# Total Object Relationships in Salesforce

* Master-detail relationship
* Lookup relationship
* Self-relationship
* External lookup relationship
* Indirect lookup relationship
* Many-to-many relationship (junction object)
* Hierarchical relationship

A relationship is a bi-directional association between two objects. Relationships allows us to create links between one object and another. The platform supports following relationship types

* Master-Detail
* Lookup
* Many-to-Many

## **Master-Detail (1: n)**

A parent-child relationship in which the master object controls certain behaviours of the detail object.

* When a record of the master object is deleted, its related detail records are also deleted.
* The Owner field on the detail object is not available and is automatically set to the owner of its associated master record. Custom objects on the detail side of a master-detail relationship cannot have sharing rules, manual sharing, or queues, as these require the Owner field.
* The detail record inherits the sharing and security settings of its master record.
* The master-detail relationship field is required on the page layout of the detail record.
* By default, records can’t be reparented in master-detail relationships. Administrators can, however, allow child records in master-detail relationships on custom objects to be reparented to different parent records by selecting the Allow reparenting option in the master-detail relationship definition.

You can define master-detail relationships between custom objects or between a custom object and a standard object. However, the **standard object cannot be on the detail side of a relationship with a custom object**. In addition, you cannot create a master-detail relationship in which the **User** or **Lead** objects are the master.

## **Lookup Relationship (1: n)**

This type of relationship links two objects together but has no effect on deletion or security. Unlike master-detail fields, lookup fields are not automatically required. When you define a lookup relationship, data from one object can appear as a custom related list on page layouts for the other object. See the Salesforce online help for details.

1. Child row not automatically deleted when a parent row is deleted
2. No inherited sharing.
3. 25 lookup relation relationships allowed per object.
4. Lookup field on child not necessarily required.

## **Many-to-Many**

You can use master-detail relationships to model *many-to-many* relationships between any two objects. A many-to-many relationship allows each record of one object to be linked to multiple records from another object and vice versa.

To create a many-to-many relationship, simply create a custom junction object with two master-detail relationship fields, each linking to the objects you want to relate.

## **Interview Questions about object relationships in Salesforce:**

**How many relationships included in SFDC & What are they?**

We are having three types of relationships, they are

* Lookup Relationship
* Master-Detail Relationship
* Many-to-Many relationship

**What is a “Lookup Relationship”?**

Up to 25 allowed for object

Parent is not a required field.

No impact on a security and access.

No impact on deletion.

Can be multiple layers deep.

Lookup field is not required.

**What is “Master-Detail Relationship”?**

**Master Detail relationship** is the Parent child relationship. In which Master represents Parent and detail represents Child. If Parent is deleted, then Child also gets deleted. **Rollup summary** fields can only be created on Master records which will calculate the SUM, AVG, MIN of the Child records.

Up to 2 allowed to object.

Parent field on child is required.

Access to parent determines access to children.

Deleting parent automatically deletes child.

A child of one master detail relationship cannot be the parent of another.

Lookup field on page layout is required.

**Does an object can have both relationships (Lookup Relationship & Master Detail Relationship) at a time?**

Yes, single object can have both relationships at a time.

**What is a “Self-Relationship”?**

A self-Relationship is a lookup relationship to the same object.

Suppose let’s take an object “Merchandise”. Here we can create relationship in between the Merchandise to Merchandise (same object) object. That is called “Self-Relationship”.

**What is “Many to Many Relationships”?**

Allow for the relationship of two objects in a many-to-many fashion.

**How we achieve the “Many-to-Many Relationship”?**

By using Junction Object we can achieve this relationship, here junction object is having Master- Detail Relationship with different objects (Ex. Students & Courses). Using this Master to Detail Relationship, we can create the Many-to-Many Relationship in between the objects.

**What are the main things need to consider in the “Master-Detail Relationship”?**

Record level access is determined by the parent,

Mandatory on child for reference of parent, cascade delete (if you delete the parent, it can cascade delete the child).

**What are the main things need to consider in “Lookup Relationship”?**

Loosely coupled, not going to have all the above rules which are existed in master detail relationship.

*Self-relationship is always lookup to self relation object type. In many to many we actually create a junction object in the middle, it does lookup for the bound objects that are ultimately being bounded in a many to many ways.*

**Can we convert the lookup relationship to Master Detail relationship?**

Yes, we can convert the lookup relationship to master detail relationship only if all the existing record has valid lookup field values.

**Can we create Master Detail relationship on existing records?**

No. first we have to create the lookup relationship then populate the value on all existing record and then convert it.

