Abfragecode und Ausdrücke

Zugehörige Informationen			
Abfragecode und Ausdrücke, Ausdrücke in Prüfregeln, Ausdrücke in Skripten, Reguläre Ausdrücke	Allgemeine Bedienungshinweise	—	—
Abfragen			

Diese Seite informiert über die Möglichkeiten der **Formulierung von Abfragecode**.

Die Bedienung der Administrator-Maske zur Definition einer Abfrage wird hier nicht erklärt!

Grundsätzlicher Aufbau

Die Abfragen in ASYS werden nicht als SQL-Ausdrücke definiert, sondern - an die Komponenten eines SQL-Ausdrucks angelehnt - als Listen zeilenweiser Teilausdrücke für die verschiedenen Hauptbestandteile einer Datenbankabfrage angelegt. Ein Hauptgrund hierfür ist der modellbasierte Verarbeitungsmechanismus von ASYS, welcher zur Laufzeit aus modellierten Informationen ausführbaren Code generiert. Hierzu wird bei der Definition von Abfragen nicht mit den physischen Datenbanknamen von Tabellen, Sichten (Views), Attributen und (z.T.) Funktionsnamen gearbeitet, sondern mit den Modellnamen. Diese sind im ASYS-Datenmodell (Fachobjektemodell FOM) hinterlegt[1)]. Erst zum Zeitpunkt der Ausführung werden die Modellnamen gegen die physischen Namen der Datenbank ersetzt. Ebenso werden Abfrageparameter gegen die entsprechenden Werte aus dem jeweiligen Datensatz oder der Systemumgebung ersetzt. Je nach genutzter Datenbank werden dabei auch spezifische Datenbankeigenschaften (z.B. das Datumsformat) berücksichtigt. Das Resultat dieser Verarbeitung ist ein SQL-Ausdruck, passend zur jeweils genutzten Datenbank und zum Kontext, der datenbankunabhängig definiert werden kann[2)].

Die Hauptbestandteile einer Abfrage sind - in Analogie zu SQL-Ausdrücken:

Abschnitt	Inhalt	Anmerkung zum Administrator/Repository für ASYS V6.x/7.x
CLASSES	Klassen (Tabellen oder Views) des ASYS-Fachobjektemodells, deren Attribute in den nachfolgenden Abschnitten verwendet werden inkl. ihrer Beziehung und einem beliebigen Alias–Namen. (FROM-Abschnitt in SQL)	→ Tab-Reiter Code 1
RESULTS	Ergebnisse, die zurückgeliefert werden sollen inkl. ihres beliebigen Alias–Namens. (SELECT-Abschnitt in SQL)	→ Tab-Reiter Code 1
CONDITIONS	Bedingungen, die berücksichtigt werden sollen (WHERE-Klausel in SQL).	→ Tab-Reiter Code 1

Abschnitt	Inhalt	Anmerkung zum Administrator/Repository für ASYS V6.x/7.x
GROUP	Gruppierungsanweisung (GROUP BY-Klausel in SQL)	→ Tab-Reiter Code 1
ORDER	Sortieranweisung für die Ergebnisse (ORDER BY-Klausel in SQL)	→ Tab-Reiter Code 2
HAVING	Bedingungen, die Ergebnisse von SQL-Funktionen auswerten (HAVING-Klausel in SQL).	→ Tab-Reiter Code 2
VARIABLES	Nur für freie Abfragen: Formatierungsanweisungen für im CONDITIONS-Bereich verwendete Parameter	→ Tab-Reiter Code 2
VARIABLES.Value	Nur für Regelabfragen: Ermittlung eines Parameter-Wertes über eine ASYS-Funktion	→ Tab-Reiter Code 2
UNION	Abfragenverknüpfung (UNION select...)	→ Tab-Reiter Code 2
SCRIPTS	Anweisungen zur Erzeugung zusätzlicher Ergebnisspalten aus den ermittelten Daten (Beispiele s.u.).	→ Tab-Reiter Code 2

Die beiden **fett** gesetzten Abschnitte einer Abfrage **müssen** definiert werden. Alle anderen Abschnitte *können* genutzt werden, sind aber nicht verpflichtend zu füllen. Allerdings besteht - je nach abgefragten Klassen - bei einem leeren CONDITIONS-Abschnitt die Gefahr, dass sehr umfangreiche Treffermengen zu Stande kommen, da die Ergebnismenge nicht gefiltert wird.

Grundsätzliche Konventionen

Verwenden Sie für die in den nachfolgend erläuterten Abschnitten einzutragenden Codefragmente ausschließlich Standardzeichen des Alphabets!

Deutsche Sonderzeichen (äÄ, öÖ, üÜ, ß) sind unbedingt zu vermeiden, da die nicht als Platzhalter für Datenbankattribute dienenden Bestandteile der Abfragendefinition unverändert in die generierten SQL-Kommandos an die Datenbank durchgereicht werden. Die Datenbanken können aber die vom lateinischen Standardalphabet abweichenden Zeichen meist nicht korrekt verarbeiten.

Die Alias-Namen für Tabellen und Views (Abschnitt CLASSES) bzw. für Attribute (Abschnitt RESULTS) dürfen nur aus **Ziffern**, **Standardbuchstaben** und dem **Unterstrich** (_) bestehen.

Leerzeichen, Satzzeichen (!?,;. etc.) und Sonderzeichen (&%$§ etc.) sind nicht erlaubt!

Abschnitte

Abschnitt CLASSES

Beachten Sie die grundsätzlichen Konventionen.

Im CLASSES-Abschnitt werden die Objekte der Datenbank angegeben, welche die Quellen der abzufragenden Daten bilden. Für eine Abfrage **muss mindestens ein Objekt** angegeben werden, aus der die Datensätze für das Ergebnis genommen werden sollen. Die Klassen in ASYS-Abfragen entsprechen den im **Fachobjektemodell (FOM)** modellierten Objekten. Sie müssen in diesem Abschnitt mit dem **Namen aus dem FOM** eingetragen werden[3].

Klassen und Views im FOM

Einer Klasse im FOM sind in der Datenbank immer zwei Strukturen eindeutig zugeordnet: Eine Tabelle und eine Sicht (View). Der Unterschied zwischen beiden ist, dass in der Sicht nur die nicht als gelöscht markierten Datensätze enthalten sind während die Tabellen Zugriff auf alle Datensätze - auch die gelöscht markierten - bieten. Wird in ASYS ein Datensatz gelöscht, findet keine physische Löschung des Datensatzes in der Tabelle statt[4]. Statt dessen wird der Datensatz aktualisiert, wobei ihm eine Löschmarkierung mitgegeben wird. Derartig markierte Datensätze lassen sich in der ASYS-Anwenderoberfläche, mit Abfragen oder Detaillierten Suchen nicht mehr aufrufen. Sämtliche lesenden Zugriffe finden also immer nur auf den View statt.

