SAP
C_ABAPD
SAP Certified - Backend Developer - ABAP Cloud (Beta Version)

Latest Version: 4.0

Topic: 1
Dunmara Pharmacies Builds a Medication Shelf-Life Helper
Business Context
Dunmara Community Pharmacies operates a network of forty-one neighbourhood pharmacies across the Irish midlands, each dispensing prescriptions and stocking over-the-counter lines for its local community. A small internal development team, working in a mentored capacity, has been asked to build a lightweight shelf-life helper on the SAP BTP ABAP environment. The application gives branch pharmacists a quick view of which medication batches are approaching their use-by date, alongside a short summary of what each branch has dispensed recently, so that stock rotation and reordering stay ahead of local demand.
System Landscape

The helper is being developed entirely with ABAP Cloud tooling and is expected to stay within clean-core boundaries, using released objects and public interfaces rather than touching standard SAP artefacts. Batch, branch, and dispensing data already exist in the underlying tables, and the team is layering a thin set of CDS read models and a simple business object on top. Because the chain runs a single shared instance, the same data model serves all forty-one locations, and any reusable view is likely to be consumed by more than one developer over time.

Reporting Goal

Two outputs matter most to the branches. The first is a near-expiry list that flags batches within a configurable window of their use-by date, ordered so the most urgent appear first. The second is a dispensing summary that totals what a chosen branch has handed out over a recent period, giving the pharmacist a feel for movement before placing the next order. Both are meant to be glanceable on a busy counter, refreshing quickly and showing only what is relevant to the branch in front of the user.

Data Foundations

Use-by moments are recorded against each batch as timestamps captured when stock is booked in, and the chain's system holds those timestamps in a single coordinated time reference rather than in per-branch local time. Presentation details for the list — the column labels and the order in which fields appear — were added directly onto the shared read model during early prototyping. The dispensing summary is built by first collecting a branch's batch keys into an internal table and then reading the matching dispensing records in one set-based pass.

Pilot Observations

During pilot use, a handful of batches surfaced on the near-expiry list a day before the date printed on their packaging, while a few others dropped off a day after. On one quiet morning, a branch manager opening the dispensing summary saw entries that plainly belonged to other locations. Separately, a second developer who wanted to reuse the batch read model found it already carried display formatting, and was unsure whether changing a label would disturb the counter view that depended on it.

Delivery Considerations

The pilot feedback is due back before the wider rollout, and the team would prefer corrections that keep the shared model stable as more branches come online. Leadership has been clear that the helper should remain a clean-core extension so future platform updates do not disturb it, even where a quicker local shortcut might close a pilot comment sooner. The mentored developer has been asked to settle the calculation, presentation, and read behaviours so the same outputs can be trusted at every branch.

Your assignment

You are working as a mentored developer on Dunmara Community Pharmacies' shelf-life helper in the SAP BTP ABAP environment. All work must remain within ABAP Cloud and clean-core boundaries, consuming released objects and public interfaces only. The helper serves forty-one branches from one shared instance, so any reusable artefact may be consumed by other developers. Complete the task expectations below and confirm each through its stated checkpoint. Do not modify standard SAP objects

1. CHALLENGE 1 — Calculating Days to Expiry from Stored Timestamps

• Derive a days-to-expiry value for each batch from its stored use-by timestamp.

• Align the timestamp to the branch's local calendar date before computing the difference against the current local date.

• Use the platform's date and timestamp handling for the calculation rather than manual arithmetic on the raw values.

• Order the near-expiry list so the most urgent batches appear first and apply the configured window consistently.

CHECKPOINTS

• A batch whose use-by moment falls just before local midnight and one just after both report a days-remaining value matching the branch calendar.
• The days-to-expiry figure is recomputed against the current date each time the list is shown.

2. CHALLENGE 2 — Separating Display Annotations into a Metadata Extension

• Identify the presentation and semantic annotations currently inline on the shared batch read model.
• Relocate those annotations into a metadata extension layered on the view, leaving the core view free of display concerns.
• Confirm the application reads its labels and column order from the metadata extension.
• Keep the core view a stable, reusable model that other developers can consume without inheriting display edits.

CHECKPOINTS

• The core view activates with no presentation annotations remaining.
• The counter view still renders the expected labels and column order through the metadata extension.

