**Tables and Fields Lesson**

What is the ServiceNow infrastructure?

Data in ServiceNow is stored and managed according to a database structure that administrators can view and configure:

|  |  |
| --- | --- |
| • | **Tables** are a data structure or database component, which contain records. |
| • | **Records** are the data stored on tables, which contain fields. |
| • | **Fields** are individual pieces of data within a record. |

Tables can be accessed using the following modules within the **System Definition** application:

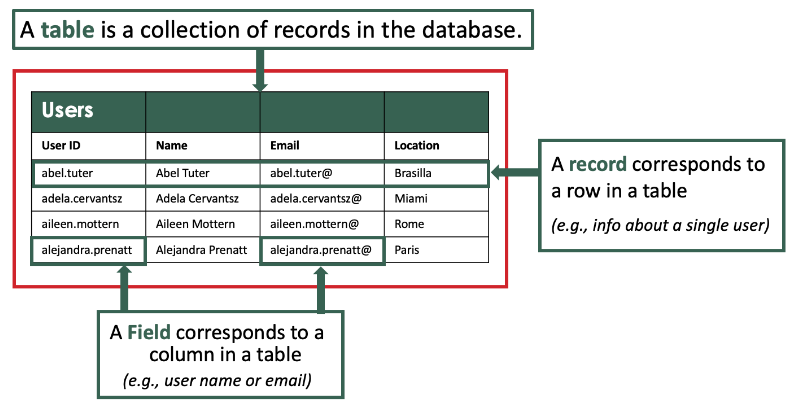
|  |  |
| --- | --- |
| • | **Dictionary** - defines every table and field in the system. Table records are identified as a Collection type. |
| • | **Tables** - contains a record for each table in the database. Custom tables can be created when the New button is created. |
| • | **Tables & Columns** - lists existing tables in the database. Selecting a table name displays its contents. |

The System Dictionary contains the **definition for each and every table and field in the database**. Navigate to **All >** **System Definition > Dictionary**to access the system dictionary to modify table and field attributes.

Table components

A table is a **collection of records** in the database into which information can be entered.

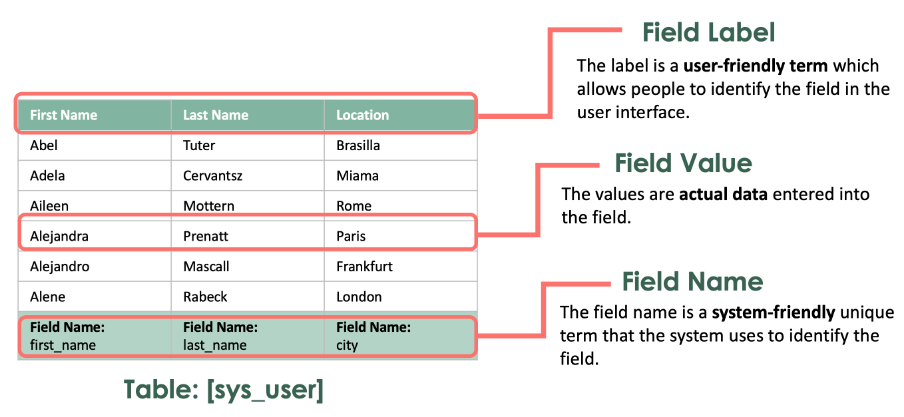
|  |  |
| --- | --- |
| • | Each record corresponds to a **row** in a table. |
| • | Each **field** on a record corresponds to a column on that table. |



Applications use tables and records to manage data and processes.

|  |  |
| --- | --- |
| • | **Records**: Are identified by a 32-character, globally unique ID, called a **sys\_id**. Record numbers are automatically incremented, and the number format per table in the system can be changed by visiting the **All >** **System Definition > Number Maintenance** application. |
| • | **Fields**: Store the actual data. They are available in a variety of different field types, such as: Choice, Date/Time, Journal, Reference, and more. Field types define how a field is interacted with through the interface, as well as the type and format of data it can store. |

Field attributes



**Field Label**

A user-friendly term for the field.  
  
In this example, the Field Labels are: **User ID**, **Name**, and **Location**.

**Field Name**

A unique term for the field. It is very important to note the Field Name does NOT always match the Field Label.  
  
