Complete SOQL and SOSL Guide – Beginner to Advanced

Comprehensive Interview Questions with Practical Query Examples

This guide covers SOQL (Salesforce Object Query Language) and SOSL (Salesforce Object Search Language) from basics to advanced concepts with real-world query examples.

Table of Contents

SOQL (Salesforce Object Query Language)

SOQL Basics

Question 1: What is SOQL?

Answer: SOQL (Salesforce Object Query Language) is a query language used to retrieve records from the Salesforce database. Similar to SQL, but designed specifically for Salesforce.

Key Characteristics:

- Case-insensitive
- Returns typed Lists of sObjects
- Cannot use SELECT *
- Maximum 50,000 records per query
- Cannot perform joins (uses relationships)

Basic Query Examples:

```
// Basic SELECT query
List<Account> accounts = [SELECT Id, Name FROM Account];

// Query with all required fields
List<Contact> contacts = [SELECT Id, FirstName, LastName, Email FROM Cont

// Query custom object
List<Custom_Object__c> customRecords = [SELECT Id, Name, Custom_Field__c

// Query specific record by Id
Account acc = [SELECT Id, Name, Industry FROM Account WHERE Id = '001xx0(

// Execute in Developer Console
System.debug('Accounts: ' + accounts);
System.debug('Total Accounts: ' + accounts.size());
```

Question 2: What are the required clauses in SOQL?

Answer: Every SOQL query must have:

1. **SELECT** - Fields to retrieve
2. **FROM** - Object to query

Optional clauses:

- WHERE - Filter conditions
- ORDER BY - Sort results
- LIMIT - Restrict number of records
- OFFSET - Skip records for pagination
- GROUP BY - Group results
- HAVING - Filter grouped results

Query Examples:

```
// Minimum required - SELECT and FROM
List<Account> allAccounts = [SELECT Id, Name FROM Account];

// With WHERE clause
List<Account> techAccounts = [SELECT Id, Name FROM Account WHERE Industry

// With ORDER BY
List<Contact> sortedContacts = [SELECT FirstName, LastName FROM Contact O

// With LIMIT
List<Opportunity> topOpps = [SELECT Name, Amount FROM Opportunity LIMIT 1

// All clauses together
List<Account> filteredAccounts = [
    SELECT Id, Name, Industry, AnnualRevenue
    FROM Account
    WHERE Industry = 'Technology'
    AND AnnualRevenue > 1000000
    ORDER BY AnnualRevenue DESC
    LIMIT 50
];
```

Question 3: How do you find field API names for SOQL queries?

Answer: Methods to find API names:

1. **Setup → Object Manager** → Select object → Fields & Relationships
2. **Schema Builder** - Visual tool
3. **Developer Console** - Query Editor (Ctrl+Space for autocomplete)
4. **Describe methods** in Apex

Query Examples:

```
// Standard fields - same as label
[SELECT Id, Name, Phone, Website FROM Account];

// Standard fields with different API names
[SELECT Id, NumberOfEmployees, AnnualRevenue FROM Account];

// Custom fields - end with __c
[SELECT Id, Name, Custom_Field__c, Another_Custom_Field__c FROM Custom_Ob

// Relationship fields - use relationship name, not field name
[SELECT Id, Name, Account.Name FROM Contact]; // Account, not AccountId

// Owner field
[SELECT Id, Name, Owner.Name, Owner.Email FROM Account];

// Using describe in Apex to get field names
Map<String, Schema.SObjectField> fieldMap = Account.sObjectType.getDescri
for(String fieldName : fieldMap.keySet()) {
    System.debug('Field API Name: ' + fieldName);
}
```

Filtering Data

Question 4: How do you filter SOQL queries using WHERE clause?

Answer: The WHERE clause filters records based on conditions. You can use comparison operators and logical operators to create complex filters.

Query Examples:

```
// Single condition - exact match
List<Account> accounts = [SELECT Id, Name FROM Account WHERE Name = 'Acme

// Single condition - number
List<Opportunity> bigDeals = [SELECT Id, Name, Amount FROM Opportunity WH

// Single condition - boolean
List<Contact> validEmails = [SELECT Id, Name FROM Contact WHERE HasOptedO

// Single condition - null check
List<Account> noIndustry = [SELECT Id, Name FROM Account WHERE Industry =
```

```
// Single condition - not null
List<Account> withIndustry = [SELECT Id, Name FROM Account WHERE Industry

// Case insensitive
List<Account> acmeAccounts = [SELECT Id, Name FROM Account WHERE Name = '

// Date field
List<Opportunity> recentOpps = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CloseDate = 2024-01-15
];

// DateTime field (format: YYYY-MM-DDTHH:MM:SSZ)
List<Account> todayAccounts = [
    SELECT Id, Name, CreatedDate
    FROM Account
    WHERE CreatedDate >= 2024-01-01T00:00:00Z
];
```

Question 5: What are comparison operators in SOQL?

Answer: Comparison operators compare values:

- `=` Equal to
- `!=` or `<>` Not equal to
- `<` Less than
- `<=` Less than or equal to
- `>` Greater than
- `>=` Greater than or equal to
- `LIKE` Pattern matching
- `IN` Match any value in list
- `NOT IN` Exclude values in list
- `INCLUDES` Multi-select picklist (at least one)
- `EXCLUDES` Multi-select picklist (none)

Query Examples:

```
// Equal to
[SELECT Id, Name FROM Account WHERE Industry = 'Technology'];
```

```
// Not equal to
[SELECT Id, Name FROM Account WHERE Industry != 'Banking'];
[SELECT Id, Name FROM Account WHERE Industry <> 'Banking']; // Same as ab

// Greater than
[SELECT Id, Name, AnnualRevenue FROM Account WHERE AnnualRevenue > 50000(

// Less than or equal
[SELECT Id, Name, NumberOfEmployees FROM Account WHERE NumberOfEmployees

// Greater than or equal
[SELECT Id, Amount FROM Opportunity WHERE Amount >= 50000];

// LIKE - Pattern matching
[SELECT Id, Name FROM Account WHERE Name LIKE 'A%']; // Starts with A
[SELECT Id, Name FROM Account WHERE Name LIKE '%Corp']; // Ends with Corp
[SELECT Id, Name FROM Account WHERE Name LIKE '%Tech%']; // Contains Tech
[SELECT Id, Email FROM Contact WHERE Email LIKE '%@gmail.com']; // Gmail

// IN - Match any value
[SELECT Id, Name FROM Account WHERE Industry IN ('Technology', 'Finance',
[SELECT Id, Name FROM Contact WHERE LeadSource IN ('Web', 'Phone', 'Email

// NOT IN - Exclude values
[SELECT Id, Name FROM Account WHERE Industry NOT IN ('Banking', 'Insuranc

// INCLUDES - Multi-select picklist (has at least one)
[SELECT Id, Name FROM Account WHERE Categories__c INCLUDES ('Partner', 'C

// EXCLUDES - Multi-select picklist (has none)
[SELECT Id, Name FROM Account WHERE Categories__c EXCLUDES ('Competitor',

// Combining with variables in Apex
String searchIndustry = 'Technology';
List<Account> accounts = [SELECT Id, Name FROM Account WHERE Industry = :

Set<String> industries = new Set<String>{'Technology', 'Finance'};
List<Account> multiAccounts = [SELECT Id, Name FROM Account WHERE Industr
```

Operators

Question 6: How do you use logical operators (AND, OR, NOT) in SOQL?

Answer: Logical operators combine multiple conditions:

- **AND** - All conditions must be true
- **OR** - At least one condition must be true
- **NOT** - Negates a condition

Query Examples:

```
// AND - Both conditions must be true
List<Account> techWithRevenue = [
    SELECT Id, Name, Industry, AnnualRevenue
    FROM Account
    WHERE Industry = 'Technology'
    AND AnnualRevenue > 1000000
];

// Multiple AND conditions
List<Opportunity> qualifiedOpps = [
    SELECT Id, Name, Amount, Probability
    FROM Opportunity
    WHERE Amount > 50000
    AND Probability > 75
    AND StageName = 'Negotiation'
];

// OR - At least one condition must be true
List<Account> selectedIndustries = [
    SELECT Id, Name, Industry
    FROM Account
    WHERE Industry = 'Technology'
    OR Industry = 'Finance'
];

// Multiple OR conditions
List<Contact> multiLeadSource = [
    SELECT Id, FirstName, LastName, LeadSource
    FROM Contact
    WHERE LeadSource = 'Web'
    OR LeadSource = 'Email'
    OR LeadSource = 'Phone'
];
```

```
// Combining AND & OR (use parentheses for clarity)
List<Account> complexFilter = [
    SELECT Id, Name, Industry, AnnualRevenue
    FROM Account
    WHERE (Industry = 'Technology' OR Industry = 'Finance')
    AND AnnualRevenue > 5000000
];

// More complex combinations
List<Opportunity> advancedFilter = [
    SELECT Id, Name, Amount, StageName, Probability
    FROM Opportunity
    WHERE (StageName = 'Prospecting' OR StageName = 'Qualification')
    AND Amount > 100000
    AND (Probability > 50 OR Amount > 500000)
];

// NOT operator
List<Account> notBanking = [
    SELECT Id, Name, Industry
    FROM Account
    WHERE NOT (Industry = 'Banking')
];

// NOT with other operators
List<Contact> notFromWeb = [
    SELECT Id, FirstName, LastName
    FROM Contact
    WHERE NOT (LeadSource = 'Web' OR LeadSource = 'Email')
];

// Complex real-world example
List<Opportunity> targetOpportunities = [
    SELECT Id, Name, Amount, CloseDate, StageName, AccountId
    FROM Opportunity
    WHERE (StageName = 'Prospecting' OR StageName = 'Qualification')
    AND Amount > 25000
    AND CloseDate >= TODAY
    AND CloseDate <= NEXT_90_DAYS
    AND Account.Industry IN ('Technology', 'Finance', 'Healthcare')
];
```

Sorting and Limiting

Question 7: How do you sort query results in SOQL?