3. CHALLENGE 3 — Reading Dispensing Records Through a Driver Table

• Build the dispensing summary by collecting the chosen branch's batch keys into a driver internal table and reading matching records in one set-based pass.
• Guard the read so an empty driver table never causes records from other branches to be returned.
• Ensure the summary totals only the selected branch's dispensing records.
• Keep the read responsive for branches with little or no recent activity.

CHECKPOINTS

• A branch with no recent dispensing returns zero rows and exposes no other branch's records.
• A busy branch returns only its own records and a total consistent with its activity.

Question: 1

CHALLENGE 1 — Calculating Days to Expiry from Stored Timestamps

Branch staff report that a few batches appear in the near-expiry list one day before their printed use-by date, while others drop off a day late. The days-remaining value is computed by subtracting today's date from the date part of the stored use-by timestamp.
What is the most likely cause of the one-day discrepancy?

A. The batch master already holds the use-by value as a plain calendar date, so the extra conversion step shifts every computed result by one full day during the read.
B. The near-expiry threshold is configured too tightly, so borderline batches move in and out of the list as the day-count rounding changes between runs.
C. The use-by moment is stored in UTC, so reading its date part without a local-date conversion makes batches near midnight shift by a day.
D. The list query orders batches by their text description instead of by the use-by value, so near-expiry rows surface in an unstable position on each run.

Answer: C

Explanation:
The use-by value is held as a UTC timestamp, so reading only its date part ignores the offset to the branch's local zone. For batches expiring close to midnight the UTC date and the local date differ by one day, which is exactly why some rows appear a day early and others a day late. Converting to the local date before extracting the day removes the shift.

Question: 2

CHALLENGE 1 — Calculating Days to Expiry from Stored Timestamps
The app must show an accurate days-to-expiry figure for each batch that is consistent across all forty-one branches.
Which approach computes the figure correctly?

A. Convert the stored UTC timestamp to the branch's local date, then take the difference from the current local date using built-in date arithmetic.
B. Subtract the two raw timestamp values, divide the elapsed seconds by the seconds in a day, and round the result to the nearest whole number for display.
C. Compare the character form of the use-by timestamp against today's date as text, since both share a year-month-day ordering and therefore sort correctly.
D. Store a days-remaining number on each batch at creation time and display that saved value directly, without recomputing it on later runs of the list.

Answer: A

Explanation:
Converting the stored UTC timestamp to the branch's local date first anchors the comparison to the calendar the staff actually use, and taking the difference with built-in date arithmetic avoids manual errors. This produces a consistent days-to-expiry value across every branch regardless of where midnight falls.

Question: 3

CHALLENGE 1 — Calculating Days to Expiry from Stored Timestamps
You are ready to release the expiry calculation and need to confirm it behaves correctly for batches expiring around midnight in the branch's time zone.
Which check best confirms correct behaviour?

A. Confirm the near-expiry list still loads within the agreed response time when every branch queries it at once during morning opening hours.
B. Verify that each batch record carries a non-initial use-by timestamp, so the calculation never runs against a blank or default value.
C. Check that the list returns exactly the same row count as the previous release, confirming that no batches were unexpectedly added or later dropped.
D. Test batches whose use-by moment sits just before and just after local midnight, and confirm the days-remaining figure matches the branch calendar.

Answer: D

Explanation:

The fault only appears at the local-midnight boundary, so the decisive test uses batches whose use-by moment sits just on either side of local midnight and confirms the days-remaining value matches the branch's calendar date. Passing this case proves the time-zone conversion is applied correctly.

Question: 4

CHALLENGE 2 — Separating Display Annotations into a Metadata Extension
The batch list view currently mixes presentation annotations — column labels and display ordering — directly into the core CDS view that other developers also reuse. Clean-core guidance asks you to keep the reusable model stable.
What should you do?

A. Leave the annotations inside the core view but add comments marking the presentation-only lines, so future developers know not to rely on them.
B. Move the presentation and semantic annotations into a separate metadata extension on the view, keeping the core model free of display concerns.
C. Duplicate the core view into a second copy that carries the annotations, point the app at the copy, and let other developers keep using the original.
D. Embed the annotations in the service binding instead, placing presentation nearer the user interface than the underlying data model where it sits today.

