PassQueen
a.k.a dumpsqueen.com

# Appian Associate Developer

## Appian ACD101

**Version Demo**

**Total Demo Questions: 15**

**Total Premium Questions: 158**

**Buy Premium PDF**

https://passqueen.com

support@passqueen.com

# Topic Break Down

| Topic | No. of Questions |
|---|---|
| Topic 1, Application Design | 29 |
| Topic 2, Data Management | 33 |
| Topic 3, User Interfaces | 45 |
| Topic 4, Process Models | 37 |
| Topic 5, Security | 8 |
| Topic 6, Deployment | 4 |
| Topic 7, Mix Questions | 2 |
| Total | 158 |

QUESTION NO: 1

An expression rule calculates a processing fee. Requests below $1,000 have a fee of 0%, while requests of $1,000 or more have a fee of 5%.

Which two test cases provide the most useful boundary coverage for this rule? (Choose two.)

**A.** A request amount of $999 with an expected fee of 0%

**B.** A request amount of $1,000 with an expected fee of 5%

**C.** A request amount of $100 with an expected fee of 0%

**D.** A request amount of $10,000 with an expected fee of 5%

**ANSWER: A B**

**Explanation:**

The most useful boundary tests use values immediately below and exactly at the threshold. A request amount of $999 should return a 0% fee, while an amount of $1,000 should return a 5% fee. These tests verify both sides of the decision boundary and confirm that the threshold is included in the 5% condition. Values far from the threshold provide less evidence that the boundary comparison is correctly implemented.

## QUESTION NO: 2

Which use case is appropriate as a record list action?

**A.** To delete a record

**B.** To create a new record of that record type

**C.** To update an existing record

**ANSWER: B**

**Explanation:**

To create a new record of that record type is the appropriate use case for a record list action. In Appian, record list actions are displayed in the actions area of a record type's list view, where users are looking across the entire collection of records rather than working in the context of one selected record. This placement makes a creation workflow intuitive: a user can open the record list, choose the action, complete a form, and create a new item that belongs to that record type. For example, from a Customer record list, an authorized user could select a "New Customer" action to start an interface or process model that captures customer information and writes the resulting record. Record actions are configured on the record type and can be made available to users through appropriate security and action configuration. Because the action begins from the list-level context, it does not need an existing record identifier as input; instead, it initiates creation of a new record instance. Appian's record documentation describes configuring actions for record types and distinguishes actions available from record lists from those used in an individual record context. See Appian Record Actions documentation and Appian Records documentation.

## QUESTION NO: 3

Which two groups can be set withinApplication Properties? (Choose two.)

**A.** Administrators Groups

**B.** Developers Groups

**C.** Users Groups

**D.** Designers Groups

**ANSWER: A C D**

**Explanation:**

Application Properties supports configuring application-level security through administrator, designer, and user groups. Therefore, “Administrators Groups,” “Users Groups,” and “Designers Groups” are all valid group types that can be set for an application. Administrators have broad authority to manage the application and its security settings. Designers are granted access needed to create and maintain application objects, while users are granted access to use the application’s interfaces and functionality. These group assignments provide a structured way to control administration, development, and runtime access without assigning permissions individually to every person. Because the question asks for only two groups but includes three valid Application Properties group categories, all three valid choices should be marked correct. Appian’s application-management documentation describes application security and the use of administrator, designer, and user access groups; see Appian Documentation: Managing Applications and Appian Documentation: Application Security.

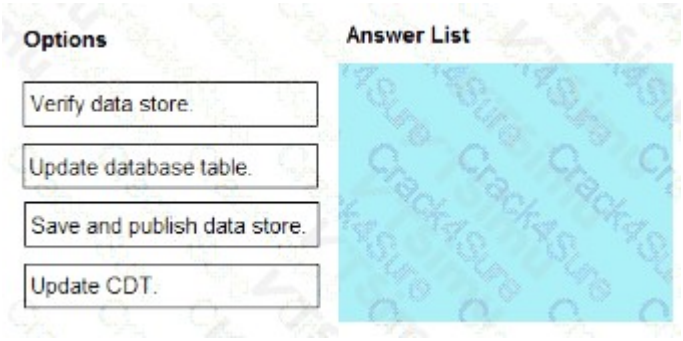
## QUESTION NO: 4 - (DRAG DROP)

-

You need to implement a new column on an existing database table. This table, its Custom Data Types (CDT), and data store entity are currently in use by an Appian application.

