*UDEMY*

**SERVICENOW DEVELOPMENT 201**

**COURSE NOTES**

GENERAL:

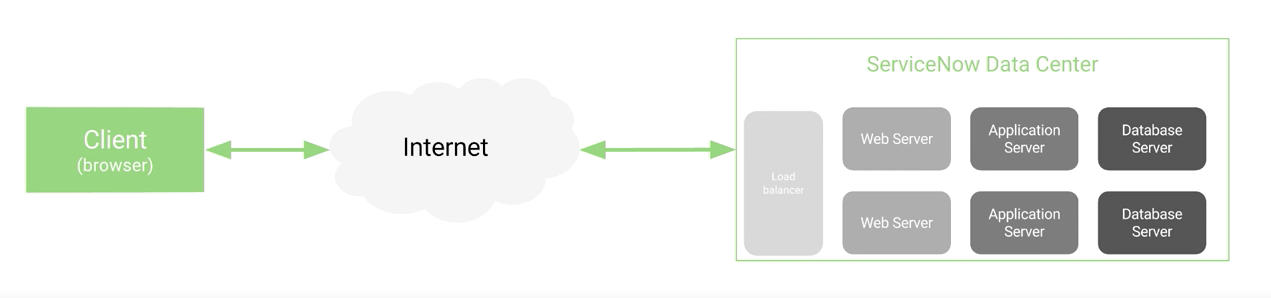
* A business rule will always be a business rule in ServiceNow, regardless of the version you’re using.
* Software-as-a-Service (SaaS) subscription model
* Each subscription includes:
  + URL: https://<instance-name>.service-now.com
  + Instance data
  + Application logic
  + Custom components
* Everything is a record in the database
* Out-of-box state
* Releases:
  + ~10-12 month release cycle
  + Feature release (new UI, new apps, new features)
  + patch release (problem fixes, includes a collection of hotfixes)
  + Hotfix (fix to a specific problem, implemented very quickly)
  + All named after cities: Istanbul, Jakarta, Helsinki, etc.
* Three main instances: dev, test, and production

DEVELOPMENT OVERVIEW:

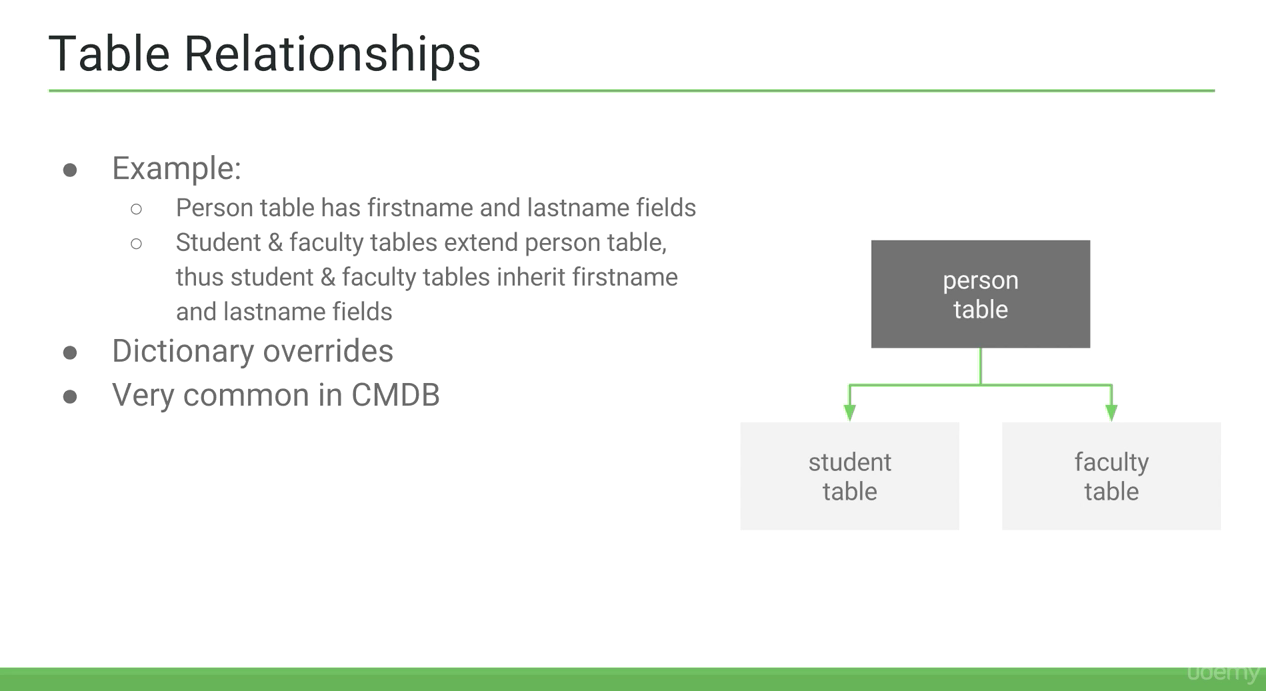
* Three main instances: dev, test, and production
  + Work is done on development environment
  + Changes are pushed to test, where users go though a user acceptance testing phase
  + After testing is complete, changes are pushed to production
  + Test should be as close to production as possible
  + Admins can clone instances, so if development instance is too different from production, a new development instance can be cloned to mirror production
* Update Sets: update sets are used to record **most** customizations and configurations