Answer: B

Explanation:
Moving presentation and semantic annotations into a metadata extension keeps the core CDS view a clean, reusable data model while the display details live separately. Other developers can keep depending on the stable core, and the app still gets its labels and columns from the extension.

Question: 5

CHALLENGE 2 — Separating Display Annotations into a Metadata Extension
A teammate argues that inlining the annotations is faster to deliver and asks why the metadata extension is worth the extra object.
Which reason best supports the clean-core choice?

A. The metadata extension makes the near-expiry query run faster, because the database can skip the presentation annotations while it executes the read.
B. The metadata extension lets the app sidestep the access controls on the core view, which simplifies how each branch's data is read.
C. Keeping presentation out of the core view leaves it a stable, reusable model that other teams can evolve without breaking display behaviour.
D. Storing the annotations separately means the column labels will be translated into each language automatically, with no further configuration by the team.

Answer: C

Explanation:
The value of the metadata extension is structural: keeping presentation out of the core view leaves that view a stable, reusable model that SAP and other teams can evolve without breaking anyone's display logic. That upgrade stability is the clean-core benefit, not a one-off delivery saving.

Question: 6

CHALLENGE 2 — Separating Display Annotations into a Metadata Extension
You have relocated the annotations and need to confirm the core view stayed a clean, reusable model while the list still displays correctly.
Which check best confirms a correct separation?

A. Confirm the core view activates with no presentation annotations left, and the app — reading through the metadata extension — still shows the expected labels and columns.
B. Confirm the core view now returns more rows than before, which would prove the annotations had previously been hiding some of the batches from view.
C. Confirm the metadata extension still compiles entirely on its own, even after the core view that it is supposed to extend has been deleted from the system.
D. Confirm the app continues to work once the annotations are deleted altogether, since presentation metadata is optional for simply returning the batch data.

Answer: A

Explanation:
The separation is correct when the core view activates carrying no presentation annotations and the app, reading through the metadata extension, still renders the expected labels and columns. That confirms both that the core stayed clean and that display behaviour was preserved.

Question: 7

CHALLENGE 3 — Reading Dispensing Records Through a Driver Table
A branch manager opens the dispensing summary on a quiet morning and unexpectedly sees dispensing rows from several other branches. The summary reads dispensing records with a key-driven set-based read, driven from an internal table of that branch's batch keys.
What is the most likely cause?

A. The dispensing table lacks a secondary index on the branch field, so the database widens the read to every branch in order to satisfy the original query during opening hours.
B. The branch filter sits in the display layer rather than in the query itself, so all rows are fetched first and only some of them are hidden afterwards.
C. The internal table of keys holds duplicate entries, so the read multiplies its result set and drags in records that belong to other branches entirely.
D. The driver table of batch keys was empty for that quiet branch, and a key-driven set-based read over an empty table returns all rows, not none.

Answer: D

Explanation:
A key-driven set-based read returns all rows when its driver internal table is empty. On a quiet morning the branch had no batch keys, so the driver table was empty and the read pulled every branch's dispensing rows instead of none. The empty-driver case is the second-order cause behind the unexpected foreign rows.

Question: 8

CHALLENGE 3 — Reading Dispensing Records Through a Driver Table
You must make the dispensing summary reliable for branches with little or no recent activity.
Which correction is complete?

A. Sort the driver table of keys and strip out duplicates before the read, so the result set stays clean and no foreign-branch rows slip in.
B. Check that the driver table of keys is not empty before issuing the read, and return nothing when there are no keys to drive it.
C. Add the branch as an extra condition inside the read so the database also filters on branch, while leaving the driver-table handling exactly as it is now.
D. Raise the app's result limit so the larger row set loads fully, then total only those rows whose branch matches the current user's branch afterwards.

Answer: B

Explanation:
The reliable fix is to confirm the driver table of keys is not empty before issuing the read and to return nothing when there are no keys, which removes the empty-driver case that exposes every branch. It addresses the root cause rather than masking the symptom.

Question: 9

CHALLENGE 3 — Reading Dispensing Records Through a Driver Table
The corrected read must protect both the branch data boundary and response time for low-activity branches.
Which test best confirms this?

