a date value to the DateValue field, etc. The underlying indexes of MT_Indexes are standard, non-unique database indexes. When an internal system query includes a search parameter that references a structured field in an object, Force.com's custom query optimizer uses MT_Indexes to help optimize associated data access operations.

Note: Force.com can handle searches across multiple languages because the system uses a *case-folding algorithm* that converts string values to a universal, case-insensitive format. The StringValue column of the MT_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, which in turn corresponds to the literal provided in the search request.

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 MT_Data and shared usage of the Value columns for field data, it is not practical to create unique database indexes for the object. (This situation is similar to the one discussed in the previous section for non-unique indexes.)

To support uniqueness for custom fields, Force.com uses the *MT_Unique_Indexes* pivot table; this table is very similar to the MT_Indexes table, except that the underlying native database indexes of MT_Unique_ 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.

In rare circumstances, Force.com's external search engine (explained later in this paper) can become overloaded or otherwise unavailable, and may not be able to respond to a search request in a timely manner. Rather than returning a disappointing error to a user that has requested a search, Force.com falls back to a secondary search mechanism to furnish reasonable search results.

A *fall-back search* is implemented as a direct database query with search conditions that reference the Name field of target records. To optimize global object searches (searches that span tables) without having to execute potentially expensive union queries, Force.com maintains a *MT_Fallback_Indexes* pivot table that records the Name of all records. Updates to MT_Fallback_Indexes happen synchronously as transactions modify records so that fall-back searches always have access to the most current database information.

The *MT_Name_Denorm* table is a lean data table that stores the ObjID and Name of each record in MT_Data. When an application needs to provide a list of records involved in a parent/child relationship, Force.com uses the MT_Name_Denorm table to execute a relatively simple query that retrieves the Name of each referenced record for display in the app, say, as part of a hyperlink.

Multitenant Relationships

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

To optimize join operations, Force.com maintains an *MT_Relationships* pivot table. This system table has two underlying database unique composite indexes that allow for efficient object traversals in either direction, as necessary.

Multitenant Field History

With just a few mouse clicks, Force.com provides history tracking for any field. When a tenant enables auditing for a specific field, the system asynchronously records information about the changes made to the field (old and new values, change date, etc.) using an internal pivot table as an audit trail.

