LimeSurvey-Tutorial

Gleichungen,
Zu- und andere Fälle

November 2023

Inhaltsverzeichnis

Vorbemerkung

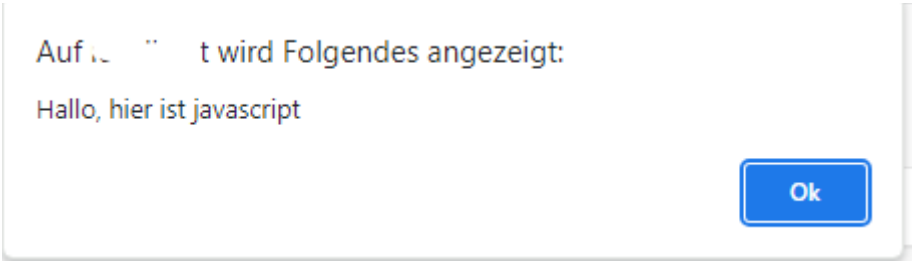
In diesem Tutorial geht es nicht um Look & Feel, sondern um die „im Verborgenen arbeitenden Geister“. Daher werden einige oft gefragte Methoden zum zufälligen Auswählen von Fragen und/oder Antwortoptionen gezeigt.

Da in einigen gezeigten Beispielen javascript benutzt wird, ist natürlich die Grundvoraussetzung, dass dieses eingesetzt werden kann.

Ein einfacher Test ist dies in den Quellcode der ersten Frage einzufügen

```
<script type="text/javascript" charset="utf-8">
$(document).on('ready pjax:scriptcomplete',function(){
    alert("Hallo, hier ist javascript");
});
</script>
```

Wenn dann dieses Fenster aufploppt, ist alle in Ordnung; wenn nicht, ...



Und dann noch dieses:

Solutions, code and workarounds presented in these text are given without any warranty, implied or otherwise.

Lösungen, Code und „Work-arounds“, die in diesem Text präsentiert werden, werden ohne jegliche stillschweigende oder sonstige Gewährleistung gegeben.

1. Begriffserklärungen

1.1. „Randomisierung“

Das Wort „Randomisierung“ oder „randomisiert“ wird im Forum mit ganz verschiedenen Bedeutungen benutzt.

„Ich möchte Fragen/Gruppen randomisiert darstellen“

Das kann nun bedeuten

- alle Fragen/Gruppen werden angezeigt, aber in zufälliger Reihenfolge
- es wird nur eine oder mehrere Fragen/Gruppen zufällig ausgewählt, und diese angezeigt
  (Dies ist ja gleichbedeutend mit „Die Teilnehmer werden per Zufall in Gruppen aufgeteilt, denen dann verschiedene Fragen gestellt werden“)

Und für beides gibt es in LimeSurvey verschiedene Lösungen, wobei – dies sei vorausgeschickt – es auch zwei Lösungen für ein Problem geben kann.

Grundsätzlich kann man aber sagen:

Wenn es darum geht, ein Element aus mehreren auszuwählen, ist das Stichwort „Zufallszahl“.

Wenn es darum geht, alle Elemente in unterschiedlicher Reihenfolge anzuzeigen, ist das Stichwort „Randomisierungsgruppe“.

1.2. „Relevanzgleichung“ (seit Version 5.x. „Bedingung“ genannt)

Hinter diesem etwas sperrigen Begriff verbirgt sich nichts anderes als ein logischer Term, der entweder WAHR oder FALSCH ist.

Damit wird festgelegt, ob eine Gruppe/Frage/Teilfrage angezeigt wird (wenn WAHR) oder nicht (wenn FALSCH). Dieser Term kann nun beliebig kompliziert sein. Der einfachste ist wieder „1“, was man oft als Vorgabe bereits vorfindet. In der benutzten Programmiersprache gilt nämlich intern „0“ = FALSCH, alles andere WAHR.

Bedeutet also:

Wenn in der „Relevanzgleichung“ etwas steht wie „Q1a==4“, dann wird die Frage nur angezeigt, wenn die Frage Q1a mit dem Code 4 beantwortet wurde, sonst nicht.
Zusatz: Sollte der Vergleichswert nicht numerisch sein, sondern Text, so muss er in Anführungszeichen eingeschlossen werden; also zum Beispiel „Q1a==‘Mercedes’“.

Machen wir es etwas komplizierter: In der Relevanzgleichung von Q1d steht

„(Q1a==1 OR (Q1a==4 AND Q1b_SQ001==“Y“)) AND (Q1c>29 AND Q1c<60)“

Lösen wir ein bisschen auf

„(Q1a==1 OR (Q1a==4 AND Q1b_SQ001==“Y“)) AND (Q1c>29 AND Q1c<60)“

UND-Verknüpfung: Beide Teile (grün und rot) müssen WAHR sein, damit der Gesamtterm WAHR ist.

„(Q1a==1 OR (Q1a==4 AND Q1b_SQ001==“Y“))

ODER-Verknüpfung: Einer der beiden Teile (grün und rot) muss WAHR sein, damit der Gesamtterm WAHR ist.

(Q1a==4 AND Q1b_SQ001==“Y“)

Wieder eine UND-Verknüpfung

Damit Q1d also angezeigt wird, muss der Teilnehmer

die Frage Q1a mit „1“ beantwortet haben

ODER

die Frage Q1a mit „4“ beantwortet haben UND die erste Teilfrage der Frage Q1b ausgewählt haben

UND auf jeden Fall

zwischen 30 und 59 Jahre alt sein (Q1c war wohl die Frage nach dem Alter)

Im Grunde schreibt man also den Term so hin, wie man ihn sprechen würde (natürlich die Klammerhierarchie beachten). Dahingegen würde eine solche Bedingung, wenn man versuchen würde sie im sogenannten „Bedingungs-Designer“ zu erstellen, ausgesprochen umständlich.

Aus diesem Grunde benutzen wird diesen auch nicht mehr; er stammt noch aus einer Zeit, in welcher es den ExpressionManager / ExpressionScript noch nicht gab.

1.3. Die Frage vom Typ „Gleichung“

Aus alter Programmiergewohnheit nenne ich Variable immer so, dass bereits am Namen ihr Typ zu erkennen ist; Fragen von diesem Typ heißen also immer „eqxxx“ (eq für Equation)

Wie der Name sagt, werden hier keine Fragen gestellt, sondern irgendetwas berechnet oder manipuliert. Um dies für das System kenntlich zu machen, werden die hier eingetragenen Funktionen von geschweiften Klammern eingeschlossen.

> Wichtig dabei ist, dass nach der öffnenden geschweiften Klammer kein Leerzeichen ist, genauso wenig wie vor der schließenden geschweiften Klammer. Daran erkennt ExpressionManager/Script nämlich, dass es hier etwas zu tun gibt.
>
> Im Gegensatz dazu müssen in scripten wie javascript oder css dort Leerzeichen sein, damit ExpressionManager/Script eben nicht zuschlägt.

Dieser Fragetyp hat zunächst einmal zwei verschiedene Funktionsweisen.

1. Erzeugen einer neuen Variablen
2. Ändern bzw. Zuweisen von Werten zu einer bestehenden Variablen

1.3.1. Erzeugen einer neuen Variablen

Es kann ein Wert mittels einer Funktion berechnet werden, der dann unter dem Namen der Frage im Datensatz gespeichert wird.

Wenn man die Frage vom Typ „Gleichung“ also „eqTralala“ nennt, und in den Fragetext {1+3} schreibt, wird im Datensatz in der Spalte „eqTralala“ eine „4“ gespeichert.

Das war jetzt wirklich trivial.