A. Run the summary for a branch with no recent dispensing and confirm it returns zero rows quickly, revealing no other branch's records.
B. Run the summary for the busiest branch and confirm its totals match the figures produced by the previous version of the same report.
C. Run the summary repeatedly for a single active branch and confirm the response time stays within the agreed limit even under sustained, repeated load.
D. Run the summary across every branch at once and confirm the combined total equals the chain-wide dispensing figure for the whole period.

Answer: A

Explanation:
The defect surfaces specifically for low-activity branches, so running the summary for a branch with no recent dispensing and confirming it returns zero rows quickly — with no other branch's records — proves both the boundary and the response time are protected. This directly exercises the empty-driver path.

Topic: 2

Saarvik Resource Recovery Builds a Collection-Site Levy Helper

Business Context

Saarvik Resource Recovery operates kerbside collection and a network of staffed drop-off sites across the Tartu region in eastern Estonia. The company weighs incoming waste at each site, classifies it by material stream, and bills municipalities and commercial customers a disposal levy based on tonnage. A small development team, working in a mentored setup alongside a lead developer, has been asked to build a lightweight extension that helps site clerks capture loads, validate the site reference on each delivery, and produce a levy figure that finance can stand behind. The work is deliberately modest in scope, but it touches data that ends up on customer invoices, so the clerks expect it to be dependable from the first day it is used.

System Landscape

The extension runs in the company's SAP BTP ABAP environment and follows clean-core development practices, so the team builds only with released objects and public interfaces rather than reaching into standard internals. Site master data — region, operating company, and address — already exists in the connected SAP S/4HANA system and is published through a released data source. The new helper adds its own load and levy records while drawing site context from that existing source, and the lead developer has been clear that the extension must stay upgrade-stable as the underlying systems move forward.

Development Setup

Each site delivery is identified by a collection-site code that pairs a regional prefix with a year and a running sequence. The codes were introduced when the company served three districts, and the original entry check was written to read the prefix from a fixed position in the string. As Saarvik has expanded, several newer sites have begun issuing codes whose prefix differs in length, and clerks at those sites report that some of their correctly formed codes are turned away at entry while a few malformed ones slip through unnoticed.

Levy Calculation Context

The levy is derived from the weighed quantity and a published rate expressed per tonne. Weights are recorded in kilograms, and the helper converts them to tonnes inside the calculation before applying the rate. On loads that weigh a whole number of tonnes the figures reconcile cleanly with the finance team's manual checks, but on mixed loads the helper's amount occasionally lands a few cents under the manually computed value, and the gap has started to surface on month-end summaries where the totals no longer tie out.

Reporting Context

The calculation itself lives in a small class that several reporting programs reuse. Those programs currently read and set the class's working fields directly, and during a review the lead developer noticed one report that displayed a levy figure which did not correspond to the weight and rate shown beside it. The team wants the calculation to behave the same way no matter which program drives it, so that a figure on one report cannot quietly disagree with the inputs printed next to it.

Data Sourcing Context

To speed up site lookups, an earlier version of the helper copied the site master records into a custom table refreshed by a nightly job. Recently a site changed its operating company, and the helper

continued to show the previous operator for most of the following day. With a wider rollout approaching, the team is weighing whether to keep maintaining the local copy or to read site attributes directly from the released source, while finance has asked that any figures shown on a finalized levy remain reproducible later if a customer queries them.

Release Preparation

The lead developer wants the helper hardened before it is offered to additional sites. The team should be able to explain why each part behaves as it does, show evidence that the levy figures are correct on the kinds of loads that arrive in practice, and keep the whole extension within the clean-core boundaries the company has committed to.

Your assignment

You are contributing, in a mentored role, to a clean-core extension in the SAP BTP ABAP environment for Saarvik Resource Recovery. The extension captures site deliveries, validates the collection-site code, calculates a disposal levy from weight and a per-tonne rate, and draws site master data from a released source. Work only with released objects and public interfaces. Complete the tasks below as you would on a system-based assessment, and be ready to show the evidence that each task is correct

1. CHALLENGE 1 — Validating the Collection-Site Code Format Reliably

• Establish the permitted structure of a collection-site code: a regional prefix, a year, and a running sequence, separated as the business defines them.

• Replace the fixed-position prefix reading with a check that evaluates the code against its permitted structure as a whole, so prefixes of differing length are handled.