For example:

|  |  |
| --- | --- |
| • | The **Field Name** of the **Location** field on the **User** table is **sys\_user.city**, whereas its **Field Label** is **Location**. |
| • | The **Field Name** of the **Name** field on the **User** table is **sys\_user.first\_name**, whereas its **Field Label** is **First Name**. |

In the example above, **[sys\_user]**is a table reference to the **User** table and anything that comes after the dot**[.]**is the **Field Name** (a unique identifier).

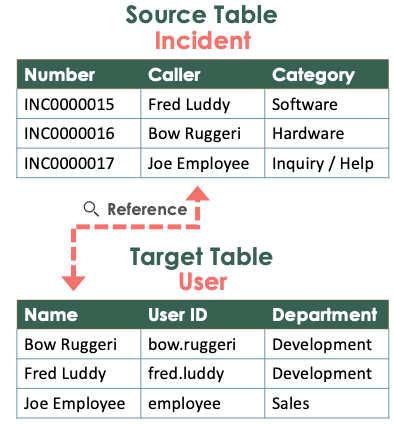
**Value**

The actual data stored in the field, such as this user’s name, Aileen Mottern, or her location, Rome. In some cases, the value may be empty, or null.  
  
In this example the Name field displays a combination of the **First name** and **Last name** fields of the **User** record.

**Reference fields**

A reference field stores a unique system identifier *(known as the sys\_id)* of a record on another table which is what establishes the reference relationship. For example, the Caller field on the Incident table is a reference to a record on the User table.

When you define a reference field, the platform creates a relationship between the two tables. Adding a reference field to a form makes the other fields in the referenced table available to the form.

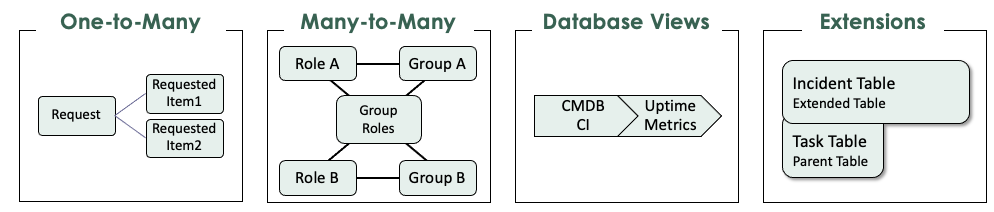


Administrators can create new reference fields and configure several options for reference fields.

Reference fields are identified with the **Reference Lookup** icon. When selected, the reference lookup icon opens a dialog box for locating a record to reference, presented as a list of the referenced (target) table. Wildcard searches can be used to locate a record.

If a record is specified in the reference field on the source table, you can hover over the **Information** icon to preview the referenced record (on the target table).

**Table Relationships Lesson**



One-to-Many

Within a table, a field can hold a reference to a record on another table. There are three one-to-many relationship fields:

|  |  |
| --- | --- |
| 1. | **Reference Fields** - Allows a user to select a record on a table defined by the reference field.  *(Example: The Caller field on the Incident table allows a user to select any record on the User table.)* |
| 2. | **Glide List** - Allows a user to select multiple records on a table defined by the glide list.  *(Example: The Watchlist field on the Incident table allows the user to select any record or records on the User table.)* |
| 3. | **Document ID Fields** - Allows a user to select a record on any table in the instance.  *(Example: Document field on the Translated Text table.* |

**Many-to-Many**

Two or more tables can be related in a bi-directional relationship, so that the related records are visible from both tables in a related list. Think ahead! Your data model is important. Plan so that you get the model right the first time. A common example of a many-to-many relationship is one between vendors and products. Software vendors can sell multiple products and products can be sold by multiple vendors. If you are working with many-to-many relationships in ServiceNow, you may find sys\_collections.list and sys\_m2m.list useful.

**Database Views**

Two tables can be joined virtually to allow for reporting on data that might be stored in more than one table. Certain views are included in the base system with the Database Views and Database Views for Service Management plugins.  
  
Database Views are read-only.

**Extensions**

A table can extend another table. The extended table includes unique fields plus all the fields and their properties from the parent table.

**Table Types Lesson**

When creating a new table, one of the first decisions a System Administrator or Developer must make is whether to create a table that stands alone, or a table that **extends** another table.

