1. Data Dictionary

* Table Creation

DATA: BEGIN OF itab OCCURS 0,

field1 TYPE i,

field2 TYPE string,

END OF itab.

CREATE TABLE zemployee ( field1 TYPE i, field2 TYPE string, PRIMARY KEY field1 );

In this example, a table named **zemployee** is created with two fields (**field1** of type integer and **field2** of type string). The primary key is set on **field1**.

* Indexes

Indexes are used to optimize the performance of database queries. You can create indexes on one or more fields of a table. Here's an example:

* CREATE INDEX idx\_employee\_field2 ON zemployee (field2).

This creates an index named **idx\_employee\_field2** on the **field2** column of the **zemployee** table.

DATA: lt\_employee TYPE TABLE OF zemployee,

lv\_field2 TYPE string.

" Set a value for lv\_field2 based on your requirement

lv\_field2 = 'SomeValue'.

" Use the SELECT statement with WHERE clause

SELECT \* FROM zemployee

INTO TABLE @lt\_employee

WHERE field2 = @lv\_field2.

\* HINT FOR INDEX ('idx\_employee\_field2').

If you want to ensure that a specific index is used for a particular SELECT statement, you can use the **HINT** clause

* Data types
* Lock Objects   
  Lock objects are used to manage concurrent access to data. Here's an example of using lock objects:

DATA: lv\_key TYPE i,

lv\_lock\_result TYPE lockresult.

lv\_key = 123.

CALL FUNCTION 'ENQUEUE\_READ'

EXPORTING

gname = 'ZEMPLOYEE'

gobject = lv\_key

IMPORTING

lockresult = lv\_lock\_result.

IF lv\_lock\_result = 'E'.

WRITE: 'Record is locked by another user.'.

ELSE.

" Perform your update or read operations here.

CALL FUNCTION 'ENQUEUE\_WRITE'

EXPORTING

gname = 'ZEMPLOYEE'

gobject = lv\_key.

ENDIF.  
In this example, the function modules **ENQUEUE\_READ** and **ENQUEUE\_WRITE** are used to check and set locks on a record.

* Search Helps

To display various possible values for any input field.  
2 types

* 1. Collective- combination of elementary search helps.
  2. Elementary- single search help

Search helps provide assistance to users when entering data in input fields. Here's a simple example:  
PARAMETERS: p\_employee TYPE zemployee-field1 OBLIGATORY

MATCHCODE OBJECT zemployee.

SELECT \* FROM zemployee INTO TABLE itab

WHERE field1 = p\_employee.  
a parameter **p\_employee** is defined with a search help based on the **zemployee** table. When the user enters a value in the input field, they can use the matchcode object to search and select a value from the **zemployee** table.

1. Reports

* Events

Events in SAP ABAP are points in the program execution where specific actions are triggered.

* ALV Reporting

ALV (ABAP List Viewer) reporting is used to enhance the output of reports by providing features such as sorting, filtering, and formatting. ALV Grid and ALV List are common types.

1. ALV Object Model:

The ALV Object Model provides a set of classes and methods that enable developers to create ALV reports. The main classes include:

CL\_GUI\_ALV\_GRID: This class is widely used for displaying tabular data. It provides a grid-based display with features like sorting, filtering, and column resizing.

CL\_SALV\_TABLE: This is a more modern ALV class introduced in SAP NetWeaver 7.02. It simplifies ALV programming and provides additional features.

2. Creating ALV Reports:

#### Using CL\_GUI\_ALV\_GRID:

REPORT ZALV\_REPORT.

DATA: gt\_data TYPE TABLE OF spfli,

go\_alv TYPE REF TO cl\_gui\_alv\_grid.

START-OF-SELECTION.

SELECT \* FROM spfli INTO TABLE gt\_data.

CREATE OBJECT go\_alv

EXPORTING

i\_parent = cl\_gui\_container=>screen0.

CALL METHOD go\_alv->set\_table\_for\_first\_display

EXPORTING

i\_structure\_name = 'SPFLI'

CHANGING

it\_outtab = gt\_data.

#### Using CL\_SALV\_TABLE

REPORT ZALV\_REPORT.

DATA: gt\_data TYPE TABLE OF spfli,

go\_alv TYPE REF TO cl\_salv\_table.

START-OF-SELECTION.

SELECT \* FROM spfli INTO TABLE gt\_data.

cl\_salv\_table=>factory(

IMPORTING

r\_salv\_table = go\_alv

CHANGING

t\_table = gt\_data ).

go\_alv->display( ).