### **REST API**

REST API is a simple and powerful web service based on RESTful principles. It exposes all sorts of Salesforce functionality via REST resources and HTTP methods. For example, you can create, read, update, and delete (CRUD) records, search or query your data, retrieve object metadata, and access information about limits in your org. REST API supports both XML and JSON.

Because REST API has a lightweight request and response framework and is easy to use, it’s great for writing mobile and web apps.

### **SOAP API**

SOAP API is a robust and powerful web service based on the industry-standard protocol of the same name. It uses a Web Services Description Language (WSDL) file to rigorously define the parameters for accessing data through the API. SOAP API supports XML only. Most of the SOAP API functionality is also available through REST API. It just depends on which standard better meets your needs.

Because SOAP API uses the WSDL file as a formal contract between the API and consumer, it’s great for writing server-to-server integrations.

### **Bulk API**

Bulk API is a specialized RESTful API for loading and querying lots of data at once. By lots, we mean 50,000 records or more. Bulk API is asynchronous, meaning that you can submit a request and come back later for the results. This approach is the preferred one when dealing with large amounts of data. There are two versions of Bulk API (1.0 and 2.0). Both versions handle large amounts of data, but we use Bulk API 2.0 in this module because it’s a bit easier to use.

Bulk API is great for performing tasks that involve lots of records, such as loading data into your org for the first time.

### **Streaming API**

Streaming API is a specialized API for setting up notifications that trigger when changes are made to your data. It uses a publish-subscribe, or pub/sub, model in which users can subscribe to channels that broadcast certain types of data changes.

The pub/sub model reduces the number of API requests by eliminating the need for polling. Streaming API is great for writing apps that would otherwise need to frequently poll for changes.

### **REST API:** REST Resources and Methods

A REST resource is an abstraction of a piece of information or an action, such as a single data record, a collection of records, or a query. Each resource in REST API is identified by a named Uniform Resource Identifier (URI) and is accessed using standard HTTP methods (HEAD, GET, POST, PATCH, DELETE). REST API is based on the usage of resources, their URIs, and the links between them.

You use a resource to interact with your Salesforce org. For example, you can:

* Retrieve summary information about the API versions available to you.
* Obtain detailed information about a Salesforce object, such as Account, User, or a custom object.
* Perform a query or search.
* Update or delete records.

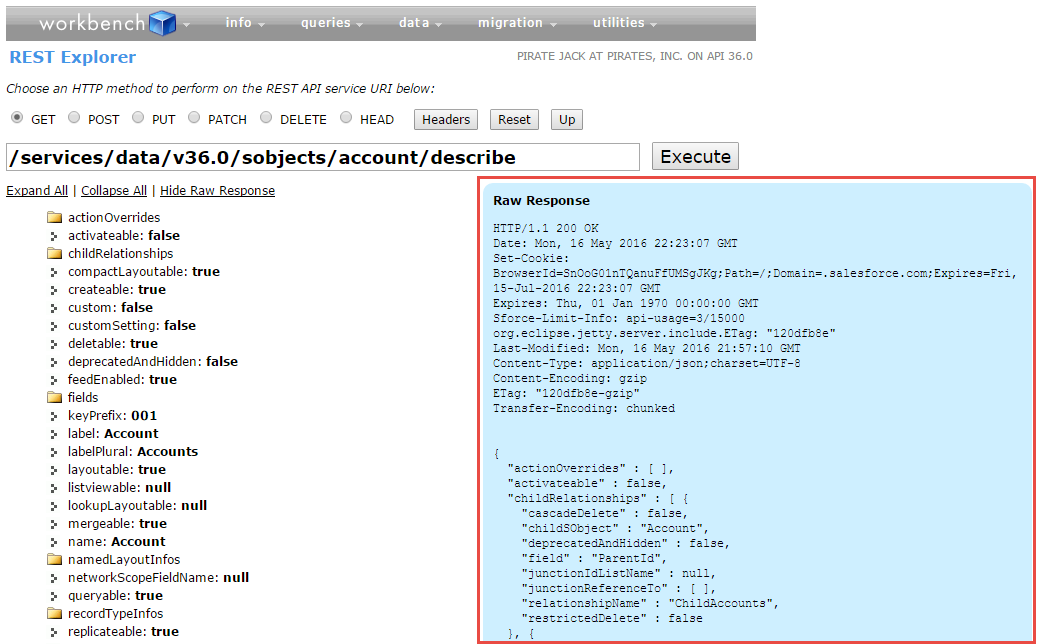
A REST request consists of four components: a resource URI, an HTTP method, request headers, and a request body. Request headers specify metadata for the request. The request body specifies data for the request, when necessary. If there’s no data to specify, the body is omitted from the request.

