MS CRM points to remember:

- **Switching Areas:** Moving between different sections of Microsoft Dynamics 365, such as Sales, Service, or Marketing.

- **Solutions:** A container for customizations and configurations in Dynamics 365. Used to manage, deploy, and transport customizations between environments.
    - U can't import a managed version of an unmanaged solution in the same environment as an unmanaged one.
    - **Types:**
        - Unmanaged- one where the developer works, Editable and modifiable after import.
        - Managed- sent for testing and UAT, Locked, cannot be edited after import.
        - Default: System-generated solution containing all customizations.
    - **Export and Importing Solution**- Moving a solution between different environments by exporting and importing.
    - **Cloning a solution**: creates a complete clone. Cloning is used when you want to create a new version of a solution.
        - When you have made multiple patches and need to combine them into a new version.
        - When you want to add new features but don't want to affect existing users.
        - Cloning removes the need for old patches.
        - If you don't clone the main solution, patches cannot be added after a certain point.
    - **Patching a solution**: A **patch** is a small update to an existing solution. Instead of replacing the entire solution, you only modify specific components. Patches are imported to the main solution only if the main solution is 'managed'
        - When you want to make a small fix or update without affecting other components.
        - When your main solution is **managed**, and you **need to update it without creating a new version.**
        - A patch does NOT work if the main solution is deleted!
        - Patches depend on the main solution and cannot be used separately.
    - Best Practice to make updates:
        - Import the managed version of the unmanaged solution into your test/production environment.
        - Make the necessary patches in the unmanaged solution in the development environment.

        - Export the patch as managed and import it into test/production.
        - If you need a major update, clone the solution and import it as a new version.

    - **Publisher**: Identifies the creator of a solution. Create your own publisher to make solutions.
    - **Prefix**: A unique identifier added to components for differentiation.

    - Managed Solutions are **locked-down** packages designed for deployment in production environments. This means deleting components inside them comes with limitations.
    - Managed solutions are meant to be **immutable** — they are deployed, not developed.
    - If you must delete a component that's part of a managed solution, you have **two options**:
        - Delete the Entire Managed Solution
        - Create a Patch or Update the Source Unmanaged Solution
    - System components (e.g., default tables like Account or Contact) cannot be deleted even from unmanaged solutions.
    - Always use Solution Layers to check if a component is part of another solution.
    - Use dependencies checker before deleting.

- **Model-Driven Apps:** an application that is data-driven and built using Dataverse tables (entities). It provides a structured UI based on relationships, forms, views, and business processes.
    - Sitemaps:
        - Navigation menu of a model-driven app that allows users to access different areas, tables, and dashboards.
        - Are customizable
        - Select Sitemap → Click on Edit Sitemap.
            - Add Areas, Groups, and Subareas:
                - Area: Main category (e.g., Sales, Service).
                - Group: Section within an area.
                - Subarea: Links to tables (entities), dashboards, or web resources.
    - **Ribbons** are the **command bars or toolbars** you see at the top of forms, views, subgrids, and dashboards.
        - Buttons for new commands can be added to use JS(for advanced logic) or Power-fx formulae (simple logic- enabling, disabling, etc)

- **Entity:** A database-like object that stores records in Dynamics 365.
    - A table

    - Types-
        - **System:** Predefined entities (e.g., Account, Contact).
        - **Custom:** User-created entities.
        - **Activity:** Entities representing tasks (e.g., Emails, Calls).
        - **Virtual**: External data, read-only.

- **Fields:** Fields store data within entities and follow specific naming rules.
    - Columns within a table (entity)
    - Types-
        - **Simple**
        - **Rollup Fields:** Aggregate data from related records.
            - Calculation Job Time Interval for Rollup:
                - Rollup Fields do not update instantly.
                - Instead, Dataverse calculates them at regular intervals using a background system job called the Rollup Calculation Job.
                - This job runs periodically to refresh the rollup field values in records. The time interval for this job defines how frequently the system updates rollup field values.
                - Default Job Interval:
                    - The default system job runs every 12 hours.
                    - Admins can manually trigger a recalculation if needed. (Refresh after making manual changes)
        - **Calculated Fields:** Perform real-time calculations using formulas.

Calculated Field	Rollup Field
Works within a single record	Works across multiple related records
Updates immediately when fields on the record change	Updates at specified intervals (e.g., daily, or when related records are updated)
Uses formulas or expressions (e.g., math, logic)	Aggregates data from related records (e.g., sum, count, max, etc.)
Only calculates values based on fields within the same record	Calculates values based on related records (e.g., opportunities related to an account)
Can perform complex calculations (e.g., IF statements, date calculations)	Typically aggregates values like sum, count, or maximum/minimum from related records
Can include fields like Single Line of Text, Number, Date, Currency, etc.	Works mainly with Number, Currency, Date, Duration, and Lookup fields

Field Type	Schema Name
Lookup E.g. 1. Contact 2. Account	ID ContactId AccountId
Option-Set (Choise) E.g. Gender	Type Code GenderTypeCode
Boolean E.g. Head Office	Is, Has, Are etc. IsHeadOffice

- **Activities:** any interactions or actions taken with a customer or within your CRM process. They're used to track communication and tasks, forming the timeline of a record (like Lead, Account, Opportunity, etc.).
    - rack communication history with customers

- Plan and assign follow-ups and meetings
- Provide a 360° view of customer interaction through the timeline
- Automate reminders and escalations in workflows/BPFs
- Activities can be associated with records like Contact, Account, Opportunity, Case, etc.
- Timeline control in forms shows activities chronologically.
- Workflows, Business Rules, and BPFs often create or interact with activities.