3. **ALV Features:**

**Sorting and Filtering:**

ALV allows users to sort columns in ascending or descending order. Filtering options enable users to display only specific data based on defined criteria.

**Column Resizing and Rearranging:**

Users can resize columns and rearrange their order for better visibility.

**Cell Editing:**

Depending on the ALV configuration, users can edit cell values directly within the ALV grid.

**Toolbar and Function Modules:**

ALV provides a default toolbar with standard functions like Save, Print, and Excel export. Custom function modules can be added to the toolbar for additional functionalities.

4. Event Handling:

5. Field Catalog:

6. Layout Management:

7. ALV Variants:

8. Adaptive ALV Grid (ALV Integrated Data Access):

9. ALV with IDoc and BAPI:

10. ALV in Web Dynpro ABAP:

ALV Grid:

Class:

Class Name: CL\_GUI\_ALV\_GRID

Description: The ALV Grid is implemented using the CL\_GUI\_ALV\_GRID class.

Appearance:

Grid-Based Display: ALV Grid provides a tabular grid-based display of data with rows and columns. It resembles an Excel-like grid.

Features:

Interactive Grid: Users can interact with the grid, perform sorting, filtering, and column resizing interactively.

Cell Editing: Depending on the ALV configuration, users can edit cell values directly within the grid.

Custom Toolbar: Developers can customize the toolbar by adding custom function modules for additional functionalities.

User-Specific Layouts: ALV Grid supports user-specific layouts, allowing users to personalize column settings and layout.  
REPORT ZALV\_GRID\_EXAMPLE.

DATA: gt\_data TYPE TABLE OF spfli,

go\_alv TYPE REF TO cl\_gui\_alv\_grid.

START-OF-SELECTION.

SELECT \* FROM spfli INTO TABLE gt\_data.

CREATE OBJECT go\_alv

EXPORTING

i\_parent = cl\_gui\_container=>screen0.

CALL METHOD go\_alv->set\_table\_for\_first\_display

EXPORTING

i\_structure\_name = 'SPFLI'

CHANGING

it\_outtab = gt\_data.

ALV List:

Class:

Class Name: CL\_SALV\_TABLE

Description: The ALV List is implemented using the CL\_SALV\_TABLE class.

Appearance:

Tabular Display: ALV List provides a tabular display of data similar to ALV Grid but with a slightly different appearance.

Features:

Modern API: ALV List uses a more modern API and simplifies ALV programming.

Built-In Toolbar: ALV List comes with a default toolbar that includes standard functions like Save, Print, and Excel export.

Adaptive ALV (ALV IDA): ALV List is integrated with Adaptive ALV Grid (ALV IDA) in SAP NetWeaver 7.40 and later versions, providing a more responsive user interface.

REPORT ZALV\_LIST\_EXAMPLE.

DATA: gt\_data TYPE TABLE OF spfli,

go\_alv TYPE REF TO cl\_salv\_table.

START-OF-SELECTION.

SELECT \* FROM spfli INTO TABLE gt\_data.

cl\_salv\_table=>factory(

IMPORTING

r\_salv\_table = go\_alv

CHANGING

t\_table = gt\_data ).

go\_alv->display( ).

The **REUSE\_ALV\_GRID\_DISPLAY** and **REUSE\_ALV\_LIST\_DISPLAY** function modules are part of the SAP ALV (ABAP List Viewer) framework and are used to display ALV Grid and ALV List reports, respectively. These function modules simplify the process of creating and displaying ALV reports by providing a set of default settings and configurations.

1. Enhancements

Enhancements in SAP ABAP allow you to add additional functionality to standard SAP applications without modifying the original code.

* User Exits

User exits are predefined exit points in the standard SAP code where you can add custom logic.  
**Identify User Exit in Standard Code:**  
" Standard SAP code

CALL CUSTOMER-FUNCTION 'EXIT\_SAPLMCS2\_001'.

" More standard SAP code  
**Implement User Exit Function Module:**FUNCTION EXIT\_SAPLMCS2\_001.

" Your custom logic here

ENDFUNCTION.

* Customer Exits

Customer exits are similar to user exits but allow customers to implement custom logic at predefined points in the standard SAP code.

* BADIs  
  BADIs provide a way to enhance the functionality of SAP applications in a flexible manner.
* Enhancement Spots  
  Enhancement spots provide a way to group multiple enhancements in one place.  
  In the ABAP Workbench (SE80), create an Enhancement Spot.