We have a URI here: /services/data/v43.0/sobjects/account/describe

where v43.0 maps to the API version you’re using.

Let’s break down this resource’s URI.

* /services/data—Specifies that we’re making a REST API request
* /v36.0—API version number
* /sobjects—Specifies that we’re accessing a resource under the sObject grouping
* /account—sObject being actioned; in this case, account
* /describe—Action; in this case, a describe request



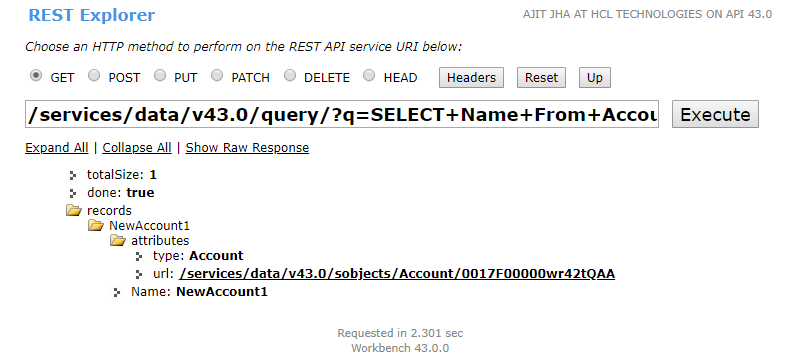
The Account metadata is displayed in JSON below some HTTP response headers. Because REST API supports both JSON and XML, let’s change the request header to specify an XML response. Next to the HTTP methods, click **Headers**. For the Accept header value, replace application/json with application/xml. Your request headers look like this.



## **Executing a Query**

URI text box with the following text:

/services/data/v43.0/query/?q=SELECT+Name+From+Account+WHERE+ShippingCity='San+Francisco' where XX maps to the API version you’re using.



### **Bulk API:** Bulk API and Asynchronous Requests

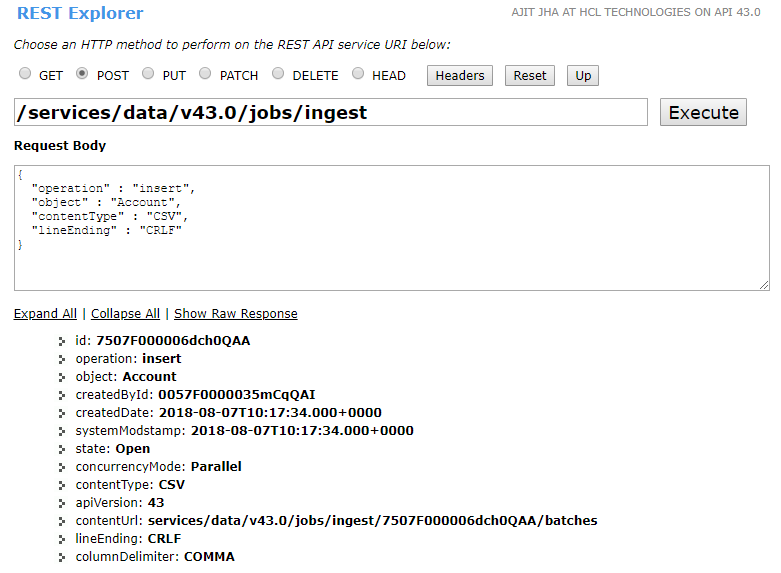
Bulk API is based on REST principles and is optimized for working with large sets of data. You can use it to insert, update, upsert, or delete many records asynchronously, meaning that you submit a request and come back for the results later. Salesforce processes the request in the background.

In contrast, SOAP and REST API use synchronous requests and are optimized for real-time client applications that update a few records at a time. You can use both of these APIs for processing many records, but when the data sets contain hundreds of thousands of records, they’re less practical. Bulk API’s asynchronous framework is designed to make it simple and efficient to process data from a few thousand to millions of records.

To create a job, we submit a POST request to /jobs/ingest with the job’s properties in the request body. Because Bulk API is REST-based, the request takes the familiar form of a REST request with four components: URI, HTTP method, headers, and body. The method is POST, as we just mentioned.

For the URI, replace the text in the URI text box with the following: /services/data/v*XX*.0/jobs/ingest, where *XX*.0 corresponds to the API version you’re using. Let’s note a few things about this URI.

