SAP
C_ABAPD_2601
SAP Certified - Backend Developer - ABAP Cloud (Beta Version)

For More Information – Visit link below:

https://www.examsempire.com/

Product Version

1. **Up to Date products, reliable and verified.**
2. **Questions and Answers in PDF Format.**



Latest Version: 4.0

Topic: 1
Unified Scenario Simulation
Business Context
Helvent Aerospace Systems manufactures precision turbine blades and structural fittings for regional aircraft programmes, operating machining and finishing plants in Stuttgart and Wrocław. Quality is the company's regulatory lifeline, and incoming-inspection records must be defensible during audits by both aviation authorities and tier-one customers. To modernise how inspectors capture inspection lots and nonconformance reports, Helvent built a custom application on the SAP BTP ABAP environment using ABAP Cloud and the RESTful Application Programming Model, sitting beside its SAP S/4HANA Cloud core.
System Landscape
The application exposes a small set of business objects: an inspection lot object, a nonconformance object, and a supplier-rating projection used by procurement. The data model is built from Core Data Services views layered over dictionary tables, with associations from inspection lots to their nonconformance items. A service binding publishes the objects to two Fiori-style consumers, one for shop-floor inspectors and one for the central quality team. The build team followed ABAP Cloud rules so that only released objects and public APIs were used.
Implementation Progress
Go-live was four weeks ago, and the project is now in hypercare. Volumes climbed faster than the pilot suggested: the Stuttgart plant alone now posts several thousand inspection results per shift, and the central quality dashboard aggregates them across both plants. During the pilot the dataset was small enough that nobody scrutinised how the reads were written, and the inspection-lot list felt instant.
Operational Observations
In the third week of hypercare the inspector list began to lag noticeably during the late shift, when posting volume peaks. A developer tracing one of the slow interactions noticed the list logic reads a large internal table of inspection results and walks it line by line to match each lot, repeating the walk for every lot on the screen. The same logic also pulls more columns than the list displays. Procurement separately reported that the supplier-rating figures lag behind what inspectors see, and that the gap widens exactly when the plants are busiest.
Access Context
A second observation came from the Wrocław quality lead. During an audit rehearsal she opened the central dashboard with a plant-quality role and could read nonconformance details for Stuttgart suppliers that her plant does not handle. The build team had added an imperative authorisation check inside one ABAP method that backs a single action, but the list and analytical reads reach the data through CDS views that carry no read restriction. Inspectors on either plant can therefore retrieve the other plant's records through the list, even though the action button is guarded.
Behavior Context
The nonconformance object was originally specified as fully managed, letting the RAP runtime provide the standard create and update operations. Late in the build, the team added a requirement that posting a nonconformance must call an external supplier-notification service and stamp a certificate number that the runtime cannot derive. Part of the posting logic was hand-written to do this, but it was wired in inconsistently, so some nonconformance records are saved with a certificate number and some without, depending on which entry path the inspector used.
Governance Context
Helvent's architecture board has one firm rule: extensions must stay upgrade-stable and clean-core compliant, because the company cannot afford to re-validate custom code against every S/4HANA Cloud

upgrade in a regulated setting. During hypercare a contractor proposed two quick fixes — copying a standard supplier-master object into the custom namespace to add a field, and reading an internal, not-released table directly to speed up the supplier rating. Both would make the symptoms disappear this week. The ABAP Test Cockpit run that the pipeline enforces flags the second approach, and the board has asked whether either is acceptable.

Decision Pressure

The hypercare exit gate is in ten days. The steering group wants the inspector list responsive at peak, the plant data boundaries provably enforced for every read path, nonconformance posting consistent regardless of entry point, and no extension that a future upgrade could break. The quality director has made clear that an audit finding caused by a cross-plant data leak would be far more damaging than a slow screen, but that a list which stalls during the late shift is also unacceptable. The team must decide which corrections are sound and which merely move the problem.

Validation Outlook

Each correction will be checked before the gate: the list behaviour under peak load, the read paths an inspector can actually reach, the certificate stamping across every posting path, and the ATC verdict on every changed object. The team's task is to choose corrections that hold up under all four checks at once, rather than fixes that pass one and fail another.

Your assignment

You are a backend ABAP Cloud developer assigned to the Helvent Aerospace quality application during hypercare. You work in the SAP BTP ABAP environment with ABAP Cloud rules enforced, beside an SAP S/4HANA Cloud core. The application's business objects are built with CDS, RAP behaviour, and a published service binding. Your corrections will be evaluated against peak-load behaviour, reachable read paths, posting consistency, and the ATC verdict. Use only released objects and public APIs. Where a specific transaction, app identifier, or released-API name is not provided, reason about the behaviour rather than assuming an exact name

1. CHALLENGE 1 — Stabilizing Inspection List Reads Under Peak Posting Load

• Review the list interaction that reads inspection results into an internal table and matches lots line by line.