Um die Anzahl an Tabellen, in denen gleichartige Daten verwaltet werden etwas überschaubarer zu halten sind an einigen Stellen im FOM mehrere Views definiert, die alle von derselben Klasse erben. Hierfür ist in der jeweiligen Klasse immer ein Attribut Rolle (Ganzzahl) definiert und jeder View erbt von der Klasse in einer ganz bestimmten Rolle. In der Datenbank gibt es für diese Views eine eigene Sicht (View) aber keine eigene Tabelle. Objektbeziehungen (abhängige Daten) im FOM erfolgen sowohl zu einer Klasse als auch zu einem View. Bei der Definiton einer Abfrage ist folgendes zu beachten: Ein View kennt alle Objektbeziehungen, die direkt auf ihn verweisen sowie alle Objektbeziehungen derjenigen Klasse von der der View erbt. Eine Klasse dagegen kennt nur die Objektbeziehungen, die direkt auf sie verweisen. Für die Definition im CLASSES-Abschnitt bietet es sich daher wahrscheinlich am häufigsten an, mit den Views (und nicht den Klassen) zu arbeiten. Nur für übergreifende Abfragen (z.B. alle Betriebe oder alle Abfalltransportbeteilgten - unabhängig von ihrer Rolle) kann es ggf. manchmal sinnvoll sein, mit der Klasse zu arbeiten und die Rolle im CONDITIONS-Abschnitt selber zu berücksichtigen oder auch nicht. Bzgl. der weiteren Defintion in einer Abfrage spielt es ansonsten keine Rolle, ob mit den Klassen oder Views gearbeitet wird. **Im folgenden wird daher immer nur von Klassen gesprochen.**

Die Namen einer Klasse und ihrer Attribute im FOM weichen gegenüber den Namen der Tabellen/Sichten(Views) in der Datenbank und ihrer Attribute voneinander ab. Sie werden von der mittleren Schicht zur Ausführungszeit einer Abfrage aus den FOM-Namen in die Datenbanknamen übersetzt. Theoretisch können für verschiedene Datenbanken (Oracle, SQL-Server, Access) unterschiedliche Datenbanknamen für die gleichen Tabellen, Views oder Attribute definiert sein (in der Praxis wird von dieser Möglichkeit kein Gebrauch gemacht).

Sämtliche Attribute aus der Datenbank, die in den weiteren Abschnitten RESULTS, CONDITIONS, GROUP, ORDER oder HAVING verwendet werden sollen, **müssen** aus einer Klasse stammen, die in diesem Abschnitt eingetragen wurde! Andersherum müssen Sie daher alle Klassen, aus denen Sie Attribute in den weiteren Abfrageabschnitten verwenden wollen, in diesem Abschnitt angeben.

Viele Fachobjekte - z.B. der Entsorgerbetrieb - bestehen nicht nur aus einem Datensatz in einer Tabelle bzw. Klasse, sondern besitzen eine komplexere Struktur aus einem Hauptdatensatz und einer variablen Anzahl von mit diesem Hauptdatensatz verknüpften Unterdatensätzen in eigenen Klassen[5)]. Derartige hierarchische Beziehungen zwischen den Klassen bzw. Datensätzen in den Datenbanktabellen können mehrstufig sein - z.B. FKB → Entsorgerbetrieb → Entsorger → Teilanlage → zugelassener Abfall.

Die Ausgangsklasse einer Abfrage wird einfach mit ihrem FOM-Namen in diesem Abschnitt eingetragen. Die weiteren Klassen werden - soweit möglich - so eingetragen, dass die **Beziehungen zwischen den Klassen** über einen **Klassenpfad in Punktnotiation** deutlich werden:

```
Klasse_A
Klasse_A.Klasse_B
```

- Die Art der Verknüpfung zwischen den beiden Tabellen ist im FOM modelliert und wird von ASYS automatisch zur Ausführungszeit in den passende SQL-Teilausdruck übersetzt, wenn die Beziehung zwischen den Klassen in dieser Punktnotation eingetragen wird.
- Damit diese Übersetzung funktioniert, muss es auch tatsächlich eine Beziehung zwischen den beiden Klassen im FOM geben. Die Eingabehilfe bietet nur FOM-Beziehungen an.
- In vielen Fallen ist die Klasse A der übergeordnete Datensatz, die Klasse B repräsentiert untergeordnete (abhängige) Datensätze. Vielfach besteht eine 1:0..n-Beziehung, d.h. zu einem übergeordneten Datensatz der Klasse A gehören Null bis beliebig viele abhängige Datensätze. Die untergeordneten Datensätze enthalten die Information, zu welchem übergeordneten sie gehören.
- Die Punktnotation ist nicht auf zwei Klassen beschränkt. In einer Zeile kann eine komplette Kette von der Klassenwurzel bis zu einem 'Blatt' über mehrere Stufen angegeben werden (entsprechend dem obigen Beispiel: *FKB.Rolle Betrieb Ent.Entsorger.Teilanlage.Abfall TA=Ata*).

Hinter dem Klassenpfad in Punktnotation kann für die letzte im Pfad angegebene Klasse ein **Klassenalias** angegeben werden:

```
Klasse_A=Alias_A
Klasse_A.Klasse_B=Alias_B
```

Die Aliasnamen müssen innerhalb einer Abfrage eindeutig sein! Die Alias-Namen werden in den nachfolgenden Abschnitten bei der Identifizierung der Attribute der FOM-Klassen verwendet.

Wir empfehlen, in jedem Fall Aliasnamen zu vergeben! (ohne Angabe eines Aliasnamen, vergibt der Administrator beim Verlassen des Eingabefeldes automatisch einen Aliasnamen XX[Zahl]) In der Praxis haben sich 3-Zeichen-Aliasnamen bewährt, weil einerseits genügend eindeutige dreibuchstabige Zeichenkombinationen zur Verfügung stehen und andererseits bei geeigneter Wahl der Buchstabenfolge der Name der Klassen hinreichend selbsterklärend abgekürzt werden kann.

1. Wie im vorstehenden Beispiel angedeutet, ergeben die Klassen in einer Abfrage mit ihren Pfaden eine Art Baum. Die Wurzel dieses Baumes muss

dabei nicht unbedingt die Hauptklasse eines Datensatzes sein - also für Stammdaten z.B. *FKB* - sondern kann auch eine beliebige andere Klasse bilden.
Wichtig: Die einzige Einschränkung bilden Left-Join-Bedingungen zwischen Klassen, die weiter unten erklärt werden.

2. Ist eine *Klasse_A* die Wurzel des Klassenbaums und eine *Klasse_D* das 'Blatt', welche über *Klasse_B* und *Klasse_C* miteinander verknüpft sind, so müssen alle vier Klassen in diesem Abschnitt in einer eigenen Zeile aufgeführt werden (siehe nachfolgendes Beispiel).
3. Die Pfade einer Klasse können nach Bedarf vollständig oder verkürzt angegeben werden. Die verkürzte Form ist nur zulässig, wenn die fortgelassenen Klassenbeziehungen in anderen Zeile des CLASSES-Abschnitts definiert sind.
   1. vollständig:
      Klasse_A=Alias_A
      Klasse_A.Klasse_B=Alias_B
      Klasse_A.Klasse_B.Klasse_C=Alias_C
      Klasse_A.Klasse_B.Klasse_C.Klasse_D=Alias_D
   2. verkürzt:
      Klasse_A=Alias_A
      Klasse_A.Klasse_B=Alias_B
      Klasse_B.Klasse_C=Alias_C
      Klasse_C.Klasse_D=Alias_D
      Wichtig: Wird die gleiche Klasse in einer Abfrage über verschiedene Pfade eingebunden (z.B. *Adresse.Gemeindekatalog* einmal via *FKB* und einmal via *Betrieb*), so müssen die Pfade auch entsprechend eindeutig angegeben werden!
4. Die Klassen in diesem Abschnitt **müssen nicht** alle über die Punktnotation miteinander verknüpft werden! Es können auch unverbundene Klassen oder zwei oder mehr Verknüpfungsbäume parallel in einer Abfrage definiert werden:
   Klasse_A=Alias_A
   Klasse_A.Klasse_B=Alias_B
   Klasse_C=Alias_C
   Klasse_D=Alias_D
   Klasse_D.Klasse_E=Alias_E
   Wie die Klassen A&B mit Klasse C bzw. den Klassen D&E verknüpft sind, muss dann im CONDITIONS-Abschnitt festgelegt werden! Damit lassen sich Klassen verknüpfen, die im ASYS-Datenmodell keine Verbindung besitzen.