Etwas weniger trivial

Normalerweise möchte man aber irgendetwas mittels einer Formel berechnen. Dazu kann man alle in LimeSurvey implementierten Funktionen benutzen.

https://manual.limesurvey.org/ExpressionScript_-_Presentation#Implemented_functions

a. {sqrt(Q1)}	Wurzel aus dem in Frage Q1 angegebenen Wert
b. {sum(Q1.NAOK,Q2.NAOK)}	Die Summe der Antworten in Frage Q1 und Frage Q2
c. {sum(that.Q1.NAOK)}	Die Summe der Teilfragen in Frage Q1 (Matrix)

Oder zum Beispiel der Body Mass Index.

In der Frage Qcm wurde die Größe in cm, in der Frage Qkg das Gewicht in kg erfragt.

Der BMI wird berechnet nach

$$\frac{Gewicht\,[kg]}{Größe\,[m]^2}$$

Gewicht ist vorhanden, die Größe (in cm) muss noch auf m umgerechnet werden.

Damit ergibt sich als Fragetext der Frage eqBMI (unter Benutzung der Funktion „pow(x,y)“):

{Qkg / pow(Qcm/100,2)}

Und als letztes Beispiel ein simpler Score einer Matrix.
Die Frage sei nicht obligatorisch und die Antwortoptionen mit 1,2,3,4,5 codiert.

Es gibt keine Funktion „Mittelwert“; also wird er „mit der Hand“ berechnet. (Summe geteilt durch Anzahl)

{sum(that.Q1.NAOK)/count(that.Q1.NAOK)}

Da die Frage nicht obligatorisch ist, muss die Anzahl der beantworteten Teilfragen gezählt werden; bei einer Pflichtfrage ist dies ja bekannt und kann als Konstante eingesetzt werden.

Beim letzten Beispiel kommen nun zwei neue Dinge ins Spiel „that“ und „NAOK“.

1. „that“

Diese Variable bezieht sich immer auf eine andere Frage. Sie dient als Kurzschreibweise und würde hier expandieren zu: {sum(Q1_SQ001.NAOK,Q1_SQ002.NAOK,Q1_SQ003.NAOK,…)}
Man kann diese Expandierung einschränken durch Terme wie
that.Q1.sq_A: Es werden nur diejenigen Teilfragen in Betracht gezogen, deren Code ein „A“ enthält.
Dadurch kann man durch geschickte Vergabe der Teilfragencodes sehr kurze und mächtige Gleichungen erzeugen.

2. „NAOK“

Ich schrieb, die Frage sei nicht obligatorisch. Dann würde die Summenbildung bei fehlenden Antworten kein Ergebnis liefern (count auch nicht; alle Funktionen, die aggregieren).
Daher benutzt man NAOK (No answer is OK)

1.3.2. Ändern bzw. Zuweisen von Werten zu einer bestehenden Variablen

Eine Zuweisung von Werten kann beispielsweise dazu dienen, in einer Frage bereits Werte vorzubesetzen. Dies kann zum Beispiel eine Vorbesetzung von Zellen einer Matrix(Texte/Zahlen) sein, oder auch eine aufeinander aufbauende Berechnung von Werten

Ein Beispiel wäre:

In einer Matrix(Texte) werden die Zellen der Diagonalen mit „1“ vorbesetzt; sie könnten dann auf „readonly“ gesetzt werden.

Wie üblich sind die Zellen der y-Achse mit „Y001“, „Y002“,…, die der x-Achse mit „X001“, „X002“, … kodiert.

Dann würde in die Gleichungsfrage eingetragen:

{Q1_Y001_X001=1}
{Q1_Y002_X002=1}
{Q1_Y003_X003=1}

Hier ist zu beachten, dass in diesem Fall **nur ein einzelnes** Gleichheitszeichen gesetzt wird.

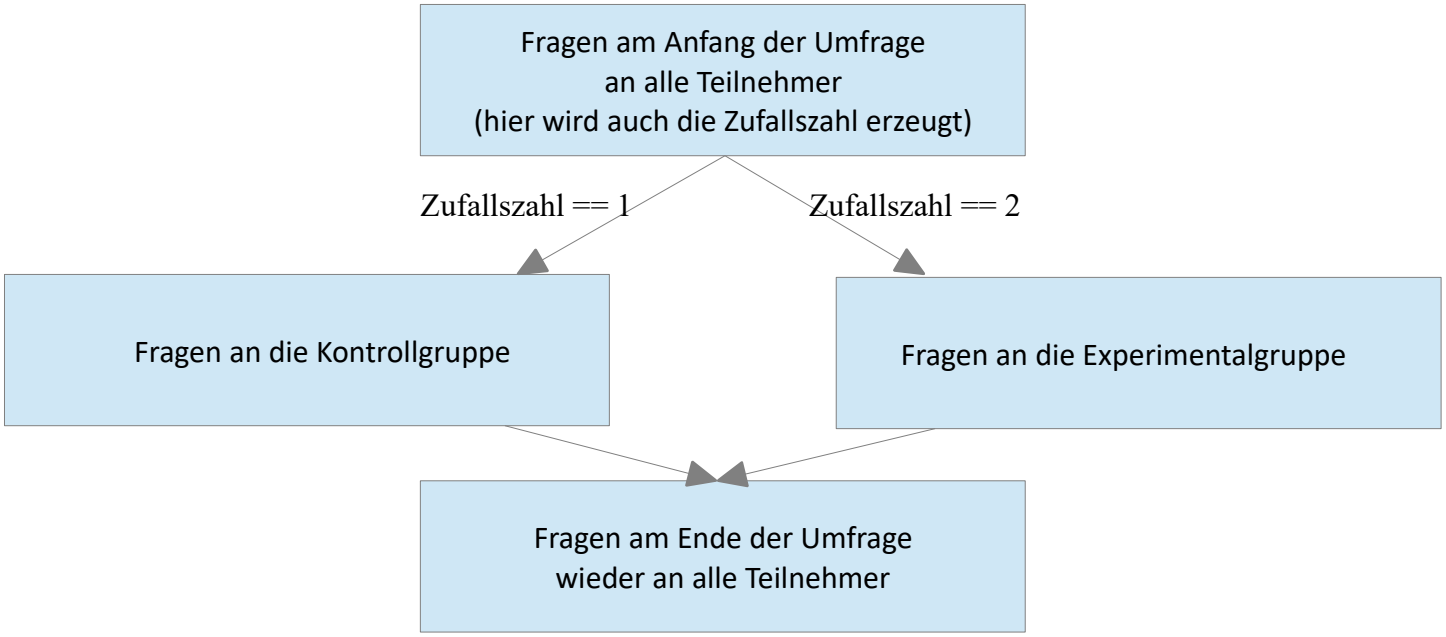
Dies bedeutet, es wird etwas zugewiesen; im Gegensatz dazu bedeuten die doppelten Gleichheitszeichen den Vergleich auf Gleichheit.

Solche Fragen werden natürlich grundsätzlich versteckt.

2. Die beiden häufigst vorkommenden Fälle

2.1. Aufteilung der Teilnehmer in zwei oder mehr Gruppen

Ein sehr häufig vorkommender Fall ist die zufällige Aufteilung der Teilnehmer in zwei oder mehr Gruppen (eine Kontrollgruppe und eine Experimentalgruppe).



2.1.1. Standardlösung

Wie bereits beschrieben wird also eine Zufallszahl erzeugt, und zwar von 1-2. Nennen wir die Frage „eqZufall“.

