

01	A	B	C	D	E
02	A	B	C	D	E
03	A	B	C	D	E
04	A	B	C	D	E
05	A	B	C	D	E
06	A	B	C	D	E
07	A	B	C	D	E
08	A	B	C	D	E
09	A	B	C	D	E
10	A	B	C	D	E
11	A	B	C	D	E
12	A	B	C	D	E
13	A	B	C	D	E

14	A	B	C	D	E
15	A	B	C	D	E
16	A	B	C	D	E
17	A	B	C	D	E
18	A	B	C	D	E
19	A	B	C	D	E
20	A	B	C	D	E
21	A	B	C	D	E
22	A	B	C	D	E
23	A	B	C	D	E
24	A	B	C	D	E
25	A	B	C	D	E
26	A	B	C	D	E

27	A	B	C	D	E
28	A	B	C	D	E
29	A	B	C	D	E
30	A	B	C	D	E
31	A	B	C	D	E
32	A	B	C	D	E
33	A	B	C	D	E
34	A	B	C	D	E
35	A	B	C	D	E
36	A	B	C	D	E
37	A	B	C	D	E
38	A	B	C	D	E
39	A	B	C	D	E

PRACTICE QUESTIONS

OutSystems

AW11J

Associate Traditional Web Developer O11 (JA)

EDITION

Demo

QUESTIONS

9

FORMAT

Limited Edition PDF

Before you begin

What this is

9 practice questions for OutSystems **AW11J**, each with its answer key and an explanation. Answer a question before you read its key — the explanation is worth more once you have committed.

If something looks wrong

Send us three things and we will check it:

- the exam code, **AW11J**
- the question number
- the email address on your order

Write to **support@mometrix.net**. We reply within one business day.

QUESTION 1

MULTIPLE CHOICE

A Traditional Web application needs to display a list of active Customers along with the total number of Orders each one has placed. Which two actions are required to build this in an Aggregate? (Choose two.)

- ☒ **A** Add the Customers entity to the Aggregate and select only active customers using a filter.
- ☒ **B** Group the Aggregate by Customer.Id so that counts are computed per customer.
- ☐ **C** Create a separate Aggregate for each customer at runtime using a For Each loop.
- ☐ **D** Use a calculated attribute on the Aggregate to count the related Orders.

ANSWER A - B

WHY To aggregate related data, the Aggregate must fetch from the Customers entity (A) and group by Customer.Id so that per-customer counts can be computed (B). A calculated attribute can then expose the count of related Orders. Creating a separate Aggregate per customer (C) is wasteful and not how Aggregates work, and adding a calculated attribute without grouping (D) would not produce a count per customer.

QUESTION 2

MULTIPLE CHOICE

Which two statements about using SQL nodes inside OutSystems Traditional Web applications are true? (Choose two.)

- ☒ **A** SQL nodes can be used in actions to execute custom SQL against the database of an entity.
- ☐ **B** SQL nodes are typically edited and managed in Service Studio using the SQL toolbox.
- ☐ **C** SQL nodes replace the need for Aggregates in every screen preparation.
- ☒ **D** The output parameter list of a SQL node can be dragged onto the action flow like any other OutSystems structure.

ANSWER A - D

WHY SQL nodes allow developers to run custom SQL statements on the entity's database (A), and once defined they expose output parameters that behave like records lists that can be referenced from other elements (D). They are managed in Service Studio, but the statement that SQL replaces Aggregates entirely (C) is false — Aggregates remain the preferred, model-driven approach for most data fetching. Option B mentions the toolbox in a generic way that is not how SQL nodes are described in the official curriculum.

QUESTION 3

SINGLE CHOICE

In a Traditional Web screen, a Form widget submits its values to the destination defined in which property?

- ☒ **A** Destination
- ☐ **B** Method
- ☐ **C** Validation
- ☐ **D** Source Record

ANSWER A

WHY The Form widget's 'Destination' property defines the screen or endpoint to which the form data is submitted when the user triggers the submit action. 'Method' controls the HTTP verb, 'Validation' controls validation messages, and 'Source Record' binds the form to an existing record.

