niota

best practice

Web Applikation ohne Programmierung

Inhalt

[Einleitung 3](#_Toc199747174)

[Grundlegendes 3](#_Toc199747175)

[Datenbank Design 3](#_Toc199747176)

[Datenbank Layer 3](#_Toc199747177)

[Source & Staging 3](#_Toc199747178)

[Config & Executeables. 4](#_Toc199747179)

[Transactions DB 4](#_Toc199747180)

[Datenmodellierung 5](#_Toc199747181)

[Primärschlüssel 5](#_Toc199747182)

[Boolean – Felder 5](#_Toc199747183)

[Feldlängen 6](#_Toc199747184)

[Feldtypen 6](#_Toc199747185)

[Sprache 6](#_Toc199747186)

[Namenskonventionen 6](#_Toc199747187)

[Tabellen-/ Viewnamen – generell 7](#_Toc199747188)

[Tabellennamen – Präfix 7](#_Toc199747189)

[Viewnamen – Präfix 7](#_Toc199747190)

[Tabellennamen/Viewnamen - Hauptteil 8](#_Toc199747191)

[Tabellennamen/Viewnamen - Suffix 8](#_Toc199747192)

[niota Design 12](#_Toc199747193)

[Namens- und Nummernkonventionen 12](#_Toc199747194)

[Nummernkreise 12](#_Toc199747195)

[Namen 13](#_Toc199747196)

[Berechtigungen 14](#_Toc199747197)

[Applikationen/Themen 14](#_Toc199747198)

[Tabellen-/Feldformatierungen 17](#_Toc199747199)

[Instanzen von Tabellen/Views 24](#_Toc199747200)

[LookUp Instanzen 25](#_Toc199747201)

[Anhang 31](#_Toc199747202)

# Einleitung

<fehlt>

# Grundlegendes

Im Systemmandanten keine Anwendungen definieren. Für eigenen Anwendungen einen eigenen Mandanten anlegen. Ein Mandant sollte pro „homogener“ Landschaft / Kundengruppe angelegt werden, wenn die Prozesse, die hier verwendet werden sich NICHT überschneiden.

Ev. eigener

Mandant für IT

später einmal

eigener Mandant für Controlling Abteilung.

# Datenbank Design

## Datenbank Layer

Es sollte eigene Datenbanken für Datenpflege (von Steuertabellen) und Datenverwendung bzw. Datenpflege für „Transaktionstabellen“ geben.

Wenn in der Tabelle gepflegt wird, die auch unmittelbar in der Anwendung verwendet wird, bestimmt immer eine gewisse Inkonsistenz, solange eine Tabelle editiert wird.

Werden Daten aus externen System importiert, dann empfiehlt sich eine STAGING-Datenbank.

Das Datenbank-Design ist so zu wählen, daß nach Möglichkeit ein Restore von einem Backup von der Produktion möglich ist, ohne das System massiv zu stören.

Empfohlene Datenbank-Kategorien

* Source & Staging
* Config & Executables
* Transaction DB
* Reporting DB

## Source & Staging

Enthält alle Tabellen die für den Ladeprozeß oder für die automatisierte Datentransformation benötigt werden. Die Inhalte dieser Datenbank sollten im Idealfall jeden Tag auf Basis der Quelldaten wiederhergestellt werden können.

## Config & Executeables.

Enthält im Idealfall ALLE manuell gepflegten Konfigurationstabelle und alle notwendigen Functions / stored procedures.

Enthält im Idealfall KEINE Logtabellen oder automatische befüllten Tabellen. Die Änderungsrate auf dieser Datenbank ist somit naturgemäß gering. Es sollten keine automatisiert befüllten Tabellen vorhanden sein, damit auch keine Fehler entstehen können und somit auch kein Restore notwendig ist.

Restore von Produktion sollte „nur“ Entwicklungsstände von Konfigurationstabellen oder SP/UDFs beeinflussen, aber niemals Daten „vernichten.

## Transactions DB

Enthält alle Logtabelle und alle Tabellen die über Prozesse voll- / halbautomatisiert geändert werden.

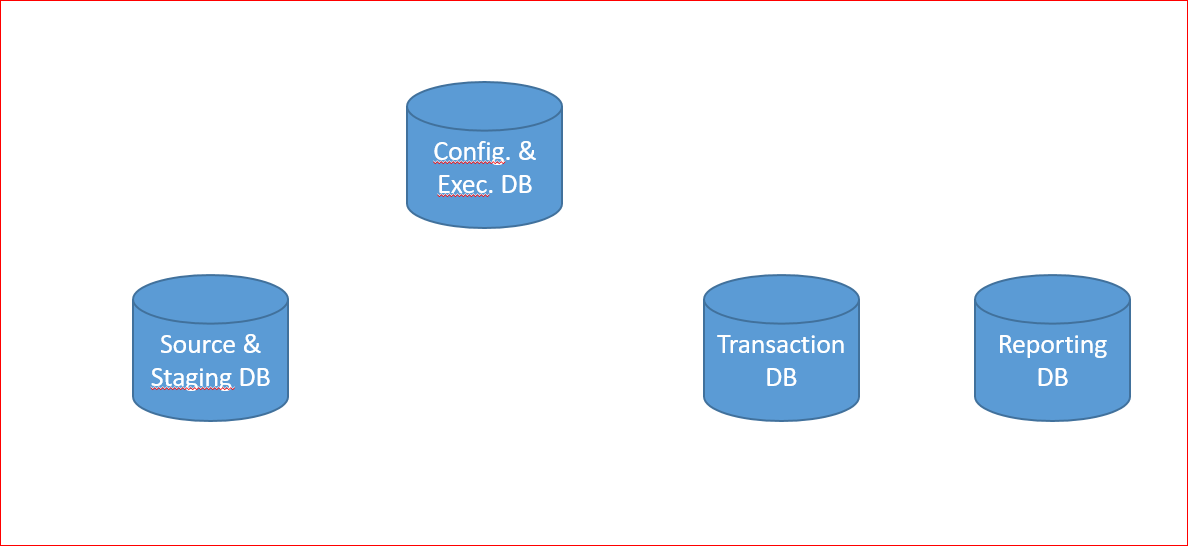
Enthält jeweils „Kopien“ der Konfigurationstabellen, wo Inhalte für den Prozeßablauf von Belang sind. Konfigurationstabellen werden von Config – DB manuell oder zyklisch übertragen.

Einige Beispiele

XYZ …. ist ein zu wählendes Projektkürzel / Mandantenkürzel

|  |  |  |  |
| --- | --- | --- | --- |
| Datenbankname(n) |  |  |  |
| Verwendung | eher guter Name | eher schlechter Name | Begründung |
| allgemeine Verwaltungsdaten – Reporting/Layer | XYZ\_STAGE  XYZ\_CONFIG  XYZ\_TRANSACTION  XYZ\_REPORTING | XYZ\_ADMINISTRATION XYZ\_PRODUCTION XYZ\_SECURITYDATA | Administraton – schränkt ein,  Production – stimmt nicht auf DEV  SECURITYDATA …. stimmt vermutlich nur am Anfang |
| allgemeine |  |  |  |
|  |  |  |  |

Datenbankmodell - schematisch



# Datenmodellierung

## Primärschlüssel

niota kann Tabellen nur im Schreibmodus (insert/update/delete) verwalten, wenn die Tabelle einen Primärschlüssel besitzt.

Der Primärschlüssel kann aus einer oder mehreren Spalten bestehen.

Der Primärschlüssel MUSS die Realität abbilden. Es dürfen nicht auf „Verdacht“ Felder in den Primärschlüssel aufgenommen werden.