* + Used for moving changes between instances
  + Everything modified in ServiceNow is a modification of a table
  + Update sets are an XML snapshot of the last modified record
  + Update sets have versions, and you can merge 2 or more update sets into one
  + If two update sets modified the same record, and the update sets are merged, then the merged update set will contain the last modified record
  + When you load an update set to an instance, you can preview the update set before “committing” the update set
  + Previewing the update set before committing will help catch most compatibility issues and errors if they arise
* Things that are captured in update sets:
  + Customizations
  + Tables & fields
  + Reports
  + Workflows
  + Forms
* Things that ARE NOT captured in update sets:
  + Data, new records
  + CIs
  + Schedules
  + Users
  + Groups
* The ServiceNow Stack (list not exhaustive):
  + Apache Tomcat web server
  + J2EE application server
  + MySQL database
  + Mozilla Rhino JavaScript engine



* Table Overview:
  + Over 2,000 tables in every instance
  + Each application has 1 or many tables
  + Each table has many fields
  + Tables can extend other tables
  + Naming convention: My Custom Table => “u\_my\_custom\_table”
  + Admins can create/modify tables
* Table Relationships:
  + Children tables inherit attributes from the parent table
  + Dictionary overrides allow you to change the attribute in a specific child table
  + Tables can either be “one to many”, or “many to many”



* Major Tables:
* Task
* Incident
* Problem
* Change\_request
* Change\_task
* Sys\_user
* Sys\_user\_group
* Sys\_user\_role
* Cmn\_location
* Core\_company
* Kb\_knowledge
* Kb\_category
* Kb\_knowledge\_base
* Sc\_catalog
* Sc\_cat\_item
* Sc\_task
* Sc\_request
* Sc\_req\_item
* Cmdb\_ci
* cmdb\_ci\_server
* Schema map:
  + Visual schema map
  + Shows extended and related tables
  + Ability to *focus* on specific tables
* GUID:
  + Each record in ServiceNow is identified by a unique 32-character GUID (Globally Unique ID) called a sys\_id.
  + A GUID is a 32-character hexadecimal string
  + Used all throughout the system
  + Commonly paired with a table name to locate a specific record
* Databases, Tables & Fields (basic stuff):
  + A database contains many tables, and tables contain many fields
  + Records are stored in tables
* Records:
  + Stored in a database table
  + A single entity defined by a tables fields
  + Each record has a unique sys\_id
* Reference Fields:
  + Reference fields store a reference to a specific row in another table, similar to foreign keys in SQL
  + Gives flexibility to create relationships between records
  + Fields with magnifying glass are Reference Fields
  + sys\_id is stored in reference field
  + Reference fields must match an exact record
* Scripting Overview:
  + ServiceNow includes APIs called ‘Glide classes’
  + The ServiceNow Glide classes expose JavaScript APIs that enable you to conveniently work with tables using scripts. Using the Glide APIs, you can perform database operations without writing SQL queries, display UI pages, as well as define UI actions.
  + Scripting isn’t always necessary, it is best practice to avoid using scripting unless it is a necessity.
  + Customizing without scripting includes: UI policies, workflows, creating new tables and fields
  + Good for: creating custom integrations, client scripts, UI actions, complex transform maps, complex business rules, Service Portal widgets, and custom applications.
* Client Side:
  + Where: User’s browser
  + What: Makes request
  + Access to:
    - Current form, fields & values
    - UI elements (DOM)
    - Client-side APIs
* Server Side:
  + Where: ServiceNow data centers
  + What: Sends response
  + Access to:
    - Databases
    - Server-side APIs
    - Script includes
* JavaScript:
  + Version: Rhino - ECMAScript 5 (ES6 and ES7 not available in ServiceNow)
  + Access to ServiceNow API, the AngularJS framework, and jQuery among a few other libraries
  + Background scripts are a location in ServiceNow where server-side code can be run on-demand, similar to a browser’s console with access to the ServiceNow API
  + Caution: background scripts should be used with caution as it could cause performance issues and result in data loss
* ServiceNow Studio: IDE
  + Can only be used for scoped applications
  + Cannot be used on the global scope
  + Can only be used for custom applications within ServiceNow
* Important Concepts:
  + Naming conventions
    - ServiceNow: underscores
    - Vanilla JavaScript: camel casing
    - Generally, if creating a new variable in a script, use the camelcase convention
    - Use underscores if referencing an existing variable in ServiceNow
  + Most fields are objects, not strings
  + There’s a difference between the display value and the field value