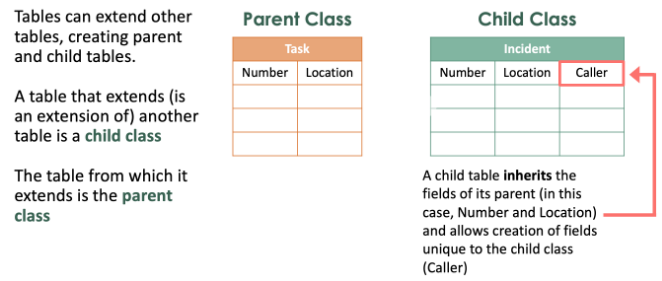
|  |  |
| --- | --- |
| • | For tables that stand alone, only the **global default**fields are automatically created. |
| • | Extending a table incorporates all of the fields of the original table and allows for unique fields to be created on the new table. The child table inherits the fields of its parent and enables the creation of subcategories of data. |

A table that extends another table is called a **child class**.

The table it extends is the **parent class**.

For example, the **Task [task]** and **Configuration Item [cmdb\_ci]** tables are parent classes that are extended to child classes:

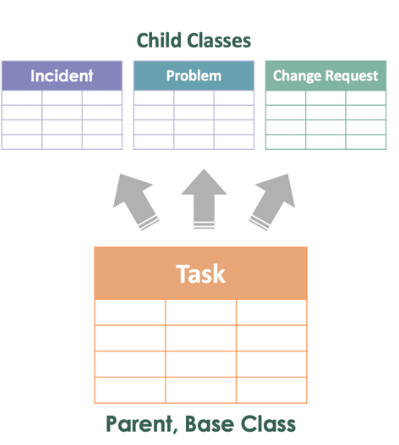
|  |  |
| --- | --- |
| • | Child tables extended from Task [task] include Change Request, Incident, and Problem. |
| • | Child tables extended from Configuration Item [cmdb\_ci] include Database, Hardware, and Software. |



**Table labels**

A different label can be defined for each extended table, such as Incident, Problem, or Change Request for example.

To add a different label for an extended table, navigate to **All >** **System Definition > Language File**. Then create a new entry for the extended table.



**Permission to extend**

A table's **Extensible** field is used to control whether a table can be extended.

|  |  |
| --- | --- |
| • | If it is set to **true** (selected/checked), the table can be extended. |
| • | If it is set to **false** (NOT selected/checked), the table cannot be extended. |

**Dictionary**

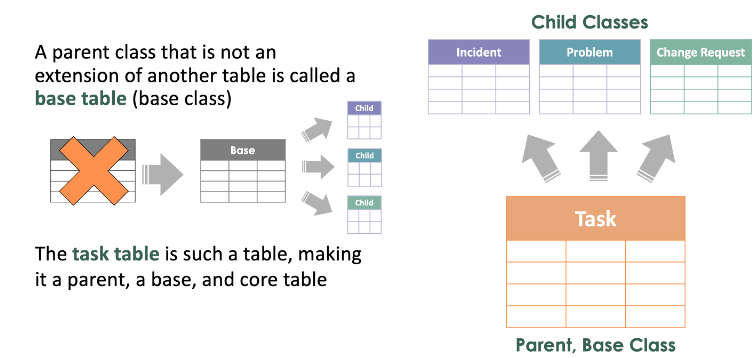
Using the **Dictionary overrides** feature provides the ability to define a field on a child table differently from the field on the parent table.

Examples includes overriding the default values, field dependencies, or read-only status of a field.

**Base tables**

If a table is extended but itself is not extending another table, it is called a **base** table. Every **child** table is a specialization of its base table or previous child table.

The **Task [task]** table is such a table, making it both a base class *(not extending another table)*, and a parent class *(extended)*.



A table can be both a parent and child class both extending and providing extensions for other tables. A parent class that is not an extension of another table is called a base class.

**Understanding Base vs. Core Tables:**

**A Core Table** is a table that exists in the ServiceNow base system.  In other words, core tables come WITH the system, they are there from the start. This is probably the easiest distinction of this table type. If it is a table that comes with ServiceNow, it’s a core table. But it is important to understand that a core table can also be a parent table (e.g., Task), a child table (e.g., Incident) or a base table (e.g., Task).

**A Base Table** is a table that serves as a “base” from which other tables may extend. From the base table, you can establish parent/child relationships in the database. The child (extended) table includes unique fields PLUS all of the fields and properties which were inherited from the parent table. In the previous example, the Incident table (a child) of the Task table (parent), inherited the fields Number and Location from the parent and then added its own unique field Caller. What makes a base table different from a core table is that the base table is not an extension of another table (it has no parent).  A base table can also be considered a parent and/or a core table depending on the circumstances.

