Triggers in Apex: Explained Like a 5-Year-Old

Imagine you have a magic notebook. Every time someone writes something in it, the notebook knows what to do next automatically. For example, if someone writes "new toy," the notebook creates a thank-you note to send to the toy shop.

In Salesforce, a **trigger** is like that magic notebook. It listens for certain actions, like when a record is created, updated, or deleted, and then it does something automatically based on your instructions.

Triggers in Apex: Industry-Level Explanation

A **trigger** in Apex is a piece of code that executes automatically in response to specific events on an object in Salesforce. Triggers can be used to:

- Automate processes.
- Enforce complex business rules.
- Interact with related records.

Trigger Contexts

Triggers can run in two contexts:

1. **Before Triggers**: Used to validate or modify records before they are saved to the database.
2. **After Triggers**: Used to perform actions after records are saved to the database, such as creating related records.

Trigger Events

Triggers can respond to the following events:

- `before insert`
- `before update`
- `before delete`
- `after insert`
- `after update`
- `after delete`
- `after undelete`

Example Trigger: Automatically Create a Task When an Opportunity is Created

Code:

```
trigger CreateTaskOnOpportunity on Opportunity (after insert) {
    List<Task> tasks = new List<Task>();
    for (Opportunity opp : Trigger.new) {
        Task t = new Task(
            Subject = 'Follow up on Opportunity',
            WhatId = opp.Id,
            Status = 'Not Started'
        );
        tasks.add(t);
    }
    insert tasks;
}
```

Word-by-Word Explanation:

1. **`trigger`**: This keyword tells Salesforce that you are defining a trigger.
2. **`CreateTaskOnOpportunity`**: The name of the trigger. You can name it anything, but it should describe what the trigger does.
3. **`on`**: Specifies the object the trigger is associated with.
4. **`Opportunity`**: The object on which the trigger operates. Here, it's the `Opportunity` object.
5. **`(after insert)`**: Defines the event(s) that fire the trigger. In this case, it's after an Opportunity is inserted into the database.
6. **`{`**: Opens the body of the trigger.
7. **`List<Task> tasks`**: Declares a list named `tasks` that will store multiple `Task` records.
8. **`= new List<Task>();`**: Initializes the list so it can store Task records.
9. **`for (Opportunity opp : Trigger.new)`**: A loop that goes through each new Opportunity record being processed.
    - **`Opportunity`**: Specifies the type of each item in the loop (an Opportunity record).
    - **`opp`**: A variable name for the current Opportunity record in the loop.
    - **`:`**: Separates the variable (`opp`) from the collection (`Trigger.new`).
    - **`Trigger.new`**: A built-in context variable that holds the list of new records.
10. **`{`**: Opens the body of the loop.
11. **`Task t = new Task(`**: Creates a new Task record in memory and assigns it to the variable `t`.

- `Task`: The Salesforce object being created.
- `t`: A variable name for the new Task record.
- `=`: Assigns the new Task to the variable `t`.
- `new Task(`: Initializes a new Task object.

12. `Subject = 'Follow up on Opportunity',`: Sets the `Subject` field of the Task to "Follow up on Opportunity."
    - `Subject`: A field in the Task object.
    - `=`: Assigns a value to the field.
    - `'Follow up on Opportunity'`: The value being assigned to the `Subject` field.
    - `,`: Indicates there are more fields to set.
13. `WhatId = opp.Id,`: Links the Task to the Opportunity by assigning the Opportunity's ID to the `WhatId` field.
    - `WhatId`: A field in the Task object that links it to another record.
    - `=`: Assigns a value to the field.
    - `opp.Id`: Refers to the ID of the current Opportunity record.
    - `,`: Indicates there are more fields to set.
14. `Status = 'Not Started'`: Sets the `Status` field of the Task to "Not Started."
    - `Status`: A field in the Task object.
    - `=`: Assigns a value to the field.
    - `'Not Started'`: The value being assigned to the `Status` field.
15. `);`: Closes the Task initialization.
16. `tasks.add(t);`: Adds the newly created Task to the `tasks` list.
    - `tasks`: The list of Task records.
    - `.add`: A method to add an item to the list.
    - `(t)`: Specifies the item to add (the Task record `t`).
    - `;`: Ends the statement.
17. `}`: Closes the loop.
18. `insert tasks;`: Inserts all the Task records in the `tasks` list into the database.
    - `insert`: A DML operation to save records to the database.
    - `tasks`: The list of Task records to insert.
    - `;`: Ends the statement.
19. `}`: Closes the trigger.

