2.10 Bedingungen, Datenmapping und Verzweigungen*

Automatisierte Workflows müssen Daten prüfen, verändern und abhängig vom Ergebnis unterschiedliche Wege ausführen.

Dafür werden verwendet:

- Bedingungen
- Datenmapping
- Verzweigungen
- Filter
- Standardwerte
- Schleifen
- Zusammenführungen

> Bedingungen entscheiden über den Ablauf. Datenmapping passt Informationen an das Zielsystem an.

Lernziele

Nach dieser Seite solltest du erklären können:

- wie Bedingungen in Workflows funktionieren
- was If- und Switch-Verzweigungen unterscheiden
- was Datenmapping bedeutet
- wie Felder umbenannt und neu aufgebaut werden
- wie fehlende Werte behandelt werden
- wie mehrere Datensätze verarbeitet werden
- wie Fehler durch falsche Datentypen vermieden werden

Bedingung

Eine Bedingung prüft, ob eine Aussage wahr oder falsch ist.

Beispiel:

> Ist der Benutzer aktiv?

Eine Bedingung besitzt meistens zwei mögliche Ergebnisse:

- wahr
- falsch

Typische Prüfungen

Prüfung	Beispiel
gleich	Abteilung ist `IT`
ungleich	Status ist nicht `inaktiv`
größer	Anzahl ist größer als 10
kleiner	Preis ist kleiner als 100
enthält	Text enthält `Fehler`
beginnt mit	E-Mail beginnt mit `support`
Feld vorhanden	E-Mail-Adresse existiert
Feld leer	Telefonnummer ist leer
Boolean	`active` ist `true`
Datum	Startdatum liegt in der Zukunft

If-Verzweigung

Eine If-Verzweigung eignet sich für eine Entscheidung mit zwei möglichen Wegen.

Beispiel:

```
Benutzer suchen  Fehler melden
```

Weitere Beispiele:

- Benutzer aktiv oder inaktiv
- Pflichtfeld vorhanden oder nicht vorhanden
- Statuscode erfolgreich oder fehlerhaft
- Datei größer oder kleiner als Grenzwert

> Eine If-Verzweigung unterscheidet normalerweise zwischen wahr und falsch.

Switch-Verzweigung

Eine Switch-Verzweigung eignet sich für mehrere mögliche Fälle.

Beispiel:

```
Abteilung prüfen
  /      |       \
 v       v        v
 IT    Vertrieb   Buchhaltung
 |       |        |
 v       v        v
IT-Gruppe CRM-Gruppe Finanz-Gruppe
```

Typische Anwendungen:

- verschiedene Abteilungen
- mehrere Statuswerte
- unterschiedliche Prioritäten
- verschiedene Dateitypen
- unterschiedliche Fehlercodes

If und Switch im Vergleich

Merkmal	If	Switch
Anzahl der Wege	meistens zwei	mehrere
typische Prüfung	wahr oder falsch	mehrere mögliche Werte
Beispiel	Benutzer aktiv?	Welche Abteilung?

Merkmal	If	Switch
Übersichtlichkeit	gut bei einfachen Prüfungen	gut bei vielen Fällen

Merksatz:

> If eignet sich für Ja-Nein-Entscheidungen. Switch eignet sich für mehrere feste Fälle.

Mehrere Bedingungen

Mehrere Bedingungen können miteinander verbunden werden.

UND-Bedingung

Alle Bedingungen müssen erfüllt sein.

Beispiel:

- Abteilung ist IT
- Benutzer ist aktiv
- E-Mail-Adresse ist vorhanden

Darstellung:

```
department = IT
    UND
active = true
    UND
email vorhanden
```

Nur wenn alle Bedingungen erfüllt sind, wird der gewünschte Weg ausgeführt.

ODER-Bedingung

Mindestens eine Bedingung muss erfüllt sein.