How should you implement this change?

Note: To answer, move all steps from the Options list to the Answer List area and arrange them in the correct order.




**ANSWER:**


**Explanation:**

Adding a column to a table that is already used through a CDT and data store entity is a coordinated schema change. The physical database table should be changed first, because it is the actual persistence layer. Creating the column there ensures that the storage location exists before Appian begins using a CDT field that maps to it. The new column should use a database type that is compatible with the type planned for the CDT field.

Next, update the CDT with a field representing that new column. The CDT is Appian’s typed representation of the record structure, so the field name and type must correspond appropriately to the database schema. This keeps Appian’s data model aligned with the table and allows interfaces, records, processes, and integrations to read or write the additional value consistently.

After both schemas have been updated, verify the data store. Verification checks the data store configuration and its entity mapping against the configured data source. Performing this check before publication is important because it confirms that Appian can recognize the revised structure and surfaces mapping issues while the change can still be corrected safely.

Finally, save and publish the data store. Publishing makes the validated data store definition available to the application, allowing the existing data store entity to operate with the updated table and CDT structure. Appian's documentation on data stores and custom data types describes how these objects provide the mapping between Appian data structures and relational database tables.

## QUESTION NO: 5

An expression rule determines whether an applicant is eligible for a program. Applicants must be at least 18 years old.

Which two test cases provide the most useful boundary coverage for this rule? (Choose two.)

**A.** An applicant aged 17 returns ineligible.

**B.** An applicant aged 18 returns eligible.

**C.** The expression rule has a descriptive object name.

**D.** The expression rule is included in a deployment package.

## ANSWER: A B

**Explanation:**

Testing an applicant aged **17** confirms that the rule rejects a value immediately below the eligibility threshold. Testing an applicant aged **18** confirms that the rule accepts the minimum valid value. Together, these cases verify the decision boundary where the rule changes from ineligible to eligible.

Testing only a descriptive label or the deployment status of the rule does not evaluate the eligibility logic. Boundary-focused test cases are especially valuable for rules that compare input values with a defined threshold.

## QUESTION NO: 6 - (SIMULATION)

Match each scenario to the correct relationship type in your data model design.

Note: Each relationship type will be used once. To change your responses, you may deselect your response by clicking the blank space at the top of the selection list.

## ANSWER: See the explanation for the answer

**Explanation:**

**Correct matches:**

**Product with basic fields and an additional detailed technical product description that is regularly updated: One-to-one**.
**Many employees working on multiple projects: Many-to-many**.
**An order with order items: One-to-many**.

A product has one corresponding detailed-description record, even if that detail record is updated often. Employees can be assigned to several projects and each project can include several employees. One order can contain multiple order-item records, while each order item belongs to one order.

## QUESTION NO: 7

You are creating a form used to order a pizza. You use a radio button component for the selection.

The pizza selection labels include a list of toppings. You do not want the selection labels to be truncated.

Which layout should you choose?

**A.** Compact

**B.** Grid

**C.** Stacked

**ANSWER: C**

**Explanation:**

Stacked is the appropriate radio button layout when each pizza choice has a long descriptive label, such as a name followed by multiple toppings. In a stacked presentation, each radio button choice is displayed on its own row, allowing the label to use the available horizontal space and wrap onto additional lines when needed. This makes the complete topping list readable without forcing users to infer text that is not visible. It also supports a clear scan pattern for a food-ordering form: users can read one pizza description at a time, compare selections, and choose the desired item confidently. The layout is especially useful on narrower form regions and smaller screens, where preserving the full content of a choice label is more important than placing several choices side by side. Appian documents the available display layouts and behavior for the Radio Button Component in its component reference. For broader guidance on choosing components and designing accessible, understandable interfaces, consult Appian's interface design guidance. See Appian Radio Button Component documentation and Appian SAIL Design Guidance.

**QUESTION NO: 8**

A loan application uses different approval limits depending on the applicant's risk rating and requested amount.

The business policy team must be able to update these approval limits without changing the process model that calls the logic.

Which Appian object should you use?