Inner-Join- und Left-Join-Ausdrücke

Herkömmliche Abfragen sind immer als **Inner-Join**-Ausdrücke formuliert. Darunter ist zu verstehen, dass nur Datensätze in die Auswahl kommen, die in allen in der Abfrage miteinander verknüpften Klassen einen oder mehrere Datensätze enthalten[6)]. Unabhängig von den selbst definierten Auswahlbedingungen für die Datensätze im CONDITIONS-Abschnitt werden nur Datensätze in die Trefferliste aufgenommen, die diese Vorbedingung erfüllen, wobei diese Auswahl automatisch von der Datenbank vorgenommen wird. Es werden also ggf. Datensätze nicht in die Trefferliste

aufgenommen, die zwar alle CONDITIONS-Ausdrücke erfüllen, aber in einer oder mehreren der verknüpften Klassen keine Daten enthalten. Ob dem so ist, kann nur festgestellt werden, wenn die 'leeren' Klassen aus dem CLASSES-Abschnitt wieder entfernt würden, was meist den Zweck der Abfrage konterkariert.

Zur bedarfsweisen Überwindung der Einschränkungen eines Inner-Join erlauben die Datenbanken auch **Left-Join**-Ausdrücke. Hierbei werden zwei Tabellen (linke Tabelle und rechte Tabelle) so miteinander verknüpft, dass alle Datensätze aus der linken Tabelle genommen werden - unter Berücksichtigung der übrigen CONDITIONS-Ausdrücke natürlich - und aus der rechten Tabelle die Datensätze, die mit den Datensätzen der linken Tabelle verknüpft sind. Falls dabei ein Datensatz in der linken Tabelle keine verknüpften Daten in der rechten Tabelle besitzt, wird er nicht automatisch entfernt sondern die Attribute der rechten Klasse werden in der Trefferliste für den Datensatz leer gelassen.

In ASYS gibt es im Datenmodell diverse Stellen, an denen einem übergeordneten Datensatz abhängige Datensätze zugeordnet werden können aber nicht müssen. Es wird dabei immer wieder Datensätze geben, für die aus unterschiedlichen Gründen keine abhängigen Daten eingetragen sind. Wird die Tabelle der abhängigen Daten mittels Inner-Join in eine Abfrage eingebunden, fallen a priori alle Datensätze aus dem Ergebnis heraus, die in dieser abhängigen Tabelle keine Einträge aufweisen (z.B. Ansprechpartner in Betrieb oder FKB, 4.BImSchV-Nummern für Teilanlagen). Wird die abhängige Tabelle hingegen mittels Left-Join eingebunden können derartige Datensätze wieder in die Treffermenge aufgenommen werden, sofern sie die übrigen Auswahlbedingungen erfüllen.

Zur Definition einer Left-Join-Beziehung wird am Ende der Zeile ein * ergänzt:

```
Klasse_A=Alias_A
Klasse_A.Klasse_B=Alias_B*
Klasse_a.Klasse_B.Klasse_C=Alias_C
```

In diesem Beispiel ist die *Klasse_B* per Left-Join mit der *Klasse_A* verbunden. Die *Klasse_C* ist per Inner-Join mit der *Klasse_B* verknüpft. Per Left-Join verknüpfte Klassen können also selbst wieder abhängige Klassen besitzen, für die unabhängig die Inner-Join oder Left-Join-Verknüpfung eingestellt werden kann.

1. Werden im CONDITIONS-Abschnitt Bedingungen auf Attribute aus einer per Left-Join verknüpften Klasse (oder ihrer nachgeordnet verknüpften Klassen) eingetragen, so wird dadurch die besondere Eigenschaft der Left-Join-Verknüpfung **aufgehoben**! Die Bedingung kann nur von vorhandenen Daten erfüllt werden, weshalb alle Datensätze ohne Einträge in der per Left-Join verknüpften Klasse ausgefiltert werden. Das Ergebnis entspricht also wieder einer Inner-Join-Verknüpfung.
2. Bei einer Left-Join-Verknüpfung zwischen Klassen kommt es auf die Reihenfolge der Klassen in der Verknüpfung an. Die am weitesten rechts stehende Klasse ist diejenige, in der nur optional Daten enthalten sein dürfen, die links davon stehende Klasse ist diejenige, aus der die Daten auch dann in die Treffermenge fließen, wenn in der rechten Klasse keine zugehörigen Daten stehen. Kehrt man die Reihenfolge um, wechseln die beiden Klassen entsprechend auch diese Eigenschaften. Dies ist beim Aufbau des Klassenbaums zu beachten!

Abschnitt RESULTS

Beachten Sie die grundsätzlichen Konventionen.

Im RESULTS-Abschnitt wird festgelegt, welche Attribute aus den im CLASSES-Abschnitt eingetragenen Datenbankklassen in der Treffermenge erscheinen. Damit eine Abfrage ein Ergebnis liefern kann **muss mindestens ein Attribut** in diesem Abschnitt eingetragen werden. Jedes Attribut ist in einer Zeile einzutragen. Die Reihenfolge der Zeilen bestimmt die Reihenfolge der Spalten in der Treffermenge.

1. **Der Attributalias ist eine notwendige Angabe!** Er **muss** innerhalb einer Abfragendefinition **eindeutig** sein. Wird im Repository-Administrator kein Attributalias vergeben, generiert das Programm automatisch einen aus dem Klassenalias und dem Attributnamen. Automatisch generierte Aliasse müssen durch den Fachadministrator auf Eindeutigkeit geprüft werden!
2. In Abfragen für **Textformulare** müssen die Attributaliasse in **GROSSBUCHSTABEN** geschrieben werden! Dies verhindert Probleme, die gelegentlich auftreten, wenn das gleiche Attribut mehrfach mit verschiedenen Aliassen ausgegeben wird!

Für die Notation der Ergebnisspalten gibt es zwei Varianten - eine einfache und eine komplexere:

Einfache Notation

Die einfache Notation ist für die einfachen Anwendungsfälle vorgesehen, bei denen der Attributwert unverarbeitet in die Ergebnisspalte überführt werden soll. Die einfache Notation lautet

```
Klassenalias.Attributname=Attributalias
```

Die Eingabehilfe im Administrator hilft bei der Auswahl der bekannten Klassenaliase und anschließend bei der Auswahl der Klassenattribute der zugehörigen Klassen.

Komplexe Notation

Die komplexere Notation ist immer dann zu verwenden, wenn der Attributwert verarbeitet werden muss, bevor er in die Ergebnisspalte geschrieben wird. Die komplexe Notation lautet

```
[*|$]Attributalias[/Typmodifikator]=[Funktion(){%Klassenalias.Attribut%}|Konstante|#@heute#[)]
```

Die Eingabehilfe im Administrator unterstützt auch bei der Formulierung derartiger Ausdrücke.

Typangabe vor dem Aliasnamen

Für das **erste Zeichen** gibt es die Alternativen * und $:

- ***** : Der Attributwert ist eine Ganzzahl (Integer, Long)
- **$** : Der Attributwert wird als Zeichenkette (String) interpretiert (für alle Attributwerte außer Ganzzahlen zu verwenden)

Die Typangabe vor dem Aliasnamen ist eine **Pflichtangabe**.

Typmodifikator hinter dem Aliasnamen

Hinter dem Attributalias kann ein **Typmodifikator** stehen. Geben Sie hinter dem Aliasnamen das Zeichen **/** ein und drücken Sie **Strg+Leertaste**, um die zugehörige kontextsensitive Eingabehilfe aufzurufen. Sie enthält die erlaubten Typmodifikatoren samt kurzer Erläuterung.

Der Typmodifikator ist optional zu verwenden, wenn der entsprechende Ergebnisspaltentyp auf Grund des Attributtyps benötigt wird.

Funktion

Die **Funktion** verarbeitet optional den Inhalt des Attributs, der Konstante oder des aktuellen Datums. Hinter den Funktionsnamen ist eine öffnende runde Klammer zu setzen, die zugehörige schließende Klammer kommt hinter den Funktionsparameter. Wird die Eingabehilfe nach dem Gleichheitszeichen geöffnet, bietet der Administrator eine Auswahl häufig genutzter Funktionen an[7)]. Die angebotenen Funktionen ergeben sich aus dem aktuell angebotenen Katalog, der durch die kontextsensitive Eingabehilfe via Strg+Leertaste aufgerufen werden kann (s. Bedienungsanleitung zu Abfragen).