* We’re using /services/data, which is the same endpoint used for the REST API. Bulk API uses the same framework that the REST API uses, which means that Bulk API supports many of the same features, such as OAuth authentication.
* /jobs/ingest indicates that we’re accessing the resource for creating Bulk API jobs.



**Sales Cloud: -**  
“When we develop product in force.com for sales then it comes in Sales Cloud Ex: - Account, Contacts, and Lead”.  
  
In Other Words “we can say that The Sales Cloud from salesforce.com is the world's #1 sales application. It gives reps, managers, and execs everything they need to connect with customers and focus on what's important —more selling and less administration. It helps you close more deals - faster, gain real-time visibility into sales, and connect with today's social customers”.

**Service Cloud: -**When we want to provide some facility and also provides support to the clients then it comes in Service Cloud. Ex: - create cases is the example of Service Cloud in which client write his problem into cases instead of call.  
   
In Other words “In service cloud when we sales product to client then we provides the many different types of facilities to client which will be helpful to client just like Call Center.  
In call Center ,client can make a call and ask his problem regarding product and can find the better solution about his problem“.

**Difference between Sales Cloud and Service Cloud:-**   
**1**. "Sales Cloud" refers to the "sales" module in salesforce.com. It includes Leads, Accounts, Contacts, Contracts, Opportunities, Products, Price books, Quotes, and Campaigns (limits apply). It includes features such as Web-to-lead to support online lead capture, with auto-response rules. It is designed to be a start-to-end setup for the entire sales process; you use this to help generate revenue.

While “Service Cloud" refers to the "service" (as in "customer service") module in salesforce.com. It includes Accounts, Contacts, Cases, and Solutions. It also encompasses features such as the Public Knowledge Base, Web-to-case, Call Canter, and the Self-Service Portal, as well as customer service automation (e.g. escalation rules, assignment rules). It is designed to allow you to support past, current, and future clients' requests for assistance with a product, service, and billing. You use this to help make people happy.  
  
**2.** Sales Cloud Implements Sales and Marketing while Service cloud implements Salesforce Knowledge.  
  
**3**. Sales Cloud ,A great solution for small and value oriented mid-sized sales groups that want to rapidly and cost effectively deploy Salesforce   While Service Cloud   provides Customer Support to the Clients and giving you the tools to provide a better customer experience for your clients.

**4**. Sales Cloud gives you the ability to open Cases (issues) and relate them to Accounts, Contacts; etc. While The Service Cloud is a superset of Sales Cloud, meaning you get everything that is in Sales Cloud PLUS some other features.

**5**. When we develop product in force.com for sales then it comes in Sales Cloud Ex: - Account, Contacts, and Lead.  While when we want to provide some facility and also provides support to the clients then it comes in Service Cloud. Ex: - create cases is the example of Service Cloud in which client write his problem into cases instead of call.

# Component Bundles

A component bundle contains a component or an app and all its related resources.

| **Resource** | **Resource** Name | Usage | See Also |
| --- | --- | --- | --- |
| Component or Application | sample.cmp or sample.app | The only required resource in a bundle. Contains markup for the component or app. Each bundle contains only one component or app resource. | [Creating Components](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_overview.htm)  [aura:application](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/ref_aura_application.htm) |
| CSS Styles | sample.css | Contains styles for the component. | [CSS in Components](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_css.htm) |
| Controller | sampleController.js | Contains client-side controller methods to handle events in the component. | [Handling Events with Client-Side Controllers](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/js_client_side_controller.htm) |
| Design | sample.design | File required for components used in Lightning App Builder, Lightning pages, Community Builder, or Cloud Flow Designer. | [Lightning Component Bundle Design Resources](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_config_for_app_builder_design_files.htm) |
| Documentation | sample.auradoc | A description, sample code, and one or multiple references to example components | [Providing Component Documentation](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_documentation.htm) |
| Renderer | sampleRenderer.js | Client-side renderer to override default rendering for a component. | [Create a Custom Renderer](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/js_renderers.htm) |
| Helper | sampleHelper.js | JavaScript functions that can be called from any JavaScript code in a component’s bundle | [Sharing JavaScript Code in a Component Bundle](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/js_helper.htm) |
| SVG File | sample.svg | Custom icon resource for components used in the Lightning App Builder or Community Builder. | [Configure Components for Lightning Pages and the Lightning App Builder](https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_config_for_app_builder.htm#components_config_for_app_builder) |

All resources in the component bundle follow the naming convention and are auto-wired. For example, a controller <componentName>Controller.js is auto-wired to its component, which means that you can use the controller within the scope of that component.