QUESTION 4 SINGLE CHOICE

A developer needs to execute a different set of actions depending on the value of a Status attribute that can be 'New', 'Approved', or 'Rejected'. Which is the most appropriate widget to use?

- ☐ A If widget with three branches
- ☒ B Switch widget based on the Status value
- ☐ C For Each widget iterating over the status list
- ☐ D Exception handler with custom status messages

ANSWER B

WHY The Switch widget is designed to branch execution based on the value of a single variable or attribute, making it the natural choice for handling multiple discrete values such as 'New', 'Approved', or 'Rejected'. An If widget is best for boolean conditions, For Each is for iteration, and Exception handlers handle errors rather than business logic flow.

QUESTION 5 SINGLE CHOICE

Which statement about the Assign node in a Traditional Web screen action is correct?

- ☒ A The Assign node is used to copy or update values of local variables and record attributes.
- ☐ B The Assign node permanently saves the modified record to the database.
- ☐ C The Assign node is only available inside Server Actions.
- ☐ D The Assign node cannot be used to assign values between two list elements.

ANSWER A

WHY The Assign node sets or copies values of variables and record attributes in memory. It does not by itself persist data to the database — that requires a Create or Update (CreateOrUpdate) action. It is available in both Screen Actions and Server Actions, and it can assign values between list elements through index-based expressions.

QUESTION 6

SINGLE CHOICE

In an Entity, what does defining a relationship of type '1 to Many' from Customer to Order imply?

- ☒ **A** A Customer can have many Orders, and each Order belongs to exactly one Customer.
- ☐ **B** A Customer can have many Orders, and each Order can belong to many Customers.
- ☐ **C** Each Order must exist independently of any Customer.
- ☐ **D** The Customer identifier is optional on every Order record.

ANSWER A

WHY A '1 to Many' relationship in OutSystems means the parent entity (Customer) can be associated with many child records (Orders), while each child record is linked to exactly one parent. Option B describes a Many-to-Many, and options C and D are inconsistent with the 1-to-Many cardinality.

QUESTION 7

SINGLE CHOICE

Which widget should be used to display a Table Records-based grid that supports built-in sorting, pagination, and selection of rows on a Traditional Web screen?

- ☒ **A** TableRecords
- ☐ **B** List widget
- ☐ **C** If widget
- ☐ **D** Container with a For Each loop

ANSWER A

WHY The TableRecords widget is specifically designed to render a tabular list bound to an Aggregate or source list and provides built-in features such as sorting, pagination, filtering, and row selection. A plain List widget renders a repeating layout but lacks the integrated table features. The If widget is for conditional rendering, and manually building a container with a For Each loop loses the built-in table functionality.

QUESTION 8

SINGLE CHOICE

A developer wants only users belonging to the 'Manager' role to be able to access a specific Screen. Which is the correct way to enforce this in a Traditional Web application?

- ☒ **A** Add the 'Manager' role to the screen's Allowed Roles property in Service Studio.
- ☐ **B** Add an If widget at the top of the screen flow that checks IsManager at runtime.
- ☐ **C** Add the user to the role at the database level manually.
- ☐ **D** Set the screen's Anonymous property to True and rely on a Server Action check.

ANSWER A

WHY OutSystems provides screen-level security through the 'Allowed Roles' property of a Screen. Listing the 'Manager' role there is the standard, declarative way to restrict access. Performing a runtime check with an If widget (B) is not recommended as the primary mechanism because the screen still loads; manual database manipulation (C) is unsupported; and using Anonymous with a runtime check (D) is not the documented pattern.

QUESTION 9

SINGLE CHOICE

When validating user input on a Traditional Web screen, where are mandatory-field validations performed by default in OutSystems?

- ☐ **A** Only on the client side using JavaScript.
- ☒ **B** On both the client side and the server side when the form is submitted.
- ☐ **C** Only in the database triggers.
- ☐ **D** Only on the server side after the screen is rendered.

ANSWER B

WHY OutSystems automatically generates client-side validation messages for mandatory fields and required formats, and it re-validates the same rules on the server side when the form is submitted, preventing bypass via disabled JavaScript. Validations do not run only in the database (C) nor only on the client (A) or only on the server (D).