Important Salesforce Developer Interview Questions (3+ Years Experience)

1. How would you handle a scenario where you need to process 50,000 records in a batch while respecting governor limits?

Answer:

- Implement `Database.Batchable` interface with proper scope size (typically 200 records)
- Use `Database.Stateful` if you need to maintain state across batches
- Implement error handling with `Database.AllowsCallouts` if needed
- Consider using Queueable chaining for complex operations
- Example scenario:

```
//Apex
public class LargeDataProcessor implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, Name FROM Account WHERE LastModifiedDate = TODAY');
    }
    public void execute(Database.BatchableContext bc, List<Account> scope) {
        // Process 200 records at a time
        List<Account> accountsToUpdate = new List<Account>();
        for(Account acc : scope) {
            // Business logic
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
```

```
    }

    public void finish(Database.BatchableContext bc) {

        // Optional post-processing

    }

}
```

2. Explain how you would implement a recursion prevention pattern in triggers.

Answer:

- Use static boolean variables in a handler class
- Implement a trigger framework with proper execution control
- Example implementation:

Apex

```
public class TriggerControl {

    private static Boolean isAccountTriggerRunning = false;


    public static Boolean isAccountTriggerRunning() {

        return isAccountTriggerRunning;

    }

    public static void setAccountTriggerRunning(Boolean running) {

        isAccountTriggerRunning = running;

    }

}


// In trigger:
```

```
trigger AccountTrigger on Account (before update, after update) {
    if(Trigger.isBefore && Trigger.isUpdate) {
        if(!TriggerControl.isAccountTriggerRunning()) {
            TriggerControl.setAccountTriggerRunning(true);
            // Trigger logic
            TriggerControl.setAccountTriggerRunning(false);
        }
    }
}
```

3. Design a solution for a complex approval process that requires dynamic approvers based on multiple criteria (amount, region, product type).

Answer:

- Implement a custom metadata type to store approval rules (criteria and approver mappings)
- Create a trigger or process builder to evaluate records against criteria
- Use `Approval.ProcessSubmitRequest` API to programmatically submit for approval
- Consider using Platform Events for cross-object approval scenarios
- Example approach:

```
apex
// Query custom metadata for matching rules
List<Approval_Rule__mdt> rules = [SELECT Approver__c, Min_Amount__c, Max_Amount__c
                                  FROM Approval_Rule__mdt
                                  WHERE Region__c = :opp.Region__c
                                  AND Product_Type__c = :opp.Product_Type__c
                                  AND :opp.Amount >= Min_Amount__c
```

```
                AND :opp.Amount <= Max_Amount__c];

    if(!rules.isEmpty()) {
        Approval.ProcessSubmitRequest req = new Approval.ProcessSubmitRequest();
        req.setObjectId(opp.Id);
        req.setNextApproverIds(new Id[] {rules[0].Approver__c});
        Approval.ProcessResult result = Approval.process(req);
    }
```

4. How would you implement a real-time sync between Salesforce and an external ERP system for order data?

Answer:

- **Pattern 1**: Use Composite API for bulk upsert with external IDs
- **Pattern 2**: Implement Change Data Capture (CDC) with Platform Events
- **Pattern 3**: Use Salesforce Connect with OData for read/write access
- **Error Handling**: Implement retry mechanism with custom object for failed records
- **Example CDC Implementation**:

```
apex
trigger OrderTrigger on Order (after insert, after update) {
    if(Trigger.isAfter && (Trigger.isInsert || Trigger.isUpdate)) {
        List<Order_Event__e> events = new List<Order_Event__e>();
        for(Order o : Trigger.new) {
            events.add(new Order_Event__e(
                Order_Id__c = o.Id,
                Status__c = o.Status,
                Total_Amount__c = o.TotalAmount
```

```
            ));
        }
        EventBus.publish(events);
    }
}
```

5. You have a complex LWC that's loading slowly. How would you diagnose and improve performance?

Answer:

Diagnosis:

- Use Chrome DevTools to analyze network requests and rendering
- Check for N+1 query problems in @wire methods
- Identify large data payloads

Optimization Techniques:

1. Implement lazy loading for sub-components
2. Use pagination or infinite scrolling for large datasets
3. Cache data using `@wire` with `{cacheable: true}`
4. Optimize SOQL queries with selective filters
5. Use `lightning-datatable` with minimal columns

Example Optimization:

javascript

```
// Before: Loading all records
@wire(getAllAccounts) accounts;

// After: Paginated approach
@wire(getAccounts, { offset: '$offset', limit: '$pageSize' })
```

```
accounts({ error, data }) {
    if(data) {
        this.accounts = data;
    }
}
```

6. Design a solution for handling high-volume inbound API calls (10,000+ requests/minute) that need to update Salesforce records.