**In the Task table example above, the Task table is:**

(a) a core table because it comes with the ServiceNow base system

(b) a base table because it’s not extended from any other table (no parent of its own), and

(c) a parent table because it has children (child classes of Incident, Problem, and Change Request)

Depicted in the image, we see that the Task table is not only a parent table (because it has children—Incident, Problem, and Change Request), but it is also a base table because the task table itself is not extended (not an extension/child of any other table), AND the task table is also a core table because it is part of the ServiceNow base system. The task table is an interesting example of how a table can be a base table, a parent table, and a core table.

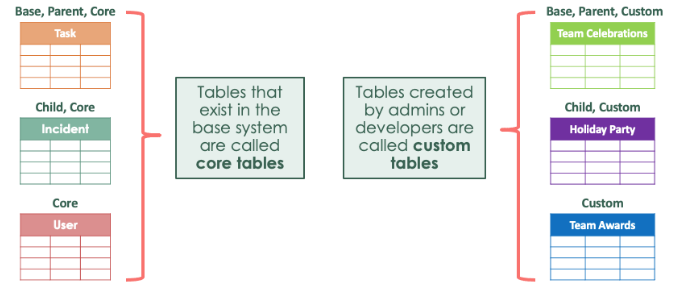
**Core tables vs. custom tables**

**Core**tables are created by ServiceNow and provided with the base system.

**Custom** tables are created by you.

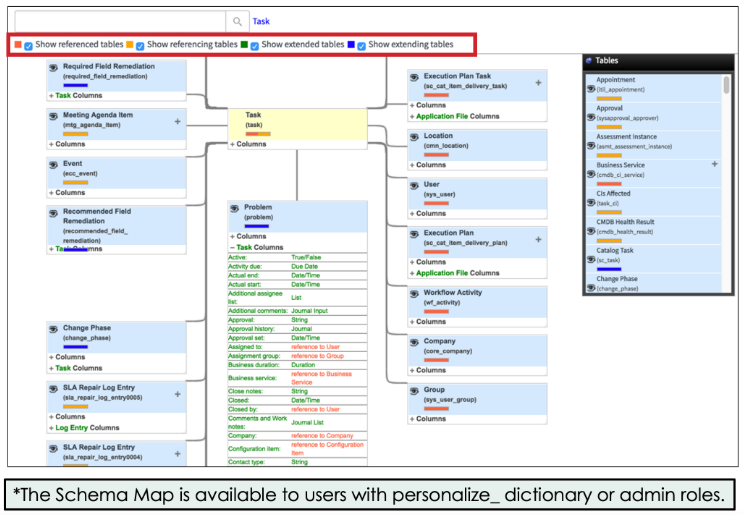
Although custom tables are not in the base system, they can still interact with existing core tables or other custom tables.

For example, a reference field on a custom table can access data stored on a core table. By doing so, a relationship between the tables is created which makes them related tables. This relationship is not exclusive between just a custom table and a core table. Related tables can be a combination of multiple core tables and/or multiple custom tables.



**Schema Map Lesson**

The **schema map** provides a graphical representation of other tables related to a specific table.  
  
Relationships can be filtered by extension or reference classes by checking the appropriate boxes at the top of the map.



In this example map:

|  |  |
| --- | --- |
| • | The Task [task] table has the focus of the map (highlighted in yellow). |
| • | Tables with **blue bars** *(e.g. Problem)*, are tables that extend the Task table. Demonstrated in this image with the *Problem* table, you can use the Schema Map to:   |  |  | | --- | --- | | ◦ | Identify which fields originate on the *Problem* table, and which columns are inherited from the Task table. | | ◦ | Look-up field types. | |
| • | Tables with **red bars**, *(e.g. Location and User)*, are tables that are referenced by the *Task* table. |
| • | A series of filters at the top of the schema map allow you to show/hide tables based on criteria such as whether they are referenced by the Task table, reference the Task table, are extended by the Task table, or extend the Task table. |
| • | A **Tables** window on the far right of the screen provides a summary of all the tables presented and their relationships. |