Üblicherweise werden in der Literatur GUID (global unique identifier) oder völlig neutrale Schlüsselbegriffe empfohlen. Das ist vernünftig, wenn für die Verwendung der Daten eine ausgereifte Applikation geschrieben wird und die Bedienung über „drag and drop“ erfolgt.

In niota werden die Daten in der Regel manuell erfaßt und es ist daher sinnvoll die Schlüsselwerte so zu wählen, dass der Anwender auch eine gewisse Bedeutung erkannt.

Nach Möglichkeit für Tabellen, deren Inhalte manuell oder über eigene Prozeduren gepflegt werden, KEINE autosequence Optionen verwenden.

Tabelleninhalte mit autosequence Feldern können nicht von Datenbank A auf Datenbank B kopiert werden. Das ist aber in der Regel notwendig, da man die Daten von PROD auf QUAL kopieren muss bzw. neue Konfigurationen auf QUAL auf PROD transportieren will.

## Boolean – Felder

Theoretisch bietet SQL Server den Datentype Boolean an. Es hat sich aber empfohlen für Flag/Indikatoren den Datentyp „integer“ oder „small integer“ zu wählen, da sich öfter als erwartet die Wert von „Wahr“ und „Falsch“ auf „Vielleicht“ oder „ein bisschen“ ausweiten. Mit integer hat man später hier noch Spielraum.

## Feldlängen

Die Feldlängen der Datenbankfelder sollten in einem vernünftigen Verhältnis zum erwarteten Inhalt stehen.

Die Feldlänge wird für die Eingabelänge in der Eingabemaske herangezogen. Wenn jedes Textfeld präventiv mit char(1000) / varchar(1000) / nvarchar(1000) angelegt wird, dann muss man jedes Eingabefeld entsprechend per Konfiguration wieder auf eine vernünftige Länge reduzieren.

## Feldtypen

Zeichenfelder – generell varchar(x) empfohlen

Beträge – generell decimal(n,m) empfohlen – (double / float nur intern für temp. Felder)

Kennzeichen – integer oder varchar(1)

Zeitstempel – einheitlich Anzahl von Millisekunden empfohlen

Autosequences – nach möglichkeit nur für LogTabellen oder ähnliches verwenden. NICHT für Tabellen mit manueller Datenerfassung.

**dzt. nicht unterstützt für niota – web-update**

Zeit – Feldtype „time“

Boolean –

# Sprache

Es sollte die Grundsatzentscheidung getroffen werden, ob alles Englisch oder alles Deutsch benannt wird. Tendenziell hat sich Englisch besser bewährt als Deutsch, da es bei Deutsch immer zu unschönen Mischformen kommt.

Bei Englisch kommt es meistens zu unglücklichen Übersetzungen zB.

Werkstatt\_Nummer -> Workshop\_ID statt Repairshop\_ID

Google Translate kennt den Hintergrund nicht!

Im Dokument werden englische Namen verwendet – aus Gründen der Einfachheit.

# Namenskonventionen

Es sollten durchgängige Namenskonventionen vereinbart werden. Die Namensgebung sollte nur die wesentlichen Informationen beinhalten und nichts „erfinden“, was sich in der Zukunft vielleicht wieder ändern kann.

## Tabellen-/ Viewnamen – generell

Grundsätzlich die Namen in der Einzahl wählen. Die Eigenschaft der Tabelle impliziert, daß es mehrere Werte / Zeilen gibt.

T\_APPLICATION statt T\_APPLICATIONS

T\_INVOICE statt T\_INVOICES

## Tabellennamen – Präfix

Üblicherweise werden einige wenige Tabellen-Präfixe verwendet, die Aufschluß über die grundsätzliche Art der Tabelle geben. Aus Gründen der Effizienz sollte man nicht mehr als drei Zeichen für das Präfix verwenden. Das Präfix ist außerdem hilfreich für die Sortierung der Tabellen im MS SQL Management Studio.

z.B.

L\_ / LOG\_ für Logtabellen

C\_ / CFG\_ für Konfigurationstabellen

F\_ / FCT\_ für Faktentabellen (hauptsächlich numerische Werte relevant

Es kann auch hilfreich sein, den grundsätzlichen Themenkomplex über das Präfix abzubilden. Hier muß man allerdings davon ausgehen, daß immer wieder zu Un-eindeutigkeiten kommen wird.

ITS\_ für IT Systeme Implementierung

APM\_ Applikationsverwaltung

Es wird empfohlen das Tabellenpräfix auf der Datenbank NICHT nach der Verwendung zu wählen. Auch wenn eine Tabelle am Beginn ausschließliche als Lookup-Tabelle verwendet wird, sollt die Tabelle nicht LOOKUP\_ oder L\_ oder LKP\_ beginnen. Es ist sehr wahrscheinlich, daß irgendwann ein Steuerungsfeld dazukommt und dann wird aus der Lookup-Tabelle eine Konfigurationstabelle oder Dimensionstabelle.

## Viewnamen – Präfix

Für das Präfix für Viewnamen gilt sinngemäß das gleiche wie für Tabellennamen. Es hat sich bewährt zusätzlich das Präfix „V\_“ bei Views vorne anzustellen.

V\_CFG\_ … Konfigurationsview (kann z.B. für stored procedures interessant sein)

V\_UIX\_ … User Interface View – damit weiß man z.B. daß diese Daten exponiert sind

V\_ETL\_ … ETL view – ist eine rein techschnische Transformation von Werten

…

## Tabellennamen/Viewnamen - Hauptteil

Es wird empfohlen sich hier an der logischen Bedeutung des Primärschlüssels zu orientieren. Der Primärschlüssel ist IMMER richtig, die Felder in einer Tabelle werden ergänzt. Wenn der Primärschlüssel für eine Tabelle z.B. die Anwendungs ID ist, dann sollte sich das auch im Tabellenamen widerspiegeln.  
Dadurch wird man auch hellhörig, wenn man eine zweite Tabelle mit gleichem Namen anlegen möchte. In diesem Fall muss man das Datenmodell überdenken.

somit z.B.

CFG\_APPLICATION … für Hauptkonfiguration der Anwendungen / z.B. irgendwelche „Vorgabewerte“

F\_APPLICATION … für die tatsächlichen Daten

## Tabellennamen/Viewnamen - Suffix

Wenn der Primärschlüssel aus mehr als aus einem Feld besteht, dann sollte sich das im Suffix widerspiegeln.

Der Primärschlüssel ist IMMER richtig, die Felder in einer Tabelle werden ergänzt.

Wenn der Primärschlüssel für eine Tabelle z.B. die AnwendungsID + Lieferanten ID ist, dann sollte sich das auch im Tabellenamen widerspiegeln.

somit z.B.

F\_APPLICATION\_VENDOR

CFG\_APPLICATION\_VENDOR

Wenn sich der Primärschlüssel im Namen der Tabelle widerspiegelt, dann bleiben logisch zusammengehörige Tabelle auch in diversen Listen und Übersichten beisammen.

Feldnamen

Feldnamen sollten unter Berücksichtigung eines „virtuellen“ Domänenkonzeptes vergeben werden. Nach diesem Konzept besteht ein Feldname ebenso aus einem Präfix (der Domäne) und einer Qualifizierung (Hauptteil+Suffix).

Für SQL Abfragen (type ahead) hat es sich bewährt die Feldnamen nicht unnötig lange zu vergeben sondern sich auf ein Limit von +/- 20 Zeichen zu beschränken.

Die Domäne eines Feldes, das Präfix sollte eine grundsätzliche Aussage über den Inhalt des Feldes geben.