Answer:

- **Architecture**:
  - API Gateway to throttle and queue requests
  - Middleware layer (MuleSoft/Heroku) to buffer requests
  - Platform Events for async processing
  - Bulk API for large data volumes
- **Salesforce Implementation**:

```
apex
// Custom REST endpoint with async processing
@RestResource(urlMapping='/highVolumeUpdates/*')
global class HighVolumeAPI {
    @HttpPost
    global static void processUpdates() {
        List<UpdateRequest> requests = (List<UpdateRequest>)JSON.deserialize(
            RestContext.request.requestBody.toString(),
            List<UpdateRequest>.class
        );
```

```
        // Queue for async processing
        System.enqueueJob(new BulkUpdateProcessor(requests));
    }
}
public class BulkUpdateProcessor implements Queueable {
    List<UpdateRequest> requests;
    public BulkUpdateProcessor(List<UpdateRequest> requests) {
        this.requests = requests;
    }
    public void execute(QueueableContext ctx) {
        // Process in batches of 200
        List<Account> updates = new List<Account>();
        for(UpdateRequest req : requests) {
            updates.add(new Account(Id=req.id, Field__c=req.value));
            if(updates.size() == 200) {
                update updates;
                updates.clear();
            }
        }
        if(!updates.isEmpty()) update updates;
    }
}
```

Security & Governance

7. How would you implement field-level security in a custom LWC that displays sensitive financial data?

Answer:

- Use WITH SECURITY_ENFORCED in SOQL
- Check FieldLevelSecurity using SObjectAccessDecision

Example:

Apex

```
public with sharing class FinancialDataController {
    @AuraEnabled(cacheable=true)
    public static List<Account> getFinancialData(Id accountId) {
        SObjectAccessDecision securityDecision = Security.stripInaccessible(
            AccessType.READABLE,
            [SELECT Id, Name, AnnualRevenue, Financial_Score__c
             FROM Account
             WHERE Id = :accountId]
        );
        return (List<Account>)securityDecision.getRecords();
    }
}
```

- Implement conditional rendering based on user permissions

- Use lightning-record-view-form for automatic FLS checks
- Example:

  html

  <template if:true={hasFinancialAccess}>

  <lightning-formatted-number value={annualRevenue}></lightning-formatted-number>

  </template>

8. A production org is experiencing CPU time limit exceptions during month-end processing. How would you investigate and resolve this?

Answer:

Investigation Steps:

1. Analyze debug logs with Developer Console or Workbench
2. Identify long-running transactions with LIMIT_USAGE_FOR_NS headers
3. Check for nested loops or inefficient SOQL in triggers
4. Review batch job execution times

Resolution Strategies:

1. Optimize SOQL queries with selective filters
2. Implement batch processing for large data volumes
3. Use Map<Id, List<Child>> pattern for parent-child relationships
4. Cache frequently accessed data in static variables
5. Consider using Platform Events to break up transactions

Example Optimization:

apex

// Before: Nested queries in loop

for(Account acc : accounts) {

```
    List<Contact> contacts = [SELECT Id FROM Contact WHERE AccountId = :acc.Id];
    // Process contacts
}
// After: Bulk query with map
Map<Id, List<Contact>> accountContactsMap = new Map<Id, List<Contact>>();
for(Contact con : [SELECT Id, AccountId FROM Contact WHERE AccountId IN :accounts]) {
    if(!accountContactsMap.containsKey(con.AccountId)) {
        accountContactsMap.put(con.AccountId, new List<Contact>());
    }
    accountContactsMap.get(con.AccountId).add(con);
}
```

---

9. Design a solution for maintaining historical data changes (audit trail) of critical fields on an Account object, where business requires:

- Tracking 10+ fields including standard and custom fields
- Storing old value, new value, who changed it, and when
- Ability to report on changes over time
- Minimal performance impact during high-volume updates

Answer:

Approach:

1. **Custom Metadata Configuration:**
   - Create Custom Metadata Type to configure which fields to track

- Allows admins to add/remove fields without code changes

2. **Trigger Implementation:**

- Use a single after-update trigger on Account
- Implement efficient field comparison using SObject.getPopulatedFieldsAsMap()

3. **Bulkified Pattern:**

- Collect all changes in a list and insert once (outside loops)
- Use static variables to prevent recursion

4. **Reporting:**

- Create a custom report type for the audit object
- Consider a Lightning Component for visual timeline

Sample Code:

apex

```
// Custom Metadata query to get tracked fields
List<Field_Audit_Setting__mdt> trackedFields = [
    SELECT Field_API_Name__c FROM Field_Audit_Setting__mdt
    WHERE Object__c = 'Account' AND Is_Active__c = true
];

// In trigger handler
List<Field_Audit__c> auditsToCreate = new List<Field_Audit__c>();
Map<String, Schema.SObjectField> fieldMap =
Schema.SObjectType.Account.fields.getMap();

for(Account newAcc : Trigger.new) {
    Account oldAcc = Trigger.oldMap.get(newAcc.Id);
```

```
    for(Field_Audit_Setting__mdt fieldSetting : trackedFields) {
        String fieldName = fieldSetting.Field_API_Name__c;
        if(fieldMap.containsKey(fieldName) &&
           newAcc.get(fieldName) != oldAcc.get(fieldName)) {
            auditsToCreate.add(new Field_Audit__c(
                Account__c = newAcc.Id,
                Field_Name__c = fieldName,
                Old_Value__c = String.valueOf(oldAcc.get(fieldName)),
                New_Value__c = String.valueOf(newAcc.get(fieldName)),
                Changed_By__c = UserInfo.getUserId(),
                Change_Date__c = System.now()
            ));
        }
    }
}
// Bulk insert outside loops
if(!auditsToCreate.isEmpty()) {
    insert auditsToCreate;
}
```

Performance Considerations:

- Use `@future` or Queueable if dealing with >10,000 records

- Consider a nightly batch to consolidate records if volume is extremely high
- Implement a retention policy to archive old records