**To generate a schema map**:

|  |  |
| --- | --- |
| 1. | Navigate to **All > System Definition > Tables & Columns**. |
| 2. | In the Table Names pane, **select a table**. |
| 3. | Select **Schema map**. The schema map for the selected table opens in a separate tab or window. |

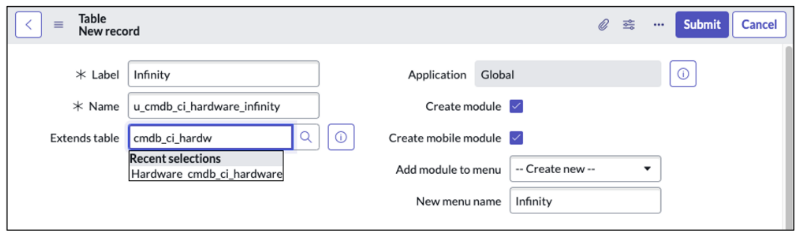
**Data Schema**

1. Create a New Table

|  |  |
| --- | --- |
| 1. | ***Note:****Before creating any table in ServiceNow, the question to always start with is:****Should the table be created from scratch or extended from an existing table?***  Navigate to **All > System Definition > Tables** to create a new table. |
| 2. | From the list header, click the **New** button. |



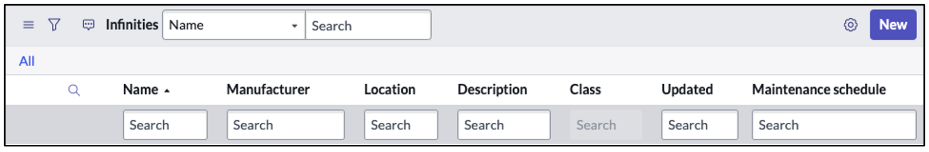
|  |  |
| --- | --- |
| 3. | Populate the top of the Table form with the following information:   * Label: **Infinity** * Name: **u\_cmdb\_ci\_hardware\_infinity** * Extends table: **Hardware [cmdb\_ci\_hardware]** * New menu name: **Infinity** *(auto fills)* |



1. Add Fields to the Infinity Form

1.Navigate to All > Infinity > Infinities.

A list with ***No records to display*** is shown with default fields *(column headers)*.



|  |  |
| --- | --- |
| 2. | Click**New** to open a form displaying default fields. |
| 3. | Open the **Form Context Menu**, select **Configure > Form Layout**. |
| 4. | Remove the **Assigned to**, **Category** and **Fault count**fields, keep **Name**, **Asset tag**, **Installed**, and **Status** in the *Selected* container.    5.Add the Owned by and Support group fields to the Selected list.  6. Using dot-walking, add Owned by.Email to the Selected list.  a) From the Available list, locate and select Owned by  b) Click the Expand selected reference field icon. |
|  | c)  Locate the .*Owned by-->User fields* and add **Email** to the Selected list.  You should now see **Owned by.Email** under the Selected list. |

|  |  |
| --- | --- |
| 7. | In the **Create new field**section, populate information as follows:      Name: **Device Number**     Type: **String** *(auto fills)*     Field length: **Small (40)** *(auto fills)* |
| 8. | Click **Add**.  9. Add the two additional fields shown below using the Create new field section.  10. After adding the new fields, rearrange the fields under the Selected list to appear in this order.  11. Click the Save button.  12. Close the Saving Form Section window.  13. The Infinity New record form displays |
|  |  |

C. Update the Infinity Application Menu

|  |  |
| --- | --- |
| 1. | In the Filter navigator, type **Infinity**. |
| 2. | Navigate to **All > Infinity**. Hover your curser over **Infinity**and click the **Edit Application** icon. |

3. Notice, this brings up the Application Menu record for Infinity.

NOTE: You could also access this record by navigating to All > System Definition > Application Menus and searching for Infinity in the list.