Basisdomänen

DATE\_ …. das ist ein Datum

DESCR\_ … das ist eine Beschreibung

TS\_ … das ist ein Timestamp

IS\_ … ist / ist nicht

HAS\_ … hat / hat nicht

Wenn es sich um sog. „Basisdomänen“ handelt wie Datum oder Beschreibung oder Benutzer, dann ist eine weitere Qualifizierung zwingend.

DATE\_CREATE …. Das ist das Erstellungsdatum

Häufig gebrauchte Felder kann man aber auch sofort als spezifische Domäne anlegen und weiter qualifizieren.

CREATE\_DATE\_ … das ist ein Erstellungsdatum

CREATE\_DATE\_APPLICATION … das ist der Erstellungsdatum für die Anwendung

CREATE\_DATE\_INVOICE … das ist das Erstellungsdatum für die Rechnung

spezifischen Domänen

APPLICATION\_ID …. das ist eine Anwendungsnummer

VENDOR\_ID … das ist eine Lieferantennummer.

Wenn eine spezifische Domäne in einer Tabelle mehrfach benötigt wird, dann erfolgt hier ebenfalls eine Qualifizierung.

APPLICATION\_ID\_PREDECESSOR … ID der Vorgängeranwendung

APPLICATION\_ID\_SUCCESSOR … ID der Folgeanwendung

VENDOR\_ID\_PREFERRED … Bevorzugter Lieferant

VENDOR\_ID\_FALLBACK … Bevorzugter Lieferant

Wenn man die Namensgebung umkehrt, z.B.

INVOICE\_DATE\_CREATE

INVOICE\_DATE\_UPATE

INVOICE\_LONG\_DESCRIPTION

…

Dann hat es den Effekt für den Anwender, daß in der Oberfläche / im Web einmal alle Felder mit INVOICE beginnen und die relevante Information eigentlich immer am Ende ist. Ebenso werden bei type ahead immer alle INVOICE – Felder angezeigt, aber die Wahrscheinlichkeit ist sehr gering, dass man INVOICE\_LONG\_DESCRIPTION mit INVOICE\_DATE verknüpfen möchte.

Viel eher will man DATE\_<abc> mit DATE\_<xyz> verknüpfen.

Ein durchgängiges Namenschema in der Datenbank hat in niota den Vorteil, daß die Konfiguration des Benutzerfrontends stark automatisiert werden kann.

Man kann zum Beispiel allen Feldern die mit „APPLICATION\_ID“ beginnen, eine Lookup auf die Applikationshaupttabelle zuweisen. Somit ist nur über die Namenskonvention gewährleistet, daß nur gültige Werte verwendet werden können. Das kann man theoretisch auch über referentielle Integritätsprüfungen erreichen, aber diese funktioniert nicht Server-übergreifend und wird in der Praxis kaum eingesetzt.

Basierend auf diesen Anforderungen könnte zum Beispiel die Tabelle wie unten angeführt definiert werden.

|  |  |  |
| --- | --- | --- |
| Feldname | Typ | Pflicht |
| Name der Anwendung | Text | x |
| Beschreibung | Text (lang) | x |
| Cloudservice | Auswahl (ja / nein) | x |
| Use Case Beschreibung | Text (lang) | x |
| Mehrwert / Nutzen | Text (lang) | x |
| Geplante Inbetriebnahme | Datum | x |
| Stichtagsrelevant | Auswahl (ja / nein) | x |
| Gesamtrisiko | Auswahl (Hoch / Mittel / Gering / Sehr gering) | x |
| DSGVO-relevant | Auswahl (ja / nein) | x |
| Interner Software-Hersteller | Text |  |
| Fremdhersteller | Text |  |
| Verantwortliche Fachabteilung | Auswahl (lt. Organigramm) | x |
| Fachexperte | Text |  |
| Vertragspartner | Text oder Auswahl bisherige Werte | ? |
| Vertragsnummer | Text | ? |
| Vertragsbezeichnung | Text (ev. Link) | ? |
| Einmalkosten | Zahl | ? |
| Laufende Kosten | Zahl | ? |
| Verrechnung | Mehrfachauswahl (Jan, Feb, Mar, Apr, …) | ? |
| Rechnungssteller | Text oder Auswahl bisherige Werte | ? |
| Lieferant |  | ? |

CREATE TABLE [dbo].[T\_Application](

[Application\_ID] [nvarchar](10) NOT NULL,

[Application\_Name] [nvarchar](100) NOT NULL,

[LongDescr\_Application] [nvarchar](500) NOT NULL,

[Is\_CloudService] [int] NOT NULL,

[Comment\_UseCase] [nvarchar](2000) NOT NULL,

[Comment\_Benefit] [nvarchar](2000) NOT NULL,

[Has\_Deadline] [int] NOT NULL,

[RiscClass\_Total] [int] NOT NULL,

[Manufacturer\_ID\_Intern] [nvarchar](10) NULL,

[Manufacturer\_ID\_Extern] [nvarchar](10) NULL,

[OrgUnit\_ID\_Responsible] [nvarchar](10) NOT NULL,

[Contractor\_ID] [nvarchar](10) NOT NULL,

[Comment\_Contract] [nvarchar](2000) NOT NULL,

[OneTimeCost] [decimal](15, 3) NOT NULL,

[RunningCost\_Original] [decimal](15, 3) NOT NULL,

[RunningCost\_Current] [decimal](15, 3) NOT NULL,

[DueMonth\_ID] [int] NOT NULL,

[DueInterval\_ID] [int] NOT NULL,

[Invoicee\_ID] [nvarchar](10) NULL,

[Supplier\_ID] [nvarchar](10) NULL,

CONSTRAINT [PK\_T\_Application] PRIMARY KEY CLUSTERED

(

[Application\_ID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

Basisdomänen sind hier

LongDescr … Langtext

Is … ist/hat Kennzeichen

Comment … Kommentar

Alles anders sind spezifiische Domänen

ohne Qualifizierung

Application\_ID … AnwendungsID

…

mit Qualifizierung

Running\_Cost\_Original … laufende Kosten – Originalwert

Running\_Cost\_Current … laufend Kosten – aktueller Wert

# niota Design

## Namens- und Nummernkonventionen

### Nummernkreise

In niota bekommen verwaltetet Tabellen / Views eine eindeutige interne Nummer. Diese Nummer wird einmalig mit dem physischen Namen des Datenbankobjekts verknüpft und dann in weiterer Folge ausschließlich diese Nummer verwendet.

Die Nummer eines Objekts ist die eindeutige Identifikation!

Eine Datenbank-Tabelle kann in niota unter unterschiedlichen Nummern mehrfach angesprochen werden.

Empfehlung – Verwendung von „stabilen“ Nummernkreisen – das sind Nummernkreise wo es nicht zu erwarten ist, daß es zu einem Sprung in den signifikanten Stellen kommt.

z.B.

10.000 – 20.000

Wenn man mit 1 beginnt, statt mit 10.000 dann kommt es schnell zu diesem Effekt in Sortierungen.

1, 10, 11, 12, 13, 14, ..., 19, 2, 3, 4, 5, 6, 7, 8, 9

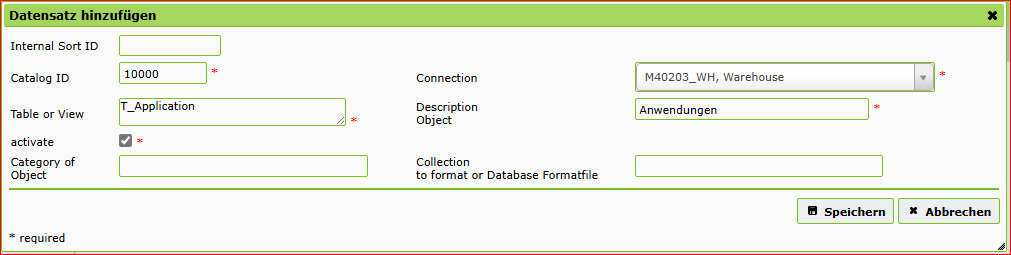
Am besten zwischen den einzelnen Nummern einen Abstand von 10 lassen und diesen dann im Bedarfsfall mit „Sonderfällen“ auffüllen.

z.B.