Partitioning of Metadata, Data, and Index Data

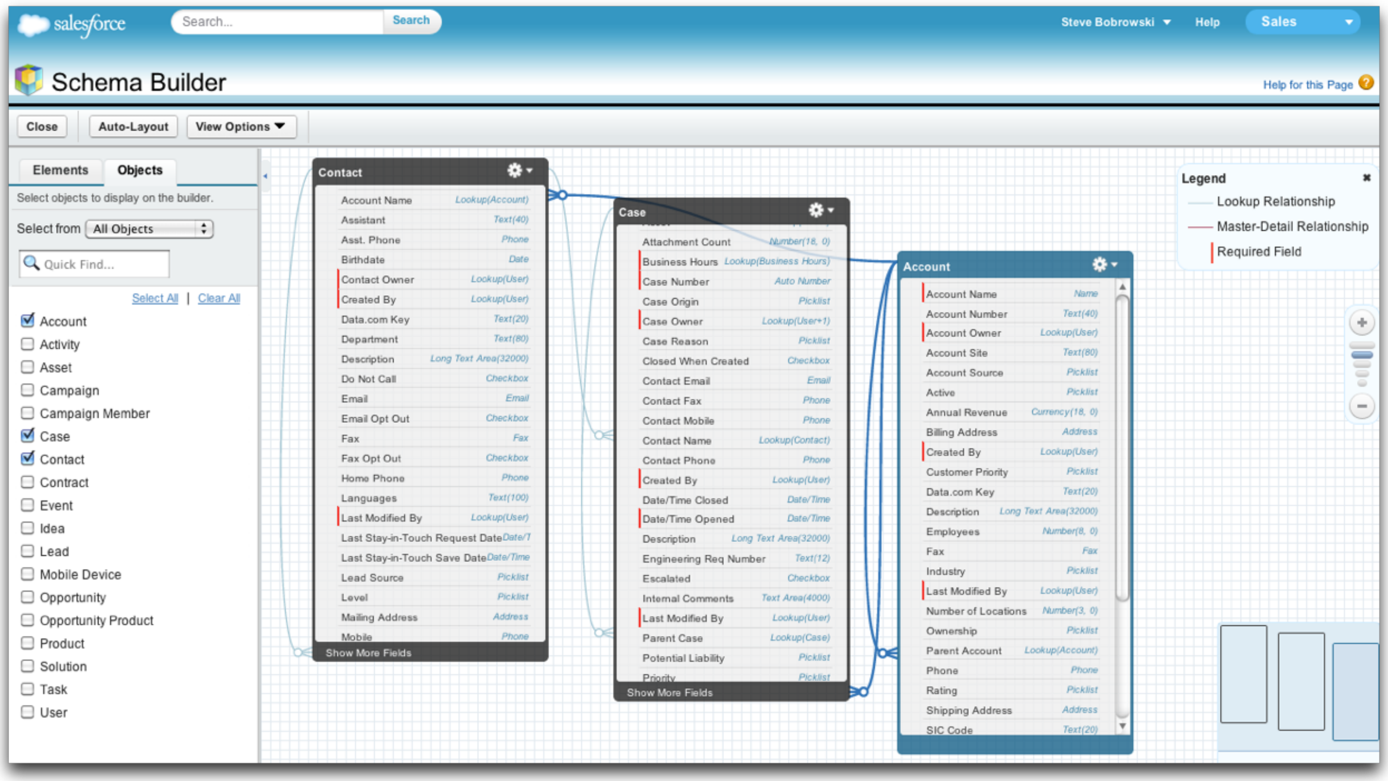
All Force.com data, metadata, and pivot table structures, including underlying database indexes, are physically partitioned by OrgID (by tenant) using native database partitioning mechanisms. Data partitioning is a proven technique that database systems provide to physically divide large logical data 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. By definition, every Force.com 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."

Multitenant App Development

The previous section explains the architecture that Force.com uses to store metadata and data. This section briefly explains how app developers can create a schema's underlying metadata and then build apps that manage data.

The Force.com Browser-Based Development Environment

Developers can declaratively build server-side application components using the Force.com Web browser-based development environment, commonly referred to as the Force.com Setup screens. This point-and-click UI supports all facets of the application schema building process, including the creation of an application's data model (objects and their fields, relationships, etc.), security and sharing model (users, profiles, role hierarchies, etc.), user interface (screen layouts, data entry forms, reports, etc.), declarative logic (workflows), and programmatic logic (stored procedures and triggers). For example, the following screen is the Force.com Schema Builder, an intuitive, ERD-like data modeling tool.



Contact

Account Name	Lookup(Account)
Assistant	Text(40)
Asst. Phone	Phone
Birthdate	Date
Contact Owner	Lookup(User)
Created By	Lookup(User)
Data.com Key	Text(20)
Department	Text(80)
Description	Long Text Area(32000)
Do Not Call	Checkbox
Email	Email
Email Opt Out	Checkbox
Fax	Fax
Fax Opt Out	Checkbox
Home Phone	Phone
Languages	Text(100)
Last Modified By	Lookup(User)
Last Stay-in-Touch Request Date	Date/T
Last Stay-in-Touch Save Date	Date/Time
Lead Source	Picklist
Level	Picklist
Mailing Address	Address
Mobile	Phone

Case

Attachment Count	Number(18, 0)
Business Hours	Lookup(Business Hours)
Case Number	Auto Number
Case Origin	Picklist
Case Owner	Lookup(User+1)
Case Reason	Picklist
Closed When Created	Checkbox
Contact Email	Email
Contact Fax	Phone
Contact Mobile	Phone
Contact Name	Lookup(Contact)
Contact Phone	Phone
Created By	Lookup(User)
Date/Time Closed	Date/Time
Date/Time Opened	Date/Time
Description	Long Text Area(32000)
Engineering Req Number	Text(12)
Escalated	Checkbox
Internal Comments	Text Area(4000)
Last Modified By	Lookup(User)
Parent Case	Lookup(Case)
Potential Liability	Picklist
Priority	Picklist