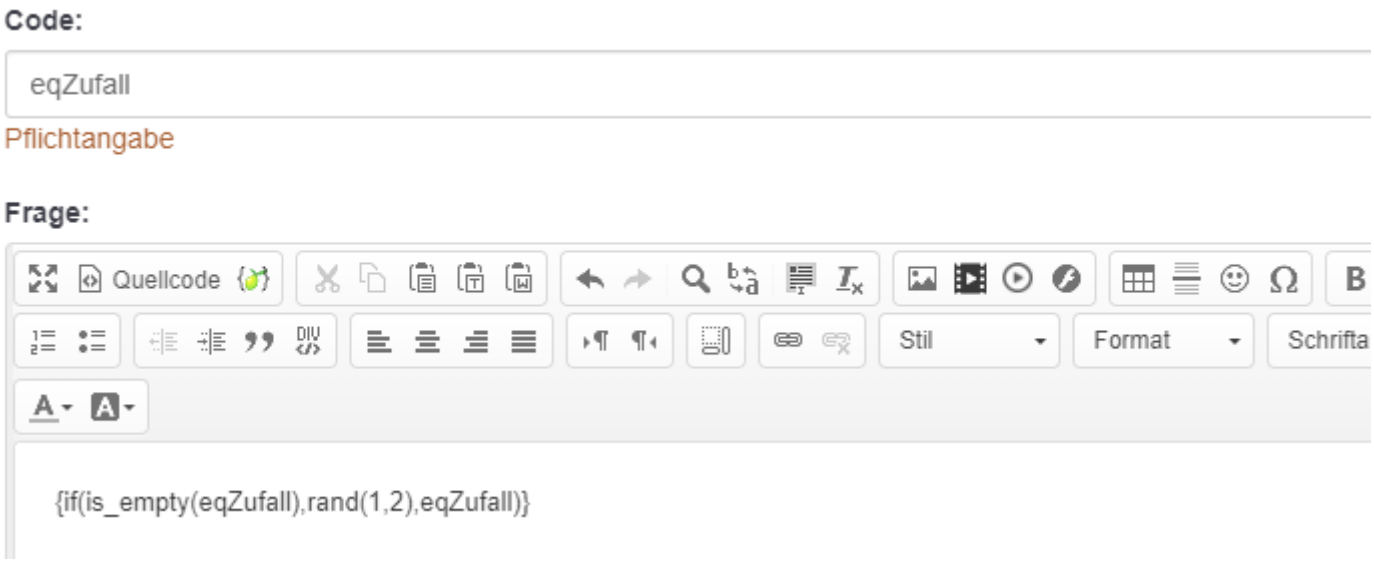
In den "implementierten Funktionen" gibt es dazu die Funktion "rand(x,y)". Dies ist die Rohform, die man aber nicht verwenden sollte.

Begründung: Wie man auch gut in EXCEL nachvollziehen kann, ändert sich eine Zufallszahl bei jedem Aufruf einer Seite. Dadurch können ungewollte Effekte auftreten, wenn z.B. mehrere Fragen auf einer Seite zusammen mit der Zufallszahlserzeugung sind.
Dasselbe würde passieren, wenn ein Teilnehmer durch Zurückblättern im Fragebogen wieder auf die Seite gerät, die die Zufallszahl erzeugt.

Um dies zu verhindern, wird ein bisschen hinzugefügt: **{if(is_empty(eqZufall),rand(1,2),eqZufall)}**

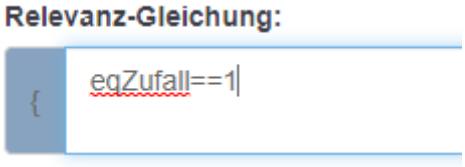
Heißt: WENN die Zufallszahl „eqZufall“ noch leer ist, erzeuge sie, SONST lasse sie, wie sie ist.

Und dieses Konstrukt sollte auch bei allen Fragen angewendet werden, welche (z.B. durch Zurückgehen in der Umfrage) erneut aufgerufen werden könnten.



Die Variable „eqZufall“ wird entweder den Wert „1“ oder den Wert „2“ annehmen.
In die „Relevanzgleichung / Bedingung“ der Frage oder auch Gruppe trägt man nun die Bedingung ein, unter welcher die Frage/Gruppe angezeigt werden soll.

Wichtig: Der Vergleichsoperator besteht aus zwei Gleichheitszeichen (==)



Die eine Gruppe erhält also die Relevanzgleichung / Bedingung „eqZufall==1“, die andere entsprechend „eqZufall==2“.

Bei dieser Methode kann man natürlich nicht erwarten, dass bei einem Sample von – sagen wir – 100 Versuchspersonen genau 50 in die Gruppe 1 und 50 in die Gruppe 2 fallen. Das lässt sich leicht mit einer Excel-Tabelle oder auch einer Münze simulieren.

2.1.2. „Zyklische Zuweisung“

Daher kann man auf die Idee kommen, ein anderes Verfahren zu verwenden, nämlich eine Art „zyklischer Zuweisung“.

Soll heißen, der erste Teilnehmer (TN) fällt in Gruppe 1, der zweite in Gruppe 2, der dritte in Gruppe 3, der vierte wieder in Gruppe 1, usw.
Damit möchte man die Zufälligkeit der Zufallszahl vermeiden, die eben gerade deswegen bei kleinen Stichproben ungleiche Besetzungen der Gruppen liefern kann.

Diese Idee klingt zunächst einmal plausibel, hat aber auch Tücken.

Denn wie geht man vor?
Bei jedem TN wird am Anfang geschaut, wie viele bereits die Umfrage beantwortet haben, und mittels der "modulo"-Funktion wird die Gruppe berechnet.
Nehmen wir also an, die Umfrage wird gestartet, der erste TN beginnt und wird der Gruppe 1 zugeordnet.
Nun beginnt auch der zweite TN - in welche Gruppe muss er?
Wenn TN 1 seinen Fragebogen abschließt, muss TN 2 in Gruppe 2. Aber wen TN 1 die Beantwortung abbricht? Dann müsste TN 2 in Gruppe 1.

Dies bedeutet, solange der erste TN seine Umfrage nicht abgeschlossen hat, werden alle folgenden ebenfalls in Gruppe 1 landen.
Dies ist besonders unangenehm, wenn TN sofort nach Erhalt der Einladung mit der Beantwortung loslegen. (Findet man häufig bei Panelmitgliedern; "Sofort antworten, sonst ist die Quote vielleicht schon voll, und ich verdiene kein Geld").
Hier muss man auch auf die Dauer der Beantwortung schauen.
Bei kurzen (5min) Umfragen ist es weniger dramatisch.
Und natürlich sendet man Einladungen langsam in Teilen aus.

Nun könnte man sich denken "Gut, rechne ich nicht nur die vollständigen Befragungen, sondern alle".
Das geht auch - sogar noch leichter als alleinige Berücksichtigung der "Completes" - aber hier könnte, wie der Zufall so spielt, eine große Anzahl derer, die der Gruppe 2 zugeordnet werden, die Befragung nicht beenden.
Und damit haben wir keinen Vorteil gegenüber der Benutzung einer Zufallszahl.

Trotzdem will ich einmal beide Verfahren vorstellen.

2.1.2.1. Berücksichtigung "aller" Teilnehmer

Da dies das weniger aufwendige Verfahren ist, kommt es zuerst.
In der Antworttabelle bei jedem Start eines Teilnehmers ein neuer Datensatz mit einer laufenden Nummer erzeugt, der sogenannten SAVEDID.
Diese kann man mittels dieser Formel benutzen

{SAVEDID-x*floor((SAVEDID-1)/x)}

wobei x die Zahl der gewünschten Gruppen ist.

Die Subtraktion von 1 dient nur dazu , den Zyklus bis "3" laufen zu lassen (1,2,3,1,2,... statt 1,2,0,1,2,…). Ist nicht relevant für die Arbeit, aber etwas „schöner“

Heißt also: Statt der Erzeugung der Zufallszahl mittels rand(x,y) wird diese Formel eingesetzt; ansonsten ist alles analog.

Beispiel:

Es gibt drei Gruppen, x ist also 3.

x	SAVEDID	(SAVEDID-1)/x	floor((SAVEDID-1)/x)	x*floor((SAVEDID-1)/x)	SAVEDID-x*floor((SAVEDID-1)/x)
3	1	0	0	0	1
3	2	0,333333333333	0	0	2
3	3	0,666666666667	0	0	3
3	4	1	1	3	1
3	5	1,333333333333	1	3	2
3	6	1,666666666667	1	3	3
3	7	2	2	6	1

2.1.2.2. Berücksichtigung "vollständiger" Teilnehmer

LimeSurvey-Version 3.x.

