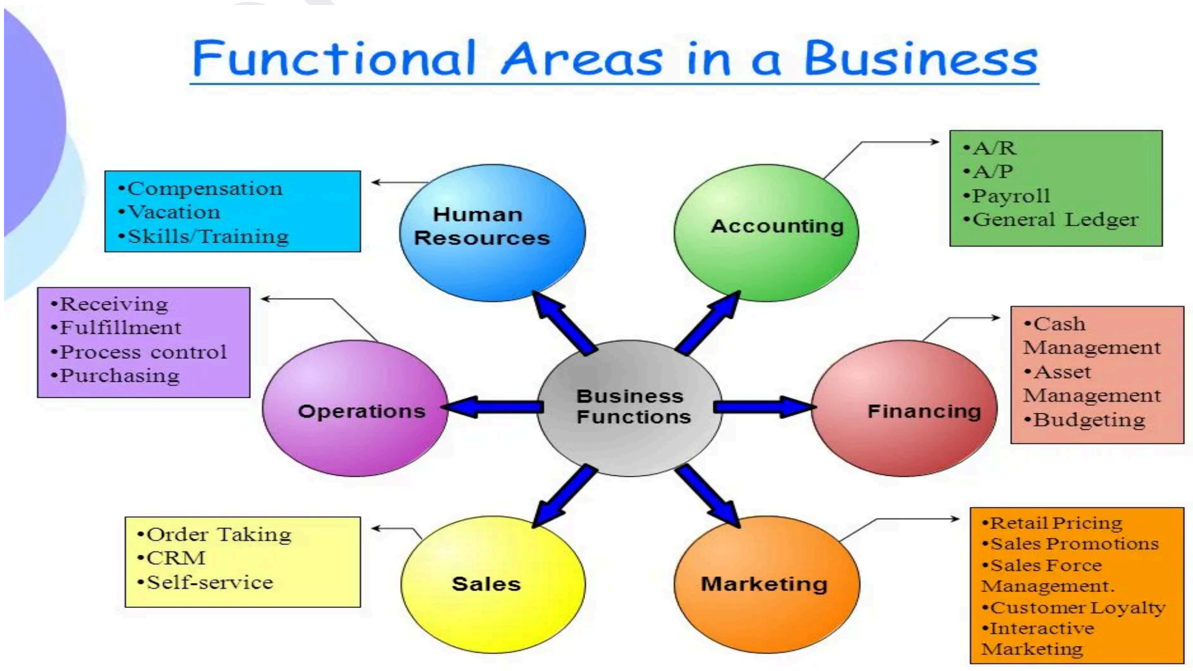
A Developer's Overview of Business Central

What is BC ?:

Microsoft Dynamics 365 Business Central is a **cloud-based** business management solution like ERP, for small and mid-sized organizations. It's designed to help businesses connect their sales, service, finance, and operations, and to help them work smarter.

Here are some of the features of Microsoft Dynamics 365 Business Central:

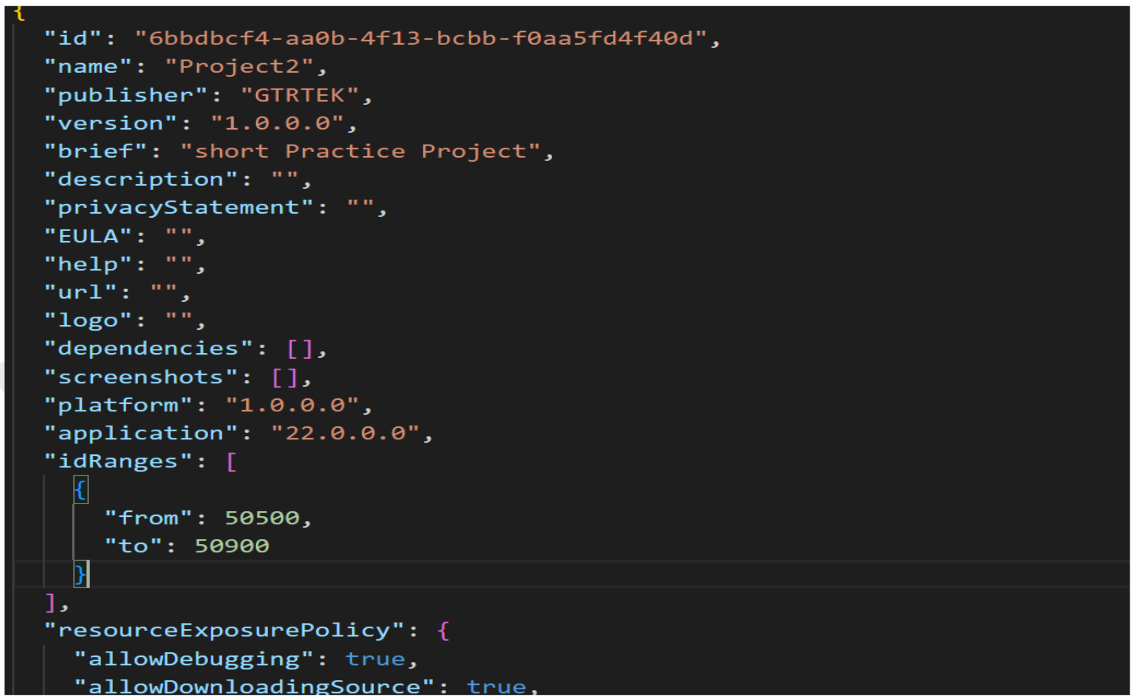
Integration: Business Central integrates ERP, CRM, and Office 365 into a single platform. It also integrates with other systems through web APIs.
Automation: Business Central automates and streamlines business processes, including financial management, supply chain management, sales and service management, project management, and operations management.
AI and BI: Business Central integrates advanced technologies such as artificial intelligence (AI) and business intelligence (BI).
User interface: Business Central has an interactive and responsive user interface.
Real-time information: Business Central provides real-time information.



As a Business Central developer, your primary tools and responsibilities include:

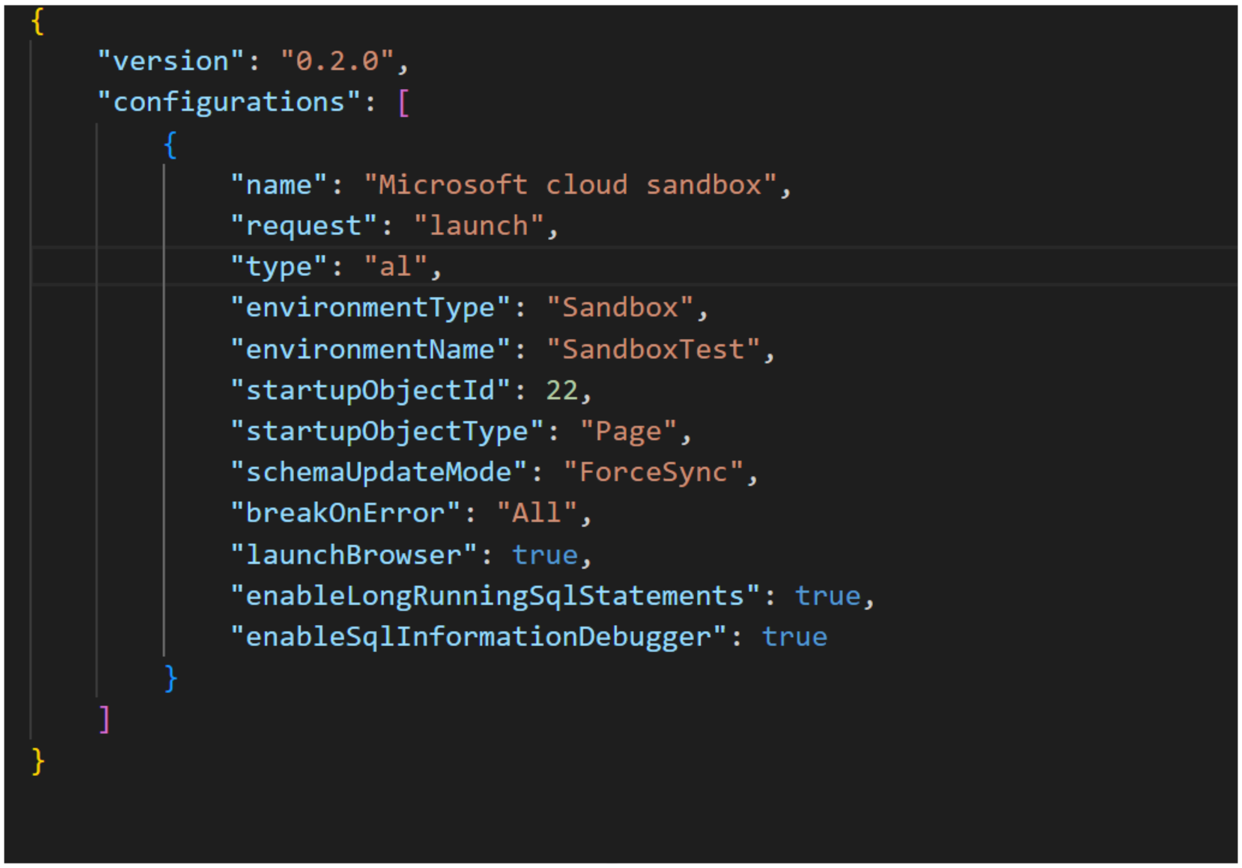
- **AL Language:** AL is the programming language used in BC for writing extensions. It is a modern, powerful language that allows developers to define the behavior of objects.
- **Visual Studio Code (VS Code):** BC development is done using VS Code with AL language extension. This environment supports development, debugging, and testing.
- **Extensions:** In BC, new functionality is delivered via extensions, which are packages that include your custom objects and code. Extensions can be installed, updated, and uninstalled without affecting the core product.
- **Business Logic:** Developers write code to manage business logic, automate processes, and handle integrations with other systems.
- **Web Client:** BC is accessed through a web client, and developers must consider UI/UX design principles when creating pages or modifying existing ones.

APP.json



```
{
  "id": "6bbdbcf4-aa0b-4f13-bcbb-f0aa5fd4f40d",
  "name": "Project2",
  "publisher": "GTRTEK",
  "version": "1.0.0.0",
  "brief": "short Practice Project",
  "description": "",
  "privacyStatement": "",
  "EULA": "",
  "help": "",
  "url": "",
  "logo": "",
  "dependencies": [],
  "screenshots": [],
  "platform": "1.0.0.0",
  "application": "22.0.0.0",
  "idRanges": [
    {
      "from": 50500,
      "to": 50900
    }
  ],
  "resourceExposurePolicy": {
    "allowDebugging": true,
    "allowDownloadingSource": true,
```

Lunch.json



```json
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Microsoft cloud sandbox",
            "request": "launch",
            "type": "al",
            "environmentType": "Sandbox",
            "environmentName": "SandboxTest",
            "startupObjectId": 22,
            "startupObjectType": "Page",
            "schemaUpdateMode": "ForceSync",
            "breakOnError": "All",
            "launchBrowser": true,
            "enableLongRunningSqlStatements": true,
            "enableSqlInformationDebugger": true
        }
    ]
}
```

Launch.json Explanation

`"name": "Microsoft cloud sandbox"`: This is just the name of the configuration to help you identify it, called Microsoft cloud sandbox.

`"request": "launch"`: Specifies the action being performed, in this case, to launch a sandbox environment for testing or debugging.

`"type": "al"`: Indicates that this configuration is for AL code (the language used for development in Dynamics 365 Business Central).

`"environmentType": "Sandbox"`: The environment type is set to Sandbox, which is a test environment separate from production.

`"environmentName": "SandboxTest"`: This is the name of the sandbox environment you're working in, which is called SandboxTest.

`"startupObjectId": 22`: This specifies the ID of the object that will be opened when the environment starts. In this case, 22 refers to the Page object that will load.

`"startupObjectType": "Page"`: Defines the type of object to be opened at startup, which is a Page (a UI object in Business Central).

`"schemaUpdateMode": "ForceSync"`: Forces synchronization of the schema when the extension is published. This makes sure that any changes to the database structure will sync automatically.

`"breakOnError": "All"`: Specifies that the debugger will break (pause) on all errors.

`"launchBrowser": true`: Automatically launches a browser window to display the sandbox environment when debugging starts.

`"enableLongRunningSqlStatements": true`: Enables monitoring of long-running SQL statements to help in optimizing performance.

`"enableSqlInformationDebugger": true`: Turns on the SQL debugger to track detailed SQL activity during debugging.

`"name": "AL: Generated Download Symbols request"`: This is an auto-generated configuration that helps download symbols, which are necessary to compile and run AL extensions.

`"request": "launch"`: This will launch a request to download symbols.

`"type": "al"`: Specifies that this configuration is for AL code.

`"environmentType": "Sandbox"`: The environment type is Sandbox.

`"environmentName": "SandboxTest"`: The name of the sandbox environment is SandboxTest.

`"tenant": "e04c56ea-acd1-4a65-b87d-a1eb9289afd4"`: This specifies the tenant ID, which identifies the environment in the cloud.

1. Basic Functionalities

Business Central is an enterprise resource planning (ERP) solution that helps businesses manage various operations such as finance, manufacturing, sales, shipping, project management, and services. As a developer, you will interact with its customizable features, including:

Customization and Extensions: BC allows customization of existing functionalities and development of new features through extensions using AL language.
Data Management: Developers work with tables, pages, reports, and code units to manage business data.
Integration: BC integrates with other Microsoft products like Office 365, Power BI, and Azure services, enabling seamless business operations.

2. Naming Objects

In Business Central, objects such as tables, pages, and reports are core components used to **define the structure and behavior of data**.

Tables: Should reflect the data they store, e.g., **Customer**, SalesInvoiceHeader.
Pages: Should indicate the purpose of the user interface, e.g., CustomerList, **SalesInvoiceCard**.
Reports: Should describe the output, e.g., SalesInvoiceReport, CustomerStatement.
Codeunits: Should indicate the functionality or logic they encapsulate, e.g., PostingManagement, **PriceCalculation**.

3. Objects Numbering

Object numbering is a critical aspect of development in BC, as it helps distinguish different types of objects and ensures there are no conflicts in extensions:

- **Standard Range:** Microsoft uses the object numbers from 1 to 50,000 for their standard objects. Developers must avoid using this range to prevent conflicts.

- **Custom Range:** Objects developed by third parties or within custom extensions typically use numbers above 50,000. Companies might have designated ranges for their custom developments.
- **Partner Extensions:** Partners often have their own object ranges assigned by Microsoft. For example, 50,001 to 99,999 might be reserved for a specific partner's use. Can be used by companies.

