In Dynamics 365 Business Central, table relations are a crucial aspect of managing data integrity and defining how different tables interact with each other. Here's an explanation of the key properties related to table relations, why we use them, and how to use them in AL code with examples in Visual Studio Code (VS Code).

1. TableRelation

- **Purpose:** Defines a relationship between a field in one table and the primary key field in another table. It ensures that the value entered in the field must exist in the related table's field.

Syntax:
al
Copy code
```
field(<FieldID>; <FieldName>; <DataType>)
{
    TableRelation = <RelatedTable>.<RelatedField>;
}
```

-

Example:
al
Copy code
```
field(1; "Customer No."; Code[20])
{
    TableRelation = Customer."No.";
}
```

-
- **Explanation:** The `Customer No.` field in your table relates to the `No.` field in the `Customer` table, ensuring that only existing customer numbers can be entered.

2. FieldRelation

- **Purpose:** Adds more specific conditions to the relationship defined by `TableRelation`, typically used to filter related records based on certain criteria.

Syntax:
al
Copy code
```
field(<FieldID>; <FieldName>; <DataType>)
```

```
{
    TableRelation = <RelatedTable>.<RelatedField> WHERE
(<ConditionField>=CONST(<Value>));
}
```

-

Example:
al
Copy code
```
field(1; "Vendor No."; Code[20])
{
    TableRelation = Vendor."No." WHERE (Blocked=CONST(false));
}
```

-
- **Explanation:** This example ensures that the `Vendor No.` field only allows vendors that are not blocked (`Blocked = false`).

3. ValidateTableRelation

- **Purpose:** Validates the relationship when the value in the field is modified or inserted, ensuring that the value exists in the related table.

Syntax:
al
Copy code
```
field(<FieldID>; <FieldName>; <DataType>)
{
    TableRelation = <RelatedTable>.<RelatedField>;
    ValidateTableRelation = true;
}
```

-

Example:
al
Copy code
```
field(1; "Item No."; Code[20])
{
    TableRelation = Item."No.";
    ValidateTableRelation = true;
```

```
}
```

- **Explanation:** This ensures that when a user inputs or changes the `Item No.`, it is validated against the `No.` field in the `Item` table.

4. OnLookup Trigger

- **Purpose:** Allows customization of the lookup behavior for a field that has a table relation. This is useful if you want to modify how users search for related records.

Syntax:
al
Copy code

```
field(<FieldID>; <FieldName>; <DataType>)
{
    TableRelation = <RelatedTable>.<RelatedField>;

    trigger OnLookup(var Text: Text): Boolean
    begin
        // Custom lookup code
    end;
}
```

Example:
al
Copy code

```
field(1; "Customer No."; Code[20])
{
    TableRelation = Customer."No.";

    trigger OnLookup(var Text: Text): Boolean
    begin
        // Custom logic for customer lookup
        // For example, filter customers based on a specific condition
    end;
}
```

- **Explanation:** This customizes how the lookup for `Customer No.` works, allowing you to add filtering or custom behavior when the user searches for a customer.

5. TableRelation Filters

- **Purpose:** Adds additional filters to the `TableRelation` property, allowing more complex conditions.

Syntax:
al
Copy code
```
field(<FieldID>; <FieldName>; <DataType>)
{
    TableRelation = <RelatedTable>.<RelatedField> WHERE
(<FilterField>=FILTER(<Value>));
}
```

-

Example:
al
Copy code
```
field(1; "Salesperson Code"; Code[20])
{
    TableRelation = Salesperson."Code" WHERE (Active=FILTER(Yes));
}
```

-
- **Explanation:** This filter ensures that only active salespersons can be selected.

6. UseTemporary

- **Purpose:** Specifies whether a table relation is temporary, meaning the data is not stored in the database but is used temporarily for calculations or processing.

Syntax:
al
Copy code
```
field(<FieldID>; <FieldName>; <DataType>)
{
    TableRelation = <RelatedTable>.<RelatedField>;
    UseTemporary = true;
}
```

-

Example:
al
Copy code
```
field(1; "Temp Sales Header"; Record "Sales Header")
{
    TableRelation = "Sales Header"."No.";
    UseTemporary = true;
}
```

-
- **Explanation:** This would be used in scenarios where you want to reference a temporary instance of a table rather than a permanent one.

Summary of Use-Cases:

- **Basic Lookup:** Use `TableRelation` to enforce that a field's value corresponds to a valid record in another table.
- **Conditional Lookup:** Use `FieldRelation` or filters within `TableRelation` to restrict the valid records based on additional conditions.
- **Custom Lookup:** Use the `OnLookup` trigger to customize how users search for and select records.
- **Temporary Data:** Use `UseTemporary` when you need a temporary table relation for calculations or processing that doesn't affect the database.

Example Table with Various Properties:

al
Copy code
```
table 50230 "Relation table1"
{
    DataClassification = ToBeClassified;

    fields
    {
        field(1; "Customer No."; Code[20])
        {
            DataClassification = ToBeClassified;
            TableRelation = Customer."No.";
        }
```

```
        field(2; "Item No."; Code[20])
        {
            DataClassification = ToBeClassified;
            TableRelation = Item."No.";
            ValidateTableRelation = true;
        }

        field(3; "Salesperson Code"; Code[20])
        {
            DataClassification = ToBeClassified;
            TableRelation = Salesperson."Code" WHERE
(Active=FILTER(Yes));
        }

        field(4; "Document No."; Code[20])
        {
            DataClassification = ToBeClassified;
            TableRelation = "Sales Header"."No." WHERE ("Document
Type"=CONST(Order));

            trigger OnLookup(var Text: Text): Boolean
            begin
                // Custom lookup logic
            end;
        }
    }

    keys
    {
        key(Pk; "Customer No.")
        {
            Clustered = true;
        }
    }
}
```

Conclusion:

These properties are essential tools in Dynamics 365 Business Central to maintain data integrity and ensure that users select valid and relevant data. Each property serves a specific purpose, from basic relationships to advanced filtering and custom behaviors, and understanding how to use them effectively is key to developing robust business solutions.