Account

Account Name	Name
Account Number	Text(40)
Account Owner	Lookup(User)
Account Site	Text(80)
Account Source	Picklist
Active	Picklist
Annual Revenue	Currency(18, 0)
Billing Address	Address
Created By	Lookup(User)
Customer Priority	Picklist
Data.com Key	Text(20)
Description	Long Text Area(32000)
Employees	Number(8, 0)
Fax	Fax
Industry	Picklist
Last Modified By	Lookup(User)
Number of Locations	Number(3, 0)
Ownership	Picklist
Parent Account	Lookup(Account)
Phone	Phone
Rating	Picklist
Shipping Address	Address
SIC Code	Text(20)

Force.com Setup screens provide access to many built-in system features that make it easy to implement common application functionality without writing the otherwise complicated and error-prone code that's required in traditional database systems.

Some such features include the following.

Force.com native user interfaces are easy to build because there's no coding necessary. Behind the scenes, a native app UI supports all the usual data access operations, including queries, inserts, updates, and deletes. Each data manipulation operation performed by native platform applications can modify one object at a time and automatically commit each change in a separate transaction.

A *workflow* is a declarative trigger–a predefined action triggered by the insert or update of a row. A workflow can update to a field or send an outbound message. Workflow rules specify the criteria that determine when to trigger a workflow action. For example, you might declare a workflow that, immediately after a record is updated, automatically updates the row's Status field to "Modified." All workflow operations occur within the context of the transaction that triggers the workflow. If the system rolls back a transaction, all related workflow operations that were executed also roll back.

When defining a text field for an object that contains sensitive data, developers can easily configure the field so that Force.com encrypts the corresponding data and, optionally, uses an input mask to hide screen information from prying eyes. Force.com encrypts fields using AES (Advanced Encryption Standard) algorithm 128-bit keys.

A declarative *validation rule* is a simple way for an organization to enforce a domain integrity rule without any programming. For example, you might declare a validation rule that makes sure that a LineItem object's Quantity field is always greater than zero.

A *formula field* is a declarative feature of Force.com that makes it easy to add a calculated field to an object. For example, you might add a field to the LineItem object to calculate a LineTotal value.

A *roll-up summary field* is a cross-object field that makes it easy to aggregate child field information in a parent object. For example, you might create an OrderTotal summary field in the SalesOrder object based on the LineTotal field of the LineItem object.

Note: Internally, Force.com implements formula and roll-up summary fields using native database features and efficiently recalculates values synchronously as part of ongoing transactions.

APIs

Force.com provides open, standards-based APIs that developers can use to build apps. Both *RESTful and Web services (SOAP-based) APIs* are available that provide access to Force.com's many features. Using these various APIs, an application can do many things such as:

Manipulate metadata that describes an application schema
Create, read, update, and delete (CRUD) business data
Bulk-load a large number of records asynchronously
Expose a near real-time stream of data in a secure and scalable way
Embed social networking functionality into an application with minimal effort
Add social collaboration features to any application, including push notifications and data feeds

For more information about each API, see the Integration (http://wiki.developerforce.com/page/Integration) section of Developer Force (http://developer.force.com/).

Query Languages

Apps can use the *Salesforce Object Query Language (SOQL)* to construct simple but powerful database queries. Similar to the SELECT command in the Structured Query Language (SQL), SOQL allows you to specify the source object, a list of fields to retrieve, and conditions for selecting rows in the source object. For example, the following SOQL query returns the value of the Id and Name field for all Account records with a Name equal to the string 'Acme'.

```
SELECT Id, Name FROM Account WHERE Name = 'Acme'
```

Force.com also includes a full-text, multi-lingual search engine that automatically indexes all text-related fields. Apps can leverage this pre-integrated search engine using the *Salesforce Object Search Language (SOSL)* to perform text searches. Unlike SOQL, which can only query one object at a time, SOSL enables you to search text, email, and phone fields for multiple objects simultaneously. For example, the following SOSL statement searches for records in the Lead and Contact objects that contain the string 'Joe Smith' in the name field and returns the name and phone number field from each record found.

```
FIND {"Joe Smith"} IN Name Fields RETURNING lead(name, phone), contact(name, phone)
```

Apex and Procedural Logic

Apex, which is similar to Java in many respects, is a powerful development language that developers can use to centralize procedural logic in their application schema. Apex code can declare program variables and constants, execute traditional flow control statements (if-else, loops, etc.), perform data manipulation operations (insert, update, upsert, delete), and transaction control operations (setSavepoint, rollback).