1. Creating and Modifying Tables

Example of Creating a Table

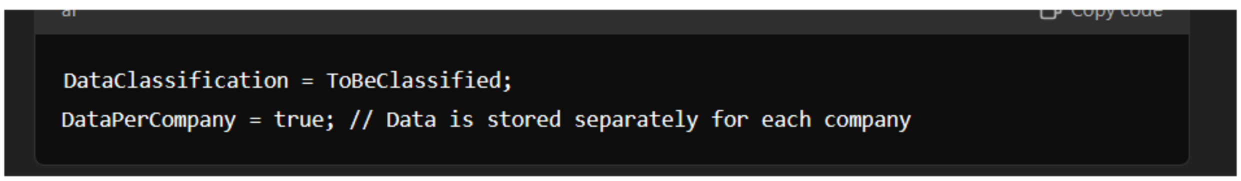
Here's how to create a simple `Employee` table:

```
table 50700 "Employee Table"
{
    DataClassification = ToBeClassified;
    Caption = 'Employee table';

    fields
    {
        field(50701; "EmployeeID"; Code[20])
        {
            DataClassification = ToBeClassified;
            Caption = 'Employee ID';
        }
        field(50702; "DepartmentID"; Code[20])
        {
            DataClassification = ToBeClassified;

            TableRelation = "Department table".DepartmentID;

        }

        field(50703; "FirstName"; Text[20])

        {

            DataClassification = ToBeClassified;

        }

        field(50704; "LastName"; Text[20])

        {

            DataClassification = ToBeClassified;

        }

        field(50705; "JoiningDate"; Date)

        {

            DataClassification = ToBeClassified;

        }

        field(50706; "Salary"; Decimal)

        {

            DataClassification = ToBeClassified;

        }

        field(50707; "ProbationEndDate"; Date)

        {

            DataClassification = ToBeClassified;

            Caption = 'Probation End Date';
        }



    }

    keys

    {

        key(Pk; EmployeeID)

        {

            Clustered = true;

        }

        key(Departmentkey; "DepartmentID")

        {

            Clustered = false;

        }

        end;
```

2. Table Properties

Table properties define the behavior and characteristics of the table as a whole. Some common table properties include:

- **DataClassification:** Indicates how the data in the table should be classified for security and privacy.
- **Caption:** The text that describes the table in the UI.
- **DataPerCompany:** Defines whether the data is stored per company or shared across all companies.
- **DrillDownPageID and LookupPageID:** Specify the pages used for drill-down and lookup functions, respectively.

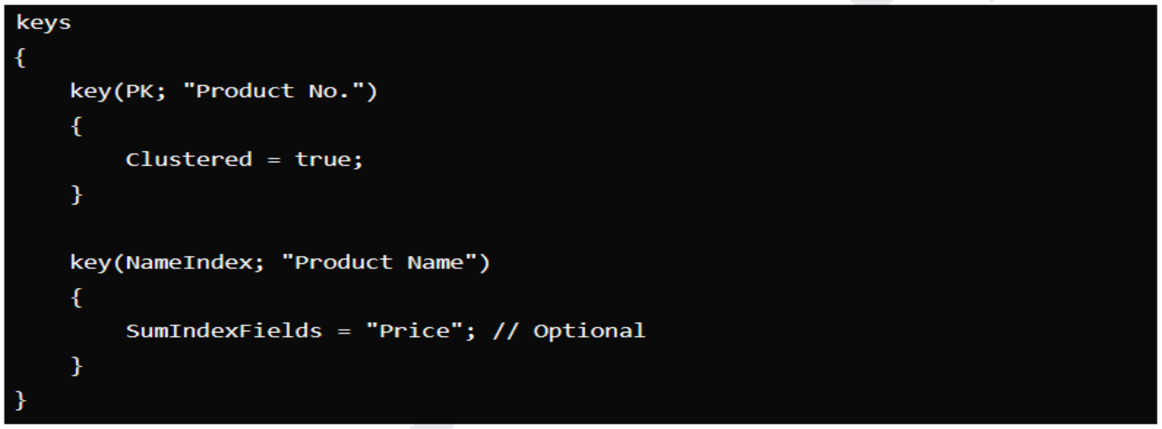


```
DataClassification = ToBeClassified;
DataPerCompany = true; // Data is stored separately for each company
```

3. Keys

Keys are essential for indexing and uniquely identifying records in a table. There are two types of keys:

- **Primary Key:** Uniquely identifies each record in the table.
- **Secondary Keys:** Used to sort and search data more efficiently.

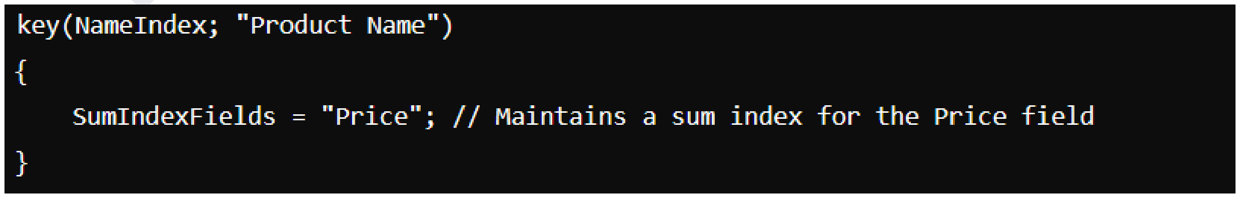


```
keys
{
    key(PK; "Product No.")
    {
        Clustered = true;
    }

    key(NameIndex; "Product Name")
    {
        SumIndexFields = "Price"; // Optional
    }
}
```

4. SumIndexFields

SumIndexFields are used in conjunction with keys to maintain totals of numeric fields, which can then be quickly retrieved. This is useful for reports and queries that need summarized data.

Example:

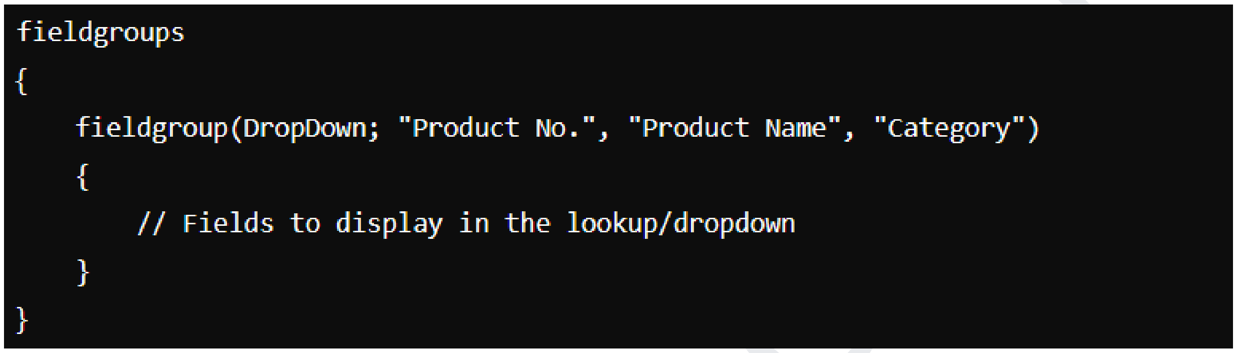


```
key(NameIndex; "Product Name")
{
    SumIndexFields = "Price"; // Maintains a sum index for the Price field
}
```

5. Field Groups

Field Groups control how fields are displayed together in certain UI components like lookup windows. A common field group is the `DropDown` group, which is used in lookup lists.

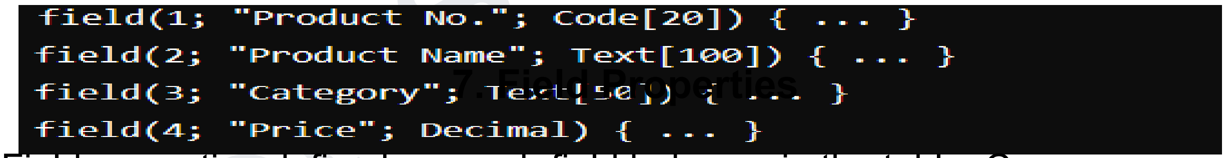
Example of a field group:



```
fieldgroups
{
    fieldgroup(DropDown; "Product No.", "Product Name", "Category")
    {
        // Fields to display in the lookup/dropdown
    }
}
```

6. Field Numbering

Fields in a table are numbered sequentially starting from 1. This numbering is crucial because it dictates the order in which fields are processed and stored. It’s also important to avoid gaps and duplicates in numbering.