Grundsätzlich lassen sich die Funktionen auch ineinander schachteln, so dass das Ergebnis der inneren Funktion ein Parameter für die äußere ist. Der Aufruf

```
[...]=@nvl(@substring({%Klassenalias.Attributname%}, 1, 3), '---')
```

liefert beispielsweise die ersten drei Zeichen des Attributs *Klassenalias.Attributname* oder - - -, falls das Attribut leer ist.

Weitere Funktionen werden durch die jeweilige Datenbank zur Verfügung gestellt. Sie können genauso wie die hier dokumentierten verwendet werden, allerdings gibt es keine Unterstützung durch den Repository-Administrator oder Governikus ITU. Hinsichtlich der Dokumentation sei hiermit auf Hilfeseiten der Datenbankhersteller im Internet verwiesen.

Konstante Werte, Anführungszeichen und Hashmarks

Neben einem Verweis auf ein Klassenattribut kann auch ein konstanter Wert an einen Attributalias übergeben werden. Alle Datensätze der Trefferliste werden diesen konstanten Wert in dieser Spalte aufweisen. Die auf diese Weise definierte Trefferlistenspalte wird wie jede andere Spalte ausgegeben, auch wenn den Inhalt nicht aus den Datensätzen stammt, welche die Trefferliste bilden.

Wird der Attributalias als Zeichenkette getypt ($ vor dem Aliasnamen), so ist der konstante Wert in Anführungszeichen zu setzen:

```
$Attributalias='Konstanter Wert'
```

Handelt es sich bei der Konstante um einen Boolschen Wert (true oder false), so ist der Wert in Hashmarks zu setzen:

```
*Attributalias=#true#  //oder #false#
```

Für Datumswerte sind ebenso Hashmarks zu setzen:

```
$Attributalias=#01.01.1970#
$Attributalias=#@heute#
```

Auch wenn das letzte Beispiel kein konstanter Wert ist, da hier das jeweils aktuelle Systemdatum ausgegeben wird, so demonstriert es doch die Verwendung von Hashmarks mit Datumswerten.

Aggregatfunktionen

Einige Funktionen sind **Aggregatfunktionen**. Sie werden regelmäßig genutzt, wenn eine Abfrage mit Gruppierungsanweisungen im GROUP-Abschnitt versehen ist.

Alle Ergebnisspalten, die nicht in der Gruppierungsanweisung aufgeführt sind, **müssen** mit einer Aggregatfunktion definiert werden!

Bei einer Gruppierung werden die Daten einer Gruppe zusammengefasst zu einem Datensatz in der Trefferliste. In der Trefferliste gibt es zwei Arten von Spalten:

1. Die Werte der Trefferlistenspalten nach denen gruppiert wird, sind für die alle Datensätze einer Gruppe in der Gruppierungsspalte definitionsgemäß identisch - das ist das Wesen einer Gruppierung. Für diese Spalten ist der Ausgabewert eindeutig und bedarf keiner Präzisierung.
2. Für alle Trefferlistenspalten, nach denen **nicht** gruppiert wird, sind die Werte pro Spalte nicht eindeutig, da jeder der Datensätze, die eine Gruppe bilden, darin einen anderen Wert enthalten kann. Hier muss der Datenbank mitgeteilt werden, wie der Spaltenwert zur Ausgabe des einen aggregierten Gruppendatensatzes in der Trefferliste ermittelt werden soll. Dafür sind die Aggregatfunktionen zu verwenden, die festlegen, wie je Gruppe der Wert der Spalte ermittelt wird. **Wichtig**: Bei mehreren aggregierten Spalten können die ausgegebenen Spaltenwerte aus unterschiedlichen Datensätzen einer Gruppe stammen[8)]!

Spezielle Notationen für ID-Felder

Für den Zugriff auf die ID-Felder einer Klasse ist eine spezielle Notation zu verwenden.

```
Klassenalias.=Attributalias
```

oder

```
*Attributalias={%Klassenalias.#%}
```

In der einfachen Notation wird also einfach hinter den Klassenalias ein Punkt geschrieben. In der komplexeren Notation muss hinter dem Klassenalias noch ein **#** geschrieben werden.

Abschnitt CONDITIONS

Im CONDITIONS-Abschnitt werden Bedingungen formuliert, denen die Datensätze in der Trefferliste genügen müssen. Diese Bedingungen haben zwei Aufgaben:

1. Die **Filterung** der Datensätze. Es werden nur die Datensätze aus den im CLASSES-Abschnitt angegebenen Klassen in die Trefferliste aufgenommen, die **allen Bedingungen** genügen.
2. Die **Verknüpfung** von Klassen. Über derartige Bedingungen wird festgelegt, wie die Datensätze zweier Klassen, die nicht im Datenmodell bzw. im CLASSES-Abschnitt verknüpft sind, miteinander in Beziehung stehen. So lassen sich beispielsweise mit einer Bedingung die Begleitscheine und die Entsorgungsnachweise über die Nachweisnummer verknüpfen.

Eine Abfrage **kann ohne** Bedingungen definiert werden. Dann werden alle Datensätze der einbezogenen Klassen zu Bestandteilen der Treffermenge. Durch das Kreuzprodukt bei mehreren Klassen (s.a. die zugehörige Fußnote auf dieser Seite) - und besonders bei unverknüpften! - können dabei sehr große Treffermengen entstehen[9)]! In der überwiegenden Mehrzahl der sinnvollen Anwendungsfälle werden daher Bedingungen definiert werden müssen.

Es gilt:

- **Jede Zeile** im CONDITIONS-Abschnitt stellt **eine Bedingung** dar.
- Mehrere Zeilen in diesem Abschnitt werden jeweils durch den logischen Operator **UND** (and) miteinander verknüpft. Jeder Datensatz der Trefferliste muss daher alle Bedingungen = alle Zeilen erfüllen.
- Alternative Bedingungen mit dem logischen Operator **ODER** (or) sind möglich, wenn sie in eine Zeile des CONDITIONS-Abschnitt geschrieben werden.

```
Bedingung A
(Bedingung B or Bedingung C)
Bedingung D
```

Dieses abstrakte Beispiel bedeutet: Alle Datensätze der Trefferliste müssen *Bedingung A* und mindestens eine von *Bedingung B* oder *Bedingung C* sowie die *Bedingung D* erfüllen. Die ODER-Bedingung der zweiten Zeile wird vor den UND-Bedingungen zwischen den drei Zeilen ausgewertet.

Diese Bedingung der Form *A* und *B* oder *C* und *D* kann auch anders gemeint sein:

```
((Bedingung A and Bedingung B) or (Bedingung C and Bedingung D))
```

Dies bedeutet, dass mindestens eines der beiden Bedingungspärchen *A und B* bzw. *C und D* erfüllt sein muss. Diese Form muss in eine Zeile geschrieben werden, weil hier die ODER-Bedingung nach den beiden UND-Bedingungen ausgewertet werden soll. Die beiden inneren Klammerpärchen sollten immer geschrieben werden, das äußere Klammerpärchen sollte geschrieben werden, wenn diese Bedingung durch weitere ergänzt wird, um die Interpretation eindeutig zu machen.