You can store Apex programs in Force.com using two different forms: as a named *Apex class* with methods (akin to stored procedures in traditional database parlance) that applications execute when necessary, or as a *database trigger* that automatically executes before or after a specific database manipulation event occurs. In either form, Force.com compiles Apex code and stores it as metadata in the UDD. The first time that an organization executes an Apex program, Force.com's runtime interpreter loads the compiled version of the program into an MRU cache for that organization. Thereafter, when any user from the same organization needs to use the same routine, Force.com can save memory and avoid the overhead of recompiling the program again by sharing the ready-to-run program that is already in memory.

With the addition of a simple keyword here and there, developers can use Apex to support many unique application requirements. For example, developers can expose a method as a custom RESTful or SOAP-based API call, make it asynchronously schedulable, or configure it to process a large operation in batches.

Apex is much more than "just another procedural language." It's an integral Force.com component that helps the system deliver reliable multitenancy. For example, Force.com automatically validates all embedded SOQL and SOSL statements within a Force.com class to prevent code that would otherwise fail at runtime. Force.com then maintains corresponding object dependency information for valid Force.com classes and uses it to prevent changes to metadata that would otherwise break dependent code.

Many standard Apex classes and system static methods provide simple interfaces to underlying system features. For example, the system static DML methods such as insert, update, and delete have a simple Boolean parameter that developers can use to indicate the desired bulk processing option (all or nothing, or partial save); these methods also return a result object that the calling routine can read to determine which records were unsuccessfully processed and why. Other examples of the direct ties between Apex and Force.com features include the built-in email classes and XmlStream classes, just to name a couple.

Multitenant Processing

In large part, Force.com performs and scales so well because the engineers at salesforce.com built the service with two important principles in mind.

Do everything as efficiently as possible.
Help developers do everything as efficiently as possible.

With these thoughts in mind, the following sections explain some of the unique processing architectures of Force.com.

Multitenant Query Processing

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 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 rows) versus users with low visibility (sales people that can only see rows related to themselves).

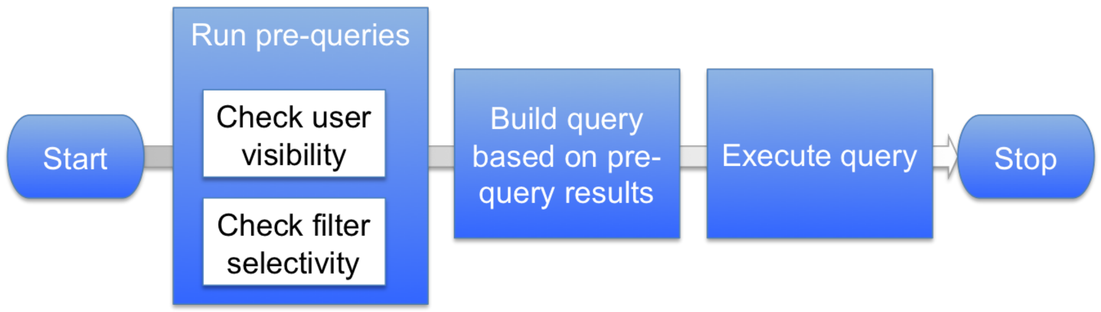
To provide sufficient statistics for determining optimal query execution plans in a multitenant system, Force.com maintains a complete set of *optimizer statistics* (tenant-, group-, and user-level) for each virtual multitenant object. Statistics reflect the number of rows that a particular query can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

Force.com also maintains other types of statistics that prove helpful with particular queries. For example, the service 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 MT_Fallback_Indexes table to efficiently find requested rows. 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 system users, including a given user's group memberships and custom access rights for object and rows. Such information is invaluable at determining the selectivity of query filters on a per-user basis. See the paper Architecting Force.com Apps A Design Primer (https://developer.salesforce.com/page/Architecting_Force.com_Apps_A_Design_Primer) for more information about the platform's embedded security model.