• Restructure the data retrieval so that data-intensive selection and aggregation occur where they are most efficient for large volumes.

• Choose an internal-table design and access path appropriate for repeated keyed lookups rather than a repeated linear search.

• Restrict the selection to the fields the list and aggregation actually require.

Checkpoints

• Confirm the corrected interaction holds its response under late-shift posting volume.

• Confirm the supplier-rating aggregation no longer lags behind inspector activity at peak.

2. CHALLENGE 2 — Enforcing Plant Data Boundaries Across Every Read Path

• Identify every path through which an inspector can retrieve inspection and nonconformance records, including list and analytical reads.

• Apply read-access protection at the layer that governs all of those read paths, scoped by the inspector's plant.

• Treat the existing imperative check on a single action as insufficient on its own for read protection.

Checkpoints

• Confirm a plant-quality role can retrieve only its own plant's records through every read path.

• Confirm the audit-rehearsal cross-plant retrieval no longer succeeds.

3. CHALLENGE 3 — Making Nonconformance Posting Consistent Across Entry Paths

• Trace each entry path through which a nonconformance can be posted.

• Place the certificate stamping and external notification where the behaviour applies to every posting, not only selected entry points.
• Decide how the non-derivable certificate value is produced within the behaviour, given that the standard runtime cannot derive it.

Checkpoints

• Confirm every newly posted nonconformance carries a certificate number regardless of entry path.
• Confirm the external notification fires once per posting and is not duplicated or skipped.

4. CHALLENGE 4 — Keeping Supplier Extensions Upgrade-Stable Under Clean Core

• Evaluate each proposed quick fix against clean-core and upgrade-stability rules.
• Prefer released objects, public APIs, and approved extension points over copying or directly reading standard objects.
• Use the enforced static-analysis verdict as the gate for whether a changed object is cloud-ready.

Checkpoints

• Confirm every changed object passes the enforced static-analysis check for cloud readiness.
• Confirm no correction depends on a not-released object or a modified standard object.

Question: 1

CHALLENGE 1 — Stabilizing Inspection List Reads Under Peak Posting Load

The list logic reads a large set of inspection results into a standard internal table and, for each lot on the screen, walks the whole table to find matching results. Response time degrades sharply as posting volume peaks.

Which change most directly removes the cause of the degradation while keeping the result correct?

A. Keep the standard table and the per-lot walk, but read the results in smaller blocks so that each individual read returns faster at peak.
B. Move the matching into a post-shift background job that pre-walks the table, so the list reads a cached result set instead of live data.
C. Push selection and aggregation into ABAP SQL or CDS so only the needed rows return, and use a keyed table access for the lookups.
D. Keep the loop but sort the standard table once before render, so each per-lot search scans a smaller portion of the table on average.

Answer: C

Explanation:
The degradation comes from moving a large result set into the application server and walking it linearly once per lot. Pushing the selection and aggregation into the database and returning only the needed rows removes the volume from the loop, and a keyed access path replaces the repeated linear search. This addresses the actual cost driver under peak load.

Question: 2

CHALLENGE 1 — Stabilizing Inspection List Reads Under Peak Posting Load

After the team agrees to push the work down, a developer notes the select still lists every column of the results table although the list shows only a few fields and the aggregation needs a few more.

Which selection approach is most consistent with performant code pushdown here?

A. Select only the fields the list and aggregation need, so the database returns a narrow result set rather than unused columns.
B. Select all columns so the same internal table can be reused if the screen is extended later, avoiding a future second round trip.
C. Select all columns, then delete the unused ones from the internal table after the read to free memory before the matching runs.
D. Select only the key field, then re-read each remaining field individually inside the loop when a row is actually displayed.

Answer: A

Explanation:
Restricting the projection to the required fields lets the database transfer a narrow result set, which is the essence of pushing data-intensive work down and minimising data movement to the application server. It directly reduces the volume that the corrected logic handles at peak.

Question: 3

CHALLENGE 1 — Stabilizing Inspection List Reads Under Peak Posting Load
The matching needs to look up results by lot key many times while rendering. The developer must choose the internal-table kind for the looked-up table.
Given repeated keyed single-record lookups by a unique lot key, which internal-table choice is most appropriate?

A. Use a standard table with a linear search, since the data is read once and the table kind rarely changes the correctness here.
B. Use a standard table that is re-sorted before each lookup so the search always runs against freshly ordered data on each call.
C. Use a standard table with a manual index held in a parallel table that stores the row number for each key value seen so far.
D. Use a hashed table keyed on the unique lot key, giving near-constant-time access for the repeated single-record lookups by key.

Answer: D

Explanation:
For many repeated single-record lookups by a fully specified unique key, a hashed table provides near-constant-time access regardless of size, which is exactly the access profile here. It removes the per-lookup scaling that the linear walk imposed.