Answer: Use ORDER BY clause to sort results. Can sort by multiple fields and specify ascending (ASC) or descending (DESC) order.

Query Examples:

```
// Sort by one field - ascending (default)
List<Account> accountsAsc = [
    SELECT Id, Name
    FROM Account
    ORDER BY Name
];

// Sort by one field - descending
List<Account> accountsDesc = [
    SELECT Id, Name, AnnualRevenue
    FROM Account
    ORDER BY AnnualRevenue DESC
];

// Sort by multiple fields
List<Contact> sortedContacts = [
    SELECT Id, FirstName, LastName, Email
    FROM Contact
    ORDER BY LastName ASC, FirstName ASC
];

// Sort with NULLS FIRST or NULLS LAST
List<Account> nullsFirst = [
    SELECT Id, Name, Industry
    FROM Account
    ORDER BY Industry ASC NULLS FIRST
];

List<Account> nullsLast = [
    SELECT Id, Name, AnnualRevenue
    FROM Account
    ORDER BY AnnualRevenue DESC NULLS LAST
];
```

```
// Sort by date
List<Opportunity> recentOpps = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    ORDER BY CloseDate DESC
];

// Sort by DateTime
List<Account> recentlyCreated = [
    SELECT Id, Name, CreatedDate
    FROM Account
    ORDER BY CreatedDate DESC
];

// Sort by related field
List<Contact> contactsByAccount = [
    SELECT Id, FirstName, LastName, Account.Name
    FROM Contact
    ORDER BY Account.Name ASC
];

// Complex sorting
List<Opportunity> complexSort = [
    SELECT Id, Name, Amount, CloseDate, StageName
    FROM Opportunity
    ORDER BY StageName ASC, Amount DESC, CloseDate ASC
];
```

Question 8: How do you limit and paginate query results?

Answer:

- **LIMIT** - Restricts number of records returned
- **OFFSET** - Skips specified number of records (pagination)

Query Examples:

```
// LIMIT - Get top 10 records
List<Account> top10 = [
    SELECT Id, Name
    FROM Account
    LIMIT 10
];
```

```
// LIMIT with ORDER BY - Top 5 opportunities by amount
List<Opportunity> top5Deals = [
    SELECT Id, Name, Amount
    FROM Opportunity
    ORDER BY Amount DESC
    LIMIT 5
];

// LIMIT with WHERE
List<Account> techTop10 = [
    SELECT Id, Name, AnnualRevenue
    FROM Account
    WHERE Industry = 'Technology'
    ORDER BY AnnualRevenue DESC
    LIMIT 10
];

// OFFSET - Skip first 10 records
List<Account> skipFirst10 = [
    SELECT Id, Name
    FROM Account
    ORDER BY Name
    LIMIT 10
    OFFSET 10
];

// Pagination - Page 1 (records 1-10)
List<Account> page1 = [
    SELECT Id, Name
    FROM Account
    ORDER BY Name
    LIMIT 10
    OFFSET 0
];

// Pagination - Page 2 (records 11-20)
List<Account> page2 = [
    SELECT Id, Name
    FROM Account
    ORDER BY Name
    LIMIT 10
    OFFSET 10
];
```

```
// Pagination - Page 3 (records 21-30)
List<Account> page3 = [
    SELECT Id, Name
    FROM Account
    ORDER BY Name
    LIMIT 10
    OFFSET 20
];

// Dynamic pagination
public class PaginationHelper {
    public static List<Account> getAccountsPage(Integer pageNumber, Integ
        Integer offset = (pageNumber - 1) * pageSize;
        return [
            SELECT Id, Name, Industry
            FROM Account
            ORDER BY Name
            LIMIT :pageSize
            OFFSET :offset
        ];
    }
}

// Usage
List<Account> firstPage = PaginationHelper.getAccountsPage(1, 20); // Rec
List<Account> secondPage = PaginationHelper.getAccountsPage(2, 20); // Re

// Maximum OFFSET is 2000
// OFFSET 2001 will throw error

// Best practice for large datasets - use WHERE with last record Id
List<Account> accounts = [
    SELECT Id, Name
    FROM Account
    WHERE Id > :lastRecordId
    ORDER BY Id
    LIMIT 100
];
```

Date Operations

Question 9: What are Date Literals in SOQL?

Answer: Date literals are predefined keywords representing relative date ranges. They make queries dynamic without calculating exact dates.

Common Date Literals:

- **TODAY** - Current date
- **YESTERDAY** - Previous day
- **TOMORROW** - Next day
- **LAST_WEEK** - Previous calendar week
- **THIS_WEEK** - Current calendar week
- **NEXT_WEEK** - Next calendar week
- **LAST_MONTH** - Previous calendar month
- **THIS_MONTH** - Current calendar month
- **NEXT_MONTH** - Next calendar month
- **LAST_90_DAYS** - Previous 90 days (includes today)
- **NEXT_90_DAYS** - Next 90 days (excludes today)
- **THIS_YEAR** - Current calendar year
- **LAST_YEAR** - Previous calendar year
- **NEXT_YEAR** - Next calendar year

Query Examples:

```
// TODAY - Records created today
List<Account> todayAccounts = [
    SELECT Id, Name, CreatedDate
    FROM Account
    WHERE CreatedDate = TODAY
];

// YESTERDAY
List<Opportunity> yesterdayOpps = [
    SELECT Id, Name, CreatedDate
    FROM Opportunity
    WHERE CreatedDate = YESTERDAY
];

// TOMORROW
List<Task> tomorrowTasks = [
    SELECT Id, Subject, ActivityDate
    FROM Task
    WHERE ActivityDate = TOMORROW
```

```
];

// LAST_WEEK
List<Contact> lastWeekContacts = [
    SELECT Id, FirstName, LastName, CreatedDate
    FROM Contact
    WHERE CreatedDate = LAST_WEEK
];

// THIS_WEEK
List<Lead> thisWeekLeads = [
    SELECT Id, Name, CreatedDate
    FROM Lead
    WHERE CreatedDate = THIS_WEEK
];

// NEXT_WEEK
List<Event> nextWeekEvents = [
    SELECT Id, Subject, ActivityDate
    FROM Event
    WHERE ActivityDate = NEXT_WEEK
];

// LAST_MONTH
List<Account> lastMonthAccounts = [
    SELECT Id, Name, CreatedDate
    FROM Account
    WHERE CreatedDate = LAST_MONTH
];

// THIS_MONTH
List<Opportunity> thisMonthOpps = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CloseDate = THIS_MONTH
];

// LAST_90_DAYS (includes today)
List<Account> recent90Days = [
    SELECT Id, Name, LastModifiedDate
    FROM Account
    WHERE LastModifiedDate = LAST_90_DAYS
];
```

```
// NEXT_90_DAYS (excludes today)
List<Opportunity> next90Days = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CloseDate = NEXT_90_DAYS
];

// THIS_YEAR
List<Account> thisYearAccounts = [
    SELECT Id, Name, CreatedDate
    FROM Account
    WHERE CreatedDate = THIS_YEAR
];

// LAST_YEAR
List<Opportunity> lastYearOpps = [
    SELECT Id, Name, Amount, CloseDate
    FROM Opportunity
    WHERE CloseDate = LAST_YEAR
];

// Comparison operators with date literals
// Greater than TODAY
List<Opportunity> futureOpps = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CloseDate > TODAY
];

// Less than or equal to THIS_MONTH
List<Task> pastTasks = [
    SELECT Id, Subject, ActivityDate
    FROM Task
    WHERE ActivityDate <= THIS_MONTH
];

// Between dates using literals
List<Opportunity> rangeOpps = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CloseDate >= TODAY
    AND CloseDate <= NEXT_90_DAYS
];
```

Question 10: What are LAST_N_DAYS and NEXT_N_DAYS literals?