BUSINESS RULES:

* Business Rules only run on the server side
  + A business rule is a server-side script that runs when a record is displayed, inserted, updated or deleted, or when a table is queried.
  + Most common scripting location
  + Server-side language: Mozilla Rhine (JavaScript runtime written in Java, since ServiceNow is written in Java)
  + 2,400 out-of-the-box business rules
* Client Scripts:
  + Client scripts run on the client (web browser). You can use client scripts to define custom behaviors that run when events occur, such as when a form is loaded or submitted or if a cell changes value.
  + JavaScript on the client side
  + Triggered when:
    - Field changes
    - Page loads
    - Form submissions
    - Cell edits
  + Client script ships to the browser when loaded
  + Form View:
    - Table: where the script is run
    - UI Type: Desktop, Mobile (Both is best practice)
    - Type: If script runs onChange, onLoad, etc.
    - Field Name: only shows when type is onChange. When that field name changes, the script will run.
    - Script: ServiceNow API is preferable to jQuery to manipulate the DOM directly
  + If the UI type is Desktop, you have access to some older, now deprecated Glide methods.
* UI Actions:
  + UI Actions add buttons, links, and context menu items on forms and lists, making the UI more interactive, customizable, and specific to user activities. UI Actions contain scripts that define user functionality.
  + Server-side or client-side
  + Typically configured for form views
* UI Policies:
  + Client side policies
  + Primarily used on forms.
  + Offer an alternative to client scripts for dynamically changing information on a from. Use UI Policies to set custom process flows for tasks.
  + Most of the time these don’t require scripting
  + Used to set forms to:
    - Read only
    - Mandatory
    - Show/Hide
* Script Includes:
  + Script includes are used to store JavaScript that runs on the server.
  + Create script includes to store JavaScript functions and classes for use by server scripts.
  + Each script include defines either an object class or a function.
  + Consider using script includes instead of global business rules because script includes are only loaded on request.
  + Features:
    - Server-side JavaScript
    - Stores JavaScript classes & functions/methods
    - Only runs when invoked
    - Unique since they can be called from anywhere (Client callable option)
  + 2 types:
    - Classless
      * Script Include name => function name
      * Server-side only
    - Class
      * Typically extend another class
      * Can be invoked on either server-side or client-side
  + Extending Script Includes:
    - Any class may be extended, which makes them very modular
    - Not tied a specific table or application; reusable
    - Script Includes can extend other Script Includes
    - AbstractAjaxProcessor is a commonly extended class used for GlideAjax, and it comes out-of-box. It provides helper functions that can call an extended Script Include from the client-side.
    - Syntax to extend another class:
    - Use Cases:
      * Case 1: Create commonly used helper functions
      * Case 2: Call a custom function via GlideAjax
  + Scheduled Jobs:
    - Scheduled jobs are automated pieces of work that can be performed either at a particular time, or on a recurring schedule.
    - Features:
      * Server-side JavaScript
      * Schedule when to run
      * **Execute Now** button for testing
    - Ability to schedule reports, scripts, charts, etc.
    - They can be scheduled once, on demand, daily/weekly/monthly, or periodically.
    - Use Case Examples:
      * Case 1: Schedule a monthly report
      * Case 2: Schedule a script to retire old records
  + Workflow Editor:
    - The workflow editor is an interface for creating and modifying workflows by arranging and connecting activities to drive processes.
    - Features:
      * Server-side JavaScript
      * Automated sequence of activities
      * Many locations to script a workflow
      * Different scopes
      * Versioning and checkout/publish features
    - Workflow contexts are created when workflow conditions evaluate to true
    - Versioning: allows a user to “checkout” an existing workflow, essentially creating a new, working version, and can be edited without affecting the currently published workflow in the system
    - Scripting in Workflows:
      * Only use scripts if out-of-box features aren’t enough.
      * Activities that support scripting:
      * Approval activities
      * Run Script activity
      * If, Switch, and Wait for condition activities
      * Create task activity
      * Notification activities
      * Scriptable Order Guide activity
      * REST Message, SOAP Message activities
    - Scope
      * Current record
      * Workflow.scratchpad object
      * Activity specific variables
      * Local variables
    - Workflow activities

GLIDE RECORD:

* Common GlideRecord Methods:
  + query()
  + newRecord()
  + insert()
  + update()
  + deleteRecord()
  + addQuery()
  + addEncodedQuery()
  + hasNext()
  + next()
  + get()
  + orderBy()
  + orderByDesc()
  + canCreate()
  + canWrite()
  + canRead()
  + canDelete()