Beispiel:

- Status ist `fehlerhaft`
- oder Status ist `abgebrochen`

Darstellung:

```
status = fehlerhaft
    ODER
status = abgebrochen
```

UND und ODER

Verknüpfung	Bedeutung
UND	alle Bedingungen müssen wahr sein
ODER	mindestens eine Bedingung muss wahr sein

Beispiel:

> Benutzer ist aktiv UND gehört zur Abteilung IT.

Beispiel:

> Fehlercode ist 500 ODER 503.

Datenmapping

Datenmapping bedeutet:

> Felder eines Quellsystems werden passenden Feldern eines Zielsystems zugeordnet.

Beispiel:

Quellsystem	Zielsystem
`firstName`	`givenName`
`lastName`	`familyName`
`mail`	`primaryEmail`
`department`	`organization.department`

Quellsystem:

```
{
 "firstName": "Max",
 "lastName": "Mustermann",
 "mail": "max.mustermann@example.com"
}
```

Zielsystem:

```
{
 "givenName": "Max",
 "familyName": "Mustermann",
 "primaryEmail": "max.mustermann@example.com"
}
```

Die Werte bleiben gleich, aber die Feldnamen werden angepasst.

Felder umbenennen

Eingangsdaten:

```
{
 "mail": "max.mustermann@example.com"
}
```

Ausgangsdaten:

```
{
 "primaryEmail": "max.mustermann@example.com"
}
```

Der Inhalt bleibt gleich. Nur der Schlüssel wird verändert.

Felder zusammenführen

Mehrere Werte können zu einem neuen Feld kombiniert werden.

Eingangsdaten:

```
{
 "firstName": "Max",
 "lastName": "Mustermann"
}
```

Ausgangsdaten:

```
{
 "fullName": "Max Mustermann"
}
```

In n8n kann dafür beispielsweise eine Expression verwendet werden:

```
{{$json.firstName + " " + $json.lastName}}
```

Felder aufteilen

Ein bestehender Wert kann in mehrere Felder getrennt werden.

Eingangsdaten:

```
{
 "fullName": "Max Mustermann"
}
```

Mögliche Ausgangsdaten:

```
{
 "firstName": "Max",
 "lastName": "Mustermann"
}
```

Diese Aufteilung ist nur zuverlässig, wenn das Format eindeutig ist.

Bei mehrteiligen Namen können zusätzliche Regeln notwendig sein.

Standardwerte

Ein Standardwert wird verwendet, wenn kein Wert vorhanden ist.

Beispiel:

```
department = nicht vorhanden
```

Standardwert:

```
"Allgemein"
```

Mögliche Expression:

```
{{$json.department || "Allgemein"}}
```

Ergebnis:

- vorhandene Abteilung wird übernommen
- fehlende Abteilung wird durch `Allgemein` ersetzt

Standardwerte dürfen Pflichtprüfungen nicht unkontrolliert ersetzen.

Fehlendes Feld

Beispiel:

```
{
 "name": "Max Mustermann"
}
```

Das Feld `department` fehlt vollständig.

Ein Workflow sollte entscheiden:

- Standardwert verwenden
- Verarbeitung stoppen
- Fehler melden
- manuelle Prüfung anfordern

Leerer Wert

Beispiel:

```
{
  "department": ""
}
```

Das Feld ist vorhanden, enthält aber keinen Text.

Ein leerer Wert ist nicht automatisch dasselbe wie ein fehlendes Feld.

Null-Wert

Beispiel:

```
{
  "department": null
}
```

`null` bedeutet:

> Für dieses Feld ist kein Wert vorhanden.

Unterschied:

Zustand	Beispiel
Feld fehlt	kein Schlüssel vorhanden
leerer Text	`""`
Null-Wert	`null`
Zahl null	`0`
falsch	`false`

Diese Werte dürfen nicht miteinander verwechselt werden.

Datentypen prüfen

Ein Zielsystem kann bestimmte Datentypen erwarten.