ENHANCEMENT-SPOT ES\_MATERIAL.

Create an enhancement implementation  
ENHANCEMENT 1 ZENHANCEMENT.

DATA: lv\_new\_field TYPE string.

" Your custom logic here

Usage in the standard SAP code  
ENHANCEMENT-POINT ES\_MATERIAL BEFORE COMMIT.

" Standard code here

ENHANCEMENT 1 ZENHANCEMENT.

" More standard code here

1. IDOCs  
   IDOCs are standardized XML or flat-file documents that facilitate communication between SAP systems and external systems. They play a crucial role in exchanging data between SAP and non-SAP systems.

Idoc architecture-:control record, data record, status record  
idoc processes-; inbound(direction=2) and outbound (direction =1)  
idoc (data exchange)methods-: ALE(application link enabling) and EDI(electronic data interchange)  
  
ale(data format not change)

* Adding New Segments  
  Use transaction code **WE31** to define a new segment type.  
  Segment Type: ZCUSTOM\_SEGMENT  
  Define the structure of the new segment in transaction **WE30**.  
  Segment Type: ZCUSTOM\_SEGMENT

Segment Definition:

ZFIELD1 (Char, Length 20)

ZFIELD2 (Numeric, Length 10)

Use transaction **WE82** to add the new segment to the existing IDOC type.  
Basic Type: ZIDOC\_TYPE

Extension: ZCUSTOM\_SEGMENT  
Create a new IDOC of the modified type, using transaction **WE19** or an outbound IDOC program.  
IDOC Type: ZIDOC\_TYPE

Data:

ZCUSTOM\_SEGMENT

ZFIELD1 = 'Value1'

ZFIELD2 = 123.

* Configuration Steps in IDOCs

Configuring IDOCs involves setting up partner profiles, ports, and distribution models to enable communication between SAP and external systems.

#### Example Configuration Steps:

1. **Define Logical System and Partner Profiles:**

In transaction **BD54**, define logical systems and create partner profiles.

1. **Configure Port in WE21:**

Use transaction **WE21** to configure the ports for inbound and outbound communication.

1. **Set Up Distribution Model in BD64:**

Configure a distribution model in **BD64** to define the message flow between logical systems.

1. **Configure Ports in WE21:**

In transaction **WE21**, configure ports for inbound and outbound processing.

1. **Define RFC Destinations:**

Use transaction **SM59** to define RFC (Remote Function Call) destinations for communication with external systems.

* Trouble Shooting during errors

#### Troubleshooting IDOC issues involves analyzing error messages, checking configurations, and monitoring system logs. Troubleshooting Steps:

1. **Check IDOC Status in WE02/WE05:**

Transaction **WE02** or **WE05** allows you to view and analyze the status of IDOCs.

1. **Review Error Messages:**

Analyze error messages in the IDOC processing log (transaction **WE05**) or in the application logs (**SM37**).

1. **Check Partner Profiles:**

Verify that partner profiles are correctly configured in **BD54** for both the sender and receiver systems.

1. **Monitor System Logs:**

Use transaction **SM37** to check for system logs and analyze any error messages related to IDOC processing.

1. **Debugging IDOC Processing:**

If necessary, set breakpoints in the IDOC processing function modules and use transaction **WE19** for debugging.

1. ADOBE Forms

Adobe Forms in SAP ABAP are used for creating print layouts and interactive PDF forms. They provide a way to design and generate documents that can be used for various purposes such as invoices, purchase orders, or any other printable document

In the ABAP Workbench (transaction SE80), create a form interface. This is used to define the data structure that the Adobe Form expects.  
DATA: lt\_employee TYPE TABLE OF ty\_employee,

gv\_total\_salary TYPE i.

TYPES: BEGIN OF ty\_employee,

emp\_id TYPE i,

emp\_name TYPE string,

emp\_salary TYPE i,

END OF ty\_employee.

INTERFACE IF\_HR\_FORM.

METHODS GET\_EMPLOYEE\_DATA

IMPORTING

et\_employee\_data TYPE TABLE OF ty\_employee

EXPORTING

ev\_total\_salary TYPE i.

ENDINTERFACE.  
Use Adobe Designer to design the form layout. Define the layout based on the form interface created in the previous step. You can add text fields, tables, and other elements to display the data. In your ABAP program, use the form interface and include logic to fetch data and call the Adobe Form.  
REPORT ZADOBEPDF.

DATA: lt\_employee\_data TYPE TABLE OF ty\_employee,

gv\_total\_salary TYPE i.