10.010, 10.020, 10.030

und später eventuell

10.021, 10.022 einfügen

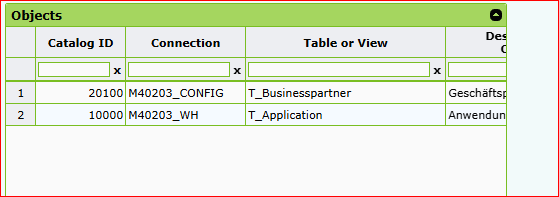


Auch wenn es technisch nicht notwendig ist, hat es sich bewährt für unterschiedlichen Datenbanken auch unterschiedliche Nummernkreise zu verwenden.

z.B.

Warehouse – DB 10.000 – 19.999

Config – DB 20.000 – 29.999



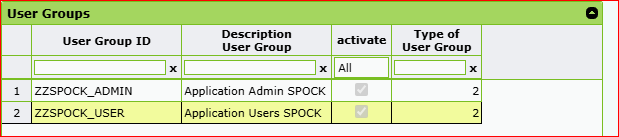
### Namen

Namen die mit „ZZ“ beginnen sind für niota interne Konfiguration reserviert.

In selbst gewählten Namen am besten sich für GroßSchreibung oder für Kleinschreibung entscheiden.

IDs von Objekten nicht in mixed case schreiben

keine Sonderzeichen oder Leerzeichen verwenden, da dies von jedem Browser unterschiedlich interpretiert werden kann und man den Unterschied zwischen „blank“ und „non breaking space“ nicht erkennen kann.



## Berechtigungen

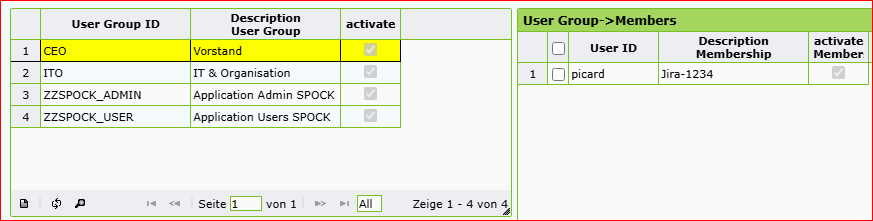
Berechtigungen IMMER auf Gruppen vergeben. Es ist technisch möglich Berechtigungen auch auf einzelne Nutzer zu vergeben, indem man die Gruppe verwendet die für jeden Benutzer angelegt wird und in der nur er selbst Mitglied sein kann.

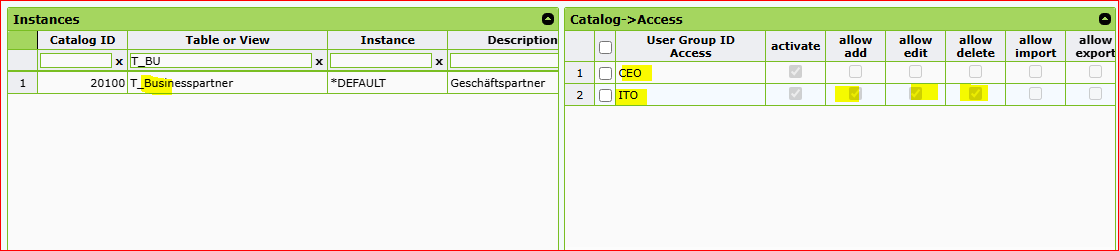
Es hat sich allerdings bewährt, auf für solche Fälle eine Gruppe zu erstellen.

z.B. wenn etwas nur der Vorstand sehen soll, dann erstellen einer Gruppe

CEO und den aktuellen Vorstand als Mitglied in diese Gruppe geben.

Wenn der Vorstand wechselt, dann kann man in der Übergangsphase beide (neuer+alter) in diese Gruppe geben.

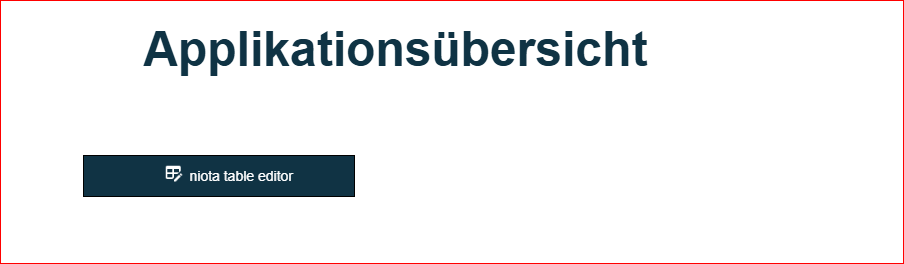


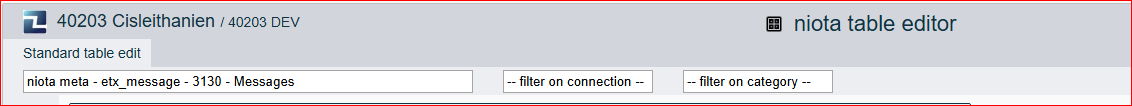


## Applikationen/Themen

Um die Tabellen und Funktionen für den Endbenutzer strukturiert anzeigen zu können, sollten gleich von Beginn an „Applikation“ und „Themenbereiche“ definiert werden.

Tabellen die eine Zuordnung zu Applikation und Themenbereich haben, werden entsprechend der Zuordnung angezeigt. Tabellen ohne Zuordnung werden in der „System-Applikation“ „niota Tabelleneditor“ im Themenbereich „Standard table edit“ in einer Auswahlliste angeboten.

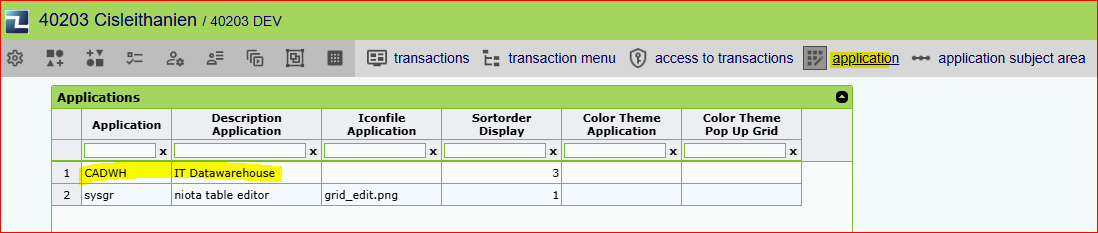


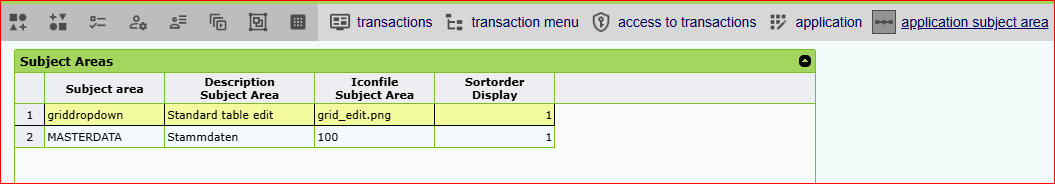


Applikationen benötigen Stammdaten, die in der Admin-Oberfläche erfaßt werden.