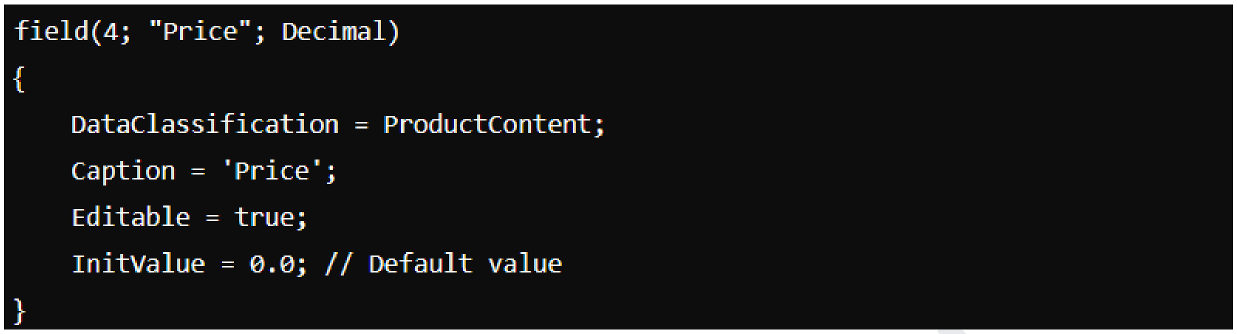


```
field(1; "Product No."; Code[20]) { ... }
field(2; "Product Name"; Text[100]) { ... }
field(3; "Category"; Text[50]) { ... }
field(4; "Price"; Decimal) { ... }
```

7. Field Properties

Field properties define how each field behaves in the table. Common properties include:

- **DataType:** Defines the type of data the field can store (e.g., `Text`, `Integer`, `Decimal`).
- **Length:** For text fields, define the maximum number of characters.
- **DataClassification:** Used for security and privacy settings.
- **Editable:** Controls whether the field can be edited by the user.
- **InitValue:** Specifies an initial value for the field.



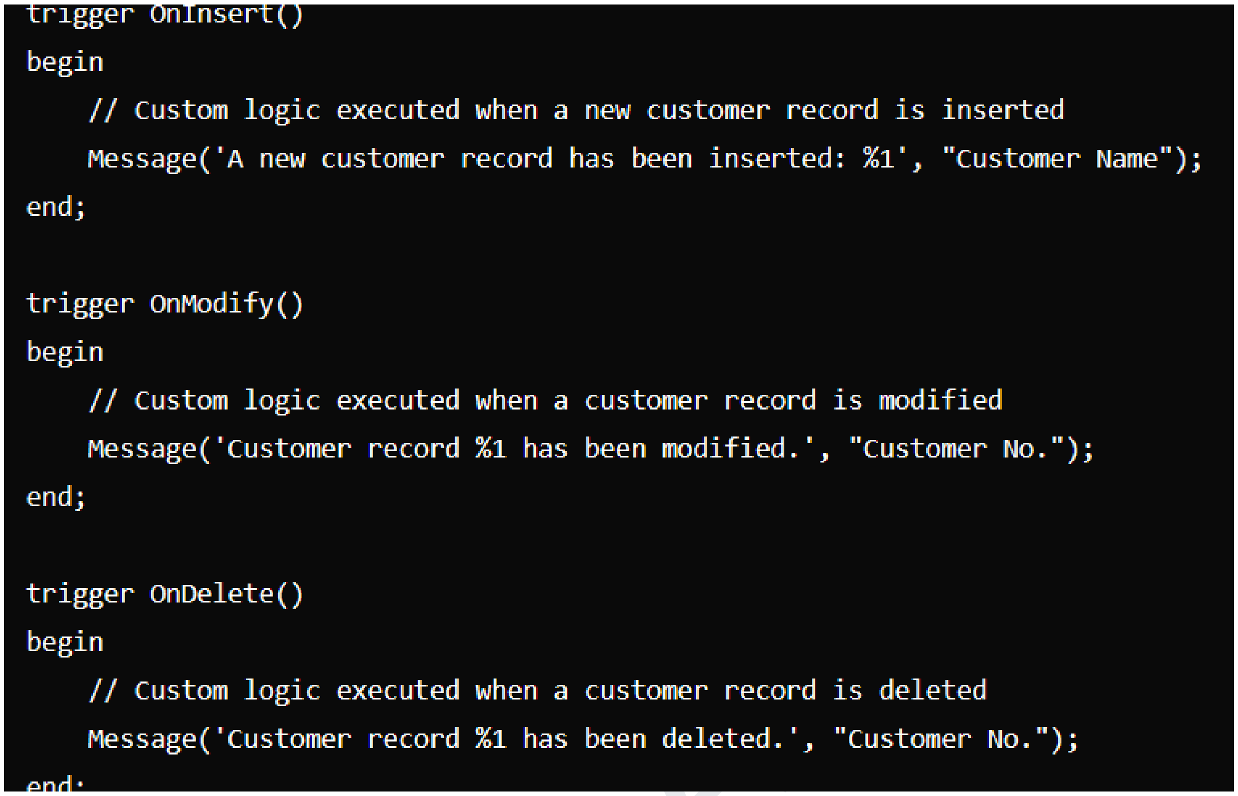
```
field(4; "Price"; Decimal)
{
    DataClassification = ProductContent;
    Caption = 'Price';
    Editable = true;
    InitValue = 0.0; // Default value
}
```

1. Table Triggers

Table triggers are event-driven methods in Dynamics 365 Business Central that execute custom code in response to specific actions on the table, such as inserting, modifying, or deleting records.

Common Table Triggers:

- **OnInsert()**: Triggered when a new record is inserted into the table.
- **OnModify()**: Triggered when an existing record is modified.
- **OnDelete()**: Triggered when a record is deleted from the table.
- **OnRename()**: Triggered when a record is renamed (if the primary key is changed).

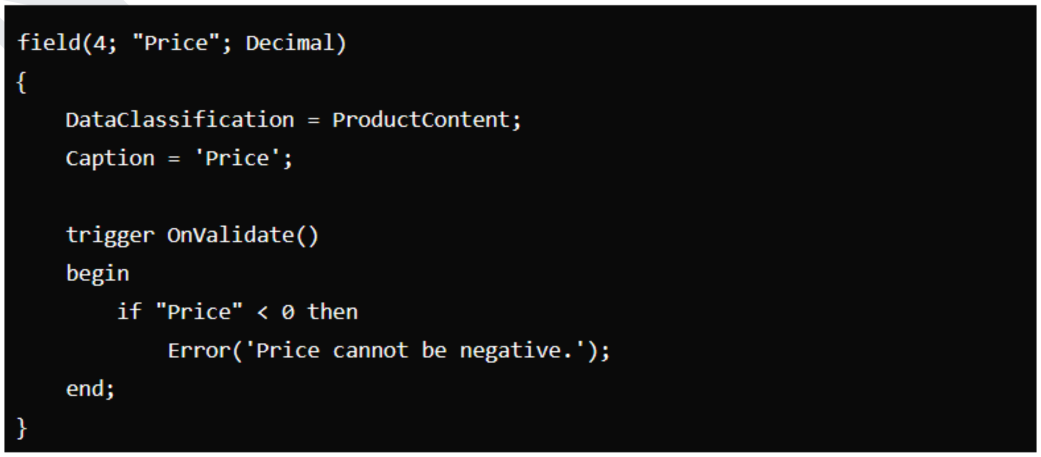

```
trigger OnInsert()
begin
    // Custom logic executed when a new customer record is inserted
    Message('A new customer record has been inserted: %1', "Customer Name");
end;

trigger OnModify()
begin
    // Custom logic executed when a customer record is modified
    Message('Customer record %1 has been modified.', "Customer No.");
end;

trigger OnDelete()
begin
    // Custom logic executed when a customer record is deleted
    Message('Customer record %1 has been deleted.', "Customer No.");
end;
```

2. Field Triggers

Field triggers are methods associated with individual fields that execute custom code when the field's value changes or when it is validated.