Im CONDITIONS-Abschnitt kommen zwei unterschiedliche Platzhalternotationen regelmäßig zum Einsatz:

1. Die **{%…%}**-Notation steht für **Attribute aus der Datenbank**. Angegeben werden diese in Punktnotation (vrgl. RESULTS-Abschnitt) als Kombination aus Klassenalias und Attributname aus dem ASYS-Fachobjektemodell, also {%Klassenalias.Attributname%}. Dieser Platzhalter wird bei der Ausführung gegen die Datenbanknamen der Tabelle/View und ihres Attributs ersetzt. Die Eingabehilfe des Administrators unterstützt bei der Auswahl korrekter Klassenaliasse und Attributnamen.
   Jede Bedingung besitzt einen Bezug auf mindestens ein Attribut der Datenbank. Dazu ist **immer** diese Platzhalternotation in den Abfragen zu verwenden.
2. Die **{*…*}**-Notation steht für **Abfrageparameter**. Abfrageparameter sind Bedingungswerte, die erst zur Ablaufzeit der Abfrage feststehen. Bei freien Abfrage müssen diese durch den Nutzer der Abfrage festgelegt werden. Bei internen Abfragen werden sie aus dem Datensatzkontext genommen, in dem die Abfrage ausgeführt wird.
    1. Für die Parameter einer freien Abfrage kann einfach ein selbstbeschreibender Name eingesetzt werden, also **{*Parametername*}**. Der Name ist eindeutig je Abfrage zu vergeben.
    2. Für die Parameter aus dem Datensatzkontext ist die Punktnotation zu verwenden, also **{*Klassenname.Attributname*}**. Dieser Platzhalter wird bei der Ausführung gegen den Wert aus dem Datensatzkontext der Maske oder der Nachricht ersetzt. Die Eingabehilfe des Administrators unterstützt bei der Auswahl korrekter Klassen- und Attributnamen.

Besonderheiten für Datumsattribute

1. Bedingungen mit Datumswerten sind **immer** durch **#$** am Beginn und **#** am Ende des Vergleichs einzurahmen. Dies aktiviert die Umformung des Datumswertes in das spezifische Datumsformat der jeweiligen Datenbank.
    1. Einzige **Ausnahme**: Das Datumsattribut wird mit **is null** oder **is not null** verglichen. Dann ist die Standardschreibweise ohne *#$…#* zu verwenden.
2. Datumsangaben sind vielfach **optional**. Leere Datumsfelder werden in Bedingungen als nicht erfüllt gewertet, egal welcher Vergleichswert angegeben wird. Sie müssen durch eine per ODER angehängte Zusatzbedingung wieder einbezogen werden, die explizit die Bedingung stellt, dass das Datumsfeld leer (*null*) ist. Diese Zusatzprüfung lässt sich durch eine Kurznotation automatisch ergänzen, wenn der Bedingung durch **#?$** (anstatt #$) eingeleitet wird.
3. Mit Datumswerten kann gerechnet werden: zu bzw. von einem Datumswert lassen sich Tage (**t**), Monate(**m**) und Jahre(**j**) addieren bzw. subtrahieren. Auch eine Kombination dieser Verrechnungszeiträume ist möglich wenn sie per Komma separiert hintereinander geschrieben werden. Vor die jeweilige Zahl ist ein **+** oder **-** für die Berechnung zu schreiben, die Einheit (t, m oder j) ist ohne Leerzeichen hinter den Wert zu schreiben.

Beipiele (der Vergleichsoperator **>** ist nach Bedarf durch den jeweils passenden zu ersetzen):

```
Besonderheit 1.: #${%Klassenalias.Datumsattribut%} > {*Vergleichsdatum*}#
Besonderheit 1.I: {%Klassenalias.Datumsattribut%} is null ==> Vergleich
gegen NULL muss nicht ausgezeichnet werden
Besonderheit 2.: #?${%Klassenalias.Datumsattribut%} > {*Vergleichsdatum*}#
==> (#${%Klassenalias.Datumsattribut%} > {*Vergleichsdatum*}#) or
{%Klassenalias.Datumsattribut%} is null
Besonderheit 3.: #${%Klassenalias.Datumsattribut%}+1j >
```

```
{*Vergleichsdatum*}-1t# ==> zum Wert von Attributname wird 1 Jahr addiert
und beim Vergleichsdatum 1 Tag abgezogen
```

Operatoren

Eine **Bedingung** besteht immer aus einem Vergleich zwischen zwei Werten (bestimmte Sonderfälle werden weiter unten behandelt). Zu einem Vergleich gehört dabei immer ein **Vergleichsoperator**. Eine Übersicht über die gebräuchlichen Operatoren findet sich in der Anwenderhilfe zur Datenbereichssuche[10]. Die Aufstellung zu den Operatoren in der dortigen Tabelle gilt in gleicher Weise für Abfragen[11]

sc-Funktionen

Manche Werte für Vergleiche lassen sich nicht aus der Datenbank, dem aktuellen Datensatz oder als konstanter Wert festlegen. Einige kontextabhängige Werte sind mittels Funktionen über das ASYS-internen Objekt sc erreichbar und können als **Abfrageparameter** in der **{*...*}**-Notation verwendet werden. Zu den sc-Funktionen siehe die Aufstellung zum Prüfregelcode.

Weitere Hinweise

1. Wenn im CLASSES-Abschnitt Klassen oder Klassenbäume enthalten sind, die nicht über die dort zu verwendende Punktnotation miteinander verbunden sind (s. Ziffer 4 des dortigen Info-Kastens), dann ist **dringend zu empfehlen**[12], in diesem Abschnitt Bedingungen für die Verknüpfung zwischen den Klassen manuell anzugeben. Hierfür kommen insbesondere Klassenattribute in Frage, die in zumindest einer der beiden zu verknüpfenden Klassen als **logischer Schlüssel** (**lkey**-Auszeichnung im Fachobjektemodell) gekennzeichnet sind. Der kombinierte Inhalt aller Attribute einer Klasse, die als logischer Schlüssel gekennzeichnet ist, muss eindeutig unter allen Datensätzen der Klasse sein.
2. Wenn im CLASSES-Abschnitt einzelne Klassen oder Teile von Klassenbäumen mittels **Left-Join** angebunden werden, so ist zu beachten, dass Bedingungen auf Attribute dieser Klassen die Left-Join-Eigenschaft der Klassenverknüpfung wieder aufheben!
3. Werden **Ankreuzfelder** (Attribute von Typ *Boolean* im Fachobjektemodell, meist mit einem *FL_*-Präfix im Attributnamen) in Bedingungen eingebunden, so sind die zulässigen Vergleichswerte **#true#** und **#false#** (inklusive der **#**!) zu verwenden, also *{%Klassenname.BoolschesAttribut%} = #true#* (oder *#false#*).
4. Wird im CONDITIONS-Abschnitt mit konstanten **Zeichenketten** gearbeitet, so sind einige Besonderheiten zu beachten:
    1. Wird als Vergleichswert eine konstante Zeichenkette verwendet, so **muss** sie in einfache Hochkommata gesetzt werden: {%Klassenname.Attributname%} = 'Vergleichswert'
    2. Wird als Vergleichswert das Ergebnis einer **sc-Funktion** verwendet, so **muss** sie in einfache Hochkommata gesetzt werden: {%Klassenname.Attributname%} = '{*sc.Funktionsname*}'
    3. Wird als Vergleichswert ein Parameter **{*...*}** mit einem Zeichenketteninhalt verwendet, so hängt es vom Einsatzzweck der Abfrage ab, ob Hochkommata notwendig sind:
        1. Bei Abfragen für **Prüfregeln**, **Empfängerermittlung**, **Vorgangssteuerung** (Ausnahme: Bearbeiterermittlung) und **Textformulare** sind **keine Hochkommata** zu setzen, da sie von der ASYS-Mittelschicht automatisch eingetragen werden:

{%Klassenname.Attributname%} = {*Parametername*} (oder {*Klassenname.Attributname*}).
2. Bei **freien Abfragen** und Abfragen, die für die **Bearbeiterermittlung einer Vorgangssteuerung** eingesetzt werden, **sind immer Hochkommata zu setzen**: {%Klassenname.Attributname%} = '{*Parametername*}' (oder '{*Klassenname.Attributname*}').

Abschnitt GROUP

Gruppierung bedeutet, dass alle Datensätze mit gleichem Inhalt in dem/den Gruppierungsattribut(en) zu einem Datensatz zusammengefasst werden und damit in der Trefferliste nur noch einen Datensatz bilden. Die Datensätze der Trefferliste einer Abfrage **können** nach einem oder mehreren Attributen gruppiert werden. Wird nicht gruppiert, bleibt dieser Abschnitt leer.