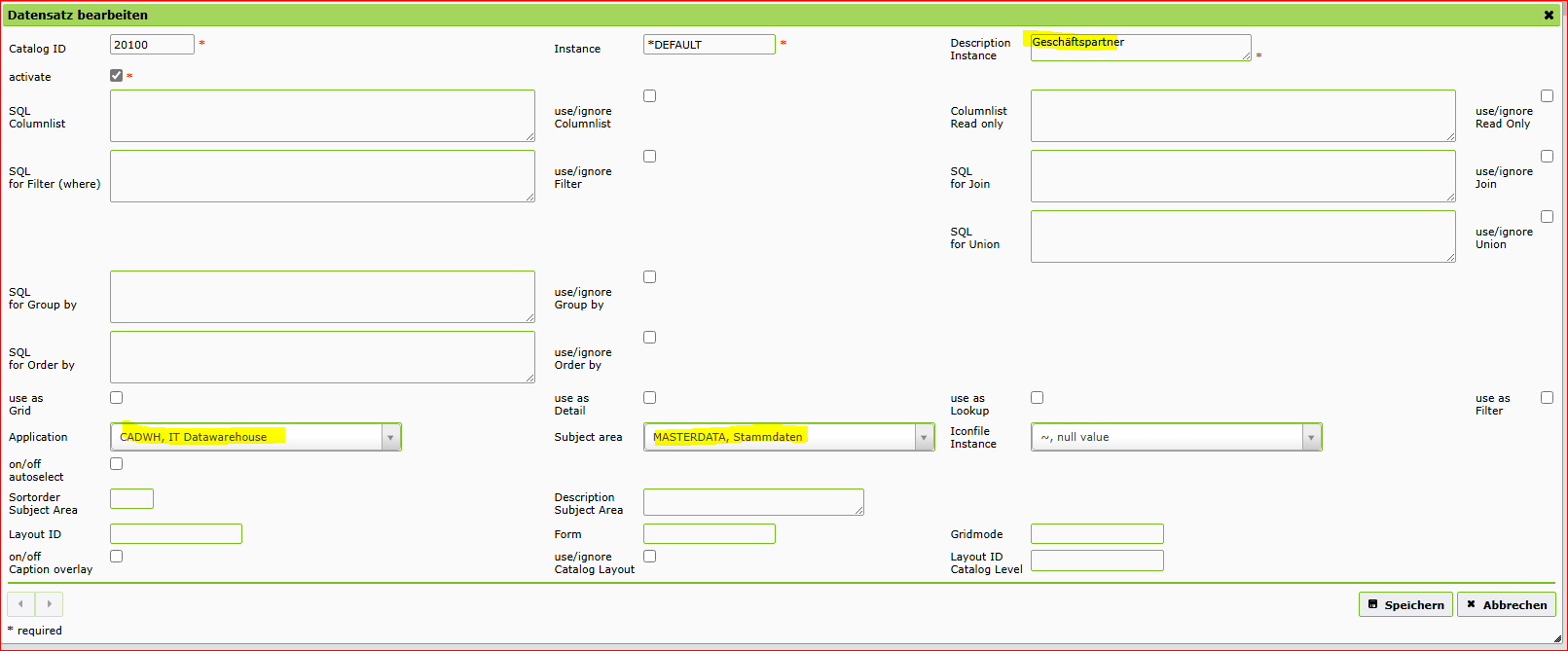
Stammdaten wie

Applications-ID und Themen-Bereich ID

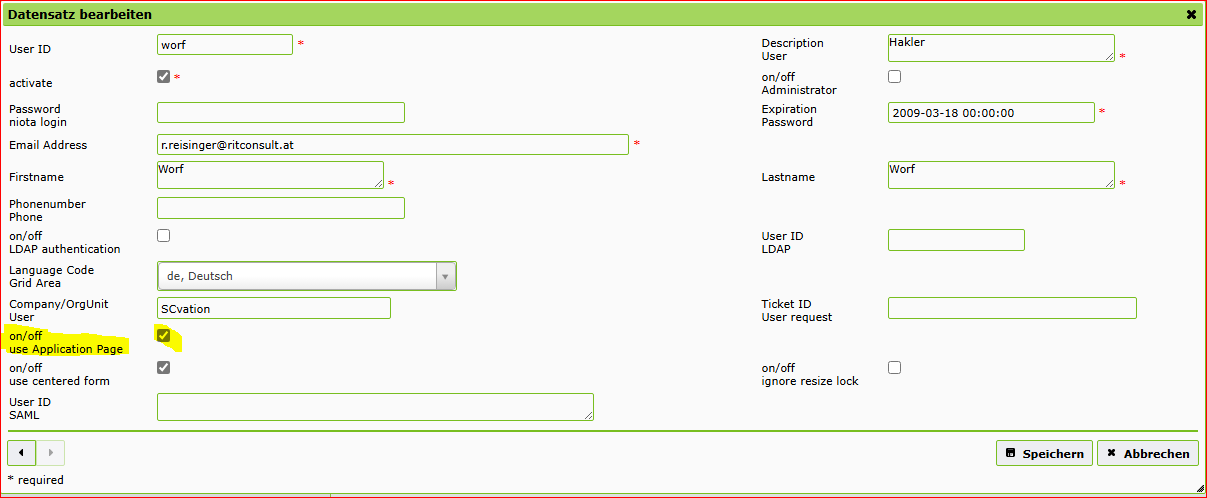


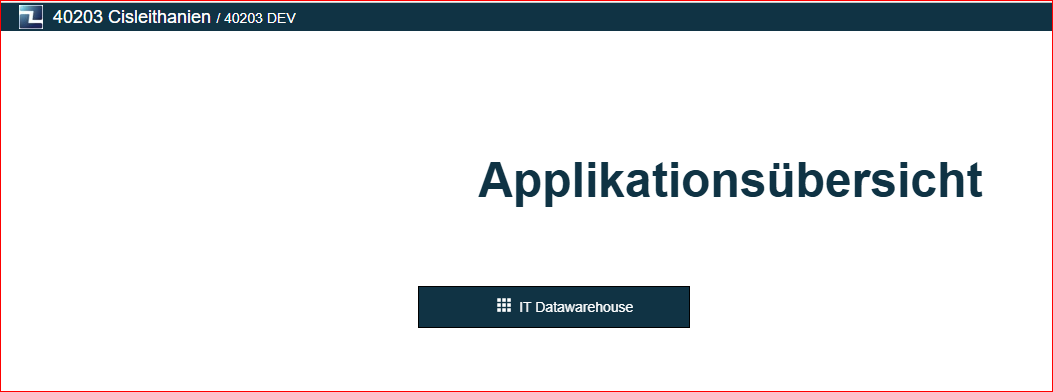


Auf Ebene der Tabelle erfolgt dann die Zuordnung.

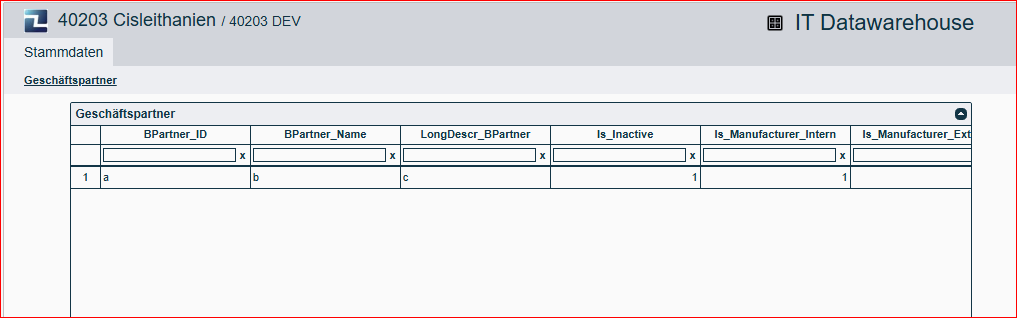


Wenn beim Benutzer die Verwendung der Applikationsübersicht aktiviert ist, dann wird eine Auswahlseite für alle seine Anwendung angeboten.