4. Update the Title to Infinity Inventory.

|  |  |
| --- | --- |
| 5. | **Save** the record to remain on the form. |
| 6. | Navigate to **All > System Definition > Modules.** Search for and open the **Infinities**record. |
|  |  |

|  |  |
| --- | --- |
| 7. | Update the record with the following information:      Title: **All Devices**     Order: **200** |

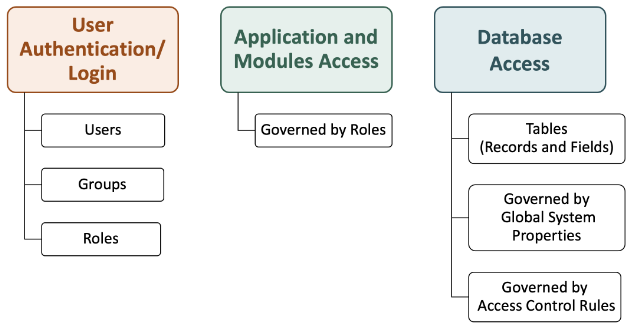
1. Configure a New Infinity Module

|  |  |
| --- | --- |
| 1. | Click**New**from the Modules section.  2. Populate the form using the following information:   * Title: Add Inventory * Application menu: Infinity Inventory * Order: 100   3. Click the Link Type tab and populate the fields as shown:   * Link type: New Record * Table: Infinity [u\_cmdb\_ci\_hardware\_infinity] |
|  | 4. Verify the **Title**, **Order**, **Link type**, and **Table** are complete, as shown. |

**Access Control Overview Lesson**

ServiceNow provides several levels of security before an end user has the capability to perform CRUD (Create, Read, Update, Delete) operations on a table:

|  |  |
| --- | --- |
| • | **User Authentication/Login**: Users, Groups, and Roles. |
| • | **Application and Modules Access**: Controlled by roles configured at the Application and Module level. |
| • | **Database Access**: Access to tables, records, and fields is controlled via globally defined system properties; as well as table and field level Access Controls |



There are three security modules on the **All menu** typically used by the System Administrator:

|  |  |
| --- | --- |
| • | **System Properties > Security** |
| • | **System Security > Access Control (ACL)** |
| • | **System Security > High Security Settings** |

An **access control** is a security rule defined to restrict the permissions of a user from viewing and interacting with data. Most security settings are implemented using **access controls**.  
  
It is executed when access to any ServiceNow table is attempted. An access control may be set at the **row** or **column** level.

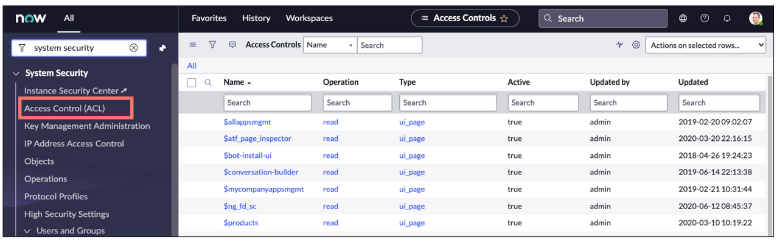
In addition to restricting CRUD operations, access control rules can restrict ServiceNow-specific operations on tables and fields.

ServiceNow operation examples include:

|  |  |
| --- | --- |
| • | **execute**: user cannot execute scripts on a record or UI page. |
| • | **edit\_ci\_relations**: user cannot define relationships between Configuration Item [cmdb\_ci] tables. |
| • | **save\_as\_template**: controls the field that should be saved when a template is created. |
| • | **report\_on**: user cannot create reports on the object. |
| • | **personalize\_choices**: user cannot right-click a choice list field and select Configure Choices. |

The access control list (ACL) contains all of an instance’s access control rules.

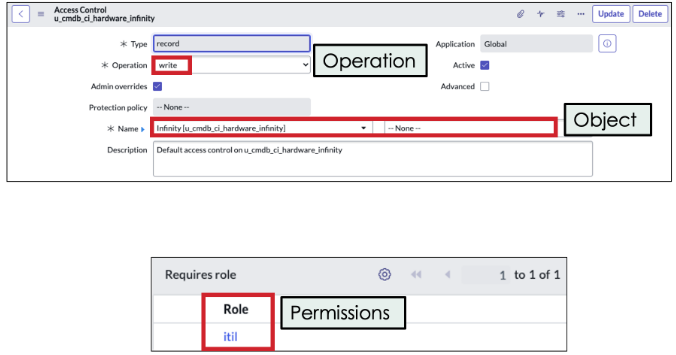
Users with the appropriate permissions can modify rules and their definitions.



Users with the **admin** role have special access to all platform features, functions, and data because admins can override access controls and pass all role checks so grant this privilege carefully! With this said, in order for a user to create or update access control roles, they must have the **security\_admin** role.