Activity Type	Description
Email	Tracks emails sent/received. Can be manual or automatic.
Phone Call	Logs phone conversations. Includes "direction" (incoming/outgoing).
Task	Generic action item. Often used for follow-ups, reminders, etc.
Appointment	Scheduled meetings, supports recurrence and invites.
Letter	Physical or digital letters exchanged with a contact.
Fax	Tracks fax communications (legacy use mostly).
Service Activity	Used in customer service for scheduling resources like facilities, people, etc.
Campaign Response	Response received from a campaign (marketing).
Custom Activities	Developers can create new activity types for specific needs.

- **Relationships:** define how tables (entities) interact with each other. They allow data to be linked, ensuring consistency and easy access.
  - Types:
    - One-to-Many (1:N)
      - One record in Table A can be related to multiple records in Table B.
      - Example: One Account can have many Contacts.
    - Many-to-One (N:1)
      - The reverse of One-to-Many.
      - Example: Many Contacts belong to one Account.
    - Many-to-Many (N: N)
      - A record in Table A can have multiple records in Table B, and vice versa.
      - Can be included in a form using subgrids
      - Example: Many Students can enroll in many Courses.

- Field Mapping in Relationships:
  - Helps auto-fill values from one table (entity) to another when creating related records.
  - When you create a related record, specific fields get automatically copied from the parent record.
- Behaviors (parental v/s referential):

Relationship Behavior	Parent Deletion Effect	Child Record Update	Example
Parental	Deleting the parent deletes all child records	Child records automatically update	Deleting an Account deletes all related Contacts
Referential	Deleting a parent does NOT delete child records	Child records remain unchanged	Deleting an Account keeps the Contacts
Referential with Restrict Delete	A parent cannot be deleted if child records exist	Child records remain unchanged	Cannot delete an Account if Contacts exist

- **Cascading behavior**: controls how actions taken on a parent record affect its related (child) records in a relationship (1:N or N:1).
- When you perform an action (like Assign, Delete, Share, etc.) on a parent record (e.g., **Account**), cascading behavior decides what happens to related child records (e.g., **Contacts** linked to that Account).

Behavior Type	What it Does
Cascade All	Child records inherit the parent's action.
Cascade Active	Only active child records are affected.
Cascade User-Owned	Only child records owned by the same user are affected.
Cascade None	No impact on child records.
Referential	Relationship is maintained but no cascade occurs.

Custom	You define custom rules for cascading.

- **Forms:** used to display, enter, and edit data for a specific table (entity). They act as the UI (User Interface) for users to interact with records.
    - Components of a form:
        - Header
            - Displays key fields that stay visible when scrolling
            - Commonly used for important information (e.g., Name, Status).
        - Body (Main Section)
            - The main data entry area contains fields, sections, and tabs.
            - Can be customized with business rules, JavaScript, and automation.
        - Tabs
            - Used to organize sections into collapsible groups.
            - Example: "General", "Details", "Additional Information".
        - Sections
            - Used inside tabs to group related fields.
            - Example: "Contact Information", "Address Details".
        - Fields
            - Data entry points on the form.
            - Can be of different types:
                - Text (Single-line, Multi-line)
                - Number (Whole, Decimal, Currency)
                - Date/Time
                - Optionset (Dropdowns, Choices)
                - Lookup (Reference to another table/entity)
        - Sub-Grids
            - Displays related records from another table.
            - Example: A Student Form showing Enrolled Courses in a sub-grid.
        - Business Rules
            - Automate form behavior without code.
            - Example: If Order Amount > $10,000, set Approval Status = "Required".
        - JavaScript Events
            - Allows adding custom logic to forms.
            - Example: Auto-fill fields based on selected values.
        - Quick View Forms
            - Shows read-only information from a related entity.

                - Example: In a Case Form, displaying Customer Details without navigating.
            - Footer
                - Displays key information that stays visible at the bottom.
                - Example: Record Owner, Created Date.
    - Types-
        - **Main Form**:
            - Primary and most commonly used form.
            - Used for viewing, editing, and entering data.
            - Supports tabs, sections, sub-grids, business rules, and custom scripts (JavaScript).
        - **Quick Create Form**:
            - Used for fast data entry without opening a full form.
            - Usually accessed from the + (Create New) button in the navigation bar.
        - **Card Form**: Compact view used in unified interface dashboards.
        - **Quick View Form**:
            - Displays related entity information inside another form.
            - Read-only and commonly used in lookup fields.

- **Views:** predefined data filters and layouts that control how records are displayed in a list format.
    - They help users see relevant records based on specific criteria.
    - They're customizable
    - Types-
        - Public Views (Visible to Everyone in Organization):
            - Available to all users with access to the table.
            - Created by admins or customizers.
            - Example: "Active Students", "Open Leads".
        - System Views (Global - Visible to All Users):
            - The default views provided by Microsoft.
            - Cannot be deleted, but can be customized.
            - Example: "My Active Contacts", "All Accounts".
        - Personal Views (User-Specific, Private):
            - Created by individual users for personal use.
            - Can be shared with other users.
            - Example: A salesperson creates "My High-Value Leads".
        - Advanced Find Views (User-Specific, Private):
            - Created using Advanced Find (now called Modern Filters).
            - Allows complex filtering based on multiple conditions.
            - The user can export data from that view
        - Associated Views (Visible in Related Records):
            - Used to show related records in a sub-grid.

            - Example: Viewing Contacts associated with an Account.
        - Lookup Views (Context-Based Visibility):
            - Used when selecting a record in a lookup field.
            - Only relevant records are shown based on conditions.
            - Example: Only teachers appear in the lookup when selecting a Teacher for a Student.
        - Quick Find Views (Visible in Search):
            - Defines search behavior when using the search box.
            - Controls which columns are searched.