Innerhalb der Anwendung werden die Daten nach Themengebieten dargestellt.



## Tabellen-/Feldformatierungen

Um dem Endbenutzer komfortables Arbeiten zu ermöglichen, sollten die technischen Feldnamen nach Möglichkeit durch fachlichen Überschriften ersetzt werden.

Mit diesen Formatierungen sollte man gleich zu Beginn eines Projekts beginnen, da eine spätere Aufarbeitung aufwendig ist.

Am effizientesten erfolgt die Formatierung von Feldern über das Domänenmodell in niota. Ein Feld kann hier auf drei Ebenen beschrieben werden.

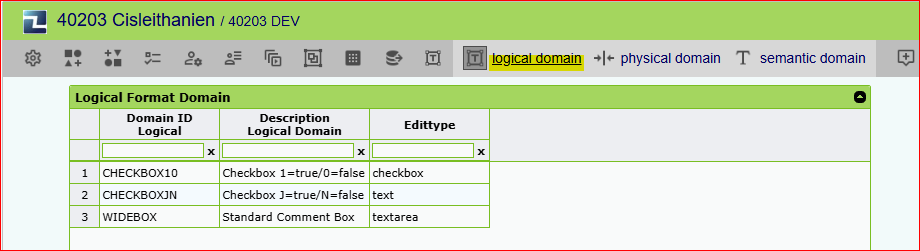
physische Domäne – Feldlängen und Feldtyp

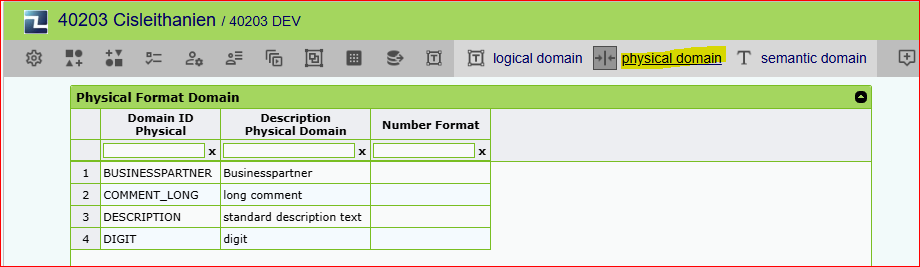
logische Domäne – Felddarstellung z.B. Textfeld, Check-Box, Text-Box,

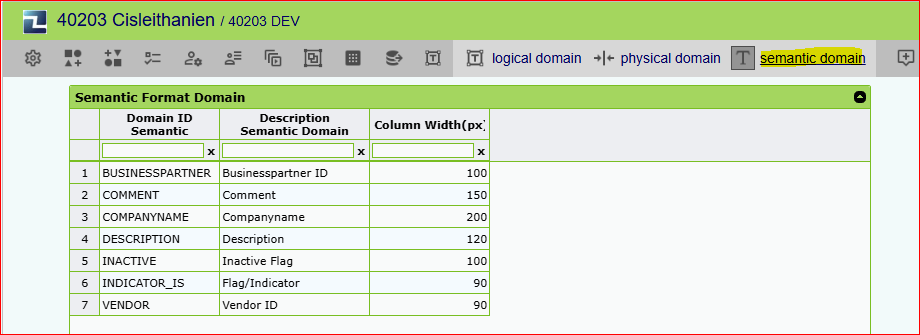
semantische Domäne – grundsätzliche, fachliche Bedeutung, z.B. Beschreibung, Kundennummer, ….

Wenn diese Domänen definiert sind, dann können sie auf Ebene der Tabelle einem Feld zugeordnet werden und regeln damit die Darstellung am Front-End.

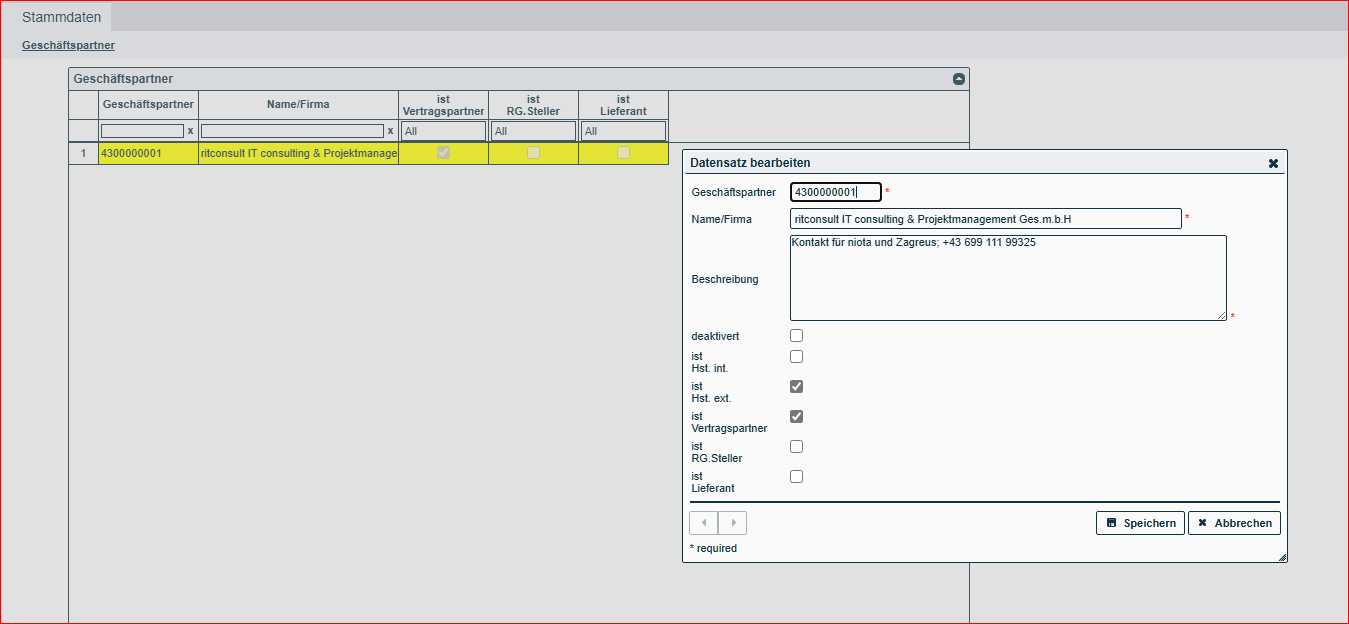
Zusätzlich können auf Ebene des Feldes oder eine Zusatzformatierung noch weitere Eigenschaften wie „protected“, „hidden“, „mandatory“, …. festgelegt werden.