Common Field Triggers:

- **OnValidate()**: Triggered when the field value is validated, usually when a user exits the field after changing its value.
- **OnLookup()**: Triggered when a lookup is performed on the field.

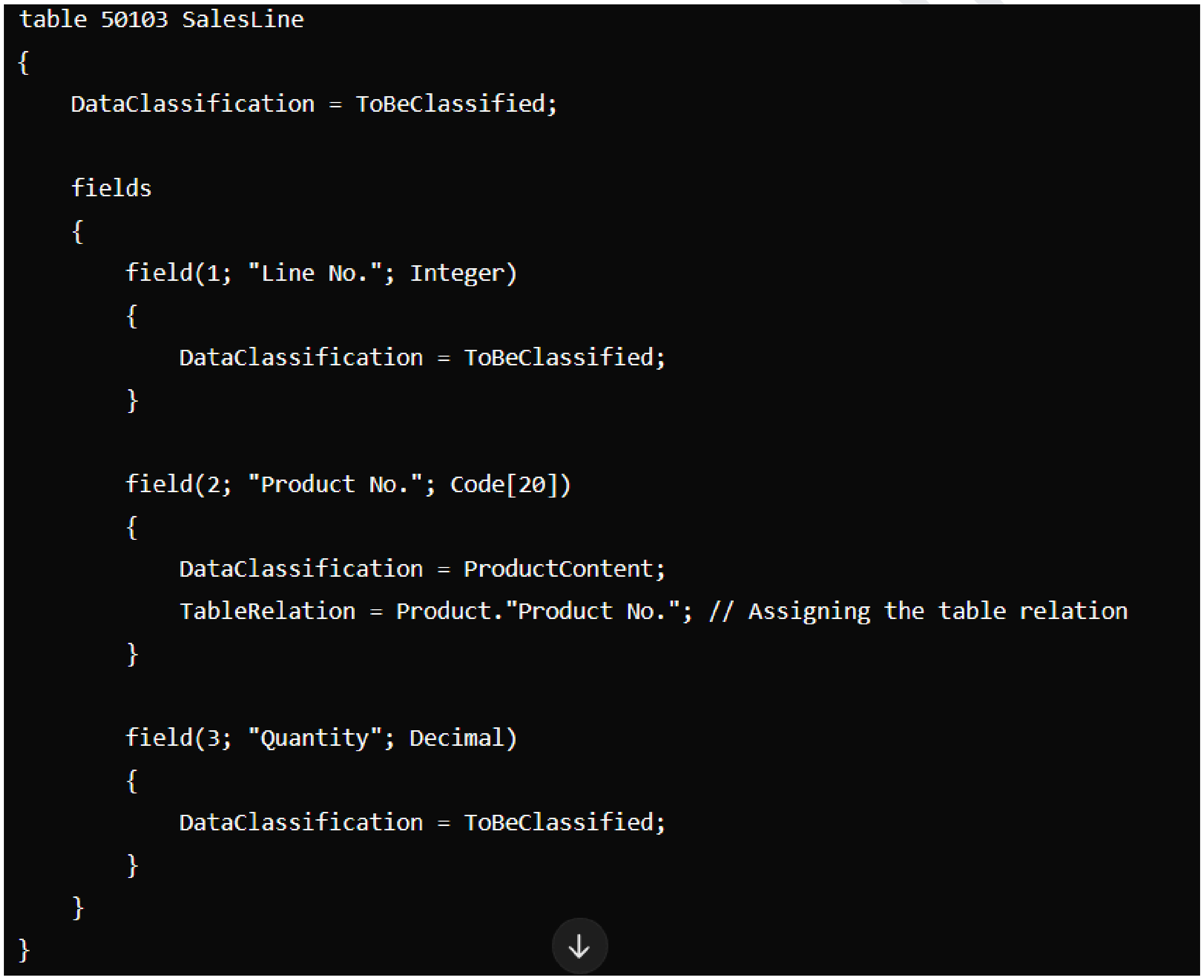

```
field(4; "Price"; Decimal)
{
    DataClassification = ProductContent;
    Caption = 'Price';

    trigger OnValidate()
    begin
        if "Price" < 0 then
            Error('Price cannot be negative.');
    end;
}
```

3. Assigning a Table Relation Property

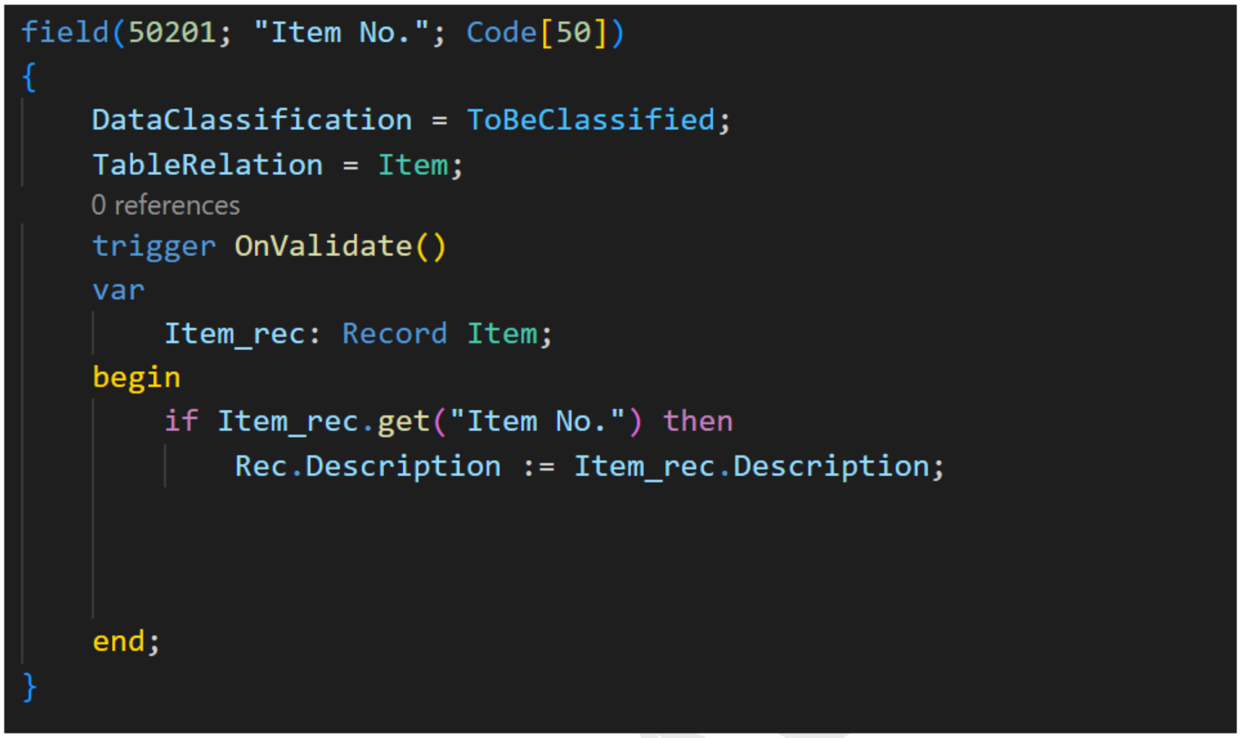
Table relations are used to define relationships between tables, enabling lookups, validations, and enforcing referential integrity.