• Ensure malformed codes are rejected at entry and that well-formed codes from every site are accepted.

CHECKPOINTS

• Confirm that codes with shorter and longer prefixes than the original three-letter form are accepted when otherwise well-formed.

• Confirm that codes missing a separator, a year, or a sequence are rejected at entry rather than reaching load capture.

2. CHALLENGE 2 — Calculating the Disposal Levy Without Losing Precision

• Trace the calculation from weight in kilograms, through the conversion to tonnes, to the application of the per-tonne rate and final rounding.

• Perform the conversion and rate application so that fractional tonnage is preserved through the calculation, rounding only once at the end to currency precision.

• Verify the result against an independently computed value for loads that are not whole tonnes.

CHECKPOINTS

• Confirm that a load with fractional tonnage produces the same levy as the independent calculation, to the cent.

• Confirm that whole-tonne loads continue to reconcile after the change.

3. CHALLENGE 3 — Encapsulating the Levy Calculation Inside Its Class

• Identify which of the class's working fields participate in the levy invariant of weight, rate, and amount.

• Restructure the class so callers cannot place those fields into a state inconsistent with the calculation, exposing the amount only as a derived value.

• Confirm that every reporting program obtains the levy the same way and cannot set it directly.

CHECKPOINTS

• Confirm that no program can display an amount that differs from the weight and rate it was calculated from.

• Confirm that the class behaves identically regardless of which reporting program drives it.

4. CHALLENGE 4 — Sourcing Site Master Data the Clean-Core Way

• Read current site attributes from the released source rather than from a maintained local copy.

• Where finance requires a finalized levy to remain reproducible, retain only a deliberate point-in-time snapshot of the values shown, not a full parallel copy of master data.
• Keep all access within released objects and public interfaces, avoiding changes to standard internals.

CHECKPOINTS

• Confirm that a change to a site's attributes at the source is reflected promptly in the helper.
• Confirm that a finalized levy can still be reproduced without maintaining a drifting local table.

Question: 10

CHALLENGE 1 — Validating the Collection-Site Code Format Reliably

Clerks at newer sites report that some correctly formed collection-site codes are rejected at entry, while a few malformed codes are accepted.
What best explains this behavior?

A. The released site source orders newer regional prefixes under a different collation, so the lookup no longer matches them, and aligning the source's sort sequence would let those codes validate again at entry.
B. The check reads the prefix from a fixed character position assuming a three-letter prefix, so codes with a different prefix length are misread, and describing the format as a pattern accepts every valid code.
C. The newer codes lose their leading zeros to an automatic numeric conversion before the length test runs, so they fail it, and restoring those zeros ahead of the check would stop the rejections.
D. The collection-site code field is defined too short to hold the newer values, so the input is truncated before validation runs, and widening that field would allow the complete code through.

Answer: B

Explanation:
Reading a fixed number of characters from a fixed position only works while every prefix is the same length; once newer sites use prefixes of a different length, the fixed read lands on the wrong characters, rejecting valid codes and sometimes accepting malformed ones. Describing the whole permitted structure as a pattern removes the dependence on a single assumed width.

Question: 11

CHALLENGE 1 — Validating the Collection-Site Code Format Reliably

The team wants the entry check to handle prefixes of differing length without breaking again at the next variation.
How should the collection-site code be validated most reliably?

A. Match the entire code against a pattern that expresses the allowed prefix, year, and sequence structure, rejecting any input that does not fit the described format.
B. Split the code on its hyphens and check that each separated part has the exact character length the first three districts originally used at entry.
C. Confirm only that the code contains two hyphens somewhere in the string, accepting whatever prefix, year, or sequence values happen to sit between them.

D. Trim the surrounding spaces from the code and compare its total character length to the fixed length the earliest district codes had, rejecting anything that differs.

Answer: A

Explanation:
A pattern that describes the permitted prefix, year, and sequence validates the structure as a whole, so codes are accepted or rejected on whether they fit the format rather than on an assumed fixed width. This handles shorter and longer prefixes alike and does not need rework each time a new variation appears.

Question: 12

CHALLENGE 1 — Validating the Collection-Site Code Format Reliably
A developer widens the prefix check from three to four characters and asks whether the validation is now sound.
What is the best assessment?