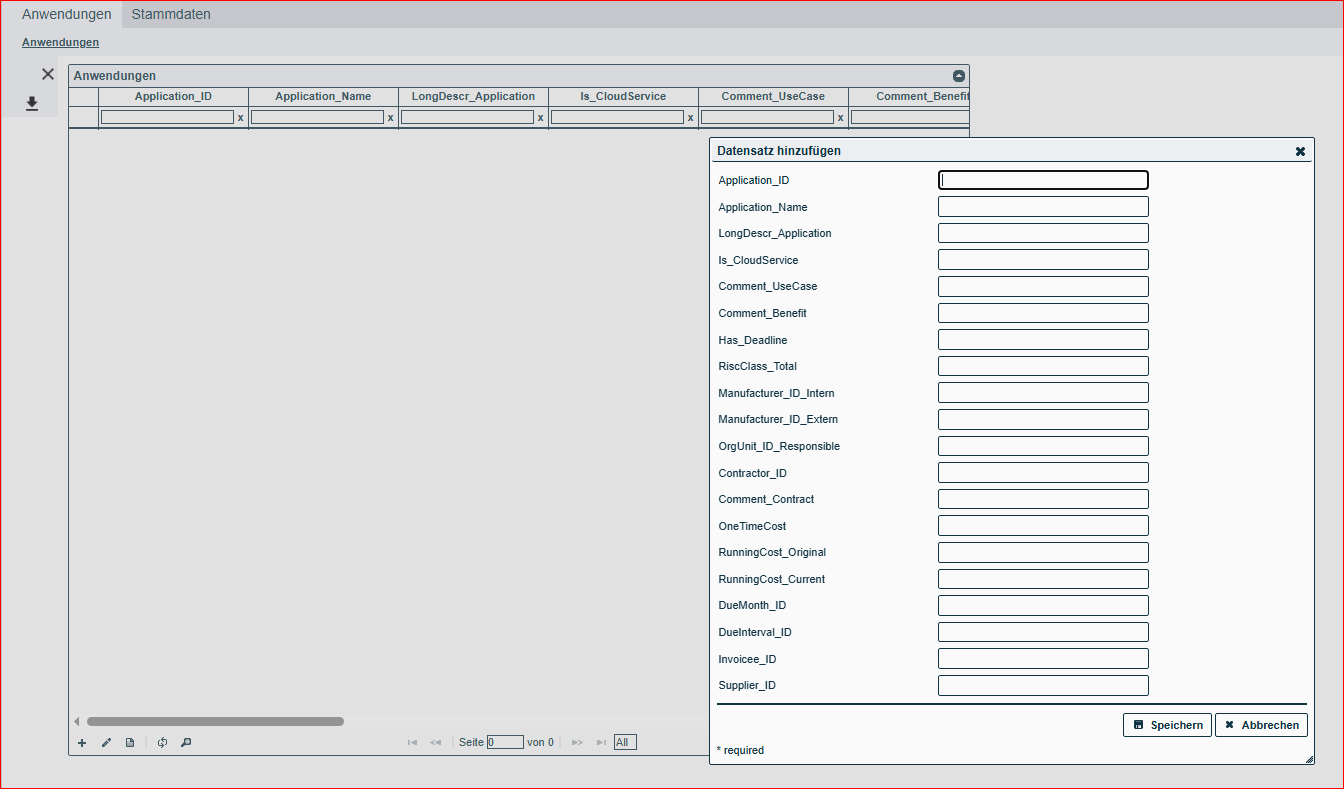


Durch einmalige Konfiguration der jeweiligen Domänen und Zuweisung an einzelne Tabellenfelder können sehr einfache komfortable Pflegemasken konfiguriert werden.



Felder können in der tabellarischen Ansicht ausgeblendet sein, aber im Änderungsformular verfügbar sein.

Gegenbeispiel – unformatierte Tabelle



Bearbeitungstabellen / (Tabellen-Anzeigen) / Tabellen-LookUp

Aus Gründen der Wartbarkeit hat sich eine strikte Trennung dieser Kategorien bewährt.

Bearbeitungstabellen sind jene Tabellen, wo mit niota Werte gepflegt werden. Diese sind direkt als entweder direkt als Tabelle in niota eingebunden oder werden über SQL-Command Aufrufe oder Aufrufe von stored procedures gepflegt.

Tabellen-Anzeigen – sind Übersichten oder Summen-Tabellen, die nach Möglichkeit über Views in niota eingebunden werden.

Dadurch ist sichergestellt, dass nicht irrtümlich Änderungen auf diese Tabellen durchgeführt werden können.

Tabellenlookup sind (meistens) Ausschnitte (Selektion/Projektion) einer Tabelle, die als LookUp/Dropdown in die Pflege-Dialoge von Bearbeitungstabellen integriert werden.   
Aus Gründen der Flexibiltät empfiehlt es sich auch hier mit Views zu arbeiten.

Es hat sich herausgestellt, daß im Laufe der Zeit auch eine sehr simple LookUp-Tabelle leichte Variationen bekommen sollen und dafür eignet sich der View-Ansatz besser, als direkter Zugriff auf die Tabelle im Lesemodus von niota.

Beide Varianten sind legitim, aber wenn man befürchten muss, dass die Tabelle für LookUp vielleicht doch nicht so einfach ist, dann gleich mit einer View beginnen.

Hinweis – da niota intern nur mit Nummern arbeitet, kann man auch später noch den Tabellennamen durch einen View-Namen ersetzen ohne weitere Konfigurationen ändern zu müssen.

Hinweis – Jede „Tabelle“ in niota benötigt einen Primärschlüssel. Auch bei Views muß es ein Feld geben/eine Kombination von Feldern, die als „virtueller Primarykey“ verwendet werden kann.

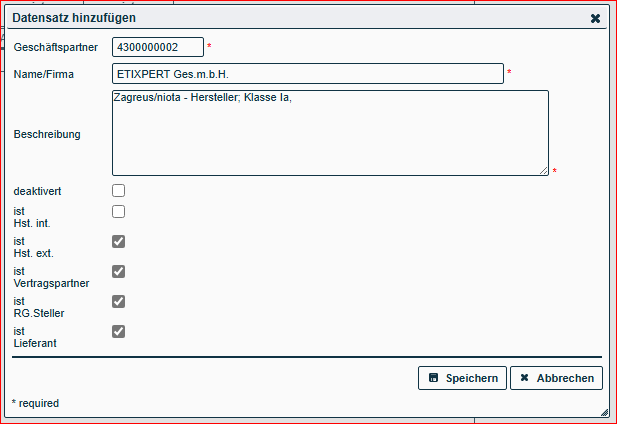
Trennung von Berarbeitung, Anzeige, LookUp am Beispiel

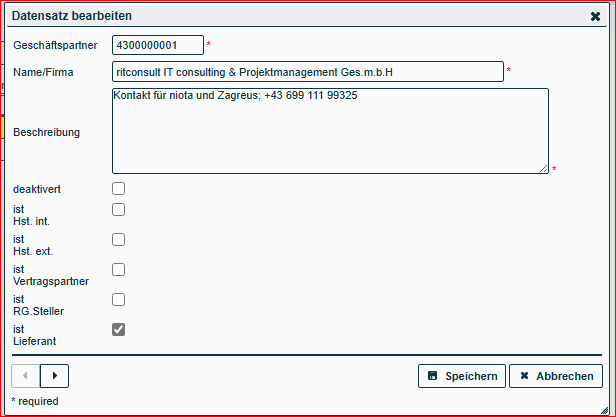
Applikationen + Partnerrollen

Eine Applikation hat hier diese Partnerrollen

Hersteller intern, Hersteller extern, Rechnungssteller, Lieferant, …

Alle Partnerrollen werden hier in einer Tabelle „Businesspartner“ / Geschäftspartner gepflegt. Da ein Geschäftspartner mehrere Rollen haben kann, werden diese Eigenschaften über Flags abgedeckt.





Die Pflege der Daten erfolgt in der allgemeinen Pflegedatenbank für Tabellenpflege.

Da es im Laufe der Zeit immer wieder vorkommt, daß sich die Herkunft der Daten ändert, ist es sinnvoll jede Tabelle so zu behandeln, als würden die Daten automatisch angeliefert werden und daß niota „nur“ die Zwischenlösung bis zur automatisierten Anlieferung ist.

Das bedeutet, daß es für jede Tabelle ein einfaches „ETL“ gibt, welche die Daten von der Änderungsumgebung in die „Verwendungsumgebung“ transportiert.