Example:

Suppose you have a `Sales Line` table that references the `Product` table. You can use the `TableRelation` property to link the two.



```
table 50103 SalesLine
{
    DataClassification = ToBeClassified;

    fields
    {
        field(1; "Line No."; Integer)
        {
            DataClassification = ToBeClassified;
        }

        field(2; "Product No."; Code[20])
        {
            DataClassification = ProductContent;
            TableRelation = Product."Product No."; // Assigning the table relation
        }

        field(3; "Quantity"; Decimal)
        {
            DataClassification = ToBeClassified;
        }
    }
}
```

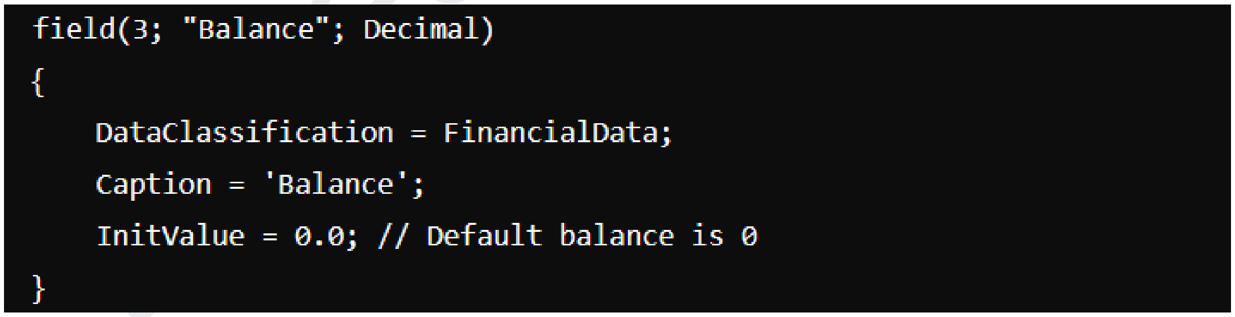


```
field(50201; "Item No."; Code[50])
{
    DataClassification = ToBeClassified;
    TableRelation = Item;
    0 references
    trigger OnValidate()
    var
        Item_rec: Record Item;
    begin
        if Item_rec.get("Item No.") then
            Rec.Description := Item_rec.Description;



    end;
}
```

4. Assigning an InitValue Property

The `InitValue` property is used to set a default value for a field when a new record is created.

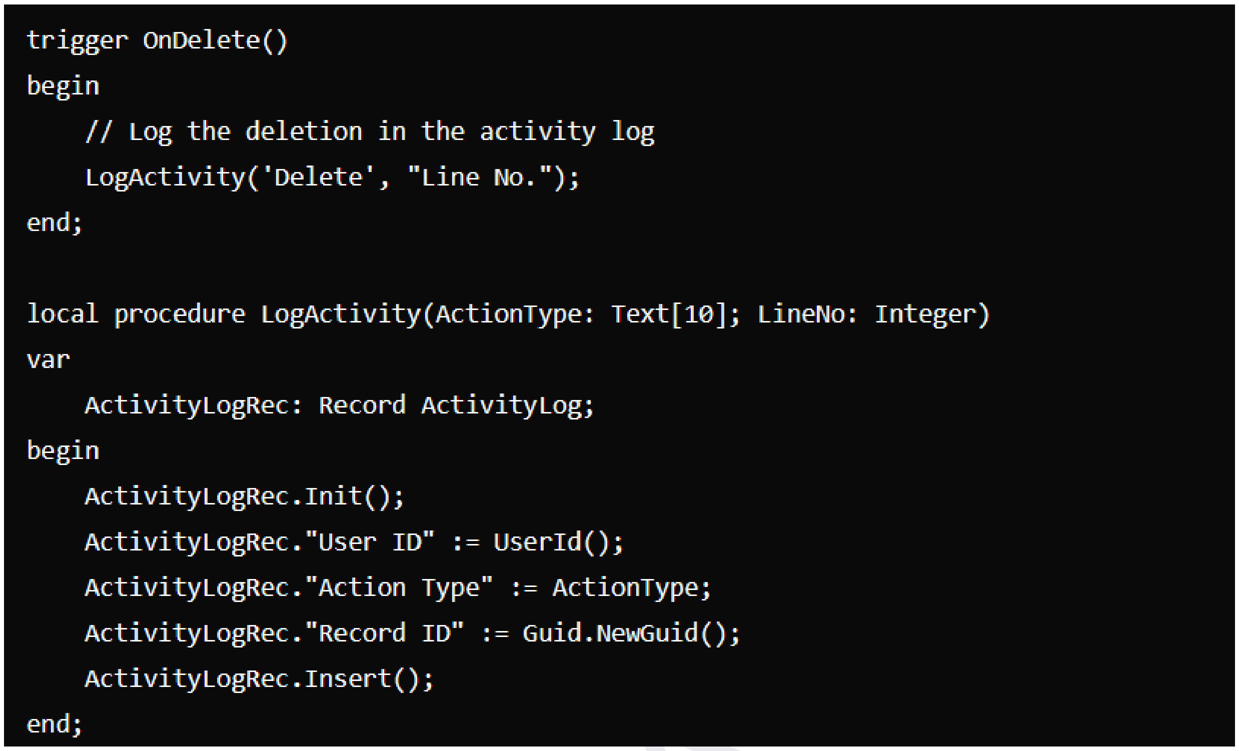
Example:



```
field(3; "Balance"; Decimal)
{
    DataClassification = FinancialData;
    Caption = 'Balance';
    InitValue = 0.0; // Default balance is 0
}
```

5. Adding a Few Activity-Tracking Tables

Activity-tracking tables can be used to log changes, actions, or other activities related to your data. Here's an example of a simple activity-tracking table.



```
trigger OnDelete()
begin
    // Log the deletion in the activity log
    LogActivity('Delete', "Line No.");
end;

local procedure LogActivity(ActionType: Text[10]; LineNo: Integer)
var
    ActivityLogRec: Record ActivityLog;
begin
    ActivityLogRec.Init();
    ActivityLogRec."User ID" := UserId();
    ActivityLogRec."Action Type" := ActionType;
    ActivityLogRec."Record ID" := Guid.NewGuid();
    ActivityLogRec.Insert();
end;
```

Types of the table

Here's an overview of the various types of tables, their roles, and examples where applicable:

1. Master Data Tables

- **Purpose**: Store core business data that remains relatively stable over time. These tables typically contain records that are used across multiple transactions and processes.
- **Examples**:
    - **Customer** table: Stores customer information.
    - **Vendor** table: Stores supplier information.
    - **Item** table: Stores product or inventory information.

2. Journal Tables Purpose: Used for temporary storage of transactional data before posting. These tables often contain unposted transactions that are yet to be finalized and posted to the ledger.

- **Examples**:
    - **General Journal Line** table: Stores general ledger entries before posting.
    - **Item Journal Line** table: Stores inventory movements before posting.

3. Template Tables

- **Purpose**: Used to store default values or configurations that can be applied to new records. Templates help standardize data entry and ensure consistency.
- **Examples**:
    - **Customer Template** table: Stores default settings for creating new customers.
    - **Item Template** table: Stores default settings for creating new items.