- **Advanced Find:** allows users to search, filter, and export data from different tables (entities) using custom conditions.
    - Use of Advanced Find:
        - Search for specific records across tables (entities).
        - Apply complex filters with multiple conditions.
        - Customize column selection for reports.
        - Export data to Excel for further analysis.
    - Adding Filters: Filters help refine the search results based on conditions.
    - Related Entities Filter:
        - Allows filtering records based on related entities.
        - Example: Find Contacts linked to Accounts with a specific status.
    - Editing Columns (Customizing Results): By default, Advanced Find shows pre-selected columns, but you can customize them.
    - Exporting Data: After applying filters and editing columns, you can export the data for analysis.

- **Business Rules:** allow no-code automation for data validation and logic execution within forms and views.
    - BR v/s JS:
    - Feature Business Rule JavaScript
    - Code No-code (UI based) Custom code
    - Scope Entity or Form Form only
    - Portability Works in model-driven apps Web client-specific
    - Complexity Limited logic Advanced logicpossible

Parameter	BR	JS
Scope	- Table/Entity Scope - Specific Form Scope - All Form Scope	Specific Form Scope

Related Data Reference	Cannot use for this scenario.	JS can do this.
OnSave Event	- Only runs on Page Load or OnChange. - OnSave not possible	Can be used for OnSave and also OnChange, OnLoad
Complex Conditions	Can’t	Can do
Clearing Values from form fields	Work workaround is not an easy way	Easy
Dependency	Works within CRM	Requires web resources and script setup
Execution Speed	Runs on the server-side	Runs on the client-side (faster UI updates)

- Scope and Conditions:
  - Conditions (When the Business Rule Runs):
    - Based on field values or record status.
    - Example: "If Status = Active, then unlock the 'Email' field."
  - Scope (Where the Business Rule Applies):
    - Entity (Table Level) → Applies across all forms & views.
    - All Forms → Works on every form of the entity.
    - Specific Form → Applies only to a selected form.
    - Custom Controls → Can run inside embedded views or sub-grids.
- Actions in Business Rules
  - Lock/Unlock a Field
    - Prevent users from editing a field.
    - Example: Lock "Discount" field if Order Status = Finalized.
  - Set Visibility
    - Show or hide fields based on conditions.
    - Example: Hide "Company Name" field if "Customer Type" = Individual.
  - Set Field Value
    - Auto-fill fields based on other inputs.
    - Example: Set "Total Amount" = "Unit Price × Quantity".
  - Show an Error Message
    - Prevent invalid data entry.
    - Example: Show error if "Age < 18" in Student Form.

        - Set a Default Value
            - Prepopulate fields with values.
            - Example: Set "Country" = "India" for all new records.

- **Processes:** automates workflows and guides users through structured steps to ensure consistency in data entry and decision-making.
    - Types
        - Business Process Flow (BPF) → Guides users through a step-by-step process.
        - Workflows → Automates background tasks (like sending emails, updating records).
        - Actions → Custom reusable processes triggered by other processes.
        - Dialogs (Deprecated) → Previously used for interactive guided processes.

- **BPF:** visual guide in Microsoft CRM (Dataverse / Power Apps) that helps users follow a structured, step-by-step process when working with records.
    - It ensures consistency, standardization, and automation across business processes like Lead Qualification, Sales, Case Handling, and Customer Onboarding.
    - Create a Custom BPF:
        - Go to Power Apps → Select your environment.
        - Click Solutions → New → Business Process Flow.
        - Choose the Table (Entity) (e.g., Lead, Account, Case).
        - Click Create.
    - Stages in BPF ->define the major steps in a process.
    - Adding Conditions & Branching: Use "If-Else" conditions to create branching logic.

- **Workflows:** automates processes like record updates, email notifications, and approvals without user intervention. It helps eliminate manual work and ensures process consistency.
    - When u wanna use a workflow in any BPF or anywhere, convert it to an **on-demand** workflow
    - Types:
        - Real-time Workflows (Synchronous):
            - Executes immediately when a trigger event occurs (takes away user control until the workflow is executed)
            - Faster, but may slow system performance.
            - Example: Send an email when a lead is created.
        - Background Workflows (Asynchronous):
            - Runs in the background after the trigger event.
            - Ideal for non-urgent processes.

            - Example: Update lead status after 24 hours.

        - On-Demand Workflows
            - Can be manually triggered by a user from a record.
            - Example: A salesperson manually starts a workflow to send a discount approval request.
        - Child Workflows
            - A sub-workflow called by another workflow.
            - Useful for reusable logic across multiple processes.
            - Example: A parent workflow triggers a child workflow to send a reminder email every 2 minutes.
    - Workflow Scope:
        - Organization → Affects all records in the system.
        - Business Unit → Runs only within a specific business unit.
        - User → Applies to records owned by a specific user.
    - Workflow Trigger Events:
        - When a Record is Created
            - The workflow runs immediately when a new record is added.
        - When a Record is Updated
            - The workflow runs when a specific field is updated in a record.
            - Can be configured to track specific fields only.
        - When a Record is Deleted
            - The workflow runs when a record is removed from the system.
        - When a Record's Status Changes
            - The workflow runs when the status or status reason of a record is modified.

        - On Demand (Manually Triggered)
            - The workflow runs when a user manually starts it from a record.
        - Child Workflow Execution
            - The workflow is triggered when another workflow is called.
    - CRUD Operations in Workflows:
        - Create → Automatically create tasks, emails, or records.
        - Read → Fetch record details (e.g., check if an order exists).
        - Update → Modify field values (e.g., change Lead Status to "Qualified").
        - Delete → Remove records (not commonly used in workflows).

- **Actions:** Custom processes used to execute logic that you can define once and call from multiple places.
    - Types:
        - Global: Not tied to any entity. Can be reused widely.
        - Entity-bound: Tied to a specific entity. Can use data from that record directly.
    - Input & Output Parameters: **Optional.** But make your action reusable by passing data dynamically.
        - Input Parameters: values you pass into the action
        - Output Parameters: values you get from the action after it runs
    - Only **entity-bound actions** can be called from classic **workflows**.
    - Calling an action might fail if the user doesn't have permissions for the target entity.