LimeSurvey-User dieser Version, die keine plugins installieren können, können dies überspringen.

Es gibt das plugin "getStatInSurvey", mit welchem man einige kumulierte Daten der Gesamtumfrage anzeigen kann, z.B. Häufigkeit, Prozentzahl, Mittelwerte von Antworten.
gitlab.com/SondagesPro/ExportAndStats/getStatInSurvey
Dies ist zunächst einmal gedacht, um diese Werte anzuzeigen.
Um die Werte auch in einer Gleichung benutzen zu können, muss man einen kleinen Umweg gehen.

Man erzeugt eine Frage vom Typ "kurzer Text" mit folgendem javascript im Fragetext (Quellcode-Modus)

```
<script type="text/javascript" charset="utf-8">
$(document).on('ready pjax:scriptcomplete',function(){
  var total=Math.floor({' [Q0.nb]'});
  $('#question{QID} input[type="text"]').val(total);
  $('#question{QID}').hide();
});
</script>
```

Hierbei soll Q0 der Code genau dieser Textfrage sein.

Im Grunde wird einfach geguckt, wie oft diese Frage bereits beantwortet wurde und das Ergebnis im Text gespeichert.

Nun wird eine Frage vom Typ Gleichung angeschlossen (diese darf nicht auf derselben Bildschirmseite dargestellt werden)
mit folgendem Inhalt:
{1+Q0-x*floor(Q0/x)}
ebenfalls wieder x= Anzahl der Gruppen

Die Struktur ist ein klein wenig anders.

Diese Addition von 1 dient nur dazu , den Zyklus bei "1" starten zu lassen (1,2,3,1,2,... statt 0,1,2,0,1,…).
Ist nicht relevant für die Arbeit, aber etwas „schöner“.

LimeSurvey-Version 5.x.

In der Version 5.x. ist ein Plugin bereits eingebaut, nämlich „statFunctions“.

Auch hier könnte es aber sein, dass Nutzer einer Uni-Installation dieses nicht zur Verfügung haben.

Mit den hier enthaltenen Funktionen „statCount“ und „statCountif“ lassen sich die oben gezeigten Beispiele ebenfalls realisieren.

Mit der Gleichung {statCount(“id“)} erhält man die Anzahl der bereits beendeten Umfragen und kann dann wieder mit der gezeigten Funktion die zyklische Zuordnung errechnen.

Mit der Gleichung {statCount,(“id“,0)} erhält man die Anzahl aller Umfragen (beendet oder nicht).

Dann geht es weiter wie oben.

Zusätzlich lässt sich hiermit eine „leastFilled“-Methoode implementieren.

Mit zwei (versteckten) Gleichungen und einer (versteckten) Frage (Q1) von Typ „Liste(Optionsfelder)“ wird dies erledigt.

Die Frage (Q1) enthält so viele Optionen, wie es Gruppen geben soll (hier 3).

Nun wird in der ersten der beiden Gleichungsfragen – heiße sie „eqMin“ - die Antwortoption errechnet, die bis jetzt am geringsten besetzt ist, also

{min(statCountIf(Q1.sgqa,1),statCountIf(Q1.sgqa,2),statCountIf(Q1.sgqa,3))}

Es wird also nachgeschaut, wie oft die Codes 1,2 oder 3 bisher vorkamen und davon das Minimum genommen.

In der nächsten Gleichung (eqLeast) wird nun die Antwortoption dieses Minimums bestimmt. Gleichzeitig wird die Frage Q1 auf diesen Wert gesetzt.

{Q1=if(statCountIf(Q1.sgqa,1)==eqMin,1,if(statCountIf(Q1.sgqa,2)==eqMin,2,3))}

Dadurch erhält man in Q1 die zysklische Gruppenzugehörigkeit und kann dann eben die Bedingungen setzen:

Q1==1, Q1==2 oder Q1==3

2.1.3. Zusammenfassung

Es gibt also Alternativen zur Verwendung der Zufallszahl. Ob diese "besser" sind, muss man für sich selbst entscheiden.

Dazu möchte ich auf das "Feintunings" von Zuordnungen mittels Zufallszahlen hinweisen.
Man erzeugt nicht eine Zufallszahl von 1-2, sondern von 1-100 mit Startzuordnung
Gruppe 1: eqZufall<51
Gruppe 2: eqZufall>50

Wenn dann auffällt, dass die Verteilung schief wird, Gruppe 1 schon recht überbesetzt,
ändert man in so etwas wie
Gruppe 1: eqZufall<30
Gruppe 2: eqZufall>29
Dies geht auch in einer aktivierten Umfrage.

Hier kann man nun nicht mehr die Zufallszahl direkt als Trennvariable für statistische Tests in SPSS benutzen. Aber da ja offensichtlich ist, welche Fragen/Gruppen beantwortet wurden, kann man ja ohne große Verrenkungen eine solche Trennvariable erstellen.

Oder man verändert die Funktion zur Erzeugung der Zufallszahl.

Nehmen wir drei Gruppen an. Also würde die Gleichung anfänglich heißen:

{if(is_empty(eqZufall),rand(1,3),eqZufall)}

Falls Gruppe 1 bereits gefüllt ist, ändern zu: {if(is_empty(eqZufall),rand(2,3),eqZufall)}

Falls Gruppe 3 bereits gefüllt ist, ändern zu: {if(is_empty(eqZufall),rand(1,2),eqZufall)}

Falls Gruppe 2 bereits gefüllt ist, ändern zu: {if(is_empty(eqZufall),2*rand(0,1)+1,eqZufall)}

Damit ist gewährleistet, dass die gefüllte Gruppe nicht mehr als Zufallszahl auftaucht.

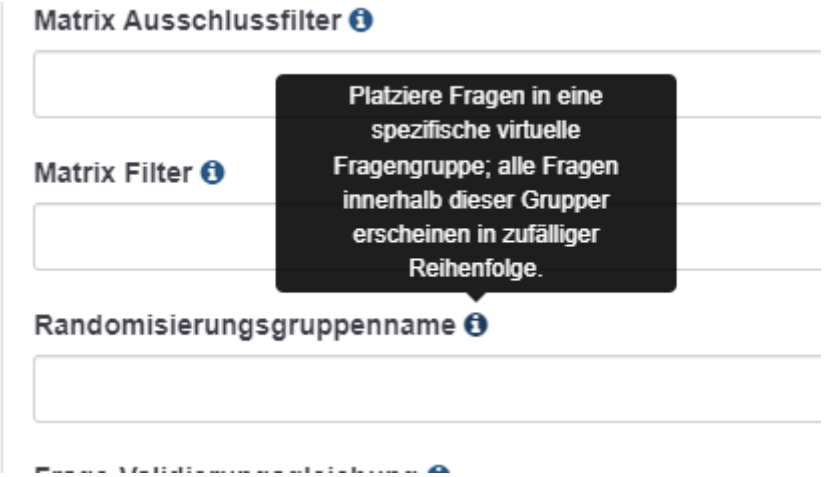
2.2. Randomisierte Reihenfolge mehrerer Fragen oder Gruppen

Auch ein häufig vorkommendes Szenario ist, dass man (zum Beispiel) Fragen(oder Gruppen von Fragen) stellt zu verschiedenen Ländern. Nun sollen diese nicht immer in derselben Reihenfolge dargeboten werden.