Wird gruppiert, so **müssen** alle Gruppierungsattribute aus Klassen stammen, die im CLASSES-Abschnitt aufgeführt sind. Die Gruppierungsattribute müssen nicht im RESULTS-Abschnitt enthalten sein (auch wenn dies in der Praxis regelmäßig der Fall ist). Sie müssen auch nicht aus nur einer Klasse kommen.

Werden **mehrere Gruppierungsattribute** angegeben, so werden sie in diesem Abschnitt **alle in eine Zeile** geschrieben und durch Kommata getrennt:

```
{%Klassenname.Gruppierungsattribut1%},
{%Klassenname.Gruppierungsattribut2%}, ...
```

Hinweise

1. Wie im Abschnitt RESULTS - Aggregatfunktionen erläutert, müssen **alle** anderen Ausgabeattribute, die nicht Gruppierungsattribute sind, durch eine Aggregatfunktion ergänzt werden, damit die Datenbank die Ausgabewerte für die Attribute in dem einen Gruppendatensatz der Trefferliste eindeutig bestimmen kann.
2. Die Gruppierung muss nicht notwendigerweise an Hand unverarbeiteter Attributwerte erfolgen. Es ist auch zulässig, den Inhalt der Gruppierungsattribute durch Funktionen zu modifizieren: *@substring({%Klassenname.Attributname%}, 1, 1)* gruppiert beispielsweise nach dem ersten Zeichen des Attributwertes

Abschnitt ORDER

In diesem Abschnitt kann die Sortierung der Datensätze in der Trefferliste festgelegt werden. Die Trefferliste **kann** nach einem oder mehreren Attributen sortiert werden. Wird nicht sortiert, so werden die Datensätze in der Reihenfolge ausgegeben, wie die Datenbank sie zurückliefert und dieser Abschnitt bleibt leer.

Wird sortiert, so **müssen** alle Sortierungsattribute aus Klassen stammen, die im CLASSES-Abschnitt aufgeführt sind. Die Sortierungsattribute müssen nicht im RESULTS-Abschnitt enthalten sein (auch wenn dies häufig der Fall ist). Sie müssen auch nicht aus nur einer Klasse kommen.

Werden **mehrere Sortierungsattribute** angegeben, so werden sie in diesem Abschnitt **alle in eine**

Zeile geschrieben und durch Kommata getrennt:

```
{%Klassenname.Sortierungsattribut1%} ASC,
{%Klassenname.Sortierungsattribut2%} DESC, ...
```

Hinter jedem Attribut ist anzugeben, wie sortiert werden soll:

- **ASC (=Ascending)**: Aufsteigende sortieren, die kleinsten Werte stehen in der Trefferliste zuoberst (diese Angabe ist optional, ohne Sortierangabe hinter dem Attribut wird aufsteigend (ASC) sortiert).
- **DESC (=Descending)**: Absteigend sortieren, die größten Werte stehen in der Trefferliste zuoberst.

Achtung: Wird nach mehreren Attributen sortiert, so kommt es auf die **Reihenfolge der Sortierungsattribute** in diesem Abschnitt an! Alle Datensätze der Trefferliste werden zunächst nach dem ersten Sortierungsattribut sortiert. Alle Datensätze mit gleichen Werten im ersten Sortierungsattribut werden anschließend nach dem zweiten Sortierungsattribut sortiert. Entsprechend geht es mit dem dritten und allen weiteren Sortierungsattributen weiter.

Abschnitt HAVING

Im HAVING-Abschnitt **können** Bedingungen definiert werden, die erst nach der Ermittlung der Treffermenge auf die Datensätze darin zur Anwendung kommen. Im Unterschied dazu werden die Bedingungen im CONDITIONS-Abschnitt auf jeden einzelnen Datensatz der Datenbank angewandt, also vor der Ermittlung der Treffermenge.

Zum Einsatz kommen HAVING-Bedingungen immer dann, wenn sich die Bedingung nicht am einzelnen Datensatz in der Datenbank, sondern erst im gefilterten und gruppierten Ergebnis prüfen lässt. Durch die Gruppierung werden Datensätze der Datenbank zu einem Gruppendatensatz in der Treffermenge zusammengefasst. Der Wert der Ausgabeattribute eines Gruppendatensatzes in der Trefferliste ergibt sich dabei erst nach Abschluss der Filterung und Gruppierung, nicht selten als Ergebnis von Aggregatfunktionen.

Aber auch hier gilt: Der Ausdrücke, die für eine HAVING-Bedingung herangezogen wird, müssen nicht im RESULTS-Abschnitt enthalten sein. Sie müssen sich aber aus den Gruppen ermitteln lassen!

Die HAVING-Bedingung(en) ist/sind in **eine Zeile** zu schreiben. Werden mehrere Bedingungen angegeben, so sind sie geeignet durch **and** (UND) und **or** (ODER) logisch zu verknüpfen.

Beispiele ({%Klassenname.Attributname%} soll je Beispiel nur genau ein Attribut identifizieren, also nicht mehrere aus einer oder mehreren Klassen):

1. count(*) > 10 ⟹ Gib nur Gruppen aus, die aus mehr als 10 Datensätzen gebildet wurden
2. count({%Klassenname.Attributname%}) = 0 ⟹ Gib nur Gruppen aus, die im angegebenen Attribut keinen Inhalt aufweisen
3. (count(*) > 10) and (@substring({%Klassenname.Attributname%}, 1, 1) < 'F') ⟹ Nur Gruppen mit mehr als 10 Datensätzen und aus den Gruppen 'A' bis 'E' (funktioniert nur, wenn auch *@substring({%Klassenname.Attributname%}, 1, 1)* im GROUP-Abschnitt eingetragen ist!)
4. (@substring({%Klassenname.Attributname%}, 1, 1) < 'F') and (@substring({%Klassenname.Attributname%}, 4, 1) <> 'T') ⟹ soll alle Gruppen mit 'A' bis 'E' an Position 1 ausgegeben, bei denen das vierte Zeichen kein 'T' ist.

1. **funktioniert nicht**, wenn im GROUP-Abschnitt *@substring({%Klassenname.Attributname%}, 1, 1)* eingetragen ist, weil nur das Zeichen an Position 1 zur Gruppierung dient.
2. **funktioniert nicht**, wenn im GROUP-Abschnitt *@substring({%Klassenname.Attributname%}, 1, 4)* eingetragen ist, obwohl die Gruppierung nun die ersten vier Zeichen umfasst. Beide Having-Bedinungen weichen aber mit *..., 1, 1)* und *..., 4, 1)* vom Gruppierungsausdruck mit *..., 1, 4)* ab.
3. **funktioniert**, wenn im GROUP-Abschnitt *@substring({%Klassenname.Attributname%}, 1, 1), @substring({%Klassenname.Attributname%}, 4, 1)* eingetragen ist, da für beide Bedingungen eine passende Gruppierungsanweisung existiert.
4. wenn im GROUP-Abschnitt *@substring({%Klassenname.Attributname%}, 1, 4)* eingetragen ist, **funktioniert auch** (@substring({%Klassenname.Attributname%}, 1, 4) like 'F*') and (@substring({%Klassenname.Attributname%}, 1, 4) not like '*T')

Abschnitt VARIABLES

Was sind Variables? Variables-Definitionen definieren Zusatzinformationen für Parameter in Abfragen. Parameter in Abfragen sind diejenigen Werte, die erst zur Laufzeit aus den Daten eines aktuellen Datensatzes oder durch den Nutzer eingegeben werden – das sind die mit **{*...*}** eingefassten Namen.