Answer: These literals allow you to specify a custom number of days for date ranges.

Syntax:

- `LAST_N_DAYS:n` - Last n days (includes today)
- `NEXT_N_DAYS:n` - Next n days (excludes today)
- `LAST_N_WEEKS:n` - Last n weeks
- `NEXT_N_WEEKS:n` - Next n weeks
- `LAST_N_MONTHS:n` - Last n months
- `NEXT_N_MONTHS:n` - Next n months
- `LAST_N_YEARS:n` - Last n years
- `NEXT_N_YEARS:n` - Next n years

Query Examples:

```
// LAST_N_DAYS - Last 7 days (includes today)
List<Account> last7Days = [
    SELECT Id, Name, CreatedDate
    FROM Account
    WHERE CreatedDate = LAST_N_DAYS:7
];

// LAST_N_DAYS - Last 30 days
List<Opportunity> last30Days = [
    SELECT Id, Name, CreatedDate
    FROM Opportunity
    WHERE CreatedDate = LAST_N_DAYS:30
];

// NEXT_N_DAYS - Next 15 days (excludes today)
List<Task> next15Days = [
    SELECT Id, Subject, ActivityDate
    FROM Task
    WHERE ActivityDate = NEXT_N_DAYS:15
];

// NEXT_N_DAYS - Next 60 days
List<Opportunity> next60Days = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CloseDate = NEXT_N_DAYS:60
```

```
];

// LAST_N_WEEKS - Last 4 weeks
List<Contact> last4Weeks = [
    SELECT Id, FirstName, LastName, CreatedDate
    FROM Contact
    WHERE CreatedDate = LAST_N_WEEKS:4
];

// NEXT_N_WEEKS - Next 2 weeks
List<Event> next2Weeks = [
    SELECT Id, Subject, ActivityDateTime
    FROM Event
    WHERE ActivityDateTime = NEXT_N_WEEKS:2
];

// LAST_N_MONTHS - Last 6 months
List<Account> last6Months = [
    SELECT Id, Name, CreatedDate
    FROM Account
    WHERE CreatedDate = LAST_N_MONTHS:6
];

// NEXT_N_MONTHS - Next 3 months
List<Opportunity> next3Months = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CloseDate = NEXT_N_MONTHS:3
];

// LAST_N_YEARS - Last 2 years
List<Account> last2Years = [
    SELECT Id, Name, Industry, CreatedDate
    FROM Account
    WHERE CreatedDate = LAST_N_YEARS:2
];

// Using with comparison operators
List<Opportunity> hotOpps = [
    SELECT Id, Name, CloseDate, Amount
    FROM Opportunity
    WHERE CloseDate <= NEXT_N_DAYS:30
    AND Amount > 100000
];
```

```
// Dynamic date literals with variables
Integer daysRange = 45;
String dateQuery = 'SELECT Id, Name FROM Account WHERE CreatedDate = LAST
List<Account> dynamicAccounts = Database.query(dateQuery);
```

Question 11: What are Date Functions in SOQL?

Answer: Date functions allow you to group or filter data by date periods. Useful in GROUP BY clauses for aggregation.

Available Date Functions:

- `CALENDAR_MONTH()` - Month number (1-12)
- `CALENDAR_QUARTER()` - Quarter (1-4)
- `CALENDAR_YEAR()` - Calendar year
- `DAY_IN_MONTH()` - Day of month (1-31)
- `DAY_IN_WEEK()` - Day of week (1-7, Sunday=1)
- `DAY_IN_YEAR()` - Day of year (1-366)
- `DAY_ONLY()` - Date part of DateTime
- `FISCAL_MONTH()` - Fiscal month
- `FISCAL_QUARTER()` - Fiscal quarter
- `FISCAL_YEAR()` - Fiscal year
- `HOUR_IN_DAY()` - Hour (0-23)
- `WEEK_IN_MONTH()` - Week in month
- `WEEK_IN_YEAR()` - Week in year (1-53)

Query Examples:

```
// CALENDAR_YEAR - Filter by year
List<Opportunity> opps2024 = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CALENDAR_YEAR(CloseDate) = 2024
];

// CALENDAR_MONTH - Filter by month
List<Account> januaryAccounts = [
    SELECT Id, Name, CreatedDate
    FROM Account
    WHERE CALENDAR_MONTH(CreatedDate) = 1
```

```
];

// CALENDAR_QUARTER - Filter by quarter
List<Opportunity> q1Opps = [
    SELECT Id, Name, CloseDate
    FROM Opportunity
    WHERE CALENDAR_QUARTER(CloseDate) = 1
];

// DAY_IN_MONTH - Filter by day
List<Contact> contactsOn15th = [
    SELECT Id, FirstName, LastName, CreatedDate
    FROM Contact
    WHERE DAY_IN_MONTH(CreatedDate) = 15
];

// DAY_IN_WEEK - Filter by day of week (1=Sunday, 7=Saturday)
List<Task> mondayTasks = [
    SELECT Id, Subject, ActivityDate
    FROM Task
    WHERE DAY_IN_WEEK(ActivityDate) = 2
];

// DAY_ONLY - Convert DateTime to Date
List<Event> todayEvents = [
    SELECT Id, Subject, ActivityDateTime
    FROM Event
    WHERE DAY_ONLY(ActivityDateTime) = TODAY
];

// HOUR_IN_DAY - Filter by hour
List<Case> morningCases = [
    SELECT Id, Subject, CreatedDate
    FROM Case
    WHERE HOUR_IN_DAY(CreatedDate) < 12
];

// GROUP BY with date functions - Count by year
AggregateResult[] yearlyAccounts = [
    SELECT CALENDAR_YEAR(CreatedDate) year, COUNT(Id) total
    FROM Account
    GROUP BY CALENDAR_YEAR(CreatedDate)
];
```

```
for(AggregateResult ar : yearlyAccounts) {
    System.debug('Year: ' + ar.get('year') + ', Total: ' + ar.get('total'
}

// GROUP BY with month - Revenue by month
AggregateResult[] monthlyRevenue = [
    SELECT CALENDAR_MONTH(CloseDate) month, SUM(Amount) totalAmount
    FROM Opportunity
    WHERE CALENDAR_YEAR(CloseDate) = 2024
    GROUP BY CALENDAR_MONTH(CloseDate)
    ORDER BY CALENDAR_MONTH(CloseDate)
];

// GROUP BY with quarter
AggregateResult[] quarterlyStats = [
    SELECT CALENDAR_QUARTER(CloseDate) quarter, COUNT(Id) totalDeals, SUM
    FROM Opportunity
    WHERE CALENDAR_YEAR(CloseDate) = 2024
    GROUP BY CALENDAR_QUARTER(CloseDate)
];

// Multiple date functions
AggregateResult[] yearMonthStats = [
    SELECT CALENDAR_YEAR(CreatedDate) year,
           CALENDAR_MONTH(CreatedDate) month,
           COUNT(Id) total
    FROM Account
    GROUP BY CALENDAR_YEAR(CreatedDate), CALENDAR_MONTH(CreatedDate)
    ORDER BY CALENDAR_YEAR(CreatedDate), CALENDAR_MONTH(CreatedDate)
];

// Convert timezone with CONVERT_TIMEZONE
List<Event> convertedEvents = [
    SELECT Id, Subject,
           CONVERTCURRENCY(Amount__c),
           CONVERTTIMEZONE(ActivityDateTime) localTime
    FROM Event
];
```

Relationship Queries

Question 12: What are Relationship Queries in SOQL?

Answer: Relationship queries retrieve data from related objects in a single query. Two types:

1. **Child-to-Parent** (Lookup/Master-Detail relationships)
2. **Parent-to-Child** (Subqueries)

Child-to-Parent Characteristics:

- Use dot notation
- Can traverse up to 5 levels
- Access parent fields directly

Parent-to-Child Characteristics:

- Use subqueries
- Can only go 1 level deep
- Returns List

Question 13: How do you write Child-to-Parent queries?

Answer: Use dot notation with relationship name (not field name) to access parent object fields.