- **Charts:** visual representations of your data from views help you analyze and interpret your CRM data quickly and visually
    - Types:
        - Column Chart: Vertical bars
        - Bar Chart: Horizontal bars
        - Pie Chart: Circular graph, used for proportions
        - Line Chart: Trends over time
        - Funnel Chart: Visualize pipeline stages
        - Area Chart: Like a line chart, but filled
    - How They Work:
        - Charts are tied to views (like "All Contacts", "My Opportunities")
        - Use aggregates: Count, Sum, Avg, Max, Min
        - Grouped by: Owner, Status, City, etc.
    - Used in **views** of an entity, embedded in **Dashboards, Clickable —** clicking chart filters the view.
    - Charts **always depend on a view**.
    - You can create **personal charts** (user-level) or **system charts** (visible to all)
    - Use **advanced find** to build custom views → then chart.

- **Dashboards:** Dashboards are interactive canvases that display: Charts, Views, Web resources, iFrames, Power BI visuals (these are components of dashboards)
    - They give a real-time overview of CRM data like a control panel.
    - Types:
        - User Dashboard: Created and visible only to the user
        - System Dashboard: Created by admin/customizer and shared org-wide
        - Interactive Dashboard: Includes streams, filters, and quick actions (highly interactive UI)

    - Where Used? => Home screen of CRM, Entity-specific dashboards, Mobile/tablet views (adapted)
    - **Dashboards ≠ Reports** — they're real-time and interactive.
    - **Charts are reusable** — can be part of multiple dashboards.
    - You need a **security role** to view/edit dashboards.
    - Dashboards **auto-refresh** when the underlying data/view changes.
    - Interactive dashboards are **perfect for customer service scenarios**

- **Security:** ensures that users only see and act on data they're authorized to
    - It's managed through Business Units, Security Roles, Teams, and additional layers like Field Security or Hierarchy Security.
    - Adding a User
        - Users must be added via Microsoft 365 Admin Center (must have a license).
        - Once added, assign them a security role in CRM to give access.
        - A user belongs to one Business Unit.
    - Business Units
        - Represents the organizational structure.
        - Every user, team, and record belongs to one BU.
        - Used to partition data access and structure roles.
        - Top BU = root; you can create child BUs.
    - Users and Ownership
        - Each record (like Account, Contact) has an Owner (a user or team).
        - Owners control who can read/update/delete the record, based on security roles.
        - Ownership is important for workflows, emails, and access control.
    - Security Roles
        - Define what actions (CRUD) a user can perform on which tables/entities.
        - Permissions are defined at different levels:
            - User – Only own records.
            - Business Unit – All records in the same BU.
            - Parent: Child BU – Records in own + child BUs.
            - Organization – All records system-wide.
        - Users can have multiple roles (permissions stack).
    - Owner Teams
        - The team that can own records.
        - Members inherit the team's permissions.
        - Useful when multiple people share ownership of records.
    - Access Teams
        - Temporary or flexible teams used for sharing specific records.
        - Do not own records.

        - You define Access Team Templates with permissions like Read, Write, etc.
    - Field-Level Security
        - Protects specific sensitive fields (e.g., Salary, SSN).
        - Only users with a Field Security Profile can view or edit those fields.
        - Applied on a per-field basis on entities.
    - Hierarchy Security
        - Allows managers to access records owned by their subordinates (based on the manager field or position).
        - Two types:
            - Manager Hierarchy
            - Position Hierarchy
        - Enhances visibility without giving org-wide access.

- **Sales Literature** *(Sales Module):* way to store and manage **marketing materials** or documents (brochures, price lists, product sheets) that can be shared with leads/opportunities.
    - To organize and distribute content used during the sales process.
    - Linked to products or competitors
    - Documents can be uploaded and categorized
    - Can be associated with email activities or opportunities

- **SLA- Service Level Agreement** (*Customer Service Module):* defines the **response and resolution time** expectations for customer service cases or activities.
    - To ensure customer issues are resolved within a committed timeframe.
    - Applicable to cases or custom entities
    - Tracks KPIs like "First Response By" or "Resolve By"
    - Can trigger workflows or actions when SLA times are violated
    - Works with entitlements and timer controls

- **Goal:** a target that a user, team, or business unit wants to achieve.
    - Example: A sales rep wants to close deals worth ₹5,00,000 this quarter.
    - It is time-bound (monthly, quarterly, yearly)
    - Can be based on revenue, number of cases closed, leads generated, etc.
    - Assigned to an owner (user/team)
    - Actual v/s In-Progress:
        - Actual: Closed records (e.g., closed-won opportunities)
        - In-Progress: Open or active records
        - Goals can be configured to show progress toward both.

- **Goal Metric:** defines:

    - What is being measured (e.g., revenue, number of cases)
    - How it is measured (via Rollup Fields)
    - Goal metric type: Amount or Count

- **Rollup Queries**
    - Used when you want to filter specific records (e.g., only count opportunities from a certain region).
    - They define which data to include in the goal calculation.

- **Duplicate Detection:** a feature that checks for records that match certain criteria (like same email, phone number, or name) when creating a new record, importing data, saving, or updating records

Component	Description
Duplicate Detection Rule	Defines the conditions to detect duplicates
Duplicate Detection Job	Allows scanning the system manually for duplicates
Publishing the Rule	Makes the rule active for detection

    - Duplicate detection works only on published rules
    - Doesn't work in real-time for all integrations (like some API inserts)
    - For more advanced deduplication, consider Data Quality tools or custom plugins

- **Templates:** predefined documents or email formats that help standardize communication and data representation. They save time and ensure consistency across the organization.

Template Type	Description
Email Templates	Reusable formats for sending standardized emails.
Word Templates	Documents for exporting data like quotes, invoices, or letters in a professional format.
Excel Templates	Predefined Excel files to analyze and report CRM data.