Wozu dienen die? Wenn Sie Abfragen für die Nutzer definieren (freie Abfragen), so werden diese meistens nicht (nur) mit statischen Bedingungen versehen sein, sondern die Nutzer sollen eigene Werte für die Bedingungen eingeben können – z.B. ein Startdatum, einen Landeskenner etc. Mit den Variables-Definitionen lassen sich unterschiedliche Eingabeunterstützungen für die Nutzer der Abfragen definieren, um das Verständnis, wie die Eingabe von Abfrageparametern erfolgen soll, zu vereinfachen. Variables-Definitionen sind also – mit einer Ausnahme (siehe unten) – für freie Abfragen vorgesehen. Bei internen Abfragen haben sie keine Funktion.

VARIABLES erlauben es, bei freien Abfragen (und mit dem Attribut 'Value' auch für Regelabfragen) für vom Anwender auszufüllende Parameter des CONDITIONS-Abschnitts eine Reihe von Formatierungsanweisungen festzulegen, mit denen sich der Aufruf einer Abfrage benutzerfreundlicher gestalten lässt, da die vom Anwender auszufüllenden Parameter (**{*Parametername*}**) mit Zusatzinformationen und Formatierungen versehen werden können. Die Syntax für einzelne Formatierungsanweisungen lautet hierbei (Eingabe ohne spitze Klammern!):

```
<Parametername>.<Eigenschaft>=<Wert>
```

Für die Eintragung der Parameter und der Eigenschaften kann via **Strg+Leertaste** eine Eingabeunterstützung aufgerufen werden (vgl. Abfragen).

Folgende Eigenschaften können gesetzt werden:

Eigenschaft	Bedeutung / Wert
Aliasname	Für die vom Nutzer einzutragenden Abfrageparameter kann ein Aliasname vergeben werden. Hintergrund: Die Abfrageparameter werden dem Anwender in einer kleinen zweizeiligen Tabelle angeboten, die nach dem Namen der Parameter sortiert ist. Um die Parameter in eine fachlich sinnvolle Reihenfolge bringen zu können, musste bislang i.d.R. mit vorangestellten Ziffern gearbeitet werden, die in der Tabelle als Überschriften mit angezeigt werden. Über den Aliasnamen kann eine sinnvollere Beschriftung (auch mit Leerzeichen) für die Abfrageparameter erfolgen, ohne die Sortierung der Parameter - weiterhin nach den im CONDITIONS-Abschnitt vergebenen Parameternamen - zu beeinflussen.
Default	Default-Wert, der bei Auswahl der Abfrage für diesen Parameter als Wert vorgeschlagen wird. Wird hinter <Parametername>.Default= mit Strg+Leer die Kontexthilfe aufgerufen, so öffnet sich eine Auswahlliste der Skriptdefinitionen, die für die Defaultwertermittlung von freien Abfragen definiert wurden.
Type	Datentyp: Mögliche Einträge sind String (Zeichenkette; Default, wenn kein Type definiert wird), Date (Datum), Boolean (Ankreuzfeld), Double (Fließkommazahl). Je nachdem, welcher Typ angegeben wird verändert sich das Verhalten an der Oberfläche. *Date*: Eingabe wird auf korrektes Datum überprüft. Es kann ein Datum aus einem Kalender ausgewählt werden. *Boolean*: Anzeige als Checkbox; nur true oder false sind möglich. *Double*: Eingabe wird auf korrekte Zahl überprüft.
Info	Tooltiptext, der bei Auswahl des Parameters erscheint.
SQD	Name einer anderen Abfrage. Diese wird bei der Auswahl der Abfrage ausgeführt und füllt eine Auswahlliste, aus der der Anwender dann einen Wert auswählen kann. Überregelt bei Auswahl des Datentyps *Date* die Anzeige des Kalenders und zeigt stattdessen die Auswahlliste.
RegExp	Regular Expression, die vor der Ausführung der Abfrage für den Parameterwert kontrolliert wird. Wird die Regular Expression nicht eingehalten, erhält der Anwender eine Mitteilung und muss den Wert anpassen.
Value (nur für Regelabfragen)	Es gibt die Möglichkeit, für Parameter aus dem CONDITIONS-Bereich einer Regelabfrage den Parameter vor Ausführung der Abfrage über eine ASYS-Funktion ermitteln zu lassen. Zur Verfügung stehen hierbei die im Kapitel Prüfregeln aufgeführten ASYS-Funktionen. **Besonderheit**: Der Zugriff auf Daten des zugrunde liegenden Datenkontextes hat hierbei immer über die **dc**-Funktionen zu erfolgen (z.B. dc.getDateValue('Klassenname.Attributname')); die **{*...*}**-Notation kann hierfür nicht verwendet werden! Diese Funktionaliät wird für die Regelabfragen der Standardprüfpläne umfangreich genutzt. Beispiele können dort eingesehen werden (z.B. in der Abfrage: *IKA STD BGS AVV gefaehrlich*)
ZeitraumVon ZeitraumBis	Nur für Datumsfelder in freien Abfragen, wenn für eine Datumsangabe zwei Parameter (Zeitraum von ... bis ...) gefüllt werden müssen. Sorgt dafür, dass bei der Ausführung der Abfrage die Funktionalität zur Jahres-, Quartals- und Monatsauswahl angeboten wird. Das Parameter-Pärchen aus 'Beginn des Zeitraum' und 'Ende des Zeitraums' wird durch Angabe der gleichen Zahl bei 'ZeitraumVon' und 'ZeitraumBis' zusammengeführt.

Jede VARIABLES-Definition ist **in eine Zeile** zu schreiben.

Abschnitt UNION

In diesem Abschnitt können Abfragen angegeben werden, deren Ergebnisse während der Ausführung zusammengefasst werden.

Werden **mehrere Abfrage** angegeben, so werden sie in diesem Abschnitt **alle in eine Zeile** geschrieben und durch Kommata getrennt:

```
Abfrage 1, Abfrage 2, ...
```

Abschnitt SKRIPTS

SKRIPTS erlauben es, auf den in den RESULTS ermittelten Spalten und Werten (Rechen)Operationen auszuführen, die weiteren Ergebnisspalten liefern.

Angegeben wird im SKRIPTS-Abschnitt der Name der Ergebnisspalte und hinter dem Gleichheitszeichen die auszuführende Anweisung. Wie bei den RESULTS darf der Name keine Leer- oder Sonderzeichen enthalten.

Hinter dem Namen der Ergebnisspalte muss mit / getrennt der Typ angegeben werden. Wird kein Typ angegeben, wird *String* (Zeichenkette) verwendet. Weitere Typen neben *double* (Fließkommazahl) sind *int* (Ganzzahl) und *date* (Datum).

Unter RESULTS sollte ebenfalls der Typ mit angegeben werden, zumindest bei *double* und bei *date*.

Um die Resultate verwenden zu können, muss jeweils die Methode dc.get…Value(„…“) verwendet werden (dc.getDoubleValue, dc.getIntValue, dc.getMemoValue, dc.getStringValue).

Es können auch mehrere SKRIPTS in einer Abfrage verwendet werden. In diesem Fall wird jedes Skript eine Zeile geschrieben.

Die Anweisungen, die in einem SKRIPTS-Abschnitt verwendet werden, werden wie Prüfregeln, also in Java definiert. Statt der Variablen in der {% … %}-Notation wird die o.g. dc.get…-Notation verwendet.