A. Yes; four characters covers the longest prefix in use today, so fixing the width to four fully resolves the rejection of valid codes across every current and future site without further change.
B. Yes, once the four-character prefix is combined with a database index on the code field, which is what actually allowed the malformed codes to pass the earlier check at entry.
C. No; the width is fine, but the code must first be converted to upper case before the check, because the stored prefixes use a mix of letter cases that breaks the comparison.
D. No; hard-coding another fixed width still fails the next variation, such as a two-letter prefix or a longer sequence, while a pattern describing the structure covers all valid forms.

Answer: D

Explanation:
Replacing one fixed width with another fixed width only moves the boundary; a two-letter prefix or a longer sequence would break it again, so the change is a partial fix. A pattern that describes the permitted structure validates any well-formed code without being tied to a specific length.

Question: 13

CHALLENGE 2 — Calculating the Disposal Levy Without Losing Precision
Finance reports that the helper's levy is exact on whole-tonne loads but a few cents low on mixed loads.
What is the most likely cause?

A. The currency field holding the levy is defined with too few decimal places, so the last cents are dropped when the amount is stored after the calculation has already completed correctly.
B. The output formatting rounds the displayed levy downward for presentation, so the stored amount is correct while only the figure shown on the month-end summary appears a few cents short.
C. The kilogram-to-tonne conversion divides using whole-number operands, so the fractional tonnage is dropped before the rate applies, and a decimal type for that step keeps the fraction.

D. The database rounds decimal results during the read, so the tonnage arrives already truncated, and reading the weight as a different numeric type on the way in would avoid the loss before calculation.

Answer: C

Explanation:
When the conversion to tonnes divides using whole-number operands, the fractional tonnage is discarded before the rate is applied, so mixed loads lose value while whole-tonne loads are unaffected. Performing that step with a decimal type preserves the fraction so the rate applies to the true tonnage.

Question: 14

CHALLENGE 2 — Calculating the Disposal Levy Without Losing Precision
The team needs the conversion and rate application to produce an amount that matches finance to the cent.
How should the calculation be implemented?

A. Reorder the arithmetic to multiply by the rate before dividing by the tonnage factor, while keeping the existing whole-number operands throughout the intermediate calculation steps.
B. Carry out the division and multiplication with decimal-typed operands so the fraction survives, then round once to the currency's precision at the final step.
C. Leave the whole-number division in place and cast only the final result to a decimal type afterward, which is expected to restore the fractional cents lost earlier in the calculation.
D. Increase the number of decimal places shown for the levy on the report output, so the previously hidden fractional amount becomes visible again to the finance reviewers.

Answer: B

Explanation:
Using decimal-typed operands keeps the fractional tonnage through the division and the rate application, so no value is lost mid-calculation; rounding once at the end then produces a correct currency amount. This fixes the loss at the point where it occurs.

Question: 15

CHALLENGE 2 — Calculating the Disposal Levy Without Losing Precision
The current tests all use whole-tonne weights and pass, yet finance still finds discrepancies.
Which test case would best expose the precision problem?

A. A load whose weight is not a whole number of tonnes, such as 1,250 kilograms, where the dropped fraction shifts the levy that whole-tonne test data cannot reveal.
B. A very large load near the top of the recorded weight range, which would push the resulting levy amount beyond the limit the currency field is able to represent on the screen.
C. A load recorded at a newly opened site, whose collection-site code uses the longer prefix that the entry validation has been rejecting for the clerks working there.

D. A load entered with a zero weight, which would drive the levy to nothing and show whether the calculation handles an empty delivery without raising a runtime error.

Answer: A

Explanation:
Whole-tonne weights convert without any fraction to lose, so they cannot reveal truncation; a fractional tonnage like 1,250 kilograms exercises exactly the step where value is dropped, making the shortfall visible against an independent calculation. That is the case the existing tests are missing.

Thank You for Trying Our Product

For More Information – Visit link below:

https://www.examsboost.com/

15 USD Discount Coupon Code:

G74JA8UF

FEATURES

- ✓ 90 Days Free Updates
- ✓ Money Back Pass Guarantee
- ✓ Instant Download or Email Attachment
- ✓ 24/7 Live Chat Support
- ✓ PDF file could be used at any Platform
- ✓ 50,000 Happy Customer