Query Examples:

```
// Basic child-to-parent - Contact to Account
List<Contact> contacts = [
    SELECT Id, FirstName, LastName, Account.Name
    FROM Contact
];

for(Contact con : contacts) {
    System.debug('Contact: ' + con.FirstName + ', Account: ' + con.Accour
}

// Multiple parent fields
List<Contact> contactsWithAccount = [
    SELECT Id, FirstName, LastName,
           Account.Name, Account.Industry, Account.Phone, Account.Website
    FROM Contact
];

// Filter using parent field
List<Contact> techContacts = [
    SELECT Id, FirstName, LastName, Email, Account.Name
    FROM Contact
```

```
    WHERE Account.Industry = 'Technology'
];

// Sort by parent field
List<Contact> sortedByAccount = [
    SELECT Id, FirstName, LastName, Account.Name
    FROM Contact
    ORDER BY Account.Name ASC
];

// Two levels - Contact → Account → Owner
List<Contact> contactsWithOwner = [
    SELECT Id, FirstName, LastName,
           Account.Name, Account.Owner.Name, Account.Owner.Email
    FROM Contact
];

// Three levels - Contact → Account → Owner → Manager
List<Contact> multiLevel = [
    SELECT Id, FirstName, LastName,
           Account.Owner.Manager.Name
    FROM Contact
    WHERE Account.Owner.Manager.Name != null
];

// Custom object relationship - CustomChild__c to CustomParent__c
List<CustomChild__c> children = [
    SELECT Id, Name,
           CustomParent__r.Name,
           CustomParent__r.CustomField__c
    FROM CustomChild__c
];

// Opportunity to Account relationship
List<Opportunity> oppsWithAccount = [
    SELECT Id, Name, Amount,
           Account.Name, Account.Industry, Account.AnnualRevenue
    FROM Opportunity
    WHERE Account.AnnualRevenue > 1000000
];

// Case to Account and Contact
List<Case> casesWithRelated = [
    SELECT Id, Subject, Status,
```

```
            Account.Name, Contact.FirstName, Contact.LastName
    FROM Case
];

// Task to related objects (polymorphic - What field)
List<Task> tasks = [
    SELECT Id, Subject,
           What.Name,
           Who.Name
    FROM Task
];

// Using Owner relationship
List<Account> accountsWithOwner = [
    SELECT Id, Name,
           Owner.Name, Owner.Email, Owner.Phone, Owner.Title
    FROM Account
];

// CreatedBy and LastModifiedBy
List<Account> auditInfo = [
    SELECT Id, Name,
           CreatedBy.Name, CreatedDate,
           LastModifiedBy.Name, LastModifiedDate
    FROM Account
];

// RecordType relationship
List<Account> accountsWithRecordType = [
    SELECT Id, Name,
           RecordType.Name, RecordType.DeveloperName
    FROM Account
    WHERE RecordType.DeveloperName = 'Customer'
];
```

Question 14: How do you write Parent-to-Child queries?

Answer: Use subqueries in SELECT clause to retrieve child records. Child relationship name ends with 's' for standard objects or '__r' for custom.

Query Examples:

```
// Basic parent-to-child - Account to Contacts
List<Account> accountsWithContacts = [
    SELECT Id, Name,
           (SELECT Id, FirstName, LastName, Email FROM Contacts)
    FROM Account
];

for(Account acc : accountsWithContacts) {
    System.debug('Account: ' + acc.Name);
    for(Contact con : acc.Contacts) {
        System.debug('  Contact: ' + con.FirstName + ' ' + con.LastName);
    }
}

// Account to Opportunities
List<Account> accountsWithOpps = [
    SELECT Id, Name,
           (SELECT Id, Name, Amount, StageName, CloseDate FROM Opportunit
    FROM Account
];

// With filter in subquery
List<Account> accountsWithOpenOpps = [
    SELECT Id, Name,
           (SELECT Id, Name, Amount FROM Opportunities WHERE StageName !=
    FROM Account
];

// With ORDER BY in subquery
List<Account> accountsWithSortedOpps = [
    SELECT Id, Name,
           (SELECT Id, Name, Amount FROM Opportunities ORDER BY Amount DE
    FROM Account
];

// With LIMIT in subquery
List<Account> accountsWithTop5Opps = [
    SELECT Id, Name,
           (SELECT Id, Name, Amount FROM Opportunities ORDER BY Amount DE
    FROM Account
];

// Multiple child relationships
List<Account> multipleChildren = [
```

```
    SELECT Id, Name,
           (SELECT Id, FirstName, LastName FROM Contacts),
           (SELECT Id, Name, Amount FROM Opportunities),
           (SELECT Id, Subject FROM Cases)
    FROM Account
];

// Custom object parent-to-child
List<CustomParent__c> parentsWithChildren = [
    SELECT Id, Name,
           (SELECT Id, Name, CustomField__c FROM CustomChildren__r)
    FROM CustomParent__c
];

// Filter parent and child
List<Account> filteredBoth = [
    SELECT Id, Name,
           (SELECT Id, Name, Amount FROM Opportunities WHERE Amount > 500
    FROM Account
    WHERE Industry = 'Technology'
];

// Check if child records exist
List<Account> accounts = [
    SELECT Id, Name,
           (SELECT Id FROM Contacts)
    FROM Account
];

for(Account acc : accounts) {
    if(acc.Contacts.size() > 0) {
        System.debug('Account has contacts: ' + acc.Name);
    } else {
        System.debug('Account has no contacts: ' + acc.Name);
    }
}

// User to created records
List<User> usersWithAccounts = [
    SELECT Id, Name,
           (SELECT Id, Name FROM CreatedAccounts)
    FROM User
    WHERE Id = :UserInfo.getUserId()
];
```

```
// Campaign to Campaign Members
List<Campaign> campaignsWithMembers = [
    SELECT Id, Name,
           (SELECT Id, ContactId, Status FROM CampaignMembers)
    FROM Campaign
];

// Processing child records
List<Account> accountsWithContacts2 = [
    SELECT Id, Name,
           (SELECT Id, FirstName, LastName, Email FROM Contacts)
    FROM Account
    LIMIT 10
];

for(Account acc : accountsWithContacts2) {
    Integer contactCount = acc.Contacts.size();
    System.debug('Account ' + acc.Name + ' has ' + contactCount + ' conta

    if(contactCount > 0) {
        for(Contact con : acc.Contacts) {
            System.debug('  - ' + con.FirstName + ' ' + con.LastName);
        }
    }
}
```

Question 15: What are Polymorphic Relationships in SOQL?

Answer: Polymorphic relationships are fields that can reference multiple object types. Examples:

- **Owner** (User or Group)
- **Who** (Lead or Contact)
- **What** (Account, Opportunity, Campaign, etc.)

Use **TYPEOF** clause to query different fields based on actual object type.

Query Examples:

```
// Basic polymorphic query - Who field (Lead or Contact)
List<Task> tasks = [
    SELECT Id, Subject, Who.Name, Who.Email
    FROM Task
```

```
];

// TYPEOF clause - different fields for different types
List<Task> tasksWithTypeof = [
    SELECT Id, Subject,
           TYPEOF Who
               WHEN Contact THEN FirstName, LastName, Account.Name
               WHEN Lead THEN Company, Status
           END
    FROM Task
];

// Processing TYPEOF results
for(Task t : tasksWithTypeof) {
    if(t.Who instanceof Contact) {
        Contact con = (Contact)t.Who;
        System.debug('Contact: ' + con.FirstName + ' ' + con.LastName);
    } else if(t.Who instanceof Lead) {
        Lead l = (Lead)t.Who;
        System.debug('Lead: ' + l.Company);
    }
}

// What field (Account, Opportunity, etc.)
List<Task> tasksWithWhat = [
    SELECT Id, Subject,
           TYPEOF What
               WHEN Account THEN Name, Industry
               WHEN Opportunity THEN Name, Amount, StageName
               WHEN Case THEN Subject, Status
           END
    FROM Task
];

// Owner field (User or Group)
List<Account> accountsWithOwner = [
    SELECT Id, Name,
           TYPEOF Owner
               WHEN User THEN FirstName, LastName, Email, Phone
               WHEN Group THEN Name, Email
           END
    FROM Account
];
```

```
// Filter by polymorphic type
List<Event> userEvents = [
    SELECT Id, Subject, Owner.Name
    FROM Event
    WHERE Owner.Type = 'User'
];

// Event Owner (Calendar or User)
List<Event> events = [
    SELECT Id, Subject,
           TYPEOF Owner
               WHEN User THEN FirstName, Email
               WHEN Calendar THEN Name, Type
           END
    FROM Event
];

// Using instanceof in Apex
List<Task> allTasks = [SELECT Id, Subject, Who.Name, What.Name FROM Task]

for(Task t : allTasks) {
    // Check Who relationship
    if(t.Who instanceof Contact) {
        System.debug('Related to Contact');
    } else if(t.Who instanceof Lead) {
        System.debug('Related to Lead');
    }

    // Check What relationship
    if(t.What instanceof Account) {
        System.debug('Related to Account');
    } else if(t.What instanceof Opportunity) {
        System.debug('Related to Opportunity');
    }
}

// Custom object polymorphic (if configured)
List<CustomObject__c> custom = [
    SELECT Id, Name,
           TYPEOF PolymorphicField__c
               WHEN ObjectType1__c THEN Field1__c
               WHEN ObjectType2__c THEN Field2__c
           END
    FROM CustomObject__c
```

```
];

// Filter and TYPEOF together
List<Task> filteredTasks = [
    SELECT Id, Subject,
           TYPEOF Who
               WHEN Contact THEN FirstName, LastName
               WHEN Lead THEN Company
           END
    FROM Task
    WHERE Who.Type = 'Contact'
];
```

Aggregate Functions

Question 16: What are Aggregate Functions in SOQL?