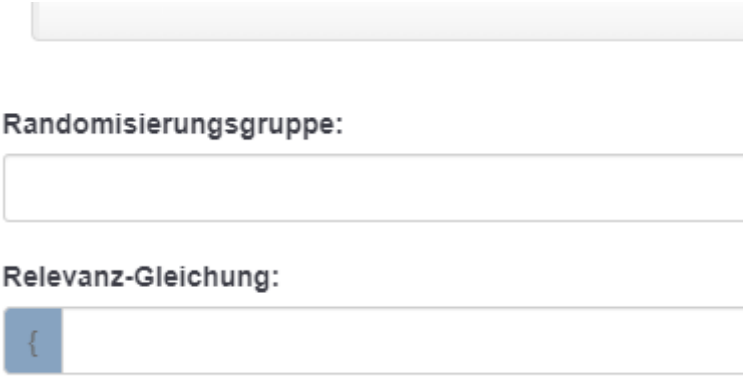
Um eine zufällige Reihenfolge zu erzeugen benutzt man die sogenannte „Randomisierungsgruppe“ (ebenfalls wieder ein sperriger Begriff).

Und es ist nichts weiter notwendig als allen betroffenen Fragen denselben „Randomisierungsgruppennamen“ zu geben.

Diese Eingabe findet man für Fragen im Bereich „Logik“ mit der mitgelieferten Erklärung



Für die „Randomisierung“ von Gruppen gibt man diese ein in der „Bearbeitung der Gruppe“.
Hier heißt es dann nur noch „Randomisierungsgruppe“



Dieser Name ist nun völlig beliebig. Ob „RG1“ oder „Fitzliputzli“ oder was auch immer.

3. Urnenziehung ohne Zurücklegen (mehrere verschiedene Zufallszahlen)

3.1. Anwendungsbeispiel

Beispiele gibt es zuhauf, hier sei sei es einmal die Auswahl von 5 Bildern aus einem Pool von 10.

Man könnte sich zunächst denken: "Gut, machen wir 5 Zufallszahlen zwischen 1 und 10".
Im Normalfall würden wohl 5 verschiedene Zahlen herauskommen, aber eben nicht zwangsläufig; es kann durchaus passieren, dass ein Zahl dabei zweimal auftaucht.

Erinnern wir uns an die Schule, haben wir es mit "Mehrmaligem Ziehen aus einer Urne ohne Zurücklegen" zu tun.

1. Lösung (die Standardlösung)

- Man legt zwei Gruppen an (G1 und G2).
- G2 wird mittels Relevanzgleichung (einfach 0 eintragen) versteckt.
- Nun legt man zehnmal die Frage zu den Bildern an.
- Alle Fragen bekommen denselben Wert in „Randomisierungsgruppenname“
- 5 Fragen werden in G1 geschoben, die restlichen 5 in G2.

Wegen des identischen Randomisierungsgruppennamens werden alle 10 Fragen über die 2 Gruppen „verstreut“, aber nur die 5, die in G1 gelandet sind, werden angezeigt.

Bei diesen Zahlen ist dieses Verfahren das Verfahren der Wahl. Sobald aber mehr (viel mehr Fragen, Objekte) zur Auswahl stehen wird es natürlich unhandlich.

Deswegen hier nun zwei andere Implementierungen.

3.2. Implementierungen

3.2.1. Mit javascript

Dazu wird eine Frage vom Typ „mehrfache kurze Texte“ angelegt mit so vielen Teilfragen wie Zufallszahlen benötigt werden.

Nun das script, welches in den Quellcode dieser Frage eingefügt wird:

```
<script type="text/javascript" charset="utf-8">


function shuffle(array) {
var currentIndex = array.length, temporaryValue, randomIndex;
// While there remain elements to shuffle...
while (0 !== currentIndex) {
// Pick a remaining element...
randomIndex = Math.floor(Math.random() * currentIndex);
currentIndex -= 1;
// And swap it with the current element.
temporaryValue = array[currentIndex];
array[currentIndex] = array[randomIndex];
array[randomIndex] = temporaryValue;
}
return array;
}
$(document).on('ready pjax:scriptcomplete',function(){
// Fill the array, hier Zahlen von 1 - 500
var arr = [];
for (var i = 1; i < 501; i++) {
arr.push(i);
}
arr = shuffle(arr);
$('#question{QID} input[type="text"]:eq(0)').val(arr[0]);
$('#question{QID} input[type="text"]:eq(1)').val(arr[1]);
$('#question{QID} input[type="text"]:eq(2)').val(arr[2]);
$('#question{QID} input[type="text"]:eq(3)').val(arr[3]);
$('#question{QID} input[type="text"]:eq(4)').val(arr[4]);
$('#question{QID}').hide();
});
</script>
```

Den meisten Raum nimmt hier die Funktion „shuffle“ ein, die nichts anderes tut als „die Kugeln in unserer Urne“ kräftig durcheinanderzuwürfeln.

Gut, es wird also zunächst ein array mit den Zahlen von 1 bis (hier 500) gefüllt (500 Kugeln in die Urne gelegt) dann dieses verwürfelt, und die ersten 5 Zahlen in die fünf Teilfragen der Frage eingetragen.

Zuletzt wieder der Befehl, um die Frage zu verstecken (sollte während des Tests mittels // ausgeschaltet sein). Daher darf diese Frage auch nicht alleine in einer Gruppe stehen, oder die Umfrage „Frage für Frage“

durchgeführt werden.

3.2.2. Nur implementierte Funktionen

Hier eine Lösung, die weder javascript noch irgendwelche plugins benötigt.
Dies könnte eine Reihe von Studenten betreffen, da viele admins in den Uni-Installationen dies verbieten.
Über die benutzten Funktionen lese man auch im Handbuch nach:
manual.limesurvey.org/ExpressionScript_-...mplemented_functions

1. Vorbereitung der "Urne"
Schaffen wir uns zunächst einmal eine gefüllte Urne.
Dazu bietet sich eine Frage vom Typ "kurzer Text" (eqBase1) an. Und als Vorgabe-Antwort tragen wir ein:
001002003004005006007008009010...497498499500
Also 500 "Kugeln" , jede mit einer Zahl beschriftet, links mit Nullen aufgefüllt, dass jede Zahl dreistellig ist.
Nun geht die Zieherei los.
Da im Folgenden viele Fragen vom Typ "Gleichung" benutzt werden, benenne ich sie eqxxx, an "Equation" erinnernd.
Als Erstes bestimmen wir die Anzahl der "Kugeln".
Das ist nicht schwer, das ist einfach "Länge des Textes" geteilt durch 3.
eqLaenge: {strlen(eqBase1)/3}

2. Ziehen der "Kugeln"

- Nun ziehen wir die erste Kugel.
  Dazu erzeugen wir eine Zufallszahl zwischen 1 und ebendieser Länge:
  eqRand1: {if(is_empty(eqRand1,(rand(1,eqLaenge),eqRand1)}
- Mit dieser Zufallszahl können wir nun die "Kugel" herausgreifen, die an dieser Stelle liegt.
  eqZahl1: {substr(eqBase1,(eqRand1-1)*3,3)}
  Übersetzt: Nimm die folgenden drei Stellen des Textes, der an der Stelle "(eqRand1-1)*3" anfängt.
- Jetzt ist diese Kugel "verbraucht" und muss entfernt werden.
  eqBase2: {str_replace(eqZahl1,"",eqBase1)}
  Übersetzt: Ersetze die soeben "gezogenen" drei Stellen des Textes durch einen leeren Text.

Und jetzt geht es analog weiter

- Zufallszahl erzeugen (der Text ist jetzt um eine Stelle verkürzt)
  eqRand2: {if(is_empty(eqRand2,(rand(1,eqLaenge-1),eqRand2)}
- Wert an der Stelle bestimmen – Kugel herausnehmen
  eqZahl2: {substr(eqBase2,(eqRand2-1)*3,3)}
- Aus dem Text entfernen
  eqBase3: {str_replace(eqZahl2,"",eqBase2)}

Usw.