The flow diagram in the following figure illustrates what happens when Force.com intercepts a request for data that is in one of the large heap tables such as MT_Data. The request might originate from any number of sources, such as an API call or stored procedure. First, Force.com executes "pre-queries" that consider the multitenant-aware statistics. Then, considering the results returned by the pre-queries, the service builds an optimal underlying database query for execution in the specific setting.

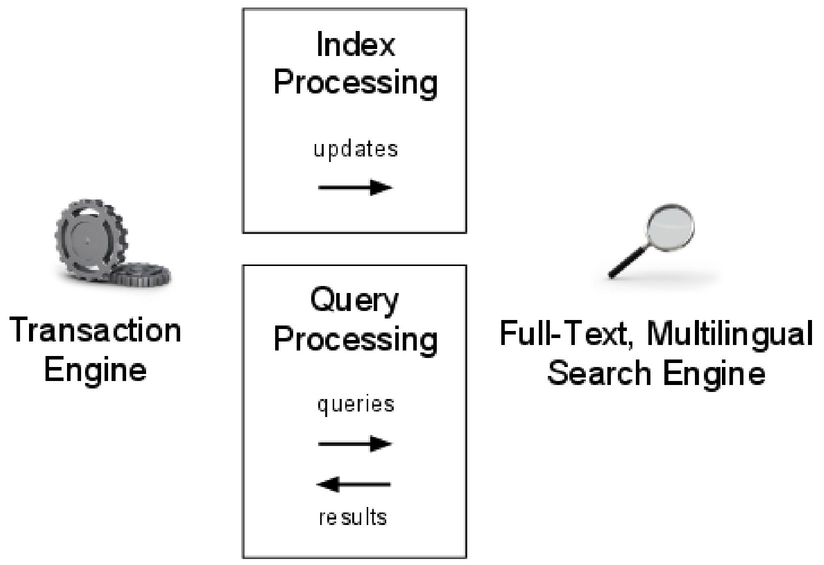


As the following table shows, Force.com can execute the same query four different ways, depending on who submits the query and the selectivity of the query's filter conditions.

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

Multitenant Search Processing

Web-based application users have come to expect an interactive search capability to scan all or a selected scope of an application's database, return up-to-date ranked results, and do it all with sub-second response times. To provide such robust search functionality for applications, Force.com uses a *search engine* that is separate from its transaction engine. The relationship between the two engines is depicted in the following figure. The search engine receives data from the transactional engine, with which it creates search indexes. The transactional engine forwards search requests to the search engine, which returns results that the transaction engine uses to locate rows that satisfy the search request.



As applications update data in text fields (CLOBs, Name, etc.), a pool of background processes called indexing servers are responsible for asynchronously updating corresponding indexes, which the search engine maintains outside the core transaction engine. 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 organization (tenant).

Depending on the current load and utilization of indexing servers, text index updates may lag behind actual transactions. To avoid unexpected search results originating from stale indexes, Force.com also maintains an MRU (most recently used) cache of recently updated rows that the system considers when materializing full-text search results. Force.com maintains MRU caches on a per-user and per-organization basis to efficiently support possible search scopes.

Force.com’s search engine optimizes the ranking of records within search results using several methods. For example, the system considers the security domain of the user performing a search and puts more weight in the rows that the current user can access. The system can also consider the modification history of a particular row and rank more actively updated rows 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 rows.

Multitenant Bulk Operations

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 might load many new rows. An inefficient approach would be to use a routine with a loop that inserts individual rows, making one API call after another for each row insert. A much more efficient approach would be to create an array of rows and have the routine insert all of them with a single API call.

Efficient bulk processing with Force.com is simple for developers because it is baked into API calls. Internally, Force.com also bulk processes all internal steps related to an explicit bulk operation.

Force.com’s bulk processing engine automatically accounts 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 successive stage 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 your discretion, an all-or-nothing mode is available for bulk operations. Also, the execution of triggers during a bulk operation is subject to internal governors that restrict the amount of work.