Beispiel:

Erwartet:

```
{
  "active": true
}
```

Falsch:

```
{
  "active": "true"
}
```

`true` ist ein Boolean.

`"true"` ist ein String.

Weitere Beispiele:

Erwartet	Falsch
Zahl `15`	Text `"15"`
Boolean `false`	Text `"false"`
Array `["IT"]`	Text `"IT"`
Objekt `{}`	Array `[]`

Datentypen umwandeln

Ein Wert kann vor der Verarbeitung umgewandelt werden.

Beispiele:

Ausgangswert	Zielwert
`"15"`	`15`
`"true"`	`true`
`"2026-08-03"`	Datumswert
`15`	`"15"`

Eine Umwandlung sollte nur erfolgen, wenn das Eingabeformat eindeutig ist.

Filter

Ein Filter lässt nur Datensätze weiter, die bestimmte Bedingungen erfüllen.

Beispiel:

Eingangsdaten:

```
[
  {
    "name": "Max Mustermann",
    "active": true
  },
  {
    "name": "Anna Beispiel",
    "active": false
  }
]
```

Filter:

> Nur aktive Benutzer weiterverarbeiten.

Ergebnis:

```
[
  {
    "name": "Max Mustermann",
    "active": true
  }
]
```

Mehrere Datensätze

n8n verarbeitet häufig mehrere Datensätze als einzelne Items.

Beispiel:

```
[
  {
    "id": 1,
    "name": "Max Mustermann"
```

```
  },
  {
    "id": 2,
    "name": "Anna Beispiel"
  }
]
```

Jeder Datensatz kann einzeln verarbeitet werden.

Mögliche Aktionen:

- für jeden Benutzer ein Konto prüfen
- für jeden Kunden eine Aufgabe erstellen
- für jede Datei einen Speicherprozess starten

Schleife

Eine Schleife wiederholt eine Verarbeitung für mehrere Datensätze.

Beispiel:

```
Benutzerliste
    |
    v
Benutzer 1 verarbeiten
    |
    v
Benutzer 2 verarbeiten
    |
    v
Benutzer 3 verarbeiten
```

n8n verarbeitet viele Items automatisch nacheinander oder gesammelt.

Bei großen Datenmengen können Teilmengen sinnvoll sein.

Batch-Verarbeitung

Batch bedeutet:

> Mehrere Datensätze werden in begrenzten Gruppen verarbeitet.

Beispiel:

- 1.000 Benutzer vorhanden
- Verarbeitung in Gruppen zu je 50 Benutzern

Vorteile:

- geringere Systemlast
- weniger Speicherverbrauch
- Rate Limits können besser eingehalten werden
- Fehler lassen sich leichter eingrenzen

Zusammenführen von Daten

Daten aus mehreren Workflow-Zweigen können wieder zusammengeführt werden.

Beispiel:

Mögliche Kriterien:

- gleiche Benutzer-ID
- gleiche E-Mail-Adresse
- gleiche Projekt-ID
- gleiche Vorgangsnummer

Beispiel: Abteilungsabhängige Rechte

Eingangsdaten:

```
{
  "name": "Max Mustermann",
  "department": "IT"
}
```

Verzweigung:

```
department prüfen
   /       |        \
  IT     Vertrieb   Sonstige
  |         |         |
  v         v         v
IT-Gruppe  CRM-Gruppe Standardgruppe
```

Mögliche Ausgabe:

```
{
  "name": "Max Mustermann",
  "groups": [
    "employees",
    "it-support"
  ]
}
```

Beispiel: Pflichtfeldprüfung

Ein Benutzer soll nur angelegt werden, wenn alle Pflichtfelder vorhanden sind.