Each access control rule specifies:

|  |  |
| --- | --- |
| 1. | The object being secured *(e.g. table, field)*. |
| 2. | The permissions required to access the object.   |  |  | | --- | --- | | • | Roles | | • | Conditional Expressions | | • | Scripts | |
| 3. | Operation - a valid action the system can take (CRUD) |
|  |  |



When a custom table is created, the system creates**four access control rules by default**:  
  •  create  
  •  delete  
  •  read  
  •  write

A **role** is also created by default and associated with these access control rules.

With system-created access controls, it is rare to have a table with no access control rules at all. This will be important to remember when you review how the rules are evaluated in the next lesson.

To view the access controls associated with a table:

  1.  Type **<table name>.config** in the Filter navigator.  
  2.  Select the **Access Controls** tab.

When you performed the **Create an Extended Table** activity in the previous lesson, you created the **u\_cmdb\_ci\_hardware\_infinity** table. In addition to the table, the following were also automatically created:

  •  The **u\_infinity\_user** role.  
  •  Table **Access Control Rules** *(create, delete, read, write).*

**Access Control Evaluation**

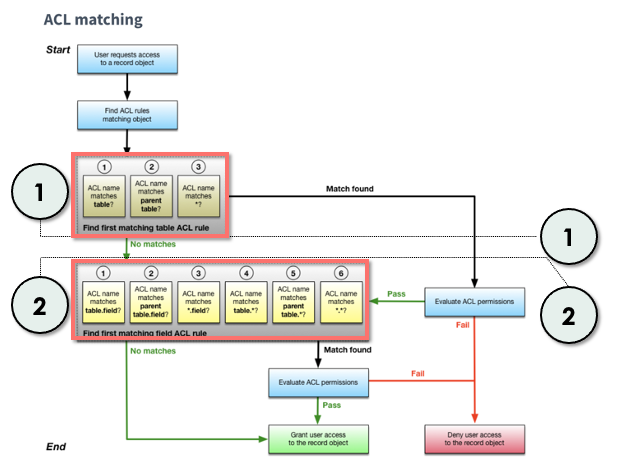
**Table access control evaluations**

When a session requests data, the system looks for matching access control rules.

Record ACL rules are processed in the following order:

|  |  |
| --- | --- |
| 1. | Match the object against **table** ACL rules - *most specific to most general*. |
| 2. | Match the object against **field** ACL rules - *most specific to most general*. |

A user must pass both table and field ACL rules to access a record object.



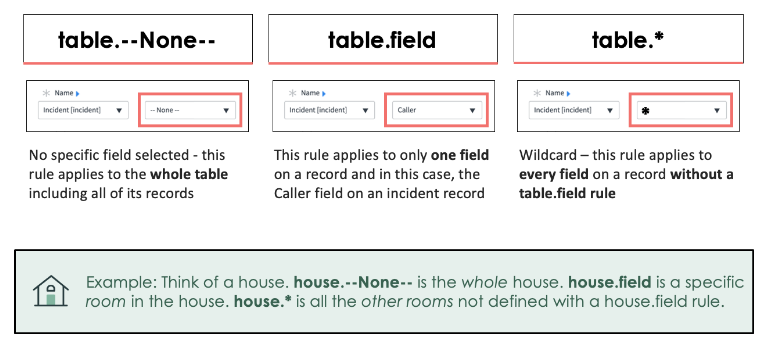
Access controls are evaluated in the order shown in the image above. This processing order ensures a user has access to the table *(most specific table rules to most general)* before evaluating access to the field *(most specific to most general)*.

|  |  |
| --- | --- |
| • | If a user fails a table access control rule, the user is denied access to all fields in the table, even if the user would pass a field ACL rule. |
| • | If a user passes a table ACL rule, but fails a field ACL rule, the user cannot access the field described by the field ACL rule. |

If there is a matching access control rule (which is almost guaranteed by system-created access controls), the system evaluates if the user has the permissions required to access the object and operation (role, conditions, script). If an access control rule specifies more than one permission, then the user must meet all permissions to gain access to the object and operation. Failing any one permission check prevents the user from accessing the matching object and operation.

If a user does not meet the permissions of the first matching rule, the system evaluates the permissions of the next matching access control rule as specified by the access control processing order. If the user fails to meet the permissions of any matching access control rule, the system denies access to the requested object and operation.

**Access control definition: rule types**



Each access control specifies the table or type of record (including fields), operation being secured, and unique object identifier.