Question: 4

CHALLENGE 1 — Stabilizing Inspection List Reads Under Peak Posting Load

After the rewrite, the developer wants objective evidence that the change actually fixed the runtime cost before the hypercare gate, not just that the screen feels faster on a small test.
Which validation approach gives the most defensible evidence for this challenge?

A. Demonstrate the corrected screen once on a small developer dataset and record that the list rendered with no visible delay at all.
B. Profile the interaction under representative peak volume and confirm the database access pattern and runtime improved versus before.
C. Ask several inspectors whether the list subjectively feels faster during a normal shift and record their shared overall impression.
D. Compare the line count of the new code against the old version to confirm that the list logic was meaningfully simplified overall.

Answer: B

Explanation:
The gate is about behavior under late-shift volume, so profiling the interaction at representative peak load and comparing the database access pattern and runtime to the prior version produces objective, reproducible evidence. It directly targets the cost driver the challenge identified.

Question: 5

CHALLENGE 2 — Enforcing Plant Data Boundaries Across Every Read Path
A plant-quality role can open the central dashboard and read the other plant's nonconformance details through the list, even though one action method runs an imperative authorisation check. Reads reach the data through CDS views.
What is the most complete correction for the read exposure?

A. Define CDS access control on the entities behind every read path so the database filters returned rows by the user's plant.
B. Add the same imperative authorisation check at the start of the list handler so the list is guarded like the action method is.
C. Hide the other plant's rows in the consuming UI so inspectors no longer see records outside their own plant on the screen.
D. Remove the supplier-rating projection from the service binding so the dashboard can no longer aggregate across both plants.

Answer: A

Explanation:
The records are reached declaratively through CDS reads, so read protection belongs at that layer: CDS access control filters returned rows by the inspector's plant for every read path at once, including list and analytical access. It closes the exposure at the layer that governs all reads.
Topic: 2
Micro Skill Drill Exam

Question: 6

A logistics company on the ABAP environment runs a transactional app for shipment records built on a RAP business object. The business object defines a determination that fills a derived total weight whenever line items change, and a validation that blocks saving when the total exceeds the route limit. A developer is asked to add a background job that bulk-loads shipments from a partner feed. To keep it fast, the developer writes the job to insert rows directly into the underlying database tables of the business object. After the load, business users report that some bulk-loaded shipments show a blank total weight and a few breach the route limit, while shipments created through the app are always consistent. The developer confirms the same tables, fields, and data values are involved in both paths. The team needs the bulk load to produce shipments as consistent as those created interactively, without weakening or duplicating the rules already defined once on the business object. The correction must route the bulk creation through the layer that runs the business object's logic, so determinations and validations execute for loaded shipments exactly as they do for interactive ones.
Which approach makes the bulk-loaded shipments consistent with interactively created ones?

A. Keep the direct table inserts for speed and add a separate report that later recalculates total weight and re-checks the route limit for every loaded shipment afterwards
B. Keep the direct table inserts and duplicate the total-weight and route-limit rules inside the load job so the same logic runs in two places
C. Create the shipments through the business object using its entity manipulation language so the existing determinations and validations run during the load
D. Keep the direct table inserts but run the load inside the app's transaction so the business object notices the new rows and processes them

Answer: C

Explanation:
Creating shipments through the business object using its entity manipulation language drives the same determinations and validations that run for interactive creation, so loaded shipments get the derived total and obey the route limit by construction, with the rule defined once on the object.

Question: 7

A retail analytics team on the ABAP environment maintains a report that lists each store with its total sales for a period. The current implementation reads the list of stores into an internal table, then loops over the stores and, for each one, issues a separate database read to sum that store's sales. With a handful of stores the report was fine, but after the chain expanded to several thousand stores it became very slow and occasionally times out, even though the underlying tables are correctly indexed and the result figures are correct. A developer must make the report scale without changing what it shows. The team has confirmed the slowness grows with the number of stores, and that each loop pass triggers its own round trip to the database. The correction must move the heavy aggregation work to where the data lives instead of pulling rows into the application server and combining them there, so the database returns the per-store totals in as few requests as possible. The result must remain identical; only the way the totals are produced should change so the report stays responsive as the store count grows.

Which change makes the report scale while returning the same totals?

A. Replace the per-store reads with a single database query that groups sales by store and returns the totals, letting the database aggregate the data
B. Keep the loop but widen each per-store read to also select all sales columns, so that fewer follow-up reads are needed for each individual store afterwards
C. Keep the per-store reads but sort the store internal table first so the database can answer each repeated read more quickly during the loop
D. Keep the loop and buffer each store's result in a secondary-key internal table so repeated passes over the stores are faster to look up later

Answer: A

Explanation:
Issuing one set-based query that groups sales by store moves the aggregation into the database and returns all totals together, removing the per-store round trips that grow with the store count. The figures are unchanged because the same data is summed, just in one request.