- **Competitor:** represents another company or solution your organization competes with in sales opportunities.

- **Customer Service Hub:**
  - Knowledge Article: a piece of content used to help internal users or customers resolve issues, answer questions, or share information.
  - Knowledge Base: a central repository where all published Knowledge Articles are stored.
  - Case (Incident): a customer issue or request that needs resolution.
    - Track customer support issues, manage resolution workflows, and analyze service performance.
    - Can be auto-created from emails, web forms, or portals.
    - Supports SLA, Entitlements, Activities, and Queues.
    - Lifecycle: Create → Assign → Resolve → Close.
  - Queue: container where cases (or activities) are held until an agent picks them up.
    - Distributes work among support agents.
    - Prioritizes or categorizes support tasks.
    - Tracks assignment and status.
    - Types:
      - Public Queue: Available to multiple users or teams.
      - Private Queue: Restricted to specific users.
  - Routing Rules: define how cases should be assigned or routed automatically to Queues or Users/Teams.
    - Routing rules only work on Create of Case unless triggered manually or via workflow/Power Automate.
    - Automatically move new cases to the right queue or assign to the right agent/team based on: Subject, Origin (Email, Portal, Phone), Customer Type, Product or Region
  - How They Work Together:
    - Case is created via email or portal.
    - Routing Rule checks the case attributes.
    - Based on rule, the case is routed to a Queue.
    - An agent picks the case from the queue.
    - Case gets assigned and tracked through resolution.
- **Connection:** Is a record that links two other records.
  - Can exist between any two entities (standard or custom).
  - Is used when a relationship isn't covered by traditional entity relationships.
  - For example:
    - A Contact might be a "Legal Advisor" for an Account.
    - An Opportunity might be "Referred By" a Contact.
- **Connection Roles:** define the nature of the relationship.
  - For example:
    - John Doe (Contact) is an ABC Corp (Account) Decision Maker.
    - Jane Smith is an Accountant for XYZ Inc.

    - Each connection has: Record A, Record B, Role A to B, (Optional) Reciprocal Role B to A
- **Product family v/s bundles:**
    - A **Product Family** is a way to organize similar products under a common category or hierarchy. Use: when you want to **manage large catalogs** with similar items and centralized control.
    - A **Product Bundle** is a package of multiple products/services sold together at a combined price. Use: when you want to **sell multiple items together** as one offering.

Feature	Product Family	Product Bundle
Purpose	Organize and inherit attributes	Sell multiple products together
Sold As	Individual products	Single package
Hierarchical?	Yes (Parent-Child)	No (Flat group of items)
Pricing	Individual product pricing	Combined/individual pricing possible
Use Case	Categorization, attribute control	Cross-sell, upsell, offer discounts

- **Auditing:** a powerful feature that allows you to track changes made to data in the system. It helps in monitoring, security, compliance, and troubleshooting.
    - Auditing Structure
        - System-Wide Audit – Turn auditing on/off for the entire system.
        - Entity-Level Audit – Enable auditing for specific tables (entities) like Account, Contact, Lead, etc.
        - Field-Level Audit – Audit changes to specific columns/fields within those tables.
    - Auditing consumes storage in your Dataverse environment.
    - Audit logs are read-only and cannot be edited or deleted.
    - You should regularly manage audit logs (archive/delete older logs).

Auditing Scope	Example
User access	When a user logs in/out
Record changes	Changes to fields like phone, email
Create/Delete	When records are added or deleted
Security role	Assign/remove security roles
Data sharing	Changes to record ownership/teams

- **Linking JS web resource to HTML web resource:**
  <script src="/WebResources/new_randomJoke.js" type="text/javascript">
  </script>
    - Replace new_randomJoke.js with the Name of the Web Resource — not the file name.

- **JS(context api):**
- Get Form Context:
  var formContext = executionContext.getFormContext();
- Get Field Value
  var fieldValue = formContext.getAttribute("fieldlogicalname").getValue();
  For lookup → returns array:
  var lookup = formContext.getAttribute("lookupfield").getValue();
  var id = lookup[0].id.replace(/[{}]/g, "");
  var name = lookup[0].name;
- Set Field Value
  formContext.getAttribute("fieldlogicalname").setValue(newValue);
  For lookup:
  var lookupValue = [{
  id: "GUID",
  name: "Display Name",
  entityType: "entitylogicalname"}];
  formContext.getAttribute("lookupfield").setValue(lookupValue);
- For date:
  formContext.getAttribute("datefield").setValue(new Date());
- Clear Field
  formContext.getAttribute("fieldlogicalname").setValue(null);
- Enable/Disable Field
  formContext.getControl("fieldlogicalname").setDisabled(true); // Disable
  formContext.getControl("fieldlogicalname").setDisabled(false); // Enable
- Show/Hide Field
  formContext.getControl("fieldlogicalname").setVisible(false); // Hide
  formContext.getControl("fieldlogicalname").setVisible(true); // Show
- Set Required
  formContext.getAttribute("fieldlogicalname").setRequiredLevel("required");
  formContext.getAttribute("fieldlogicalname").setRequiredLevel("none");
- Add Notifications
  formContext.ui.setFormNotification("Your message", "INFO", "uniqueId");
  formContext.ui.clearFormNotification("uniqueId");
- Add Pre-search (filter lookup)
  formContext.getControl("lookupfield").addPreSearch(function() {
  formContext.getControl("lookupfield").addCustomFilter("<filter></filter>",
  "entitylogicalname");});

- Most Used Events to Attach JS:

Event	Use
OnLoad	When form loads
OnChange	When field changes
OnSave	When saving form