" Populate lt\_employee\_data (fetch data from database, etc.)

" Call the Adobe Form

DATA(lo\_form) = NEW (cl\_fp).

DATA(lo\_interface) = CAST IF\_HR\_FORM(lo\_form->get\_form\_interface( ) ).

lo\_interface->get\_employee\_data(

EXPORTING et\_employee\_data = lt\_employee\_data

IMPORTING ev\_total\_salary = gv\_total\_salary ).

" Pass data to Adobe Form and display/print the form

CALL FUNCTION 'FP\_JOB\_OPEN'

EXPORTING

output\_device = 'printer'.

CALL FUNCTION 'FP\_FUNCTION\_MODULE\_NAME'

EXPORTING

formname = 'ZHR\_FORM'

IMPORTING

fm\_name = DATA(lv\_fm\_name).

CALL FUNCTION lv\_fm\_name

EXPORTING

/1bcdwb/docparams/type = 'PDF'

/1bcdwb/docparams/preview = 'X'

TABLES

/1bcdwb/docparams/data = lt\_employee\_data.

CALL FUNCTION 'FP\_JOB\_CLOSE'.

**SAP Adobe Forms vs. SAPScript or Smart Forms:**

* Adobe Forms provide more advanced formatting options and better integration with external systems.
* Adobe Forms are based on XML and PDF, offering better print quality.

1. CDS Views  
   Core Data Services (CDS) is an SAP ABAP development paradigm that enables you to define and consume semantically rich data models in the SAP HANA database  
   CDS views are defined in the Data Definition Language (DDL) using a declarative syntax.  
   They define the structure of the data model, including entities, fields, associations, and annotations.  
   CDS View Types:

Basic View: Represents a projection on a single database table or view.

Composite View: Combines data from multiple tables/views using associations.

Consumption View: Defines how data is consumed by applications or other CDS views.

Interface View: Defines an interface for reuse in other CDS views.  
Annotations provide metadata information to the CDS view, helping to define semantics, behaviors, and UI-related information.  
define a CDS view to represent a list of employees along with their department information.  
@AbapCatalog.sqlViewName: 'ZEMPLOYEE\_VIEW'

@AbapCatalog.compiler.compareFilter: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'Employee View'

define view ZEmployeeView as select from scarr as airline

inner join spfli as flight

on airline.carrid = flight.carrid

left outer join sbook as booking

on flight.connid = booking.connid

{

airline.carrid,

airline.carrname,

flight.connid,

flight.fldate,

booking.bookid,

booking.passname,

booking.smoker

}  
use the CDS view (**ZEmployeeView**) to fetch and display employee data. The data retrieval and processing are similar to working with database tables.  
DATA(lt\_employee\_data) TYPE TABLE OF ZEmployeeView.

SELECT \* FROM ZEmployeeView INTO TABLE @lt\_employee\_data.

LOOP AT lt\_employee\_data INTO DATA(ls\_employee).

WRITE: / ls\_employee-carrid,

ls\_employee-carrname,

ls\_employee-connid,

ls\_employee-fldate,

ls\_employee-bookid,

ls\_employee-passname,

ls\_employee-smoker.

ENDLOOP.

1. AMDP  
   AMDP, or ABAP Managed Database Procedures, is a feature introduced in SAP S/4HANA that allows you to embed SQLScript or Native SQL directly into ABAP classes. AMDP enables you to leverage the power of the SAP HANA database for complex calculations and data processing directly within your ABAP programs.  
   Key Concepts:

* Integration of ABAP with HANA:

AMDP allows you to write database procedures in the SAP HANA-native SQLScript language or using Native SQL in ABAP methods.

* Types of AMDP:

AMDP Methods: AMDP methods are part of ABAP classes and can include either SQLScript or Native SQL.

AMDP Function: AMDP functions allow you to define standalone database functions.

* AMDP Annotations:

Annotations are used to define the type of the AMDP method (READ, WRITE, etc.) and provide additional metadata.

* Input and Output Parameters:

AMDP methods can have input parameters, output parameters, or a combination of both. These parameters facilitate the exchange of data between ABAP and the HANA database.  
consider a simple example where we create an AMDP method to calculate the total salary of employees in a specific department directly in the HANA database.  
CLASS ZEmployeeCalculator DEFINITION.

PUBLIC SECTION.

CLASS-METHODS CalculateTotalSalary

FOR DATABASE PROCEDURE

USING ZEMPLOYEE

CHANGING TOTAL\_SALARY TYPE i.

ENDCLASS.