Additional Concepts Related to Triggers

Trigger.new

- A collection of new records being processed by the trigger.
- Available in `insert` and `update` triggers.

Trigger.old

- A collection of original records before any changes were made.
- Available in `update` and `delete` triggers.

Best Practices for Triggers

1. **One Trigger Per Object**:
    - Avoid multiple triggers on the same object to reduce complexity.
2. **Use Handler Classes**:
    - Delegate trigger logic to a separate Apex class for better organization and reusability.
3. **Bulkify Your Code**:
    - Ensure your trigger can handle multiple records efficiently.
4. **Avoid SOQL/DML Inside Loops**:
    - Minimize database operations within loops to avoid hitting governor limits.

Real-World Example: Trigger to Update a Related Record

Scenario:

When a `Case` is closed, update the related `Account`'s `Last_Case_Closed_Date__c` field.

Code:

```
trigger UpdateAccountOnCaseClosure on Case (after update) {
    Map<Id, Date> accountUpdates = new Map<Id, Date>();

    for (Case c : Trigger.new) {
        if (c.IsClosed && !Trigger.oldMap.get(c.Id).IsClosed) {
            accountUpdates.put(c.AccountId, c.ClosedDate);
        }
```

```
    }

    List<Account> accountsToUpdate = new List<Account>();
    for (Id accId : accountUpdates.keySet()) {
        accountsToUpdate.add(new Account(
            Id = accId,
            Last_Case_Closed_Date__c = accountUpdates.get(accId)
        ));
    }

    update accountsToUpdate;
}
```

Explanation:

1. `Map<Id, Date> accountUpdates = new Map<Id, Date>();`
    - Prepares a map to store Account IDs and their corresponding last case closed dates.
2. `for (Case c : Trigger.new) {`
    - Iterates through updated Case records.
3. `if (c.IsClosed && !Trigger.oldMap.get(c.Id).IsClosed) {`
    - Checks if the Case was just closed (it wasn't closed before).
4. `accountUpdates.put(c.AccountId, c.ClosedDate);`
    - Adds the Account ID and Closed Date to the map.
5. `List<Account> accountsToUpdate = new List<Account>();`
    - Creates a list to hold Account records for updating.
6. `for (Id accId : accountUpdates.keySet()) {`
    - Iterates through Account IDs in the map.
7. `update accountsToUpdate;`
    - Updates the Account records in a single database operation.

Word-by-Word and Line-by-Line Explanation:

Line 1: `trigger UpdateAccountOnCaseClosure on Case (after update) {`

1. `trigger`: Declares that this block of code is a trigger.
2. `UpdateAccountOnCaseClosure`: The name of the trigger. It describes what the trigger does: updating an Account when a Case is closed.
3. `on`: Specifies the object the trigger is associated with.
4. `Case`: The Salesforce object this trigger is tied to. It will execute for Case records.
5. `(after update)`: Defines the trigger event. It runs **after** a Case record is updated in the database.
6. `{`: Opens the body of the trigger.