4. Entry Tables

- **Purpose**: Contain detailed transaction records that have been posted and cannot be changed. These tables form the historical records of transactions in the system.
- **Examples**:
    - **G/L Entry** table: Stores posted general ledger entries.
    - **Item Ledger Entry** table: Stores posted inventory transactions.

5. Subsidiary (Supplementary) Tables

- **Purpose**: Store additional details or sub-data related to records in a primary table. They often hold supplementary information that supports the main data.
- **Examples**:
    - **Customer Bank Account** table: Stores bank account details for customers.
    - **Vendor Ledger Entry** table: Stores detailed financial transactions related to vendors.

6. Register Tables

- **Purpose**: Store information about batches of posted entries, typically to maintain an audit trail of postings.

- **Examples**:
    - **G/L Register** table: Tracks the batches of general ledger entries.
    - **Item Register** table: Tracks the batches of item ledger entries.

7. Posted Document Tables

- **Purpose**: Store records of documents that have been finalized and posted, such as invoices, orders, and shipments. These documents represent completed transactions.
- **Examples**:
    - **Sales Invoice Header** table: Stores posted sales invoices.
    - **Purchase Order Header** table: Stores posted purchase orders.

8. Singleton Tables

- **Purpose**: Designed to hold a single record that represents a global setting or configuration. Singleton tables typically manage system-wide settings.
- **Examples**:
    - **Company Information** table: Stores company-wide information like address, logo, etc.
    - **General Ledger Setup** table: Stores global settings for the general ledger.

9. Temporary Tables

- **Purpose**: Used for temporary data storage during processing. These tables exist only in memory and are discarded after the process completes.
- **Examples**:
    - Temporary tables are often used in AL code for processing data, such as calculating totals or preparing data for reports

10. System Tables

- **Purpose**: Contain metadata and system-related information used by Business Central to manage the database, user permissions, and other core functions.
- **Examples**:
    - **User** table: Stores user login information.
    - **Permission Set** table: Stores permission sets for users.

11. Virtual Tables
Purpose: Tables that do not physically exist in the database but provide data at runtime.
Examples: Date, Integer, Session, User.
==

Tasks

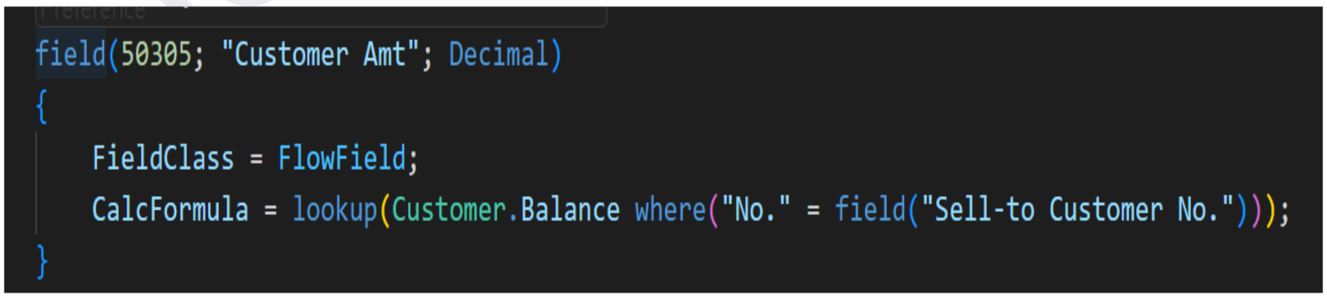
Create Table Total Amt. and use flowfield to calculate the SUM, AVG, MIN, MAX from sales Line table to sales order page.

Solutions:

```
field(50304; "Total Amt"; Decimal)
{

    FieldClass = FlowField;
    CalcFormula = sum("Sales Line".Amount where("Document No." = field("No.")));
     // calcFormula = Max("Sales Line".Amount Where("Document No." = field("No.")));
  / calcFormula = Min("Sales Line".Amount Where("Document No." = field("No.")));
    // calcFormula = Average("Sales Line".Amount Where("Document No." =
field(Upperlimit("No."))));

      }
```

Task 2 similar to above : but In that Customer amt lookup from Balance field of customer table.

Solutions

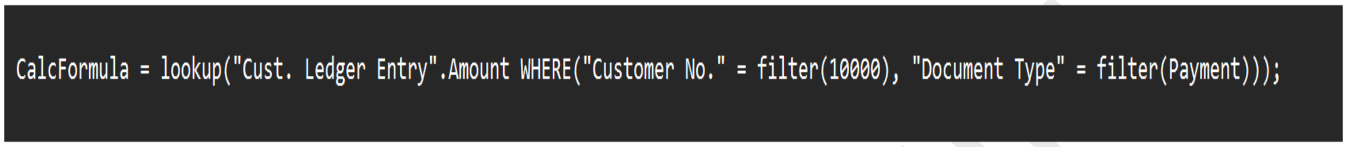


```
field(50305; "Customer Amt"; Decimal)
{
    FieldClass = FlowField;
    CalcFormula = lookup(Customer.Balance where("No." = field("Sell-to Customer No.")));
}
```

Task3

USe Flow Field and calcFormula to lookup data from Cus Ledger ENtry where Cus No =10000 and Document type :: Payment.

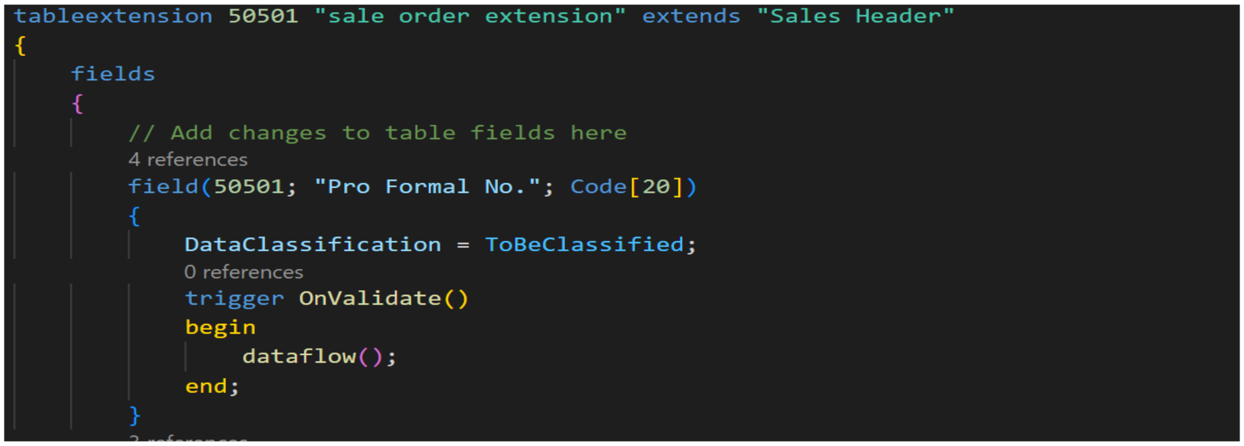
Solutions :



```
CalcFormula = lookup("Cust. Ledger Entry".Amount WHERE("Customer No." = filter(10000), "Document Type" = filter(Payment)));
```

Table Extension

Table extensions in Microsoft Dynamics 365 Business Central allow you to add extra fields or change some properties on an existing table. This enables you to extend the functionality of standard tables without modifying the base application. Here are the key points:



