Salesforce cheatsheet

A comprehensive guide for experienced Salesforce developers, covering key concepts, best practices, and code examples.

Apex Essentials

Apex Syntax and Data Types

Data Types:

`Integer`, `Decimal`, `String`, `Date`, `DateTime`, `Boolean`, `ID`, `List`, `Set`, `Map`

Example:

```
Integer count = 0;
String name = 'Salesforce';
List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 10];
```

SOQL Queries:

SELECT fields FROM object WHERE conditions ORDER BY field LIMIT number

Example:

```
List<Contact> contacts = [SELECT Id, FirstName, LastName FROM Contact WHERE AccountId = :accountId];
```

DML Operations:

`insert`, `update`, `upsert`, `delete`, `undelete`

Example:

```
Account acc = new Account(Name = 'New Account');
insert acc;
```

Control Flow:

`if-else`, `for`, `while`, `do-while`

Example:

```
if (count > 0) {
    System.debug('Count is positive');
} else {
    System.debug('Count is zero or negative');
}
```

Exception Handling:

`try-catch-finally`

Example:

```
try {
    // Code that might throw an exception
} catch (DmlException e) {
    System.debug('DML error: ' + e.getMessage());
} finally {
    // Code that always executes
}
```

Apex Classes and Methods:

`public`, `private`, `protected`, `global`, `static`

Example:

```
public class MyClass {
    public static String getName() {
        return 'MyClass';
    }
}
```

Triggers

Trigger Events:

`before insert`, `after insert`, `before update`, `after update`, `before delete`, `after delete`, `after undelete`

Example:

```
trigger AccountTrigger on Account (before insert, before update)
{
    // Trigger logic here
}
```

Trigger Context Variables:

`Trigger.new`, `Trigger.old`, `Trigger.newMap`, `Trigger.oldMap`, `Trigger.isExecuting`, `Trigger.isInsert`, `Trigger.isUpdate`, `Trigger.isDelete`, `Trigger.isBefore`, `Trigger.isAfter`

Example:

```
for (Account acc : Trigger.new) {
    // Access new account values
}
```

Best Practices:

- One Trigger per Object
- Logic-less Triggers
- Use Helper Classes

Example:

```
// Trigger
trigger AccountTrigger on Account (before insert, before update)
{
    AccountTriggerHandler.run(Trigger.new, Trigger.oldMap);
}

// Handler Class
public class AccountTriggerHandler {
    public static void run(List<Account> newAccounts, Map<Id, Account> oldMap) {
        // Trigger logic here
    }
}
```

Asynchronous Apex

Future Methods

Syntax:

`@future(callout=true)` (for callouts)
`public static void methodName(parameters)`

Example:

```
@future(callout=true)
public static void makeCallout(String param) {
    // Callout logic here
}
```

Limitations:

- Maximum 50 future calls per transaction
- Cannot take sObjects or objects as parameters

Best Practices:

- Use for long-running operations
- Handle governor limits carefully

Queueable Apex

Syntax:

`public class MyQueueableClass implements Queueable`
`public void execute(QueueableContext context)`

Example:

```
public class MyQueueableClass implements Queueable {
    public void execute(QueueableContext context) {
        // Queueable logic here
    }
}

// Enqueue
System.enqueueJob(new MyQueueableClass());
```

Chaining Queueable Jobs:

Call `System.enqueueJob()` within the `execute` method.

Example:

```
public class MyQueueableClass implements Queueable {
    public void execute(QueueableContext context) {
        // Queueable logic here
        System.enqueueJob(new AnotherQueueableClass());
    }
}
```

Batch Apex

Syntax:

`public class MyBatchClass implements Database.Batchable<sObject>`
`start(Database.BatchableContext bc)`
`execute(Database.BatchableContext bc, List<sObject> scope)`
`finish(Database.BatchableContext bc)`

Example:

```
public class MyBatchClass implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, Name FROM Account');
    }

    public void execute(Database.BatchableContext bc, List<sObject> scope) {
        // Batch logic here
    }

    public void finish(Database.BatchableContext bc) {
        // Finish logic here
    }
}

// Execute
MyBatchClass batch = new MyBatchClass();
Database.executeBatch(batch, 200);
```

Best Practices:

- Use for processing large datasets
- Specify batch size carefully
- Handle exceptions and retries

Example:

```
Database.executeBatch(new MyBatchClass(), 50);
```

Schedulable Apex

Syntax:

`public class MySchedulableClass implements Schedulable`
`public void execute(SchedulableContext sc)`

Example:

```
public class MySchedulableClass implements Schedulable {
    public void execute(SchedulableContext sc) {
        // Schedulable logic here
        MyBatchClass batch = new MyBatchClass();
        Database.executeBatch(batch, 200);
    }
}

// Scheduling
String schedule = '0 0 0 15 3 ? 2023';
// Example CRON expression
String jobID = System.schedule('My Scheduled Job', schedule, new MySchedulableClass());
```

CRON Expression:

`Seconds Minutes Hours Day_of_month Month Day_of_week Year`

Best Practices:

- Use for recurring tasks
- Monitor scheduled jobs

Visualforce

Components

`<apex:page>` - The root component.
`<apex:form>` - Represents an HTML form.
`<apex:inputField>` - Creates an input field bound to a sObject field.
`<apex:outputField>` - Displays the value of a sObject field.
`<apex:commandButton>` - Creates a button that can invoke an Apex action.