**A.** Decision

**B.** Process Report

**C.** Data Store

**ANSWER: A**

**Explanation:**

A **Decision** object is appropriate because it separates configurable business decision logic from the process model that uses it. The process model can pass the applicant's risk rating and requested amount to the decision and use its result for routing, while authorized designers can maintain the approval policy in the decision object. An expression rule can also return a result, but it is better suited to reusable expression logic that is maintained in code rather than a business decision table.

**QUESTION NO: 9 - (HOTSPOT)**

Match each node to the correct description for the node.

Note: Each description will be used once. To change your responses, you may deselect your response by clicking the blank space at the top of the selection list.

## Answer Area

Timer Event

- Can accept multiple paths: when all paths arrive, all outgoing flows are executed at the same time.
- Can be added to a flow or attached to a smart service to trigger an Exception Flow or an Escalation.
- Can run automated activities.
- Can accept one incoming path and select one outgoing path.

XOR Gateway

- Can accept multiple paths: when all paths arrive, all outgoing flows are executed at the same time.
- Can be added to a flow or attached to a smart service to trigger an Exception Flow or an Escalation.
- Can run automated activities.
- Can accept one incoming path and select one outgoing path.

Script Task

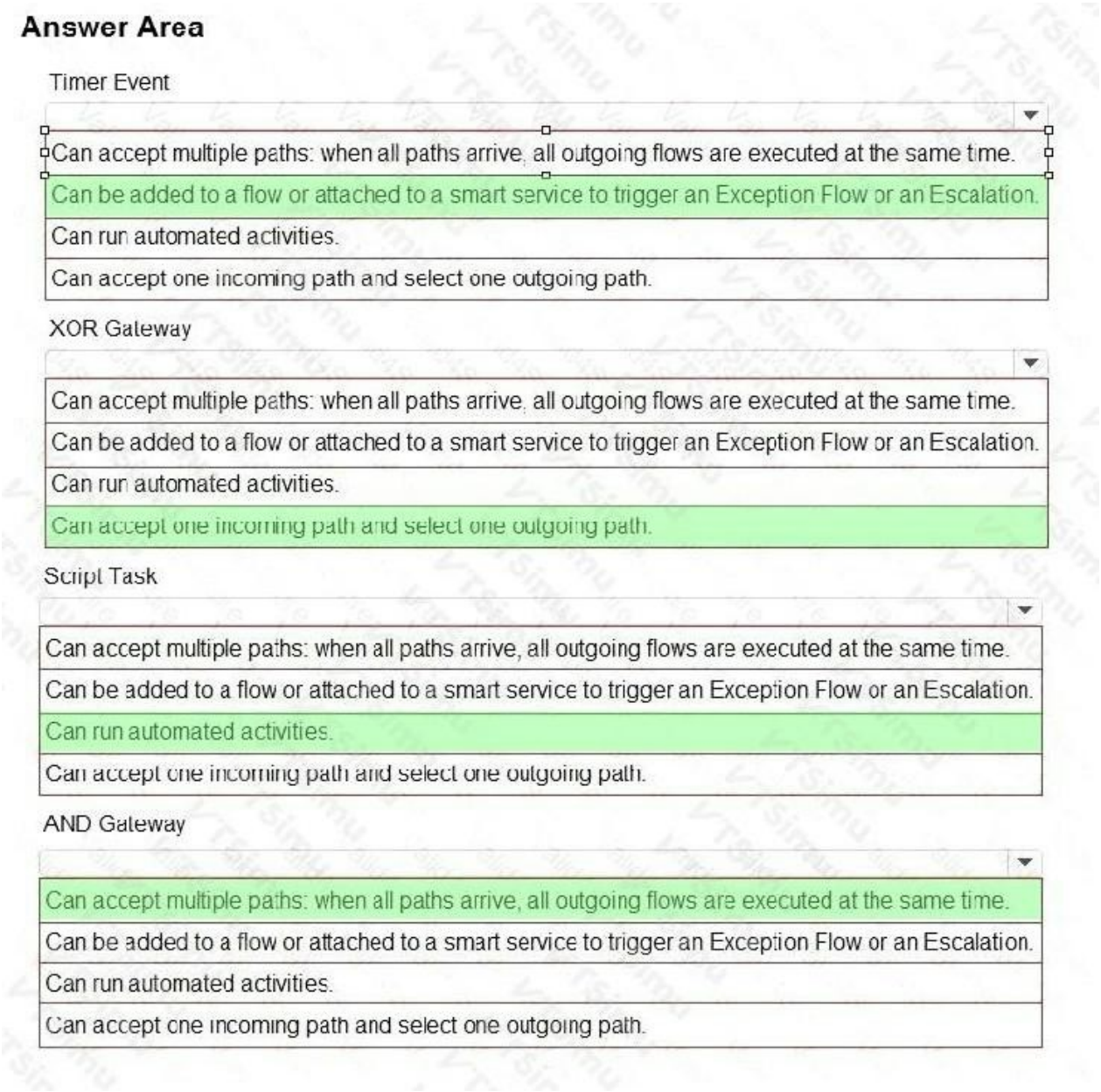
- Can accept multiple paths: when all paths arrive, all outgoing flows are executed at the same time.
- Can be added to a flow or attached to a smart service to trigger an Exception Flow or an Escalation.
- Can run automated activities.
- Can accept one incoming path and select one outgoing path.