-

- **JS XRM WebApi:**
  - **Real example/ format of using Xrm.WebApi:**

```
function retrieveContactEmail(executionContext) {
    var formContext = executionContext.getFormContext();
    var contactId = formContext.data.entity.getId().replace(/[{}]/g, "");

        Xrm.WebApi.retrieveRecord("contact",contactId,
"?$select=emailaddress1")
    .then(function(result) {
        alert("Contact Email: " + result.emailaddress1);
    })
    .catch(function(error) {
        console.log("Error: " + error.message);
    });
}
```

  - **Retrieve Single Record**
    -
```
function getRecord(entityLogicalName, recordId) {
    Xrm.WebApi.retrieveRecord(entityLogicalName, recordId)
    .then(function (result) {
        console.log(result);
    })
    .catch(function (error) {
        console.log(error.message);
    });
}
```
    - `getRecord("contact", "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx");`
  - **Retrieve Multiple Records (With Filters)**
    -
```
function getMultipleRecords(entityLogicalName, filter) {
        Xrm.WebApi.retrieveMultipleRecords(entityLogicalName,
filter)
    .then(function (result) {
        result.entities.forEach(function (record) {
            console.log(record);
        });
```

```
    })
    .catch(function (error) {
        console.log(error.message);
    }); }
```

- getMultipleRecords("contact", "?$filter=firstname eq 'John'");

- **Create Record:**
  - ```
    function createRecord(entityLogicalName, data) {
        Xrm.WebApi.createRecord(entityLogicalName, data)
        .then(function (result) {
            console.log("Record created. ID: " + result.id);
        })
        .catch(function (error) {
            console.log(error.message);
        });
    }
    ```
  - ```
    var data = {
        firstname: "John",
        lastname: "Doe",
        emailaddress1: "john@example.com"
    };
    createRecord("contact", data);
    ```
- **Update Record:**
  - ```
    function updateRecord(entityLogicalName, recordId, data) {
            Xrm.WebApi.updateRecord(entityLogicalName,   recordId,
    data)
        .then(function (result) {
            console.log("Record updated.");
        })
        .catch(function (error) {
            console.log(error.message);
        });
    }
    ```
  - ```
    var data = {
        firstname: "Jane"
    };
    updateRecord("contact",
    "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx", data);
    ```
- **Delete Record:**
  - ```
    function deleteRecord(entityLogicalName, recordId) {
        Xrm.WebApi.deleteRecord(entityLogicalName, recordId)
        .then(function (result) {
            console.log("Record deleted.");
        })
        .catch(function (error) {
    ```
    ```

```
            console.log(error.message);
        });
    }
```

- deleteRecord("contact", "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx");

- **Associate Lookup (Set Lookup Field):**
  - function setLookup(entityLogicalName, recordId, lookupField, lookupEntityLogicalName, lookupRecordId) {
    var data = {};
    data[lookupField + "@odata.bind"] = "/" + lookupEntityLogicalName + "s(" + lookupRecordId + ")";

    Xrm.WebApi.updateRecord(entityLogicalName, recordId, data)
    .then(function (result) {
    console.log("Lookup set.");
    })
    .catch(function (error) {
    console.log(error.message);
    });
    }
  - setLookup("contact", "contactID", "parentcustomerid_account", "account", "accountID");
- **Clear Lookup (Disassociate)Clear Lookup (Disassociate):**
  - function clearLookup(entityLogicalName, recordId, lookupField) {
    var data = {};
    data[lookupField + "@odata.bind"] = null;

    Xrm.WebApi.updateRecord(entityLogicalName, recordId, data)
    .then(function (result) {
    console.log("Lookup cleared.");
    })
    .catch(function (error) {
    console.log(error.message);
    });
    }
  - clearLookup("contact","contactID", "parentcustomerid_account");
-

- **When to use logical name v/s the schema name:**
  Quick Validator for Web API:
  Before using any field in createRecord / updateRecord:
  - Is it a lookup field?

        - Use schema name + @odata.bind
        - Ex: "ud_ExistingContact@odata.bind": "/contacts(GUID)"
    - Is it a simple field (text, option set, boolean)?
        - Use logical name
        - Ex: "jobtitle": "CTO"
        - Never use logical name in lookup @odata.bind → Will cause errors
    - Check field in Table > Columns in Power Apps:
        - Use “Name” (PascalCase) → Schema Name
        - Use “Display Name” only for user interface
    - Still unsure? -> Use XrmToolBox → Metadata Browser Tool to see: Logical Name, Schema Name, Lookup Target

Scenario	Field Name Format	Example	Notes
`getAttribute()` / `getControl()`	Logical Name	`"ud_existingcontact"`	Used in forms / client-side scripts
Web API `@odata.bind` (lookup)	Schema Name	`"ud_ExistingContact@odata.bind"`	Only schema name works here
Web API `createRecord` / `updateRecord`	Schema for lookups	`"transactioncurrencyid@odata.bind"`	Use schema for all lookup bindings
Web API simple fields	Logical Name	`"jobtitle": "Engineer"`	OK to use logical name for non-lookup fields
