\*\*1) One Trigger Per Object\*\*

A single Apex Trigger is all you need for one particular object. If you write multiple Triggers for a single object, you have no way of controlling the order of execution if those Triggers can run in the same contexts variable.

\*\*2) Logic Less Triggers\*\*

If you write methods in your Triggers, those can’t be exposed for the test purposes. You also can’t expose logic to be re-used anywhere else in your org.

\*\*3) Context Specific Handler Methods\*\*

Create context specific handler methods in Trigger handlers

\*\*4) Bulkify your Code\*\*

Bulkifying Apex code refers to the concept of making sure the code properly handles more than one record at a time.

\*\*5) Avoid SOQL Queries or DML statements inside FOR Loops\*\*

An individual Apex request gets a maximum of 100 SOQL queries before exceeding that governor limit.

So if this trigger is invoked by a batch of more than 100 Account records, the governor limit will throw a runtime exception error.

\*\*6) Using Collections, Streamlining Queries, and Efficient For Loops\*\*

It is important to use Apex Collections to efficiently query data and store the data in a memory. A combination of using collections and streamlining SOQL queries can substantially help writing efficient Apex code and avoid governor limits.

\*\*7) Querying Large Data Sets\*\*

The total number of records that can be returned by SOQL queries in a request is 50,000. If returning a large set of queries causes you to exceed your heap limit, then a SOQL query for loop must be used instead. It can process multiple batches of records through the use of internal calls to query and queryMore

\*\*8) Use @future Appropriately\*\*

It is critical to write your Apex code to efficiently handle bulk or many records at a time. This is also true for asynchronous Apex methods (those annotated with the @future keyword). The differences between synchronous and asynchronous Apex can be found

\*\*9) Avoid Hardcoding IDs\*\*

When deploying Apex code between sandbox and production environments, or installing Force.com AppExchange packages, it is essential to avoid hardcoding IDs in the Apex code. By doing so, if the record IDs change between environments, the logic can dynamically identify the proper data to operate against and not fail.

\*\*Few more Best Practices for Triggers\*\*

There should only be one trigger for each object.

Avoid complex logic in triggers. To simplify testing and resuse, triggers should delegate to Apex classes which contain the actual execution logic. See Mike Leach's excellent trigger template for more info.

Bulkify any "helper" classes and/or methods.

Trigers should be "bulkified" and be able to process up to 200 records for each call.

Execute DML statements using collections instead of individual records per DML statement.

Use Collections in SOQL "WHERE" clauses to retrieve all records back in single query

Use a consistent naming convention including the object name (e.g., AccountTrigger)