Multitenant Schema Modifications

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 cloud database service at large.

For example, consider what happens behind the scenes when you modify 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 you add a roll-up summary field to an object. In this case, Force.com asynchronously calculates initial summaries in the background using an efficient bulk operation. While the background calculation is happening, users who view the new field receive an indication that Force.com is currently calculating the field's value.

Multitenant Isolation and Protection

To prevent malicious or unintentional monopolization of shared, multitenant system resources, Force.com has an extensive set of governors and resource limits associated with Force.com code execution. For example, Force.com closely monitors the execution of a code 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, how many math calculations it can perform, how many outbound Web services calls it can make, and much more. Individual queries that Force.com'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 database system for all concerned applications. In the long term, these measures help to promote better coding techniques among developers and create a better experience for everyone that uses the platform. For example, a developer who 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 system 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 application from development to production status, salesforce.com requires unit tests that validate the functionality of the application's Force.com code routines. Submitted unit tests must cover no less than 75 percent of the application's source code. Salesforce.com executes submitted unit tests in Force.com's sandbox development environment to ascertain if the application code 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 wasn't executed by the test.

Once an application's code is certified for production by salesforce.com, the deployment process for the application 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.

Note: Salesforce.com reruns the unit tests for every application with each development release of Force.com to proactively learn whether new system features and enhancements break any existing applications.

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 you can review and use to tune application functionality. The system also logs and returns information about runtime exceptions to administrators to help debug their applications.

Deletes, Undeletes, and the Recycle Bin

When an app deletes a record from an object, Force.com simply marks the row for deletion by modifying the row's IsDeleted field in MT_Data. This action effectively places the row in what is known as the *Recycle Bin*. Force.com lets you restore selected rows from the Recycle Bin for up to 15 days before permanently removing them from MT_Data. Force.com limits the total number of records it maintains for an organization based on the storage limits for that organization.

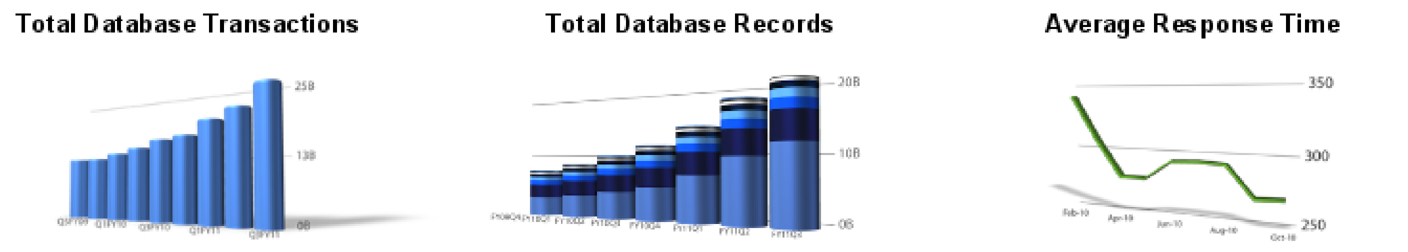
When an operation deletes a parent record involved in a master-detail relationship, Force.com automatically deletes all related child records, provided that doing so would not break any referential integrity rules in place. For example, when you delete a SalesOrder, Force.com automatically cascades the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

In contrast, when you delete a referenced parent record involved in a lookup relationship, Force.com automatically sets all dependent keys to null. If you subsequently restore 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.

Historical Statistics

Years of experience have hardened Force.com into an extremely fast, scalable, and reliable cloud database service. As an illustration of Force.com's proven capability to support Internet-scale applications, consider the following figure. Specifically notice that, over time, average page response time has decreased or held steady (a measure of performance), while average transaction volume has concurrently increased (a measure of scalability).



For more system data such as planned maintenance, historical information on transaction volume and speed, etc., visit trust.salesforce.com (http://trust.salesforce.com/), the Force.com community's home for real-time information about system performance and security.

Summary

Cloud-based services are a contemporary IT resources model 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. Services such as cloud application development platforms are attractive because they let businesses quickly access 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 and dedicated hardware, software, and accompanying administrative staff.