Pflichtfelder:

- Name
- E-Mail-Adresse
- Abteilung
- Startdatum

Ablauf:

```
Daten empfangen
      |
      v
Pflichtfelder prüfen
      |
      v
vollständig?
  /      \
 ja        nein
 |          |
 v          v
weiter    Fehler melden
```

Fehlermeldung:

```
{
  "success": false,
  "error": "Pflichtfeld email fehlt"
}
```

Beispiel: Statuscode auswerten

Nach einem API-Aufruf:

```
Statuscode prüfen
  /     |      \
 2xx    4xx     5xx
 |      |       |
 v      v       v
Erfolg  Request   Retry oder
        prüfen    Fehler melden
```

Mögliche Behandlung:

Status	Reaktion
200	Daten weiterverarbeiten
201	Erstellung bestätigen
400	Eingabedaten prüfen

Status	Reaktion
401	Token prüfen
403	Berechtigung prüfen
404	Ressource oder Endpoint prüfen
429	Wartezeit und Retry
500	Fehler protokollieren
503	später erneut versuchen

Expressions in n8n

Expressions lesen oder verändern dynamische Werte.

Beispiele:

Wert auslesen:

```
{{$json.name}}
```

Text zusammensetzen:

```
{{$json.firstName + " " + $json.lastName}}
```

Standardwert:

```
{{$json.department || "Allgemein"}}
```

Boolean prüfen:

```
{{$json.active === true}}
```

Zahl umwandeln:

```
{{Number($json.licenseCount)}}
```

Code Node

Für komplexere Datenverarbeitung kann eine Code Node verwendet werden.

Beispiel:

```
return items.map(item => {
  return {
    json: {
      fullName: `${item.json.firstName} ${item.json.lastName}`,
      department: item.json.department || "Allgemein"
    }
  };
});
```

Die Code Node:

- liest alle Items
- erstellt `fullName`
- setzt einen Standardwert für `department`
- gibt neue JSON-Daten zurück

Für einfache Aufgaben sollten möglichst Standard-Nodes verwendet werden.

Validierung vor Veränderung

Vor dem Mapping sollten Daten geprüft werden.

Empfohlene Reihenfolge:

1. Daten empfangen
2. Pflichtfelder prüfen
3. Datentypen prüfen
4. Werte validieren
5. Daten umwandeln
6. Zielstruktur erzeugen
7. API aufrufen
8. Response prüfen

> Erst prüfen, dann verändern und übertragen.

Unbekannte Werte

Eine Switch-Verzweigung sollte einen Standardpfad besitzen.

Beispiel:

```
Abteilung prüfen
  /     |      \
 IT   Vertrieb  Standard
 |      |         |
 v      v         v
IT-Gruppe CRM    manuelle Prüfung
```

Ohne Standardpfad könnten unbekannte Werte unbemerkt verloren gehen.

Groß- und Kleinschreibung

Systeme können Werte unterschiedlich schreiben.

Beispiele:

```
IT
it
It
information technology
```

Vor einem Vergleich kann eine Vereinheitlichung notwendig sein.

Beispiel:

```
{{$json.department.toLowerCase()}}
```

Ergebnis:

```
it
```

Danach kann zuverlässig mit `it` verglichen werden.

Leerzeichen entfernen

Eingabedaten können unbeabsichtigte Leerzeichen enthalten.

Beispiel:

```
" IT "
```

Bereinigung:

```
{{$json.department.trim()}}
```

Ergebnis:

```
"IT"
```

Diese Normalisierung verhindert fehlerhafte Vergleiche.

Sicherheitsaspekte

Beim Datenmapping sollten nur benötigte Felder übertragen werden.

Nicht unnötig übertragen:

- Passwörter
- Zugangstoken
- private Notizen
- vertrauliche Personaldaten
- vollständige Kundendatensätze

Beispiel:

Quellsystem enthält:

```
{
  "name": "Max Mustermann",
  "email": "max.mustermann@example.com",
  "salary": 50000,
  "password": "geheim"
}
```

Das Zielsystem benötigt nur:

```
{
  "name": "Max Mustermann",
```

```
  "email": "max.mustermann@example.com"
}
```

> Nur notwendige Daten sollten ein System verlassen.