Question: 8

A utilities provider on the ABAP environment is building several new custom tables that all store a "contract status" field. A developer types the status as a plain character field of fixed length directly on the first table and hardcodes the list of allowed values in the program that fills it. As more tables and programs are added, problems appear: one table accepts a status value the others reject, two programs disagree on the permitted codes, and a planned report cannot offer a consistent value help because each field is defined independently. The data values are valid in isolation, but there is no shared definition of what "contract status" means or which values are allowed. The developer must redefine the field so that every table and consuming program shares one definition of its technical type, its allowed value range, and its semantic meaning, and so that future tables reuse it instead of redeclaring it. The correction must centralise the field's type and permitted values in the dictionary so consistency and value help follow automatically wherever the field is used, rather than depending on each program to enforce the rules by hand.
How should the contract-status field be defined so all tables and programs share one consistent definition?

A. Keep the local character field but add the same check on allowed values into every program, so each consumer enforces an identical list independently wherever the field is used
B. Define a domain that fixes the technical type and the allowed value range, and a data element for its meaning, then reuse them on every table and field
C. Keep the local character field but document the allowed values in a comment so future developers can copy the same list whenever they add new tables
D. Define one shared structure that holds the status field and embed that structure into each table so the field layout is copied consistently across every table

Answer: B

Explanation:
A domain centralises the technical type and the fixed value range, while a data element carries the semantic meaning; reusing them on every table makes the type, allowed values, and value help consistent by definition wherever the field appears. This is the dictionary-level fix the situation needs.

Question: 9

A healthcare services company on the ABAP environment exposes patient billing data through a CDS view that several apps and reports consume. To stop unauthorized users from seeing other regions' data, a developer adds an authorization check in one report that reads the view, filtering rows by the user's assigned region. Testing through that report looks correct. Soon after, a data-protection review finds that a second app and an ad-hoc query tool read the same CDS view directly and return billing rows for every region to any user who can open them, because the region restriction lives only inside the one report. The view itself enforces nothing, so each new consumer would have to re-implement the same check, and any consumer that forgets it leaks data. The developer must restrict read access at the view so the rule applies to every consumer automatically, regardless of which app or tool reads it. The correction must attach the region restriction to the CDS entity itself so authorized scope is enforced once, centrally, rather than being repeated in each consuming program and missed by some.
How should read access to the patient billing data be restricted so every consumer is covered?

A. Move the region check into a shared helper routine and ask every current and future consumer of the view to call it before reading any billing rows from the entity
B. Add the same region authorization check into the second app and the query tool, so that all three currently known consumers now filter the billing rows by region
C. Define a CDS access control on the view so its read access is restricted by the user's region for every consumer that reads the entity
D. Restrict the underlying database tables by region so the view returns only permitted rows to whichever consumer happens to query it

Answer: C

Explanation:
A CDS access control attaches the region restriction to the entity itself, so the database returns only permitted rows to any consumer that reads the view. The rule is enforced once, centrally, and cannot be bypassed by a new reader.

Question: 10

A manufacturing customer on the ABAP environment has a pricing method in a shared class that occasionally returns the wrong discount for one combination of inputs reported by a key user. A developer can reproduce the bug manually and sees roughly where the logic is wrong. Leadership wants the fix today, but the same method is reused by several applications, and a previous "quick fix" to it reintroduced an old defect that was only noticed weeks later in production. The developer must correct the discount logic in a way that proves this specific case is handled and protects the method against future regressions, without relying on someone manually re-testing every consumer each time the method changes. The team wants the corrected behavior captured so the build itself will catch it if the

logic breaks again. The correction must add an automated check that pins the expected discount for the reported combination and then fix the logic so that check passes, giving repeatable evidence rather than a one-off manual confirmation that the case now works.
What is the most reliable way to fix the discount defect and guard against regressions?

A. Fix the logic directly and confirm the correct discount by running each affected application manually for the reported input combination before the change is shipped to users
B. Fix the logic directly and add detailed logging around the discount calculation, so that any future wrong discount can later be traced from the production logs by support
C. Fix the logic directly and schedule a manual regression review of every consuming application a few weeks after the change has reached production users
D. Write an ABAP Unit test that asserts the correct discount for the reported input combination, then fix the logic until that test and the existing tests pass

Answer: D

Explanation:
An ABAP Unit test that pins the expected discount for the reported combination turns the fix into repeatable evidence and runs with every build, so the specific case is verified and a future change that breaks it is caught automatically.

Thank You for Trying Our Product

Special 16 USD Discount Coupon: NSZUBG3X

Email: support@examsempire.com

Check our Customer Testimonials and ratings available on every product page.

Visit our website.

https://examsempire.com/