Answer: Aggregate functions perform calculations on groups of records and return a single value.

Available Functions:

- **COUNT()** - Count records
- **COUNT(fieldName)** - Count non-null values
- **COUNT_DISTINCT(fieldName)** - Count unique values
- **SUM(fieldName)** - Sum numeric values
- **AVG(fieldName)** - Average of numeric values
- **MIN(fieldName)** - Minimum value
- **MAX(fieldName)** - Maximum value

Query Examples:

```
// COUNT() - Total records
AggregateResult[] result = [SELECT COUNT() total FROM Account];
Integer totalAccounts = (Integer)result[0].get('total');
System.debug('Total Accounts: ' + totalAccounts);

// COUNT(field) - Count non-null values
AggregateResult[] phoneCount = [SELECT COUNT(Phone) total FROM Account];
Integer accountsWithPhone = (Integer)phoneCount[0].get('total');

// COUNT_DISTINCT - Unique values
AggregateResult[] uniqueIndustries = [SELECT COUNT_DISTINCT(Industry) tot
```

```
Integer industryCount = (Integer)uniqueIndustries[0].get('total');

// SUM - Total amount
AggregateResult[] sumResult = [SELECT SUM(Amount) totalAmount FROM Opport
Decimal totalRevenue = (Decimal)sumResult[0].get('totalAmount');
System.debug('Total Revenue: ' + totalRevenue);

// AVG - Average value
AggregateResult[] avgResult = [SELECT AVG(Amount) avgAmount FROM Opportur
Decimal averageAmount = (Decimal)avgResult[0].get('avgAmount');

// MIN - Minimum value
AggregateResult[] minResult = [SELECT MIN(Amount) minAmount FROM Opportur
Decimal minimumAmount = (Decimal)minResult[0].get('minAmount');

// MAX - Maximum value
AggregateResult[] maxResult = [SELECT MAX(Amount) maxAmount FROM Opportur
Decimal maximumAmount = (Decimal)maxResult[0].get('maxAmount');

// Multiple aggregate functions
AggregateResult[] stats = [
    SELECT COUNT() recordCount,
           SUM(Amount) totalAmount,
           AVG(Amount) avgAmount,
           MIN(Amount) minAmount,
           MAX(Amount) maxAmount
    FROM Opportunity
];

System.debug('Count: ' + stats[0].get('recordCount'));
System.debug('Total: ' + stats[0].get('totalAmount'));
System.debug('Average: ' + stats[0].get('avgAmount'));
System.debug('Min: ' + stats[0].get('minAmount'));
System.debug('Max: ' + stats[0].get('maxAmount'));

// With WHERE clause
AggregateResult[] techStats = [
    SELECT COUNT() total, SUM(AnnualRevenue) totalRevenue
    FROM Account
    WHERE Industry = 'Technology'
];
```

Question 17: How do you use GROUP BY in SOQL?

Answer: GROUP BY groups records by one or more fields. Must be used with aggregate functions.

Query Examples:

```
// GROUP BY single field - Count by Industry
AggregateResult[] industryCount = [
    SELECT Industry, COUNT(Id) total
    FROM Account
    GROUP BY Industry
];

for(AggregateResult ar : industryCount) {
    System.debug('Industry: ' + ar.get('Industry') + ', Count: ' + ar.get
}

// GROUP BY with SUM
AggregateResult[] revenueByIndustry = [
    SELECT Industry, SUM(AnnualRevenue) totalRevenue
    FROM Account
    GROUP BY Industry
];

// GROUP BY with multiple aggregates
AggregateResult[] industryStats = [
    SELECT Industry,
           COUNT(Id) total,
           SUM(AnnualRevenue) totalRevenue,
           AVG(AnnualRevenue) avgRevenue
    FROM Account
    GROUP BY Industry
];

// GROUP BY multiple fields
AggregateResult[] multiGroup = [
    SELECT Industry, Rating, COUNT(Id) total
    FROM Account
    GROUP BY Industry, Rating
];

// GROUP BY with ORDER BY
AggregateResult[] sortedGroup = [
    SELECT Industry, COUNT(Id) total
    FROM Account
    GROUP BY Industry
```

```
    ORDER BY COUNT(Id) DESC
];

// GROUP BY with LIMIT
AggregateResult[] top5Industries = [
    SELECT Industry, COUNT(Id) total
    FROM Account
    GROUP BY Industry
    ORDER BY COUNT(Id) DESC
    LIMIT 5
];

// GROUP BY with WHERE
AggregateResult[] filteredGroup = [
    SELECT Industry, SUM(AnnualRevenue) totalRevenue
    FROM Account
    WHERE AnnualRevenue > 100000
    GROUP BY Industry
];

// GROUP BY with relationship field
AggregateResult[] oppsByAccount = [
    SELECT Account.Name, COUNT(Id) oppCount, SUM(Amount) totalAmount
    FROM Opportunity
    GROUP BY Account.Name
];

// GROUP BY with date functions
AggregateResult[] yearlyStats = [
    SELECT CALENDAR_YEAR(CreatedDate) year, COUNT(Id) total
    FROM Account
    GROUP BY CALENDAR_YEAR(CreatedDate)
    ORDER BY CALENDAR_YEAR(CreatedDate)
];

// GROUP BY month and year
AggregateResult[] monthlyRevenue = [
    SELECT CALENDAR_YEAR(CloseDate) year,
           CALENDAR_MONTH(CloseDate) month,
           SUM(Amount) revenue
    FROM Opportunity
    WHERE CloseDate = THIS_YEAR
    GROUP BY CALENDAR_YEAR(CloseDate), CALENDAR_MONTH(CloseDate)
    ORDER BY CALENDAR_YEAR(CloseDate), CALENDAR_MONTH(CloseDate)
```

```
];

// Real-world example: Sales report
AggregateResult[] salesReport = [
    SELECT Owner.Name ownerName,
           COUNT(Id) totalDeals,
           SUM(Amount) totalRevenue,
           AVG(Amount) avgDealSize
    FROM Opportunity
    WHERE StageName = 'Closed Won'
    AND CloseDate = THIS_YEAR
    GROUP BY Owner.Name
    ORDER BY SUM(Amount) DESC
];

for(AggregateResult ar : salesReport) {
    System.debug('Owner: ' + ar.get('ownerName'));
    System.debug('Deals: ' + ar.get('totalDeals'));
    System.debug('Revenue: $' + ar.get('totalRevenue'));
    System.debug('Avg Deal: $' + ar.get('avgDealSize'));
    System.debug('---');
}
```

Question 18: How do you use HAVING clause in SOQL?

Answer: HAVING filters groups after GROUP BY, similar to WHERE but works on aggregate results.

Query Examples:

```
// HAVING with COUNT
AggregateResult[] popularIndustries = [
    SELECT Industry, COUNT(Id) total
    FROM Account
    GROUP BY Industry
    HAVING COUNT(Id) > 10
];

// HAVING with SUM
AggregateResult[] highRevenueIndustries = [
    SELECT Industry, SUM(AnnualRevenue) totalRevenue
    FROM Account
    GROUP BY Industry
    HAVING SUM(AnnualRevenue) > 10000000
```

```
];

// HAVING with AVG
AggregateResult[] highAvgIndustries = [
    SELECT Industry, AVG(AnnualRevenue) avgRevenue
    FROM Account
    GROUP BY Industry
    HAVING AVG(AnnualRevenue) > 500000
];

// HAVING with MAX
AggregateResult[] groups = [
    SELECT Industry, MAX(NumberOfEmployees) maxEmployees
    FROM Account
    GROUP BY Industry
    HAVING MAX(NumberOfEmployees) > 1000
];

// Multiple conditions in HAVING
AggregateResult[] complexHaving = [
    SELECT Industry, COUNT(Id) total, SUM(AnnualRevenue) totalRevenue
    FROM Account
    GROUP BY Industry
    HAVING COUNT(Id) > 5
    AND SUM(AnnualRevenue) > 1000000
];

// WHERE and HAVING together
AggregateResult[] filteredGrouped = [
    SELECT Industry, COUNT(Id) total
    FROM Account
    WHERE CreatedDate = THIS_YEAR
    GROUP BY Industry
    HAVING COUNT(Id) > 10
];

// HAVING with ORDER BY
AggregateResult[] sortedHaving = [
    SELECT Owner.Name, COUNT(Id) totalOpps
    FROM Opportunity
    GROUP BY Owner.Name
    HAVING COUNT(Id) > 5
    ORDER BY COUNT(Id) DESC
];
```

```
// Real-world example: Find top performers
AggregateResult[] topSalespeople = [
    SELECT Owner.Name ownerName,
           COUNT(Id) totalDeals,
           SUM(Amount) totalRevenue
    FROM Opportunity
    WHERE StageName = 'Closed Won'
    AND CloseDate = THIS_YEAR
    GROUP BY Owner.Name
    HAVING SUM(Amount) > 500000
    ORDER BY SUM(Amount) DESC
];

// Example: Accounts with many opportunities
AggregateResult[] activeAccounts = [
    SELECT Account.Name, COUNT(Id) oppCount
    FROM Opportunity
    GROUP BY Account.Name
    HAVING COUNT(Id) >= 3
];
```

Advanced SOQL

Question 19: What are SOQL Keywords for scope and security?