CLASS ZEmployeeCalculator IMPLEMENTATION.

METHOD CalculateTotalSalary BY DATABASE PROCEDURE FOR HDB

LANGUAGE SQLSCRIPT

OPTIONS READ-ONLY.

TOTAL\_SALARY = SELECT SUM(SALARY) FROM :ZEMPLOYEE WHERE DEPARTMENT = 'HR';

ENDMETHOD.

ENDCLASS.  
In this example, the **CalculateTotalSalary** method is defined as an AMDP method. It uses SQLScript to calculate the total salary of employees in the HR department. In your ABAP program, you can call the AMDP method to calculate the total salary and then display the result.  
DATA(lv\_total\_salary) TYPE i.

CALL METHOD ZEmployeeCalculator=>CalculateTotalSalary

EXPORTING

TOTAL\_SALARY = lv\_total\_salary.

WRITE: / 'Total Salary in HR Department:', lv\_total\_salary.  
  
AMDP and Code Pushdown:

AMDP supports the code pushdown paradigm, allowing complex calculations and processing to be executed directly in the database engine, leading to improved performance.

AMDP and SQLScript:

When using SQLScript in AMDP, you have access to advanced features of the SAP HANA database, such as table functions, stored procedures, and more.

AMDP Annotations:

Annotations like FOR DATABASE PROCEDURE and OPTIONS READ-ONLY are used to define the behavior and properties of AMDP methods.

AMDP Function:

AMDP functions allow you to define standalone database functions that can be called from ABAP programs.

1. OData Services   
   Key Concepts:

Service Development:

Service development involves defining data models and service operations that represent SAP entities and functionalities.

Entity Data Model (EDM):

The Entity Data Model defines the structure of data exposed by the OData service. It includes entities, properties, and associations.

Service Implementation:

Service implementation involves coding the logic for CRUD (Create, Read, Update, Delete) operations and other custom operations exposed by the OData service.

Service Registration:

After implementation, the OData service needs to be registered in the SAP Gateway system to make it available for consumption.  
consider a simple example where we expose employee data from the SAP system as an OData service.

* Define Entity Data Model (EDM) in SEGW:

In transaction SEGW, define an Entity Data Model (EDM). For example, let's create an entity called Employee with properties like EmployeeID, FirstName, LastName, and Salary.

* Implement OData Service in SEGW:

Implement the OData service operations in SEGW. For example, create operations to retrieve a list of employees and to retrieve details of a specific employee.  
CLASS zcl\_employeeservice DEFINITION.

PUBLIC SECTION.

DATA: mt\_employee TYPE TABLE OF zemployee,

ms\_employee TYPE zemployee.

METHODS get\_employee\_list

IMPORTING

VALUE(iv\_department) TYPE zemployee-department

EXPORTING

VALUE(rt\_employee) TYPE TABLE OF zemployee.

METHODS get\_employee\_details

IMPORTING

VALUE(iv\_employee\_id) TYPE zemployee-employee\_id

EXPORTING

VALUE(rs\_employee) TYPE zemployee.

ENDCLASS.

CLASS zcl\_employeeservice IMPLEMENTATION.

METHOD get\_employee\_list.

SELECT \* FROM zemployee INTO TABLE rt\_employee WHERE department = iv\_department.

ENDMETHOD.

METHOD get\_employee\_details.

SELECT \* FROM zemployee INTO rs\_employee WHERE employee\_id = iv\_employee\_id.

ENDMETHOD.

ENDCLASS.

* Register and Activate OData Service:

Register the OData service in SEGW.Activate the service to generate runtime artifacts.

* Test OData Service:

You can use tools like SAP Fiori Launchpad or tools like Postman to test the OData service endpoints. For example:

List of Employees: http://<server>/sap/opu/odata/sap/ZEMPLOYEESERVICE\_SRV/EmployeeSet?$filter=Department eq 'HR'

Employee Details: http://<server>/sap/opu/odata/sap/ZEMPLOYEESERVICE\_SRV/EmployeeSet('123')

### Additional Considerations:

* **OData Annotations:**
  + Annotations such as **@OData.publish: true** are used to expose entities and operations as part of the OData service.
* **Security Considerations:**
  + Ensure that proper security measures are implemented, and that the OData service is configured to allow or restrict access based on user roles.
* **Batch Operations:**
  + OData services support batch operations, allowing you to bundle multiple requests into a single HTTP request.
* **Error Handling:**
  + Implement proper error handling mechanisms to provide meaningful error responses to clients.