As the foundation of salesforce.com's enormously successful apps–such as Sales Cloud and Service Cloud, Force.com is a proven application development platform on which individual enterprises and service providers have built all types of business applications, including supply chain management, billing, accounting, compliance tracking, human resource management, and claims processing applications. Force.com's unique, multitenant, metadata-driven architecture is engineered specifically for the cloud, and has demonstrated since 1999 that it reliably and securely supports mission-critical, Internet-scale applications. Using standards-based Web services APIs and native development tools, Force.com developers can easily build all components of a Web-based application, including the application's data model (objects, relationships, etc.), business logic (workflows, validations, etc.), integrations with other applications, and more.

Since its inception, Force.com has been optimized by salesforce.com's engineers for multitenancy, with features that let the system deliver unprecedented Internet scalability exceeding 1 billion complex business transactions per day. Integral system features–such as the bulk data-processing API; Apex, an external, full-text search engine; and a unique query optimizer–help make dependent applications highly efficient and scalable with little or no effort from developers.

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

Related Resources

- Architect Core Resources (/page/Architect_Core_Resources)

About the Author and CCE Technical Enablement

Steve Bobrowski (https://twitter.com/sbob909) is a member of Technical Enablement within the Salesforce Customer Centric Engineering group. The team's mission is to help customers understand how to implement technically sound Salesforce solutions. Check out all of the resources that this team maintains on the Architect Core Resources (/page/Architect_Core_Resources) page of Developer Force.

Technical Library

Documentation (/index.php/Documentation)
Tools (/index.php/Tools)
Integration (/index.php/Integration)
App Logic (/index.php/App_Logic)
User Interface (/index.php/User_Interface)
Database (/index.php/Database)
Security (/index.php/Security)
Web Sites (/index.php/Websites)
Mobile (/mobile)
App Distribution (/index.php/App_Distribution)
Newsletter (/index.php/News)
Release Information (http://developer.force.com/releases/)

RSS Feeds

Featured Content (http://feeds.feedburner.com/force/featuredcontent)
Blog (http://feeds.feedburner.com/SforceBlog)
Discussion Boards (//developer.salesforce.com/forums/ForumsRSS)

Salesforce Platform (/platform)

Lightning Platform (/gettingstarted)

Heroku (https://devcenter.heroku.com/start)

MuleSoft Quick Start Guide (https://developer.mulesoft.com/tutorials-and-how-tos)

Salesforce Dev Centers

Lightning Developer Center (/devcenter/lightning)

Mobile Developer Center (/devcenter/mobile)

Heroku Dev Center (http://devcenter.heroku.com/)

Desk.com </developers> (http://dev.desk.com/)

Pardot Developer Site (http://developer.pardot.com/)

MuleSoft Developer Site (http://developer.mulesoft.com/)

Developer resources

Mobile Services (/mobile)

Lightning Platform Docs (/docs/?service=Force.com&sort=title)

Lightning Platform Downloads (/page/Tools)

Heroku Downloads (http://toolbelt.heroku.com/)

Learn Salesforce with Trailhead (/trailhead?utm_campaign=trailhead&utm_source=website&utm_medium=dsc_footer)

Community

Developer Community Groups (https://trailblazercommunitygroups.com/?&utm_source=developer&utm_medium=web-banner&utm_campaign=community_groups&utm_content=devs)

Developer Forums (/forums)

Salesforce Developer Events (/calendar)

Webinars (/content/type/Webinar)

Learn more

Salesforce AppExchange (/appexchange)

Salesforce Administrators (//admin.salesforce.com)

Salesforce.com Help Portal (//help.salesforce.com/)

Privacy Statement (//www.salesforce.com/company/privacy.jsp)

Security Statement (//www.salesforce.com/company/security.jsp)

Terms of Use (https://trailblazer.me/terms)

[+]Feedback

About Us (http://www.salesforce.com/company/)

Language: English

(https://www.facebook.com/salesforcedevs) (https://twitter.com/#!/salesforcedevs)

(https://plus.google.com/118327959233932983591)

Join over 6.0 million Salesforce developers. Subscribe Now! HIDE

/