Answer: Special keywords control query behavior and security:

- **WITH SECURITY_ENFORCED** - Respect field-level security
- **FOR VIEW** - Track recent viewed records
- **FOR REFERENCE** - Mark records as referenced
- **FOR UPDATE** - Lock records for update
- **USING SCOPE** - Control data visibility

Query Examples:

```
// WITH SECURITY_ENFORCED - Respect FLS
try {
    List<Account> accounts = [
        SELECT Id, Name, Industry, AnnualRevenue
        FROM Account
        WITH SECURITY_ENFORCED
```

```
    ];
} catch(System.QueryException e) {
    System.debug('Access denied to some fields');
}

// FOR VIEW - Track as recently viewed
List<Account> viewed = [
    SELECT Id, Name
    FROM Account
    LIMIT 10
    FOR VIEW
];

// FOR REFERENCE - Mark as referenced (for reports)
List<Contact> referenced = [
    SELECT Id, Name
    FROM Contact
    LIMIT 10
    FOR REFERENCE
];

// FOR UPDATE - Lock records (prevents concurrent updates)
List<Account> locked = [
    SELECT Id, Name, AnnualRevenue
    FROM Account
    WHERE Id IN :accountIds
    FOR UPDATE
];

// Update locked records
for(Account acc : locked) {
    acc.AnnualRevenue = 1000000;
}
update locked; // No concurrency issues

// USING SCOPE - Filter by division
List<Account> mineScope = [
    SELECT Id, Name
    FROM Account
    USING SCOPE Mine
];

// USING SCOPE options:
// - Mine (owned by user)
```

```
// - Team (owned by user's team)
// - Everything (all records)
// - Delegated (delegated to user)
// - MineAndMyGroups (user and groups)

List<Opportunity> teamOpps = [
    SELECT Id, Name
    FROM Opportunity
    USING SCOPE Team
];

// FIELDS() function - Query standard/custom/all fields
List<Account> withStandard = [
    SELECT FIELDS(STANDARD)
    FROM Account
    LIMIT 10
];

List<Account> withCustom = [
    SELECT FIELDS(CUSTOM)
    FROM Account
    LIMIT 10
];

List<Account> withAll = [
    SELECT FIELDS(ALL)
    FROM Account
    LIMIT 10
];

// Note: FIELDS(ALL) can hit query limits
```

SOQL in Apex

Question 20: How do you use SOQL in Apex?

Answer: SOQL can be used inline (static) or dynamically built as strings.

Examples:

```
// Static SOQL - Inline query
public class StaticSOQL {
```

```
    public void queryAccounts() {
        List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 10];

        for(Account acc : accounts) {
            System.debug('Account: ' + acc.Name);
        }
    }

    // Query single record
    public Account getAccount(Id accountId) {
        Account acc = [SELECT Id, Name, Industry FROM Account WHERE Id =
        return acc;
    }

    // Query with variable
    public List<Contact> getContactsByAccount(Id accountId) {
        return [SELECT Id, FirstName, LastName FROM Contact WHERE Account
    }
}

// Using bind variables
String searchIndustry = 'Technology';
List<Account> accounts = [SELECT Id, Name FROM Account WHERE Industry = :

Set<Id> accountIds = new Set<Id>{'001...', '001...'};
List<Account> specificAccounts = [SELECT Id, Name FROM Account WHERE Id I

// For loop with SOQL
for(Account acc : [SELECT Id, Name FROM Account]) {
    System.debug(acc.Name);
}

// Query all or one pattern
try {
    Account acc = [SELECT Id, Name FROM Account WHERE Name = 'Acme'];
} catch(QueryException e) {
    System.debug('No account found or multiple found');
}

// Check if exists
List<Account> accounts = [SELECT Id FROM Account WHERE Name = 'Test' LIMI
if(!accounts.isEmpty()) {
    System.debug('Account exists');
}
```

```
// Count records
Integer count = [SELECT COUNT() FROM Account WHERE Industry = 'Technology
System.debug('Tech accounts: ' + count);
```

Question 21: What is Dynamic SOQL?

Answer: Dynamic SOQL builds queries as strings at runtime using Database.query(). Useful for dynamic field selection or conditions.

Examples:

```
public class DynamicSOQL {
    // Basic dynamic query
    public static List<SObject> queryDynamic(String objectName, String fi
        String query = 'SELECT Id, ' + fieldName + ' FROM ' + objectName;
        return Database.query(query);
    }

    // Usage
    List<SObject> accounts = DynamicSOQL.queryDynamic('Account', 'Name');

    // Dynamic with WHERE clause
    public static List<Account> searchAccounts(String fieldName, String v
        String query = 'SELECT Id, Name FROM Account WHERE ' + fieldName
        return Database.query(query);
    }

    // Dynamic with multiple fields
    public static List<SObject> queryMultipleFields(String objectName, Li
        String fieldList = String.join(fields, ', ');
        String query = 'SELECT ' + fieldList + ' FROM ' + objectName;
        return Database.query(query);
    }

    // Usage
    List<String> fields = new List<String>{'Id', 'Name', 'Industry', 'Phc
    List<SObject> results = queryMultipleFields('Account', fields);

    // Dynamic ORDER BY
    public static List<Account> getAccountsSorted(String sortField, Strir
        String query = 'SELECT Id, Name, Industry FROM Account ORDER BY '
        return (List<Account>)Database.query(query);
```

```
    }

    // Usage
    List<Account> sorted = getAccountsSorted('Name', 'ASC');

    // Complex dynamic query
    public static List<SObject> advancedDynamicQuery(
        String objectName,
        List<String> fields,
        String whereClause,
        String orderBy,
        Integer limitCount
    ) {
        String query = 'SELECT ' + String.join(fields, ', ') + ' FROM ' +

        if(String.isNotBlank(whereClause)) {
            query += ' WHERE ' + whereClause;
        }

        if(String.isNotBlank(orderBy)) {
            query += ' ORDER BY ' + orderBy;
        }

        if(limitCount != null && limitCount > 0) {
            query += ' LIMIT ' + limitCount;
        }

        System.debug('Generated Query: ' + query);
        return Database.query(query);
    }

    // Dynamic with date literals
    public static List<Opportunity> getOpportunitiesByDateRange(String da
        String query = 'SELECT Id, Name, CloseDate FROM Opportunity WHERE
        return Database.query(query);
    }

    // Usage
    List<Opportunity> thisWeek = getOpportunitiesByDateRange('THIS_WEEK')
    List<Opportunity> lastMonth = getOpportunitiesByDateRange('LAST_MONTH
}

// SOQL Injection Prevention - Use String.escapeSingleQuotes()
public static List<Account> safeDynamicQuery(String userInput) {
```

```
    String safeInput = String.escapeSingleQuotes(userInput);
    String query = 'SELECT Id, Name FROM Account WHERE Name = \'' + safeI
    return Database.query(query);
}

// Better: Use bind variables when possible
public static List<Account> saferQuery(String name) {
    String query = 'SELECT Id, Name FROM Account WHERE Name = :name';
    return Database.query(query);
}
```

Question 22: What are SOQL For Loops?

Answer: SOQL For Loops query and process records in batches (200 records at a time), reducing heap size usage.