Example:

```
<apex:page controller="MyController">
    <apex:form>
        <apex:inputField value="
{!account.Name}"/>
        <apex:commandButton action="
{!save}" value="Save"/>
    </apex:form>
</apex:page>
```

`<apex:pageBlock>` - Creates a block-level section.
`<apex:pageBlockSection>` - Creates a section within a page block.
`<apex:dataTable>` - Displays data in a table format.

Example:

```
<apex:pageBlock title="Account Details">
    <apex:pageBlockSection columns="2">
        <apex:outputField value="
{!account.Name}"/>
        <apex:outputField value="
{!account.Industry}"/>
    </apex:pageBlockSection>
</apex:pageBlock>
```

Controllers

Standard Controller:
Provides access to a single record.
Custom Controller:
Allows full control over the page logic.
Controller Extension:
Extends the functionality of a standard controller.

Example (Custom Controller):

```
public class MyController {
    public Account account {get; set;}

    public MyController() {
        account = [SELECT Id, Name FROM
Account WHERE Id =
:ApexPages.currentPage().getParameters()
.get('id')];
    }

    public PageReference save() {
        update account;
        return null;
    }
}
```

Data Binding:
Use `{!variable}` to bind data from the controller to the Visualforce page.

Example:

```
<apex:page controller="MyController">
    <apex:outputText value="
{!account.Name}"/>
</apex:page>
```

Best Practices

- Use standard components whenever possible.
- Optimize Visualforce pages for performance.
- Handle exceptions gracefully.
- Follow MVC pattern by keeping logic in the controller.

Lightning Web Components (LWC)

Core Concepts

Component Structure:

- HTML: Defines the structure.
- JavaScript: Handles the logic.
- CSS: Styles the component.
- XML: Configuration file.

Example:

```
myComponent/
├── myComponent.html
├── myComponent.js
├── myComponent.css
└── myComponent.js-meta.xml
```

Data Binding:
Use `{variable}` in HTML to bind data from JavaScript.

Example:

```
<template>
    <p>{message}</p>
</template>
```

```
import { LightningElement } from 'lwc';

export default class MyComponent extends LightningElement {
    message = 'Hello, LWC!';
}
```

Event Handling:
Use `onclick` or other event attributes in HTML to call JavaScript functions.

Example:

```
<template>
    <lightning-button label="Click Me" onclick={handleClick}>
</lightning-button>
</template>
```

```
import { LightningElement } from 'lwc';

export default class MyComponent extends LightningElement {
    handleClick() {
        alert('Button clicked!');
    }
}
```

Apex Integration

Import Apex Methods:
Use `@wire` or `import` to call Apex methods from LWC.

Example (@wire):

```
import { LightningElement, wire } from 'lwc';
import getAccountList from
'@salesforce/apex/AccountController.getAccountList';

export default class MyComponent extends LightningElement {
    @wire(getAccountList)
    accounts;
}
```

Imperative Apex Calls:
Call Apex methods directly from JavaScript.

Example (Imperative):

```
import { LightningElement } from 'lwc';
import getAccountList from
'@salesforce/apex/AccountController.getAccountList';

export default class MyComponent extends LightningElement {
    handleClick() {
        getAccountList()
            .then(result => {
                this.accounts = result;
            })
            .catch(error => {
                console.error(error);
            });
    }
}
```

Best Practices

- Use LWC for building modern Salesforce UIs.
- Follow best practices for component design.
- Optimize performance by minimizing DOM manipulations.
- Leverage the Salesforce Lightning Design System (SLDS).

Governor Limits

Key Governor Limits

- **SOQL Queries:** 100 synchronous, 200 asynchronous.
- **DML Statements:** 150.
- **CPU Time:** 10,000 ms synchronous, 60,000 ms asynchronous.
- **Heap Size:** 6 MB synchronous, 12 MB asynchronous.
- **Number of future calls:** 50.

- **Total number of records retrieved by SOQL queries:** 50,000.
- **Maximum timeout for callouts:** 120 seconds.
- **Maximum stack depth:** 12.

Best Practices for Avoiding Limits

- **Bulkify Code:** Process multiple records in a single transaction.
- **Efficient SOQL:** Use WHERE clauses and LIMIT to reduce record retrieval.
- **Avoid SOQL in Loops:** Move SOQL queries outside of loops.
- **Use Collections:** Use Lists, Sets, and Maps to store and manipulate data efficiently.

- **Asynchronous Processing:** Use `@future`, Queueable Apex, or Batch Apex for long-running operations.
- **Caching:** Use platform cache to store frequently accessed data.
- **Governor Limit Monitoring:** Use debug logs and System.Limits methods to monitor governor limit usage.

- **Use Database.Stateful:** When working with Batch Apex to maintain state across transactions.

Example

SOQL Inside Loop (Bad Practice):

```
List<Account> accounts = [SELECT Id FROM Account LIMIT 10];
for (Account acc : accounts) {
    List<Contact> contacts = [SELECT Id FROM Contact WHERE
AccountId = :acc.Id]; // SOQL inside loop
    // Process contacts
}
```

SOQL Outside Loop (Good Practice):

```
List<Account> accounts = [SELECT Id FROM Account LIMIT 10];
Set<Id> accountIds = new Set<Id>();
for (Account acc : accounts) {
    accountIds.add(acc.Id);
}
List<Contact> contacts = [SELECT Id FROM Contact WHERE AccountId
IN :accountIds]; // SOQL outside loop

// Process contacts
Map<Id, List<Contact>> accountContactMap = new Map<Id,
List<Contact>>();
for (Contact con : contacts) {
    if (!accountContactMap.containsKey(con.AccountId)) {
        accountContactMap.put(con.AccountId, new List<Contact>
());
    }
    accountContactMap.get(con.AccountId).add(con);
}
```