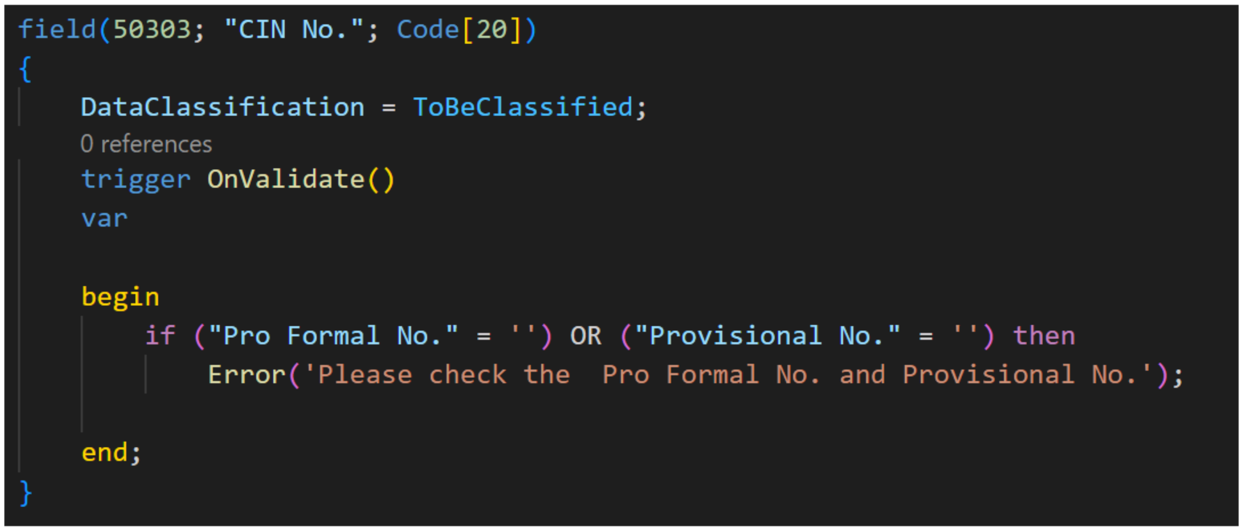
```
tableextension 50501 "sale order extension" extends "Sales Header"
{
    fields
    {
        // Add changes to table fields here
        4 references
        field(50501; "Pro Formal No."; Code[20])
        {
            DataClassification = ToBeClassified;
            0 references
            trigger OnValidate()
            begin
                dataflow();
            end;
        }
```

Keys Properties of table extension.

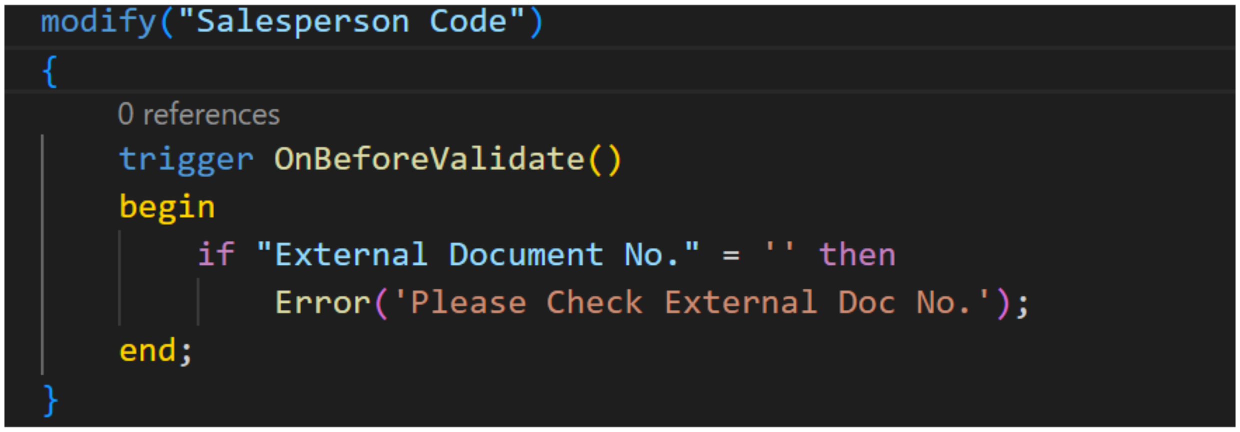
- **Properties: You can overwrite some properties on fields in the base table using table extensions. For example, you can change the `Caption`, `Editable`, or `DataClassification` properties1.**
- **Keys: You can define new keys for fields added in the table extension or extend keys provided in the base table1.**

```
tableextension 50115 MyTableExtension extends Customer
{
    fields
    {
        modify(Name)
        {
            Caption = 'Customer Name';
            Editable = false;
        }
    }
```

Validate trigger on STD field : Task (Modify Procedure)



```
field(50303; "CIN No."; Code[20])
{
    DataClassification = ToBeClassified;
    0 references
    trigger OnValidate()
    var

    begin
        if ("Pro Formal No." = '') OR ("Provisional No." = '') then
            Error('Please check the  Pro Formal No. and Provisional No.');

    end;
}
```



```
modify("Salesperson Code")
{
    0 references
    trigger OnBeforeValidate()
    begin
        if "External Document No." = '' then
            Error('Please Check External Doc No.');
    end;
}
```

Pages In BC

For Data Visualization on the Front-end ; that is UI of the Data

Page Types in Business Central

Page Type	Use	Characteristics
RoleCenter	Overview of business performance and start page for a specific role	Collection of parts (Cues, KPIs, etc.) and navigation pane contents
Card	Master, reference, and setup data management	Single entity, titled entity with FastTabs, may embed parts
Document	Transaction and other document management	Single entity, titled entity with FastTabs, document lines follow header section
ListPlus	Statistics, details, and related data management	Single entity, titled entity with at least one ListPart, fields above/below part(s)
List	Entity overviews, navigation, and inline editing of simple entities	Collection of entities/entries, single list with caption, may have field groups

Worksheet	Line-based data entry tasks (e.g., journals) and inquiries	Collection of entities, single list/table with caption, may have field groups
StandardDialog	Routine dialog that starts or progresses a task	Single or collection, cancelable dialog with instruction, can have groups of fields
ConfirmationDialog	Confirmative or exceptional dialog (e.g., warnings)	Single or collection, Yes/No dialog with instruction, can have groups of fields
NavigatePage	Multi-step dialog (also known as a "Wizard")	Single or collection, can have groups of fields and parts

[1]: Microsoft Learn - Page Types and Layouts [2]: Microsoft Learn - Pages Overview

Page Properties:

In Microsoft Dynamics 365 Business Central, page properties define the behavior and appearance of pages within the application. Here are some key properties you might encounter:

1. **Caption: This is the title displayed in the page header and in the navigation pane. It helps users understand the purpose of the page.**
2. **Application Area: Specifies the areas of the application where the page is applicable. This helps in managing visibility across different user roles.**

3. **SourceTable: Indicates the primary table that the page interacts with. It determines which data is displayed and manipulated on the page.**
4. **PageType: Defines the type of page (e.g., List, Card, Worksheet, etc.). Each type serves a different purpose, such as displaying lists of records or details of a single record.**
5. **Layout: This property manages how fields and controls are arranged on the page, including grouping, positioning, and alignment.**
6. **DataItem: Specifies the data items used in the page. It can include multiple data items if the page displays related data from different tables.**
7. **Visible: A property that can be set to true or false to control the visibility of the page in the user interface based on certain conditions.**
8. **OnOpenPage Trigger: A code trigger that executes when the page is opened. You can use this to initialize variables, set filters, or perform other actions.**
9. **Actions: Defines the buttons and menu options available on the page, allowing users to perform various operations.**
10. **Field Properties: Each field on a page can have specific properties like Required, Editable, Enabled, and Visible to control user interactions.**