Examples:

```
// Standard for loop - All records in memory
List<Account> accounts = [SELECT Id, Name FROM Account];
for(Account acc : accounts) {
    // Process account
}

// SOQL For Loop - Batches of 200
for(Account acc : [SELECT Id, Name FROM Account]) {
    // Process account in batches
    System.debug(acc.Name);
}

// SOQL For Loop with List
for(List<Account> accountBatch : [SELECT Id, Name FROM Account]) {
    // Each iteration has up to 200 records
    System.debug('Processing batch of ' + accountBatch.size() + ' account

    for(Account acc : accountBatch) {
        // Process each account
        acc.Description = 'Processed';
    }

    update accountBatch;
}
```

```
// Processing large datasets
public class LargeDataProcessor {
    public static void processAllAccounts() {
        Integer totalProcessed = 0;

        for(List<Account> accounts : [SELECT Id, Name, Industry FROM Acco
            // Process batch
            for(Account acc : accounts) {
                if(acc.Industry == null) {
                    acc.Industry = 'Other';
                }
                totalProcessed++;
            }

            update accounts;
        }

        System.debug('Total processed: ' + totalProcessed);
    }
}

// With WHERE clause
for(Contact con : [SELECT Id, Email FROM Contact WHERE Email != null]) {
    // Send email
    System.debug('Email: ' + con.Email);
}

// With related records
for(Account acc : [SELECT Id, Name, (SELECT Id, Name FROM Contacts) FROM
    System.debug('Account: ' + acc.Name);
    for(Contact con : acc.Contacts) {
        System.debug('  Contact: ' + con.Name);
    }
}

// Best practice: Use SOQL For Loop for large datasets
public static void updateAllOpportunities() {
    for(List<Opportunity> opps : [SELECT Id, Description FROM Opportunity
        for(Opportunity opp : opps) {
            opp.Description = 'Updated on ' + System.today();
        }
        update opps;
    }
}
```

SOSL (Salesforce Object Search Language)

SOSL Basics

Question 23: What is SOSL?

Answer: SOSL (Salesforce Object Search Language) performs text-based searches across multiple objects simultaneously. Uses search index for fast results.

SOSL vs SOQL:

Feature	SOQL	SOSL
Purpose	Query specific records	Text search across objects
Objects	One at a time	Multiple simultaneously
Search	Exact match	Text/fuzzy search
Performance	Slower for text	Faster (uses index)
Return	List	List<List>
Syntax	SELECT ... FROM	FIND ... RETURNING

Basic SOSL Examples:

```
// Basic SOSL query
List<List<SObject>> searchResults = [
    FIND 'Acme'
    IN ALL FIELDS
    RETURNING Account(Id, Name), Contact(Id, Name)
];

// Extract results
List<Account> accounts = (List<Account>)searchResults[0];
List<Contact> contacts = (List<Contact>)searchResults[1];

System.debug('Found ' + accounts.size() + ' accounts');
System.debug('Found ' + contacts.size() + ' contacts');

// Search specific term
List<List<SObject>> results = [
```

```
    FIND 'John Smith'
    IN ALL FIELDS
    RETURNING Contact(Id, FirstName, LastName, Email)
];

// Search in NAME fields only
List<List<SObject>> nameResults = [
    FIND 'Technology'
    IN NAME FIELDS
    RETURNING Account(Id, Name), Lead(Id, Name)
];

// Search in EMAIL fields
List<List<SObject>> emailResults = [
    FIND 'example.com'
    IN EMAIL FIELDS
    RETURNING Contact(Id, Email), Lead(Id, Email)
];

// Search in PHONE fields
List<List<SObject>> phoneResults = [
    FIND '555-1234'
    IN PHONE FIELDS
    RETURNING Account(Id, Phone), Contact(Id, Phone)
];

// Process SOSL results
List<List<SObject>> searchList = [
    FIND 'Acme'
    IN ALL FIELDS
    RETURNING Account(Id, Name), Contact(Id, FirstName, LastName)
];

for(List<SObject> objects : searchList) {
    for(SObject obj : objects) {
        if(obj instanceof Account) {
            Account acc = (Account)obj;
            System.debug('Account: ' + acc.Name);
        } else if(obj instanceof Contact) {
            Contact con = (Contact)obj;
            System.debug('Contact: ' + con.FirstName + ' ' + con.LastName
        }
    }
}
```

SOSL vs SOQL

Question 24: When should you use SOSL instead of SOQL?

Answer: Use SOSL when:

- Searching across multiple objects
- Don't know which object contains data
- Need fuzzy/partial text matching
- Searching name, email, or phone fields
- Need faster text search performance

Use SOQL when:

- Know exact object and fields
- Need exact matches
- Need complex filtering
- Require specific ordering
- Working with relationships

Examples:

```
// SOQL - When you know the object and field
List<Contact> contacts = [
    SELECT Id, FirstName, LastName
    FROM Contact
    WHERE FirstName = 'John' AND LastName = 'Smith'
];

// SOSL - When searching across multiple objects
List<List<SObject>> results = [
    FIND 'John Smith'
    IN ALL FIELDS
    RETURNING Contact(Id, Name), Lead(Id, Name), Account(Id, Name)
];

// SOQL - Specific filtering
List<Opportunity> opps = [
    SELECT Id, Name, Amount
    FROM Opportunity
    WHERE Amount > 100000
    AND StageName = 'Closed Won'
```

```
    ORDER BY Amount DESC
];

// SOSL - Broad search
List<List<SObject>> searchOpps = [
    FIND 'enterprise'
    IN ALL FIELDS
    RETURNING Opportunity(Id, Name, Amount)
];

// SOQL - Count and aggregates
AggregateResult[] stats = [
    SELECT COUNT(), SUM(Amount)
    FROM Opportunity
    WHERE CloseDate = THIS_YEAR
];

// SOSL - Cannot do aggregates (use SOQL after)
List<List<SObject>> foundOpps = [FIND 'Q1' RETURNING Opportunity(Id, Amou
```

Advanced SOSL

Question 25: What are SOSL search groups and clauses?

Answer: SOSL clauses control where and how to search:

IN Clauses (Search Groups):

- **ALL FIELDS** - All text fields (default)
- **NAME FIELDS** - Name fields only
- **EMAIL FIELDS** - Email fields only
- **PHONE FIELDS** - Phone fields only
- **SIDEBAR FIELDS** - Fields in sidebar

Other Clauses:

- **RETURNING** - Objects and fields to return
- **WHERE** - Filter results
- **ORDER BY** - Sort results
- **LIMIT** - Limit results
- **OFFSET** - Pagination
- **WITH** - Additional options

Examples:

```
// ALL FIELDS (default)
List<List<SObject>> allFields = [
    FIND 'Acme'
    IN ALL FIELDS
    RETURNING Account(Id, Name), Contact(Id, Name)
];

// NAME FIELDS only
List<List<SObject>> nameOnly = [
    FIND 'John'
    IN NAME FIELDS
    RETURNING Contact(Id, FirstName, LastName), Lead(Id, Name)
];

// EMAIL FIELDS only
List<List<SObject>> emails = [
    FIND 'gmail.com'
    IN EMAIL FIELDS
    RETURNING Contact(Id, Email), Lead(Id, Email)
];

// PHONE FIELDS only
List<List<SObject>> phones = [
    FIND '555'
    IN PHONE FIELDS
    RETURNING Account(Id, Phone), Contact(Id, Phone)
];

// RETURNING with specific fields
List<List<SObject>> specific = [
    FIND 'Technology'
    RETURNING Account(Id, Name, Industry, Phone),
              Contact(Id, FirstName, LastName, Email, Account.Name)
];

// WITH WHERE clause
List<List<SObject>> filtered = [
    FIND 'Acme'
    IN ALL FIELDS
    RETURNING Account(Id, Name WHERE Industry = 'Technology')
];
```

```
// WITH ORDER BY
List<List<SObject>> sorted = [
    FIND 'Sales'
    RETURNING Opportunity(Id, Name, Amount ORDER BY Amount DESC)
];

// WITH LIMIT
List<List<SObject>> limited = [
    FIND 'John'
    RETURNING Contact(Id, Name LIMIT 10)
];

// Multiple objects with different limits
List<List<SObject>> multiLimit = [
    FIND 'Tech'
    RETURNING Account(Id, Name LIMIT 5),
              Contact(Id, Name LIMIT 10),
              Lead(Id, Name LIMIT 3)
];

// WITH OFFSET for pagination
List<List<SObject>> page1 = [
    FIND 'Acme'
    RETURNING Account(Id, Name ORDER BY Name LIMIT 10 OFFSET 0)
];

List<List<SObject>> page2 = [
    FIND 'Acme'
    RETURNING Account(Id, Name ORDER BY Name LIMIT 10 OFFSET 10)
];

// Complex SOSL with all clauses
List<List<SObject>> complex = [
    FIND 'Enterprise'
    IN ALL FIELDS
    RETURNING Account(Id, Name, Industry, AnnualRevenue
                      WHERE Industry = 'Technology'
                      AND AnnualRevenue > 1000000
                      ORDER BY AnnualRevenue DESC
                      LIMIT 20),
              Opportunity(Id, Name, Amount
                          WHERE StageName = 'Closed Won'
                          ORDER BY Amount DESC
                          LIMIT 10)
```

```
];

// WITH DIVISION (if using divisions)
List<List<SObject>> divisions = [
    FIND 'Acme'
    IN ALL FIELDS
    RETURNING Account(Id, Name)
    WITH DIVISION = 'Global'
];

// WITH DATA CATEGORY (for articles)
List<List<SObject>> articles = [
    FIND 'Installation'
    RETURNING KnowledgeArticleVersion(Id, Title)
    WITH DATA CATEGORY Product__c AT Laptop__c
];

// WITH NETWORK (for Communities)
List<List<SObject>> community = [
    FIND 'Support'
    RETURNING Case(Id, Subject)
    WITH NETWORK = '12345'
];
```

Question 26: How do you use wildcards in SOSL?