Typische Fehler

- falscher Feldpfad
- leeres Pflichtfeld
- String statt Zahl
- String statt Boolean
- fehlender Standardpfad
- Groß- und Kleinschreibung nicht berücksichtigt
- doppelte Datensätze
- falsche Zuordnung von Quell- und Zielfeld
- sensible Daten werden unnötig übertragen
- Verzweigungen enden ohne Fehlerbehandlung

Systematische Fehlersuche

Wenn eine Verzweigung oder ein Mapping nicht funktioniert:

1. Welche Eingangsdaten erhält die Node?
2. Ist der Feldname korrekt geschrieben?
3. Ist der JSON-Pfad korrekt?
4. Ist das Feld vorhanden?
5. Ist der Wert leer oder `null`?
6. Besitzt der Wert den erwarteten Datentyp?
7. Stimmt die Groß- und Kleinschreibung?
8. Werden Leerzeichen berücksichtigt?
9. Ist die Bedingung mit UND oder ODER korrekt aufgebaut?
10. Existiert ein Standardpfad?
11. Entspricht die Zielstruktur der API-Dokumentation?
12. Werden mehrere Items korrekt verarbeitet?
13. Enthält die Ausgabe nur die benötigten Daten?

Wichtige Begriffe

Begriff	Bedeutung
Bedingung	prüft, ob eine Aussage wahr oder falsch ist

Begriff	Bedeutung
If	Verzweigung mit zwei Wegen
Switch	Verzweigung mit mehreren Fällen
UND	alle Bedingungen müssen erfüllt sein
ODER	mindestens eine Bedingung muss erfüllt sein
Datenmapping	Zuordnung von Quell- und Zielfeldern
Standardwert	Ersatzwert bei fehlenden Daten
Filter	lässt nur passende Datensätze weiter
Schleife	wiederholt eine Verarbeitung
Batch	begrenzte Gruppe von Datensätzen
Normalisierung	vereinheitlicht Eingabewerte
Validierung	prüft Struktur, Datentyp und Inhalt

Gesamtmerksatz

> Bedingungen und Verzweigungen steuern den Ablauf eines Workflows. Datenmapping passt Feldnamen, Werte und Strukturen an das Zielsystem an. Vor der Verarbeitung müssen Pflichtfelder, Datentypen und Eingabewerte geprüft werden. Unbekannte Fälle benötigen einen sicheren Standard- oder Fehlerpfad.

Kontrollfragen

Was ist eine Bedingung?

Eine Prüfung, die ein wahres oder falsches Ergebnis liefert.

Wann wird eine If-Verzweigung verwendet?

Bei einer Entscheidung mit meistens zwei möglichen Wegen.

Wann wird eine Switch-Verzweigung verwendet?

Wenn mehrere unterschiedliche Werte oder Fälle behandelt werden müssen.

Was bedeutet Datenmapping?

Felder eines Quellsystems werden passenden Feldern eines Zielsystems zugeordnet.

Was ist der Unterschied zwischen UND und ODER?

Bei UND müssen alle Bedingungen erfüllt sein. Bei ODER genügt mindestens eine.

Was ist ein Standardwert?

Ein Ersatzwert, der verwendet wird, wenn ein benötigter Wert fehlt.

Warum müssen Datentypen geprüft werden?

Weil beispielsweise `true` und `"true"` unterschiedliche Werte darstellen.

Was ist ein Filter?

Eine Prüfung, die nur passende Datensätze weiterverarbeitet.

Warum sollte eine Switch-Verzweigung einen Standardpfad besitzen?

Damit unbekannte Werte nicht unbemerkt verloren gehen.

Warum sollten nur notwendige Felder übertragen werden?

Um Datenschutz- und Sicherheitsrisiken zu reduzieren.

Quellen

- n8n-Dokumentation – If Node
- n8n-Dokumentation – Switch Node
- n8n-Dokumentation – Edit Fields Node
- n8n-Dokumentation – Expressions
- n8n-Dokumentation – Data Structure