- eqRand3: {if(is_empty(eqRand3,(rand(1,eqLaenge-2),eqRand3)}
- eqZahl3: {substr(QWerte,(eqRand3-1)*3,3)}
- eqBase4: {str_replace(eqZahl3,"",eqBase3)}

Nach 5 Runden haben wir damit 5 verschiedenen zufälligen Zahlen in "eqZahl1,..., eqZahl5".

Hier das Ganze noch einmal schematisch (mit 3 aus 8).

Hier wird die Länge der Zeichenkette immer aufs Neue in der Funktion „rand()“ bestimmt

Was tun wir									Fragencode	Gleichung
Wir füllen die Urne mit 8 Kugeln	1	2	3	4	5	6	7	8	eqBase1	{"12345678"}
Eine Zufallszahl von 1 bis zur Anzahl der Kugeln									eqRand1	{if(is_empty(eqRand1),rand(1,strlen(eqBase1)),eqRand1)}
Wir entnehmen die erste Kugel	1	2	3	4	5	6	7	8	eqZahl1	{substr(eqBase1,sum(eqRand1,-1),1)}
Wir entfernen die Kugel aus der Urne	1	2	3	5	6	7	8		eqBase2	{str_replace(eqZahl1,"",eqBase1)}
Eine Zufallszahl von 1 bis zur Anzahl der Kugeln									eqRand2	{if(is_empty(eqRand2),rand(1,strlen(eqBase2)),eqRand2)}
Wir entnehmen die zweite Kugel	1	2	3	5	6	7	8		eqZahl2	{substr(eqBase2,sum(eqRand2,-1),1)}
Wir entfernen die Kugel aus der Urne	1	2	3	5	6	8			eqBase3	{str_replace(eqZahl2,"",eqBase2)}
Eine Zufallszahl von 1 bis zur Anzahl der Kugeln									eqRand3	{if(is_empty(eqRand3),rand(1,strlen(eqBase3)),eqRand3)}
Wir entnehmen die dritte Kugel	1	2	3	5	6	8			eqZahl3	{substr(eqBase3,sum(eqRand3,-1),1)}

3.3. Weitere Anwendungen (zur Anregung der Kreativität)

3.3.1. Auswahl von 3 Elementen aus den gesamten gewählten einer Mehrfachnennung

Manchmal möchte über die in einer Mehrfachnennung ausgewählten Objekte weitere Fragen stellen. Um den Teilnehmer nicht zu sehr zu ermüden, ist es ratsam hier nur eine zufällig ausgewählte Auswahl anzuzeigen.

Gut, wie man 3 verschiedene zufällige Kugeln aus einer Urne zieht, ist im vorherigen Kapitel gezeigt.

Das Neue ist nun:

Wie füllen wir die Urne für dieses Beispiel?
Und wie beschriften wir die Kugeln?

Zur Beschriftung fallen natürlich Zahlen ein, je nach Anzahl der Kugeln ein-, zwei-, oder noch mehrstellig. Außerdem kann man auch andere Zeichen benutzen, zum Beispiel Buchstaben (große und kleine).

Persönlich verwende ich gerne Buchstaben, da mit den zur Verfügung stehenden 52 Zeichen die meisten Fälle abgedeckt werden. Und es ist einstellig; man muss nicht immer mit dem Faktor 2 oder 3 hantieren. Bei mehr Elementen bleibt aber keine andere Wahl.

Statt {substr(eqBase1,(eqRand3-1)*3,3)} einfach {substr(eqBase1,(eqRand3-1),1)}

Nun müssen wir die richtigen Kugeln in die Urne legen.
Dazu erstellen wir eine Frage vom Typ „Gleichung“ – nennen wir sie „eqUrne“.

Und um nun eine solche Zeichenkette wie im vorigen Beispiel zu erzeugen, gibt es die Funktion „join(a,b,c,…)“. Wie der Name andeutet werden hier Elemente zusammengefügt.

Was und wann fügen wir etwas zusammen?
Natürlich, wenn das Objekt in der Frage ausgewählt wurde.

Die Mehrfachnennungsfrage möge Q1 heißen mit den Teilfragencodes „Y001“, „Y002“, „Y003“,…

Dann sagen wir:

Wenn die Teilfrage 1 ausgewählt wurde (Q1_Y001==“Y“), dann füge ein „A“ hinzu, sonst nichts.
Und dies für alle Teilfragen.

{join(if(Q1_Y001==“Y“,“A“,““),if(Q1_Y002==“Y“,“B“,““),if(Q1_Y003==“Y“,“C“,““),…,if(Q1_Y012==“Y“,“L“,““))}

Damit haben wir die Urne gefüllt.

eqUrne kann dann einen Wert haben wie „BFGIL“.

Es geht weiter wie oben in 3.2.2.: Länge bestimmen, Zufallszahl erstellen, Kugel herausnehmen,…

Dann erhalten wir für die drei „Zufallszahlen vielleicht eqZahl1: „G“ eqZahl2: „B“ eqZahl3: „L“

In einer letzten Frage vom Typ „Gleichung“ (eqFinal) sollte man diese Zahlen noch einmal „joinen“
{join(„#“,eqZahl1,eqZahl2,eqZahl3)}

Dadurch erhalten wir wieder eine Zeichenkette, nämlich „#GBL“

Und nun stellt sich die Frage: Wie zeige ich die drei Folgefragen, oder gar drei Folgegruppen an?

1. Die Standardlösung würde so aussehen, dass man gleich viele Fragen/Gruppen erzeugt, wie ursprünglich Elemente vorhanden waren (hier waren es wohl 12).
Dann blendet man mittels Relevanzgleichung/Bedingung ein.

Wann wird Frage 1 angezeigt? Wenn eine der „Zufallszahlen“ gleich „A“ ist.
Wann wird Frage 2 angezeigt? Wenn eine der „Zufallszahlen“ gleich „B“ ist.
…

Das heißt also für Frage 1: eqZahl1==“A“ OR eqZahl2==“A“ OR eqZahl3==“A“

Und so weiter für jede weitere Frage.

Ist ein bisschen lang, diese Bedingung. Daher haben wir unsere drei Zahlen auch noch einmal zusammmengefasst.

Jetzt reicht als Bedingung:
Wann wird Frage 1 angezeigt? Wenn in der Zeichenkette ein „A“ ist.
Wann wird Frage 2 angezeigt? Wenn in der Zeichenkette ein „B“ ist.

Und dies kann man elegant mit der Funktion „strpos(haystack,needle)“ lösen. Diese sucht die Nadel im Heuhaufen und gibt bei Erfolg die Position der Nadel an, bei Misserfolg „0“.

Jetzt gibt es ein Dilemma. Diese Funktion beginnt die Zählung bei „0“.

strpos(eqFinal,“G“) würde im Normalfall den Wert 0 ergeben; dies bedeutet aber gleichzeitig „nicht gefunden“. Schön blöd.

Aber daher haben wir an die erste Stelle ja die Raute (#) gesetzt. Nun liefert „strpos(eqFinal,“G“) eine „1“.

Und die Bedingungen sind:

Frage 1: strpos(eqFinal,“A“)>0 (Wenn das „A“ in eqFinal gefunden wurde, egal wo)

Frage 2: strpos(eqFinal,“B“)>0

2. Die kürzere Lösung benötigt nur drei Fragen/Gruppen, in welche der Text mittels „tayloring“ angezeigt wird.

Bei einem solchen Szenario werden die Fragen für jedes Item ja wohl identisch sein und sich nur durch die Nennung des Items und/oder ein eingeblendetes Bild unterscheiden.

Dann würde man in jeder Frage den Text/ das Bild dynamisch mit einem IF-Konstrukt einblenden.