AND Gateway

- Can accept multiple paths: when all paths arrive, all outgoing flows are executed at the same time.
- Can be added to a flow or attached to a smart service to trigger an Exception Flow or an Escalation.
- Can run automated activities.
- Can accept one incoming path and select one outgoing path.

**ANSWER:**

**Answer Area**




**Explanation:**

The correct matches follow the intended behavior of these Appian process-model nodes. A Timer Event is used for time-based process behavior. It can sit directly on a process flow, where it holds progression until its configured timing condition is met, and it can also be attached to a smart service. When attached in that context, it supports time-driven exception handling or escalation behavior. This makes the description about adding the node to a flow or attaching it to a smart service to trigger an Exception Flow or Escalation the appropriate Timer Event description.

An XOR Gateway is the process decision point. It evaluates the flow situation and routes the process along one selected outgoing path, which is why the description stating that it can accept one incoming path and select one outgoing path belongs with this gateway. A Script Task represents automated work in the process model: it executes configured expressions or scripts without requiring a person to act through a task interface. Accordingly, it matches the description that it can run automated activities.

An AND Gateway controls parallel flow. At a synchronization point, it waits until all relevant incoming paths have arrived; once they have, it proceeds by executing each outgoing flow at the same time. This is the behavior described by the multiple-paths and all-outgoing-flows statement. These node behaviors are consistent with Appian's process-modeling concepts and BPMN-style flow control. See the Appian process modeling documentation and the Process Modeler objects reference for the relevant process-node concepts.

## QUESTION NO: 10 - (SIMULATION)

Review the following variables:

Match each expression rule to the expected output.

*Note: Each output will be used once or not at all. To change your responses, you may*deselect*your response by clicking the blank space at the top of the selection list.*

**ANSWER: See the explanation for the answer**

**Explanation:**

Match the expression rules as follows:

```
difference(loc!groupA, loc!groupB) → {8, 10}
union(loc!groupA, loc!groupB) → {3, 6, 8, 9, 10, 12}
intersection(loc!groupA, loc!groupB) → {6, 12}
```

`difference()` retains values found only in the first list; `union()` combines both lists without duplicates; and `intersection()` retains the values shared by both lists.

## QUESTION NO: 11

According to Appian user interface best practices, which data type should be right-aligned on grids? Assume the use of the English language.

**A.** Numbers

**B.** Text

**C.** Icons

**ANSWER: A**

**Explanation:**

**Numbers** should be right-aligned in grids when designing for English-language users. This convention helps users scan, compare, and interpret numeric values efficiently, particularly where values differ in digit length. Right alignment places the ones digits, decimal points, and digit groupings in consistent visual positions, making magnitude comparisons across rows faster and reducing cognitive effort. For example, monetary amounts, quantities, percentages, and other measurements are easier to review when their values share a common right edge. Appian's user-experience guidance follows familiar table-reading conventions: textual labels are generally read from the left, while numerical data is positioned to support rapid comparison. Applying the alignment consistently within a grid also improves readability and creates a predictable interface for users. Alignment should be selected based on the meaning and format of the displayed value rather than on the underlying storage type alone; a value presented as a numerical measure should use right alignment. See Appian's UX design guidance and its SAIL component documentation for guidance on building clear, usable interfaces.

## QUESTION NO: 12

You write an expression that checks the validity of an email address.

Which three scenarios should you configure as test cases? (Choose three.)