FetchXML / OData `$select`, `$filter`	Logical Name	`?$select=firstname,lastname`	Lookup value = `_fieldname_value`
Lookup value from Web API response	`_schemaName_value`	`_ud_ExistingContact_value`	Underscore + schemaName + `_value`

- **Plugins:** A plugin is a custom C# code that runs in response to CRM system events (like Create, Update, Delete, Assign, etc.). It allows developers to inject business logic before or after the CRM operation.
  used to:
    - Automate complex business logic (e.g., auto-calculate values, validate data)
    - Integrate external systems (API calls, push/pull)
    - Enforce business rules beyond standard workflows or business rules
    - Update related records automatically (e.g., update contact when account changes)

Pipeline stages:

Stage	Description	Common Use
Pre-Validation (10)	Before security check	Cross-entity validation
Pre-Operation (20)	Before DB save	Modify target data
Post-Operation (40)	After DB save	Create related records, call external APIs

Plugin Execution Modes:

Mode	Description	Key Point
Sandbox (Isolated)	Plugin runs in restricted environment	Default in Online; safer
None (Full Trust)	Access to local resources	Only in On-Premise deployments

- Secure/Unsecure Config: Pass config values from plugin registration (e.g., API keys)
- IPlugin Interface: All plugins must implement this interface
- IServiceProvider: Provides services like IPluginExecutionContext, ITracingService, and IOrganizationService
- Plugin Registration Tool / Plugin Registration in Power Platform Tools: Used to deploy plugins
- Profiler: Used for debugging plugins by replaying recorded context

Best Practices:

- Keep logic lean and efficient (especially in sync plugins)
- Use Pre-Operation to modify data before save
- Use Post-Operation to access full record (with ID) and handle external logic
- Avoid using Thread.Sleep, MessageBox, Console etc.
- Always wrap in try-catch and use ITracingService.Trace

- **Use of context.InputParameters["Target"]:**
  - That if (context.InputParameters.Contains("Target") && context.InputParameters["Target"] is Entity entity) check is a safety net.
  - It protects your plugin from running into `null` references or incorrect assumptions about what triggered it.
  - When to Use It:

        - The plugin is registered on a **message where "Target" might not always be present** (e.g., `Create`, `Update`, `Delete`, `Assign`, `SetState`).
        - The plugin is **shared across multiple messages/entities** and you need to ensure you're handling only the right execution context.
        - You **don't control the triggering conditions** — for example, if it could be registered by someone else or triggered by a process you didn't design.
        - You're working with **different input parameters** in different scenarios and want to safely guard your cast.
    - When You Can Skip It:
        - The plugin step is **registered for a very specific message and entity** (e.g., `Update` on `contact`) and you know for sure `"Target"` will always be present.
        - You **always get PreImage or PostImage** and will base your logic on that instead of `Target`.
        - You **control the registration** and know `"Target"` will always be the expected type.
    - **Rule of Thumb**:
        - road or shared plugin logic → Always check InputParameters.Contains("Target").
        - Highly specific, tightly scoped logic → You can skip the check, but only if you're 100% sure the context will always have "Target" of the right type.

- **Pre-Image and Post-Image**: Images are snapshots of the record's data either before (Pre-Image) or after (Post-Image) the core operation (like Update, Delete) is executed.
    - They are optional but handy for getting additional context in a plugin.
    - Used for:
        - Access field values that aren't present in the Target entity
        - Compare before vs after values
        - Perform validation or conditional logic
        - Retrieve read-only fields (e.g., calculated fields not present in Target)
    - Pre-Image
        - When: Captured before the core operation (e.g., before record is updated)
        - Use for:
            - Checking previous values
            - Validating changes
            - Preventing updates based on current data

        - Example: Check if the Status was previously “Open” before allowing an update.
    - Post-Image
        - When: Captured after the operation (record has been updated or created)
        - Use for:
            - Fetching updated values
            - Creating logs or audit data
            - Triggering external logic (e.g., send mail after status is set to Approved)
        - Example: Send an email when the Status becomes “Approved” after an update.
    - EImages are explicitly configured while registering plugin step (via Plugin Registration Tool or Plugin Reg in VS)
    - You define the columns you want in the image – avoid selecting all to reduce load
    - Use context.PreEntityImages["imageName"] and context.PostEntityImages["imageName"] to access them
    - Images are of type Entity, just like Target
    - Example:

```
// Fetch Pre-Image
Entity preImage = (Entity)context.PreEntityImages["PreImage"];
string oldStatus = preImage.GetAttributeValue<string>("statuscode");

// Fetch Post-Image
Entity postImage = (Entity)context.PostEntityImages["PostImage"];
string newStatus = postImage.GetAttributeValue<string>("statuscode");

// Compare status change
if (oldStatus != newStatus && newStatus == "Approved")
{
    // Trigger logic like sending email
}
```

- C
- C
- C
- C
- C
- C
-
-
- **Debugging a plugin (use profiler):**
    - Profiler helps simulate the plugin execution in your local machine.

- In PRT:
  - Right-click your registered plugin step
  - Select "Start Profiling" > "Profile Workflow/Plugin (Sandbox)"
  - Choose “Persist to disk” → click OK
- Trigger the plugin again, click "Debug" on the top menu
- Select the .profiler file you just downloaded. It asks for:
  - Path to your compiled plugin .dll
  - Class name & method (YourClass: Execute)
  - Click "Start Execution"
- Context Validations Are a Must: Always validate before casting or accessing data
  ```
  if (context.InputParameters.Contains("Target") &&
  context.InputParameters["Target"] is Entity)
  ```
  - Prevents runtime exceptions.
  - Avoids null reference errors.
- Never Forget Tracing
  ```
  tracingService.Trace("Reached point X: {0}", someVariable);
  ```
  - Works in the production sandbox too.
  - Shows up in plugin trace logs in CRM under System Jobs > Exception Details.
  - Crucial when debugging asynchronous plugins.
- Avoid Using context.UserId for Write Operations
  ```
  serviceFactory.CreateOrganizationService(null); // actsAsSYSTEMuser
  ```
  - Use context.UserId only if you want to execute as the user who triggered the plugin.
- Limit Expensive Operations in Synchronous Plugins
  - No long loops
  - No heavy queries
  - Avoid recursion or calling other plugin-triggering operations.
  - Use asynchronous plugins (background mode) if long processing is needed.
- Prevent Infinite Loops
  - If your plugin updates the same entity, it may retrigger itself.
  - Prevent this using:
    ```
    if (context.Depth > 1) return; // avoid recursive triggering
    ```
- Use Secure & Isolated Configuration: Use Secure/Unsecure plugin configuration during step registration. Or fetch values from Custom Configuration Entities.
- Filtered Attributes = Performance Boost. When registering step:
  - Use Filtering Attributes to trigger plugin only when relevant fields are changed.
  - e.g., if your plugin is only interested in "statuscode", don’t let it fire on every update.

- Don't use async/await Inside Plugins: The CRM plugin pipeline is synchronous. Don’t use async methods — they’ll behave unpredictably.
-

Practice	Benefit
Use `EntityImageCollection`	For **Pre/Post Images**
Log errors to Custom Entity	For production audit tracking
Use Dependency Injection (advanced)	Better plugin testing & design
Separate Logic from Plugin Class	Makes plugins cleaner & testable

-

-
-
-
-
- **C**
- **C**
- **D**
- **E**
- **F**
- **G**
- **H**
- **I**
- **J**
- **K**
- **L**
- **M**
- **N**
- **O**
- **p**

-

Power Automate

- **Types of flows:**
  - **Cloud flows**: Create a cloud flow when you want your automation to be triggered either automatically, instantly, or via a schedule.
    - **Automated cloud flow**: Create an automation that is triggered by an event such as the arrival of an email from a specific person, or a mention of your company in social media.
    - **Instant cloud flow**: Start an automation with the selection of a button.
    - **Scheduled cloud flow**: Start a scheduled automation such as daily data upload to SharePoint or a database.
  - **Desktop flows**: Use desktop flows to automate tasks on the web or the desktop.
  - **Generative actions** (preview): Use generative actions in your flow if you want to specify only the intent of the action and then have the AI choose the right set of actions in the right order based on your input, context, and intent.

- **Dataverse connectors**: It's the primary bridge between Power Automate and Microsoft Dataverse (the database behind Dynamics 365 CE).
  - Lets you create, update, delete, and retrieve records in any Dataverse table (custom or standard).
  - Fully supports custom tables, columns, and relationships without extra setup.
  - Works with solution-aware flows (important for ALM — Application Lifecycle Management).
  - Two Flavors of Dataverse Connector:
    - Microsoft Dataverse (current)
      - Preferred connector — modern, more secure, faster.
      - Supports solution-aware development.
      - Supports trigger conditions and change tracking to reduce unnecessary runs.
      - Works with both cloud and model-driven apps (Dynamics 365 CE).
      - Syntax-friendly with OData queries in actions.
    - Microsoft Dataverse (Legacy)
      - Older version — labeled as "(Legacy)" in UI.
      - Limited functionality.
      - No new features being added; use only if maintaining older flows.
    - Diff from old connectors:
      - Dataverse Connector → Supports all tables, security, and environments in one place.

- Dynamics 365 Connector (deprecated) → Limited to Dynamics-specific tables, slower, and no new features.

Always use the current Microsoft Dataverse connector unless you're forced to maintain legacy flows.

- Best Practices for Using Dataverse Connector in CRM Flows
  - Always filter triggers (either Filter attributes or Trigger Conditions) to avoid unnecessary runs.
  - Use $filter in "List rows" instead of retrieving all records (performance optimization).
  - Use Top Count to limit results if you only need a few records.
  - For related tables, use Expand Query in "List rows" to fetch lookup details in one call.
  - Store record IDs in variables if you plan to update later in the flow.
  - Avoid unnecessary "Get row" calls if you already have data from the trigger.

- **Triggers in Dataverse Connector:** Events that start your flow.
  - Filtering attributes: In "When a row is modified," you can specify fields to watch.
  - Trigger Conditions: Advanced expressions to decide whether to run.

Trigger	Description	Example in CRM
When a row is added	Fires when a new record is created.	Send a welcome email when a Contact is created.
When a row is modified	Fires when a record is updated.	Alert the Sales Manager when an Opportunity's estimated revenue changes.
When a row is deleted	Fires when a record is deleted.	Archive related records when an Account is deleted.

- Actions in Dataverse Connector: Things your flow can do in Dataverse.

Action	Purpose	Example
Add a new row	Create a record in a table.	Create an Opportunity when a Lead is qualified.
Update a row	Modify existing record fields.	Update Contact address when Account address changes.
Delete a row	Remove a record.	Delete related tasks when a project is canceled.
Get a row by ID	Retrieve one record by unique ID.	Get Account details before sending an email.
List rows	Retrieve multiple records using OData filters.	Get all open Opportunities older than 30 days.

- Other connectors:
  - Productivity & Collaboration Connectors

Connector	**Purpose**	**CRM Use Case**
Office 365 Outlook	Send/receive emails via Exchange Online	Send notifications when leads/opportunities are created
Microsoft Teams	Send chat messages, create channels, start approvals	Post in a Teams channel when a deal is won
Approvals	Create/manage approval processes	Approve quotes > ₹5L via Teams/Outlook
Microsoft 365 Users	Retrieve user profile info (email, name, manager)	Get CRM user's manager for escalations

  - Document & Storage Connectors:

Connector	**Purpose**	**CRM Use Case**
SharePoint	Store documents	Save uploaded files from CRM to a SharePoint library

OneDrive for Business	Store personal business docs	Save a generated PDF quote to user's OneDrive
Azure Blob Storage	Store large files in Azure	Store attachments or bulk export CRM data

- Integration & API Connectors

Connector	Purpose	CRM Use Case
HTTP with Azure AD	Call Dataverse Web API or any OAuth 2.0-protected API	Fetch CRM data with custom OData queries
HTTP	Call REST APIs with API keys/basic auth	Integrate with payment gateways or third-party services
Azure Service Bus	Event-based integration with other apps	Publish CRM changes for ERP system consumption
Azure Functions	Run custom server-side logic	Perform complex calculations and return results to CRM

- Data Transformation & AI Connectors

Connector	Purpose	CRM Use Case
Excel Online (Business)	Read/write Excel data	Import CRM leads from Excel sheet
AI Builder	Predictive analysis, text recognition, form processing	Predict lead scoring based on historical data
Power BI	Push data for analytics	Send CRM opportunity data to Power BI dashboard

-