Hier gezeigt für die erste der drei Fragen, in welcher der Text / das Bild für das erste Element im Text „GBL“ anzeigt.

a. Textbeispiel
„Nun möchten wir Ihnen einige Fragen zum Bier der Brauerei „{if(substr(eqFinal,1,1)==“A“,“Andechser Urbräu“,if(substr(eqFinal,1,1)==“B“,“Binding“, if(substr(eqFinal,1,1)==“C“,“Clausthaler“,…, if(substr(eqFinal,1,1)==“L“,“Leikeimer Urbock“)))))))))))}“ stellen.

Sie sagten ja, dass Sie dieses Bier schon einmal getrunken haben. Bitte, beschreiben Sie doch einmal den Geschmack…“

b. Bildbeispiel

Hier sollte man die Bilder – falls möglich – geschickt umbenennen, z.B. „BildA.png“, „BildB.png“. Dann beschränkt sich die dynamische Einbindung eines Bildes auf:

Mittels ExpressionManager/Script wird der Buchstabe an den Bildnamen angehängt

<img src=“upload/images/Bild{substr(eqFinal,1,1)}.png“>

Sollte keine Möglichkeit bestehen, das Bild umzunennen (es stammt aus einer Internet-Bibliothek), kann man es herunterladen oder aber die obige Texte-Lösung benutzen, also ein verschachteltes IF.

Z.B.

{if(substr(eqFinal,1,1)==‘A’,‘<img src=“myServer/myFolder/myImages.png“>‘, if(substr(eqFinal,1,1)==‘B’,‘<img src=“myServer/myFolder/myOtherImages.png“>‘, if(substr(eqFinal,1,1)==‘C’,...

Man beachte die abwechselnde Benutzung der Anführungszeichen.

3. Zusammenfassung

Lösung 1 ist geradeheraus, verbraucht viel Platz, dafür stehen die Daten zu jedem Objekt / jedem Bild immer in derselben Datenbankspalte → leichte Auswertung

Lösung 2 spart Platz, dafür muss man aber vor der eigentlichen Analyse die Daten umstrukturieren, da nun in jeder Datenbankspalte die Daten zu jedem Objekt stehen können.

Die Anordnung der Objekte in den Spalten erkannt man ja in eqFinal.

Ein Excel-Makro leistet dies auch recht einfach.

3.3.2. Weiterbearbeitung der x am höchsten bewerteten Items einer Matrix

Nehmen wir an, wir haben eine Matrix, in welcher Objekte irgendwie bewertet werden.

In einer Folgefrage möchten wir nun über die am 4 am höchsten bewerteten Objekte weitere Fragen stellen.

Zunächst zum Aufbau.

Frage vom Typ „Matrix“ (Q1) mit 10 Teilfragen (kodiert „Y001“, „Y002“,...“Y010“) und 5 Antwortoptionen (kodiert 1,2,3,4,5).

Die Vorgehensweise ist klar:

1. Suche alle mit „5“ bewerteten Objekte

1.a. Sind es genau 4, sind wir fertig.

1.b. Sind es mehr als 4, müssen wir per Zufall 4 auswählen (Das war im vorigen Kapitel)

1.c. Sind es weniger als 4, gehen wir weiter mit der Bewertung „4“.

2.a. Ist die Gesamtzahl der Bewertungen „5“ und „4“ genau „4“, sind wir wieder fertig.

2.b. Ist die Gesamtzahl der Bewertungen „5“ und „4“ größer als „4“, müssen wie wieder aus den „4“-Bewertungen so viele per Zufall auswählen, dass die Summe „4“ ergibt.

2.c. Sind es immer noch weniger als 4, gehen wir weiter mit der Bewertung „3“.

4. Arbeiten mit Panels

Wenn man einen Panel-Provider benutzt, welcher die Versuchspersonen „zur Verfügung stellt“, sind einige Dinge zu beachten.

Die Panel-Teilnehmer bekommen eine kleine Vergütung dafür, dass sie eine Umfrage beantworten. Diese bekommt er vom Panel-Provider. Daher muss dieser wissen, wer von seinen Schäfchen an der Umfrage teilgenommen hat – und auch mit welchem Ergebnis.

Daher werden einige Parameter an die LimeSurvey-Umfrage übergeben und am Ende dann wieder an den Panel-Provider.

4.1. Speicherung des vom Panel-Provider übergebenen Parameters

Im Handbuch handelt dieses Kapitel davon, wie man Fragen in einer Umfrage mittels der URL vorbesetzt.

URL_Felder

Gleich das erste Beispiel ist das Gesuchte

```
https://www.myServer.com/limesurvey/123456?Q1=Tralala
```

Fülle die Frage Q1 mit dem Text „Tralala“

Das bedeutet aber im Umkehrschluss: Wenn ich einen angehängten Parameter in der Umfrage speichern will, muss ich eine Frage (am besten immer „kurzer Text“) mit dem Namen des Parameters anlegen; dann wird der Wert des Parameters dort gespeichert.

Anderes Beispiel

```
https://www.myServer.com/limesurvey/123456?MeinParameter=Fitzliputzli
```

Um also „Fitzliputzli“ irgendwie in der Umfrage zu speichern, muss die Frage den Fragecode „MeinParameter“ haben.

4.1.1. Panel-Integration

Im LimeSurvey-Menue gibt es den Punkt „Panel-Integration“.

Meiner Meinung nach ist diese nur notwendig, wenn der vom Panel-Provider übergebene Parameter einen Namen hat, der als Fragecode in LimeSurvey nicht erlaubt ist (er beginnt mit einer Zahl, enthält nicht erlaubte Zeichen, usw.)

Dann kann man mit der „Panel-Integration“ eine entsprechende Zuweisung vornehmen.

Hier beginnt der Parametername mit einer Ziffer (nicht erlaubt); daher wurde die Frage „QParam“ angelegt und diese Zuweisung vorgenommen.

```
https://www.myServer.com/limesurvey/123456?12XYZ=Fitzliputzli
```

Der Wert von 12XYZ (Fitzliputzli) wird dann in der Frage QParam gespeichert.

Der Panel-Provider muss also angeben, mit welchem Namen der Parameter an die URL der Umfrage angehängt wird.
Und es kann auch vorkommen, das mehrere Parameter angehängt werden. Dann werden eben zwei Fragen angelegt.

Gut, jetzt haben wird den übergebenen Parameter (meist ist dies ja eine ID des Teilnehmers) gespeichert.
Nun möchte der Panel-Provider auch wissen, wie seine Leute abgeschnitten haben.

4.2. Rückgabe der Parameter an den Panel-Provider

Ein Teilnehmer kann die Umfrage ja mit verschiedenen Ergebnissen beenden:

- vollständig durchgeführt (Complete)
- abgebrochen wegen nicht erfüllter Kriterien (ScreenOut)
- abgebrochen wegen bereits erfüllter Quote (QuotaFull)

Es gibt noch andere, die aber meist nicht gesondert betrachtet werden.

Und diese Information erwartet der Panel-Provider neben der Teilnehmer-ID auch zurück. Davon hängt ja die Bezahlung ab.

Er wird also die Information liefern, wie diese Information zurückgeliefert wird. Grundsätzlich wird dazu eine Website bei ihm aufgerufen mit den entsprechenden angehängten Parametern.

Nehmen wir einmal folgendes Szenario an.