**A.** An invalid email address that is missing the @-character: " john.doeexample.com " .

**B.** An invalid email address: null.

**C.** A valid email address: " jane.doe@example.com " .

**D.** The mail server is unavailable.

**ANSWER: A B C**

**Explanation:**

“An invalid email address that is missing the @-character: " john.doeexample.com " .”, “An invalid email address: null.”, and “A valid email address: " jane.doe@example.com " .” are appropriate test cases because they exercise the essential expected outcomes of an email-validation expression. A validation rule should confirm that a normally formatted email address is accepted, while malformed input is rejected. Testing an address without the required @ symbol specifically verifies that the expression recognizes a fundamental structural requirement of an email address. Testing `null` is equally important because form data and rule inputs can be absent; the expression should return a predictable result without producing an evaluation error. Together, these cases cover a valid input, an invalidly formatted string, and an empty or missing value, providing useful baseline coverage for the expression’s behavior. Appian provides the `a!isEmailAddress()` function for validating whether text is formatted as an email address, and expression rules can be tested with defined inputs and expected outputs. See Appian’s documentation for a!isEmailAddress() and expression rules.

## QUESTION NO: 13

Which two scenarios are ideal for using Appian Portals? (Choose two.)

**A.** A manager wants to obtain a view of their team ' s performance.

**B.** A retail customer wants to conduct a public survey for their recently launched product.

**C.** An employee who does not have an account wants to register for their company ' s vehicle fleet a management system.

**D.** A user needs to submit support requests when they are using their mobile device in areas with bad network coverage.

**ANSWER: B C**

**Explanation:**

Appian Portals are appropriate when people need to start or participate in an Appian process without having an Appian user account. A public survey for a recently launched product is a strong portal use case because a retail customer can publish a URL that respondents access directly in a browser. The survey can collect responses through an Appian interface and initiate any associated workflow without requiring each respondent to be provisioned as an Appian user.

Registering for a company’s vehicle fleet management system is also suitable when the employee does not have an Appian account. A portal can expose a focused registration experience to that person, capture the necessary information, and start the appropriate Appian process. Portal design should keep unauthenticated access in mind: expose only the data and actions needed for the public interaction, and use process and data-security design appropriate to the use case. Appian describes portals as public, externally accessible application experiences for users who do not need Appian accounts. See Appian Portals documentation and Appian portal security documentation.

## QUESTION NO: 14

Your application calls the same authenticated external REST service from several integrations. The service URL and authentication configuration must be maintained consistently and should not be repeated in every integration.

What should you configure?

**A.** A Connected System referenced by the integrations

**B.** A separate Web API for each external REST operation

**C.** A process variable containing the service credentials

**D.** A record relationship between the integrations

**ANSWER: A**

**Explanation:**

Configure a **Connected System** and use it from the integrations. A connected system stores reusable connection details, such as the service base URL and authentication configuration. Individual integration objects can then define the specific service operation while relying on the shared connection configuration.

Duplicating authentication settings in every integration creates unnecessary maintenance work and increases the risk of inconsistent configuration. A Web API exposes an endpoint for external callers and is not used to configure Appian's connection to an external REST service.

## QUESTION NO: 15

You need to start a process using an email trigger.

How should you configure this process model using the Process Model Properties dialog?

**A.** Go to File > Properties > Alerts. Configure the Custom Alert settings.

**B.** Go to File > Properties Set the proper Process Display Name

**C.** Go to File > Properties. Select the Public Events checkbox to allow anyone to fire triggers.

## ANSWER: C

**Explanation:**

**Go to File > Properties. Select the Public Events checkbox to allow anyone to fire triggers.** is correct because an email-triggered process relies on an external event reaching the process model. Enabling Public Events permits the model's public event mechanisms to be fired without requiring the sender to be an authenticated Appian user with direct access to the process model. This is the appropriate process-model-level setting when an inbound email is intended to initiate or advance a process through an email trigger or other externally initiated event.

The setting is configured in the Process Model Properties dialog, so it is applied to the model that receives the event rather than being configured as a user-facing alert or display-label setting. Once public events are enabled, the model can accept the relevant event from the email integration while the process design determines what should happen after the event is received. This configuration supports email-based automation while preserving the intended event-driven process behavior. Appian's process-model documentation describes the properties available for configuring process behavior, and its event documentation provides the broader context for event-driven process execution. See Appian Process Model Properties and Appian Process Events.