Sonderfälle

IN-Operator für verschachtelte Abfragen

Mit dem IN-Operator lässt sich der Wert eines Bedingungsattribut im CONDITIONS-Abschnitt mit einer Ergebnismenge einer Unterabfrage vergleichen. Die Bedingung ist erfüllt, wenn der Wert des Bedingungsattributs in der Wertemenge der Unterabfrage enthalten ist (IN) bzw. nicht enthalten ist (NOT IN). Die Unterabfrage enthält hierfür **nur eine** Ergebnisspalte.

```
{%Klassenalias.Attributname%} [NOT] IN (Unterabfrage)
{%Klassenalias.Attributname%} [NOT] IN ({%*Abfrage%}) (ab Version 7.09.00)
```

Die folgende Anleitung/Einschränkung gilt nur bis zur Version 7.09.00 (ab der Version 7.09.00 kann auf eine andere Abfrage verwiesen werden)
Die **Unterabfrage** kann **nicht** als Verweis auf eine andere Abfragendefinition im Repository formuliert werden. Statt dessen muss die Unterabfrage als ausführbares SQL-Statement in die Zeile des CONDITIONS-Abschnitts eingetragen werden. Die Schritte hierzu sind:

1. Formulierung der Unterabfrage im Repository-Administrator mit den üblichen Werkzeugen zur Definition einer Abfrage.
2. Übergabe der Unterabfrage an die Administrator-Client-Anwendung zum Test.
3. Wenn die Abfrage das gewünschte Ergebnis liefert, kann das ausgeführte Datenbankkommando in der Client-Anwendung im Konsolenbereich markiert und per Strg-C in die Zwischenablage kopiert werden.
4. Im Repository-Administrator kann dieses Datenbankkommando im CONDITIONS-Abschnitt eingerahmt durch ein Klammerpärchen (...) hinter den IN/NOT IN-Operator eingefügt werden.
5. Die Unterabfrage muss in einer Zeile stehen!

UNION-Operator für verkettete Abfragen

Mit dem UNION-Operator lassen sich zwei oder mehr Abfragen zu einer Treffermenge kombinieren s. UNION-Abschnitt ab Version 7.09.00. Die Bedingungen hierfür sind:

1. Die Abfragen müssen die gleichen Ergebnisspalten im RESULTS-Abschnitt enthalten (gleiche Alias-Namen der Attribute).
2. Die Reihenfolge der Ergebnisspalten muss übereinstimmen.
3. Die Typen der Ergebnisspalten müssen übereinstimmen (Bei gleicher Position und gleichem Aliasnamen lassen sich z.B. eine Zeichenkette und ein Ankreuzfeld nicht zusammenführen!).

```
{%Klassenalias.Attributname%} operator Vergleichswert UNION (UNION-Abfrage)
```

Die folgende Anleitung/Einschränkung gilt nur bis zur Version 7.09.00 (ab der Version 7.09.00 gibt es einen eigenen UNION-Abschnitt)
Eine per UNION verbundene Abfrage kann **nicht** als Verweis auf eine andere Abfragendefinition im Repository formuliert werden. Statt dessen muss die Unterabfrage als ausführbares SQL-Statement in die letzte Zeile des CONDITIONS-Abschnitts eingetragen werden. Die Schritte hierzu sind:

1. Formulierung der UNION-Abfrage im Repository-Administrator mit den üblichen Werkzeugen zur Definition einer Abfrage.
2. Übergabe der Abfrage an die Administrator-Client-Anwendung zum Test.
3. Wenn die Abfrage das gewünschte Ergebnis liefert, kann das ausgeführte Datenbankkommando in der Client-Anwendung im Konsolenbereich markiert und per Strg-C in die Zwischenablage kopiert werden.
4. Im Repository-Administrator kann dieses Datenbankkommando im CONDITIONS-Abschnitt eingerahmt durch ein Klammerpärchen (...) **hinter der letzten Bedingung** eingefügt werden.
5. Die UNION-Abfrage muss in einer Zeile stehen!
6. Bei mehr als zwei UNION-Abfragen ist dieser Weg entsprechend rekursiv zu gehen, d.h. die Abfrage zweier verbundener Abfragen wird per UNION in den CONDITIONS-Abschnitt einer dritten Abfrage eingefügt.

Die ASYS-Standardkonfiguration enthält vier Beispiele einer Anwendung dieses Operators. Sie lassen sich im Filterdialog des Abfragenbaums über das Filterwort *union* auswählen.

Hinweis: UNION-Trefferlisten haben eine Besonderheit, welche die einzelnen Abfragen nicht aufweisen: Die Trefferliste enthält **keine Duplikate**, d.h. es werden automatisch alle Datensätze ausgefiltert, die in **allen** Ergebnisspalten identische Werte aufweisen. Derartige Datensätze werden nur einmalig ausgegeben. Die Summe der Datensätze in der UNION-Trefferliste kann daher kleiner sein, als die Summe der Datensätze addiert über die einzelnen Abfragen.

Öffnen einer Bearbeitungsmaske aus der Ergebnisliste einer freien Abfrage

Aus der Ergebnisliste einer freien Abfrage kann automatisch in eine Maske gesprungen werden, wenn folgenden Randbedingungen eingehalten werden:

1. Der Abfrage ist eine zu öffnende Maske zugewiesen, die aus der Ergebnisliste der Abfrage heraus geöffnet werden soll.
2. Als erstes Feld im RESULTS-Abschnitt der Abfrage ist das ID-Feld der Hauptklasse der unter 1 zugewiesenen Maske anzugeben und dieses ID-Feld muss den Aliasnamen **THE_ID** haben. Um welche Hauptklasse es sich hierbei handelt wird neben der zu öffnenden Maske angezeigt (die Anzeige erfolgt erst, nachdem die Änderungen der Abfrage übernommen wurden).

RESULTS

```
*THE_ID={%Klassenalias.#%}
...
```

Alte Themenseite zu Ausdrücken in Abfragen

Weitere Informationen zu diesem Thema																
Abfragen																
landesspezifische Zusatzinformationen:	SH	HH	NI	HB	NW	HE	RP	BW	BY	SL	BE	MV	ST	BB	TH	SN

---

1) Auch in Prüfregeln und Skripten werden die Modellnamen genutzt.

2) Dessen ungeachtet gibt es Einzelfälle, in denen die Besonderheiten einer Datenbank durch diesen Mechanismus nicht abgefangen werden können und daher datenbankspezifische Definitionen benötigen.

3) Zu den Eingabehilfen: siehe die Bedienungsanleitung für die Abfragendefinition

4) Also kein SQL-DELETE-Statement.

5) Sogenannte **abhängige Datensätze**; abhängig deshalb, weil sie an ihrem übergeordneten Datensatz 'hängen': sie sind in der Oberfläche nur über diesen erreichbar. Abhängige Datensätze werden genau einem übergeordneten Datensatz zugeordnet und kommen nur im Kontext mit diesem vor - sie werden daher z.B. zusammen mit diesem auch gelöscht.

6)

Wenn sie mehrere Datensätze enthalten, erstellt die Datenbank ein sogenanntes Kreuzprodukt.

7)

Achtung: Bitte den Hinweistext zur jeweiligen Funktion beachten! Einige der angebotenen Funktionen lassen sich nur mit bestimmten Datenbanken nutzen.

8)

Wird z.B. für zwei Spalten die min()-Funktion verwendet, so ist nicht gesagt, dass die kleinsten Werte der beiden Spalten aus demselben Datensatz der Gruppe stammen.

9)

Ein Beispiel: Die Klassen der Ansprechpartnerrollen - 18 Datensätze und ein leerer Defaultdatensatz - sowie der Behördenrollen - 10 + 1 Datensätze - werden im CLASSES-Abschnitt eingetragen, im RESULTS-Abschnitt werden die Bezeichnungsattribute beider Klassen als Ausgabespalten benannt, der CONDITIONS-Abschnitt bleibt leer. Die beiden Tabellen sind im ASYS-Datenmodell nicht verbunden und können daher im CLASSES-Abschnitt nicht aneinander gehängt werden. Die Trefferliste enthält dann (18 + 1) * (10 + 1) = 19 * 11 = 209 Datensätze.

10)

Achtung: Bitte die Fußnoten beachten! Hier werden die SQL-Operatoren verwendet und nicht die eingedeutschten Operatornamen der ASYS-Oberfläche.

11)

Achtung: In früherer Versionen von ASYS wurden auch die Operatoren lt, le, ge und gt (für <, < =, > = und >) zugelassen. Dies ist nicht mehr der Fall!

12)

Siehe die Fußnote weiter oben zum Kreuzprodukt!