Line 2: `Map<Id, Date> accountUpdates = new Map<Id, Date>();`

1. `Map<Id, Date>`: Declares a map data structure where:
    - `Id`: The key type. It will store Account IDs.
    - `Date`: The value type. It will store dates (the ClosedDate of Cases).
2. `accountUpdates`: The name of the map variable.
3. `=`: Assigns a value to the map.
4. `new Map<Id, Date>()`: Creates an empty map of type `<Id, Date>`.
5. `;`: Ends the statement.

Line 3: `for (Case c : Trigger.new) {`

1. `for`: Starts a loop to iterate through records.
2. `(Case c`: Declares a variable `c` of type `Case` to represent each record in the loop.
3. `:`: Separates the variable `c` from the collection being iterated over.
4. `Trigger.new`: A built-in context variable that holds the list of new Case records being processed.
5. `{`: Opens the body of the loop.

Line 4: `if (c.IsClosed && !Trigger.oldMap.get(c.Id).IsClosed) {`

1. `if`: A conditional statement to check if a condition is true.

2. `c.IsClosed`: Checks if the current Case (`c`) is closed. `IsClosed` is a field on the Case object.
3. `&&`: Logical AND operator. Combines two conditions.
4. `!`: Logical NOT operator. Negates the condition.
5. `Trigger.oldMap.get(c.Id)`: Fetches the old version of the Case record using its ID.
6. `.IsClosed`: Checks if the Case was closed before the update.
7. `{`: Opens the body of the conditional block.

Line 5: `accountUpdates.put(c.AccountId, c.ClosedDate);`

1. `accountUpdates`: Refers to the map created earlier.
2. `.put`: A method to add a key-value pair to the map.
3. `(c.AccountId`: The key being added to the map. It's the ID of the Account related to the Case.
4. `, c.ClosedDate)`: The value being added to the map. It's the ClosedDate of the Case.
5. `;`: Ends the statement.

Line 6: `}`

1. `}`: Closes the conditional block.

Line 7: `}`

1. `}`: Closes the loop.

Line 8: `List<Account> accountsToUpdate = new List<Account>();`

1. `List<Account>`: Declares a list data structure where:
    - `Account`: The type of records the list will store.
2. `accountsToUpdate`: The name of the list variable.
3. `=`: Assigns a value to the list.
4. `new List<Account>()`: Creates an empty list of type `Account`.
5. `;`: Ends the statement.

Line 9: `for (Id accId : accountUpdates.keySet()) {`

1. `for`: Starts a loop to iterate through keys in the map.
2. `(Id accId`: Declares a variable `accId` of type `Id` to represent each key in the map.
3. `:`: Separates the variable from the collection being iterated over.
4. `accountUpdates.keySet()`: Retrieves all keys (Account IDs) from the map.
5. `{`: Opens the body of the loop.

Line 10: `accountsToUpdate.add(new Account(`

1. `accountsToUpdate`: Refers to the list created earlier.
2. `.add`: A method to add an item to the list.
3. `(new Account(`: Creates a new Account record in memory and adds it to the list.

Line 11: `Id = accId,`

1. `Id`: Specifies the ID field of the Account.
2. `=`: Assigns a value to the field.
3. `accId`: The current Account ID from the loop.
4. `,`: Indicates there are more fields to set.

Line 12: `Last_Case_Closed_Date__c = accountUpdates.get(accId)`

1. `Last_Case_Closed_Date__c`: A custom field on the Account object.
2. `=`: Assigns a value to the field.
3. `accountUpdates.get(accId)`: Retrieves the ClosedDate from the map using the Account ID.
4. `)`: Closes the Account initialization.

Line 13: `));`

1. `));`: Ends the method call and the statement.

Line 14: `}`

1. `}`: Closes the loop.

Line 15: `update accountsToUpdate;`

1. `update`: A DML statement to save changes to the database.
2. `accountsToUpdate`: Refers to the list of Account records to update.
3. `;`: Ends the statement.

Line 16: `}`

1. `}`: Closes the trigger.

This document provides a comprehensive explanation of triggers, their use cases, and best practices in Salesforce Apex development.