Answer: SOSL supports wildcards for pattern matching:

- ***** - Zero or more characters
- **?** - Exactly one character

Examples:

```
// * wildcard - multiple characters
List<List<SObject>> startwith = [
    FIND 'John*'
    IN ALL FIELDS
    RETURNING Contact(Id, FirstName, LastName)
];
// Matches: John, Johnny, Johnson, etc.

// * at end
List<List<SObject>> endwith = [
```

```
    FIND '*Corp'
    IN ALL FIELDS
    RETURNING Account(Id, Name)
];
// Matches: Acme Corp, Tech Corp, etc.

// * in middle
List<List<SObject>> contains = [
    FIND '*tech*'
    IN ALL FIELDS
    RETURNING Account(Id, Name), Opportunity(Id, Name)
];
// Matches: Technology, Fintech, Techno, etc.

// ? wildcard - single character
List<List<SObject>> single = [
    FIND 'sm?th'
    IN ALL FIELDS
    RETURNING Contact(Id, LastName)
];
// Matches: smith, smyth, etc.

// Multiple ? wildcards
List<List<SObject>> multiple = [
    FIND 'a??e'
    IN ALL FIELDS
    RETURNING Account(Id, Name)
];
// Matches: acme, able, anne, etc.

// Combine * and ?
List<List<SObject>> combined = [
    FIND 'jo?n*'
    IN ALL FIELDS
    RETURNING Contact(Id, FirstName)
];
// Matches: john, johnny, joan, etc.

// Email domain search
List<List<SObject>> emailDomain = [
    FIND '*@example.com'
    IN EMAIL FIELDS
    RETURNING Contact(Id, Email), Lead(Id, Email)
];
```

```
// Phone pattern
List<List<SObject>> phonePattern = [
    FIND '555*'
    IN PHONE FIELDS
    RETURNING Contact(Id, Phone)
];

// Escape special characters with backslash
List<List<SObject>> escaped = [
    FIND 'test\\*value'
    IN ALL FIELDS
    RETURNING Account(Id, Name)
];
// Searches for literal "test*value"
```

Question 27: How do you use SOSL in Apex?

Answer: SOSL can be inline or dynamic, returns List<List>.

Examples:

```
public class SOSLHelper {
    // Basic SOSL in Apex
    public static void searchRecords(String searchTerm) {
        List<List<SObject>> results = [
            FIND :searchTerm
            IN ALL FIELDS
            RETURNING Account(Id, Name), Contact(Id, Name)
        ];

        List<Account> accounts = (List<Account>)results[0];
        List<Contact> contacts = (List<Contact>)results[1];

        System.debug('Found ' + accounts.size() + ' accounts');
        System.debug('Found ' + contacts.size() + ' contacts');
    }

    // Process results
    public static Map<String, List<SObject>> searchAndGroup(String term)
        Map<String, List<SObject>> resultMap = new Map<String, List<SObje

        List<List<SObject>> searchResults = [
```

```
            FIND :term
            IN ALL FIELDS
            RETURNING Account(Id, Name),
                      Contact(Id, FirstName, LastName),
                      Lead(Id, Name, Company)
        ];

        resultMap.put('Accounts', searchResults[0]);
        resultMap.put('Contacts', searchResults[1]);
        resultMap.put('Leads', searchResults[2]);

        return resultMap;
    }

    // Dynamic SOSL
    public static List<List<SObject>> dynamicSOSL(String searchTerm, Stri
        String soslQuery = 'FIND :searchTerm IN ALL FIELDS RETURNING ' +
        return Search.query(soslQuery);
    }

    // Advanced search
    public static List<Account> searchAccounts(String term) {
        List<List<SObject>> results = [
            FIND :term
            IN ALL FIELDS
            RETURNING Account(Id, Name, Industry, Phone
                              WHERE Industry != null
                              ORDER BY Name
                              LIMIT 50)
        ];

        return (List<Account>)results[0];
    }
}

// Usage examples
SOSLHelper.searchRecords('Acme');
Map<String, List<SObject>> grouped = SOSLHelper.searchAndGroup('Technolog
List<Account> accounts = SOSLHelper.searchAccounts('Enterprise');

// In triggers (not recommended - use batch/async)
trigger AccountTrigger on Account (after insert) {
    List<List<SObject>> duplicates = [
        FIND :Trigger.new[0].Name
```

```
        IN ALL FIELDS
        RETURNING Account(Id, Name)
    ];
}
```

Question 28: What are SOSL limits and best practices?

Answer:

Limits:

- Maximum 2,000 records returned across all objects
- Maximum 20 SOSL queries per transaction
- Search string must be at least 2 characters
- Maximum search string length: 10,000 characters

Best Practices:

1. Use specific search groups (EMAIL, PHONE, NAME)
2. Use WHERE clause to filter
3. Use LIMIT to control results
4. Avoid SOSL in loops
5. Use for broad searches, SOQL for specific queries

Examples:

```
// BAD - SOSL in loop
for(Account acc : accounts) {
    List<List<SObject>> results = [FIND :acc.Name RETURNING Contact(Id)];
}

// GOOD - Single SOSL
Set<String> searchTerms = new Set<String>();
for(Account acc : accounts) {
    searchTerms.add(acc.Name);
}
String searchString = String.join(new List<String>(searchTerms), ' OR ');
List<List<SObject>> results = [FIND :searchString RETURNING Contact(Id, N

// BAD - No LIMIT
List<List<SObject>> unlimited = [
    FIND 'a*'
    RETURNING Account(Id, Name)
```

```
];  // Could return thousands

// GOOD - With LIMIT
List<List<SObject>> limited = [
    FIND 'a*'
    RETURNING Account(Id, Name LIMIT 100)
];

// BAD - All fields, all objects
List<List<SObject>> broad = [
    FIND 'test'
    IN ALL FIELDS
    RETURNING Account, Contact, Lead, Opportunity, Case
];

// GOOD - Specific fields and objects
List<List<SObject>> specific = [
    FIND 'test'
    IN NAME FIELDS
    RETURNING Account(Id, Name WHERE Industry = 'Technology' LIMIT 50)
];

// Check limits
System.debug('SOSL Queries: ' + Limits.getSoslQueries() + '/' + Limits.ge

// Prevent SOSL injection
public static List<List<SObject>> safeSearch(String userInput) {
    String safeInput = String.escapeSingleQuotes(userInput);
    String query = 'FIND \'' + safeInput + '\' RETURNING Account(Id, Name
    return Search.query(query);
}
```

Summary

Key Takeaways

SOQL:

✅ Query specific records from database
✅ Use SELECT, FROM, WHERE, ORDER BY, LIMIT
✅ Leverage relationship queries (child-to-parent, parent-to-child)
✅ Use aggregate functions (COUNT, SUM, AVG, etc.)

✅ Apply date literals for dynamic date filtering
✅ GROUP BY for summarized data
✅ Use bind variables in Apex
✅ SOQL For Loops for large datasets

SOSL:

✅ Text search across multiple objects
✅ Use FIND...RETURNING syntax
✅ Search in specific field groups (NAME, EMAIL, PHONE)
✅ Support wildcards (* and ?)
✅ Faster for text searches
✅ Returns List<List>
✅ Use for broad searches

Best Practices:

1. Bulkify SOQL queries (avoid queries in loops)
2. Use selective WHERE clauses
3. Query only necessary fields
4. Use LIMIT to control data volume
5. Leverage relationship queries instead of multiple queries
6. Use SOQL For Loops for large datasets
7. Use SOSL for text searches across objects
8. Monitor query limits using Limits class

Document Information:

- Topics Covered: 28 comprehensive questions
- Query Examples: 300+ practical SOQL and SOSL queries
- Coverage: Beginner to Advanced level
- Includes: Filters, Relationships, Aggregates, Date Operations, Dynamic queries

Good luck with your Salesforce SOQL/SOSL journey! 🚀