Der Panel-Provider hat an die URL zum Aufruf der Umfrage den Parameter „tid“ angehängt, der auch in der Frage „tid“ gespeichert ist und diese Struktur für die Redirects mitgeteilt:

https://gateway.haifish.com/routings/participation_id/finish/completed

https://gateway.haifish.com/routings/participation_id/finish/screenout

https://gateway.haifish.com/routings/participation_id/finish/quota_full

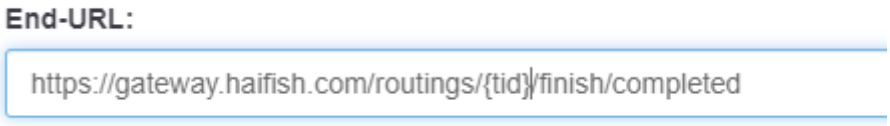
Interessant ist hier der Parameter „participation_id“. Hier soll der am Anfang übergebene Parameter „tid“ wieder zurückgegeben werden. Alles andere ist ja konstant.

a. vollständig durchgeführt (complete)

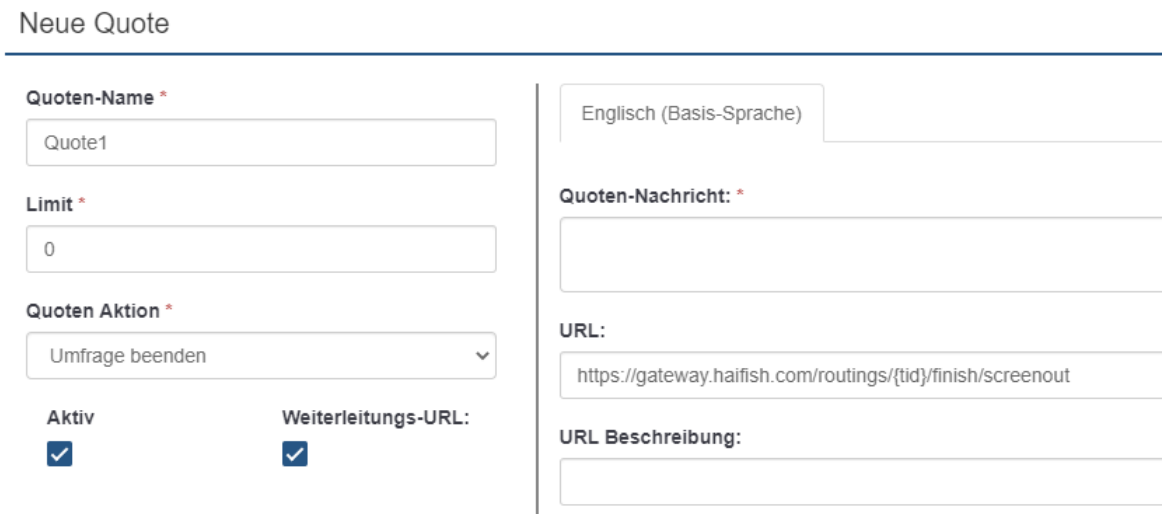
In einer vollständigen Umfrage leitet man die Teilnehmer mittels der sogenannten „end-url“ auf eine andere Seite und eine Variable bindet man mittels ExpressionManager/Script ein.

Damit würde die end-url in diesem Falle so aussehen:

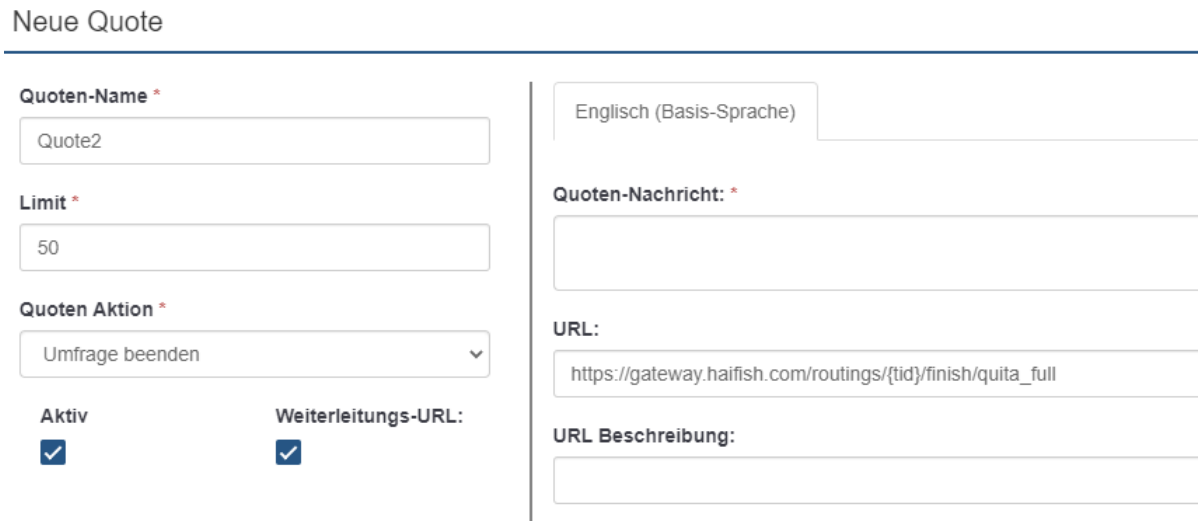
*https://gateway.haifish.com/routings/**{tid}**/finish/completed*



Sofern man Quoten angelegt hat, seien es „QuotaFull“-Quoten (50 Männer – 50 Frauen) oder „Screenout“-Quoten (0 Teilnehmer jünger als 18), die URL wird in diesem Fall im Quotenmanagemant angelegt



oder eine „QuotaFull“-Quote



Wichtig ist natürlich, dass

- Quotenmanagement: die Quote aktiv ist
- Quotenmanagement: die Weiterleitungs-URL eingeschaltet ist
- Präsentation: „Automatisch End-URL laden, wenn Studie abgeschlossen“ eingeschaltet ist

Ein anderes Beispiel (hier wird ein Parameter namens „m“ hin- und hergeschoben)

complete:

survey.minimiles.com/complete?p=82627_6b5db5d3&m={m}

screenout:

survey.minimiles.com/complete?p=82627&m={m}

5. Anhang

5.1. Wie erkennt man, welche Fragen- und Antwortcodes man (in Relevanzgleichungen u.ä) benutzen muss

Wenn man nicht genau weiß, wie Fragen/Teilfragen kodiert sind, wie Antworten gespeichert werden, kurz, wie der Zugriff darauf erfolgt, hilft dies ungemein.

Man aktiviert die Umfrage und gibt einige wohl definierte Daten ein. Wohl definiert heißt „Man notiert sich, was wo eingegeben wurde“.

Dann öffnet man die Antworttabelle in LimeSurvey und sieht die Benennung und Speicherung.

Q11 Liste(Optionsfelder)	Q11_other Liste(Optionsfelder) Overige
Option 1 [1]	
Overige [-oth-]	Tralala

Angezeigt wird immer die Nennung als Text und dahinter der Code in eckigen Klammern.

Außerdem sehen wir, dass der Code für die „Sonstige“-Option bei einer Einfachnennung „-oth-“ ist. In einer Relevanzgleichung/Bedingung könnte man nun als schreiben

Q11==1, um eine Frage anzuzeigen, wenn die Frage Q11 mit „Option 1“ beantwortet wurde.

Um irgendeine Frage anzuzeigen, wenn in Frage Q11 die „Sonstigen“-Option ausgewählt wurde, hat man nun zwei Möglichkeiten:
1. Q11==“-oth-“ oder
2. !is_empty(Q11_other) Dieses Feld ist nicht leer, wenn „Sonstiges“ gewählt wurde.

Und um den Text der „Sonstigen“-Nennung in einer späteren Frage enzublenden, benutzt man {Q11_other}.

Auch dies sieht man hier

Q12_SQ002 Mehrfach Teilfrage 2	Q12_SQ003 Mehrfach Teilfrage 3	Q12_other Mehrfach Overige
Ja [Y]		Trilili
	Ja [Y]	Trululu

Bei einer Mehrfachnennung ist es etwas anders. Hier gibt es für jede Teilfrage eine separate Spalte. Wird eine Teilfrage ausgewählt, wird „Y“ gespeichert, ansonsten bleibt die Spalte leer.
Und für das „Sonstige“ gibt es nur eine Textspalte, die entweder leer oder gefüllt ist.

Als Bedingungen können hier also in Frage kommen
Q12_SQ002==“Y“ (Teilfrage 2 wurde gewählt)
!is_empty(Q12_other) („Sonstiges“ wurde ausgewählt)