Access control rules are defined for and applied to a specific table so that the rule is within the context of the table and the type of data stored.

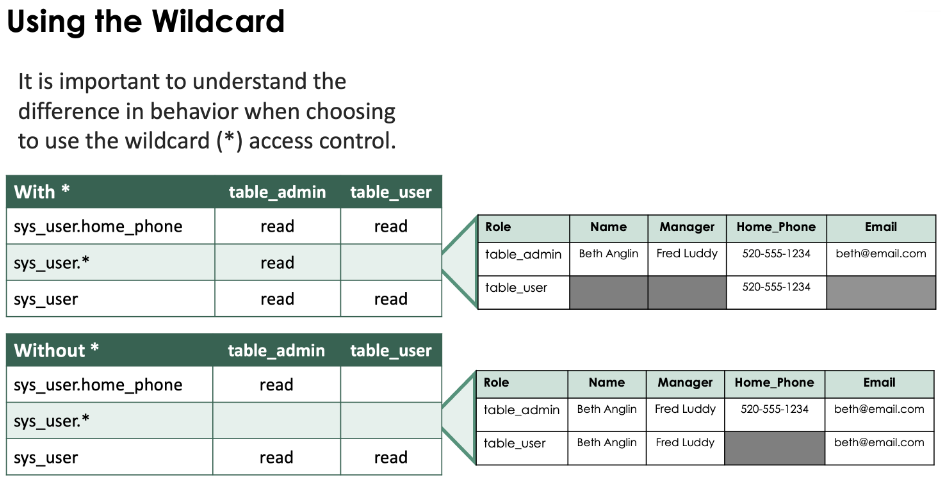
**table.--None--** applies to the entire table. Instead of a table in the platform, imagine a house. A house has rooms (fields on a record). The record **table.--None--** is **house.--None--** in our example.

**table.field** is one specific field on the table or a specific room in the house. In this example, **house.field**would represent a living room.

**table.\*** represents all other fields in the table or all other rooms in the house, not defined with a house.field rule (**house.\*)**. This method reduces the number of ACLs that need to be written for a table in the system, as it closes off any additional permissions.

To illustrate, when creating the rule and deciding the type (table.--none--, table.field, or table.\*),we’re really referring to the field options on the object. When you select --None--, notice the name of the access control. Notice that the read, create, write, and delete operations with no restrictions on the table appear as “u\_cmdb\_ci\_hardware\_infinity”, showing there is no restriction on any fields. Compare this with the rule that was created using the table.field designation restricting the Asset tag field. Notice that the name of the rule reflects the field “u\_cmdb\_ci\_hardware\_infinity.***asset.tag***.

By looking at the rule name, you can see there is a restriction specific to the Asset tag field. When we create a rule with no restrictions, the –none– is a bit misleading. It really means you’re selecting “nothing” and thereby not creating any restrictions on the table. When you create a restriction on the field, using either a specific field name or a wildcard, the name will change to reflect the restriction you’ve created.



**How does this work?**

In the first example (With \*):

|  |  |
| --- | --- |
| • | The None Access Control (which appears as just sys\_user) granted all rows and all fields to both the table\_admin and table\_user roles. |
| • | The .\* Access Control granted all rows and all fields to the table\_admin role. It seems redundant to have this Access Control because the table\_admin role already had access to all rows and all fields. The purpose of this Access Control is to deny access to all other roles, even roles granted permission by the None Access Control. |
| • | The .home\_phone Access Control explicitly gives the table\_user role access to the home\_phone field, even though table\_user was denied access to home\_phone by the \* Access Control. Why? Because field-specific Access Controls take precedence over .\* Access Controls. |

In the second example (Without \*):

|  |  |
| --- | --- |
| • | The None Access Control (which appears as just sys\_user) granted all rows and all fields to both table\_admin and table\_user roles. |
| • | The .home\_phone Access Control granted home\_phone access to the table\_admin role. Giving home\_phone explicitly to table\_admin removed home\_phone access from table\_user, even though the table\_user role was granted home\_phone access by the None Access Control. |

Best Practice:

|  |  |
| --- | --- |
| • | When creating .\* Access Controls, also create a None Access Control, because only None grants access to records. |
| • | When writing an ACL that mostly grants access, use only None. |
| • | When writing an ACL that mostly denies access, use None and .\*. |