In diesem Beispiel ist die Tabelle in der „Verwendungsumgebung“ aufgeteilt in

T\_Businesspartner

T\_Contractor

T\_Manufacturer

T\_Supplier

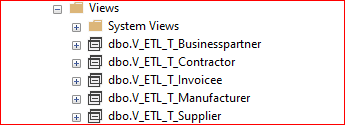
T\_Invoice

Durch diese Aufteilung wird verhindert, dass jeder Partner überall verwendet warden kann. Die Verwendung ist durch seine Eigenschaften geregelt.

Das „ETL“ besteht aus Views + einem allgemeinen Programm, das Daten aus einer View in eine Tabelle überträgt.

Die Übertragung erfolgt hier immer additiv. Es werden in der Verwendungsumgebung i.d.R. keine Daten gelöscht, auch wenn sie in der Pflegeumgebung nicht mehr vorhanden sind.

Entsprechend diesem Ansatz gibt es somit folgende Views.



Die alle vom Prinzip so aussehen

CREATE OR ALTER view [dbo].[V\_ETL\_T\_Contractor]

/\* Kommentare \*/

as

SELECT [BPartner\_ID] as [Contractor\_ID]

,[BPartner\_Name] as [Contractor\_Name]

FROM [bonaire].[dbo].[T\_Businesspartner]

where Coalesce(Is\_Contractor, 0) = 1

;

GO

Nimm aus Businesspartner nur die, wo Contractor aktiviert ist.

Das ETL erfolgt dann regelmäßig oder nach Bedarf über eine einfache Prozedur.

exec niota\_spock.dbo.mdData\_MergeFromView 'MERGE', 'NODELETE', null, 'aruba', 'V\_ETL\_T\_Supplier', '', 'curacao', 'T\_Supplier';

exec niota\_spock.dbo.mdData\_MergeFromView 'MERGE', 'NODELETE', null, 'aruba', 'V\_ETL\_T\_Businesspartner', '', 'curacao', 'T\_Businesspartner';

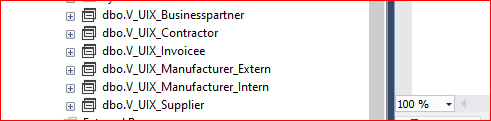
Kopiere die Daten die du in View „aruba.dbo.V\_ETL\_T\_Supplier“ findest und merge sie in „curacao.dbo.T\_Businesspartner“.

Dadurch werden die Daten in der Verwendungsumgebung „autark“ und stabil.

In der Pflegeumgebung kann wieder editiert werden, ohne die Ergebnisse zu beeinflussen. Auch wenn die Daten aus einem anderen System gelesen werden – z.B. LDAP wegen OrgEinheiten, empfiehlt es sich diesen Prozess zu wählen.

1 x pro … oder nach Bedarf die Tabelle vom Quellsystem lesen und in die „Verwendungsumgebung“ kopieren. Dadurch wird man von Änderungen im Quellsystem unabhängig und historische Daten müssen nicht an jeder Änderung in der Organisationsstruktur angepaßt werden.

Im Beispiel wird für jeden Geschäftspartner-Typ eine separate View angelegt, da sich alle Geschäftspartner die aktiv/inaktiv Eigenschaft von „Haupteintrag“ holen.



Die Views sind alle analog dieser Logik.

CREATE OR ALTER view [dbo].[V\_UIX\_Contractor]

/\* Kommentar \*/

as

select ct.Contractor\_ID

,case when bp.Is\_Inactive = 1

then cast(concat('inaktiv! ', substring(ct.Contractor\_Name, 1, 90)) as varchar(100))

else ct.Contractor\_Name end as Contractor\_Name

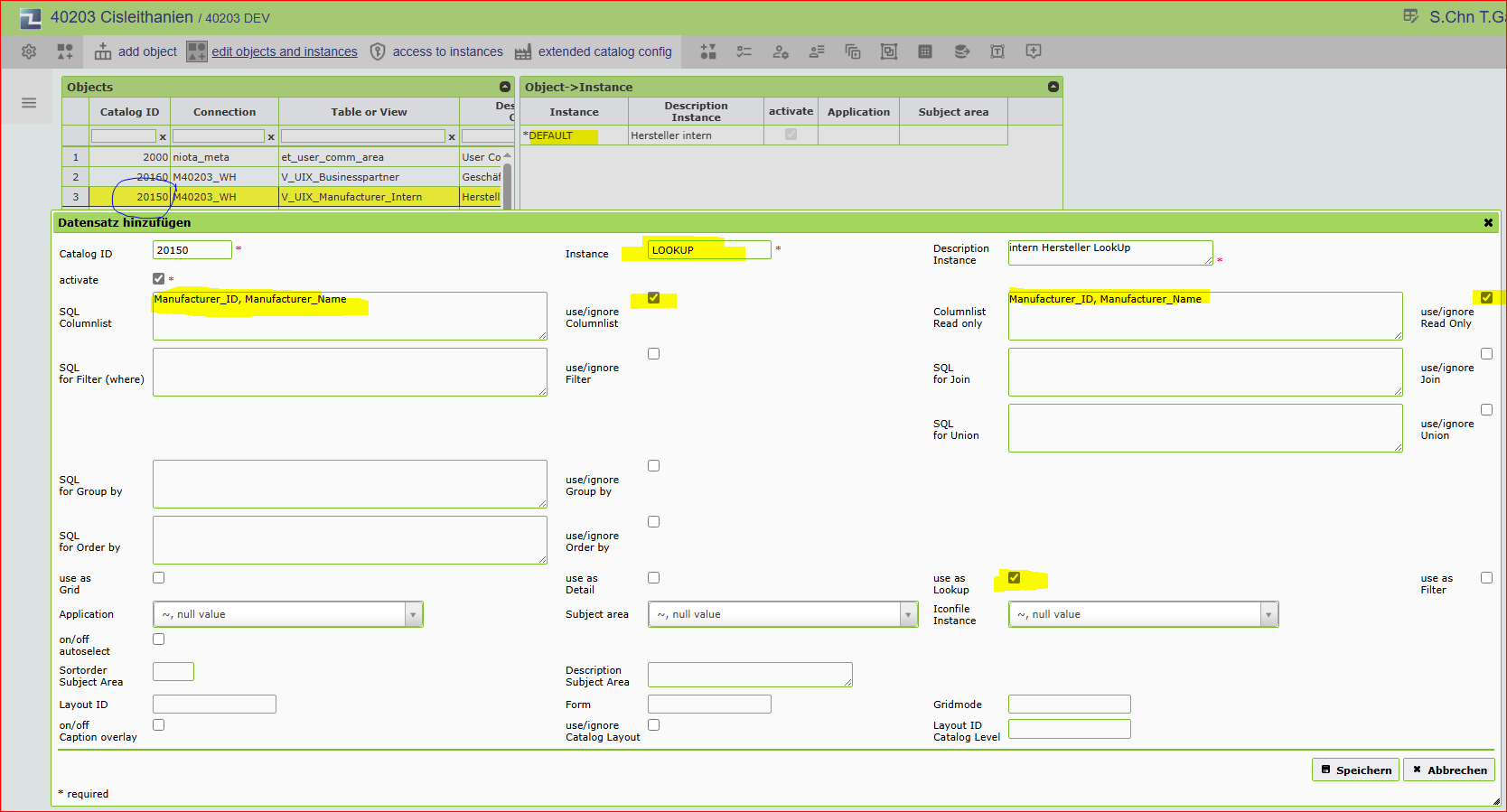
from dbo.T\_Contractor ct

left join dbo.T\_Businesspartner bp

on bp.BPartner\_ID = ct.Contractor\_ID;

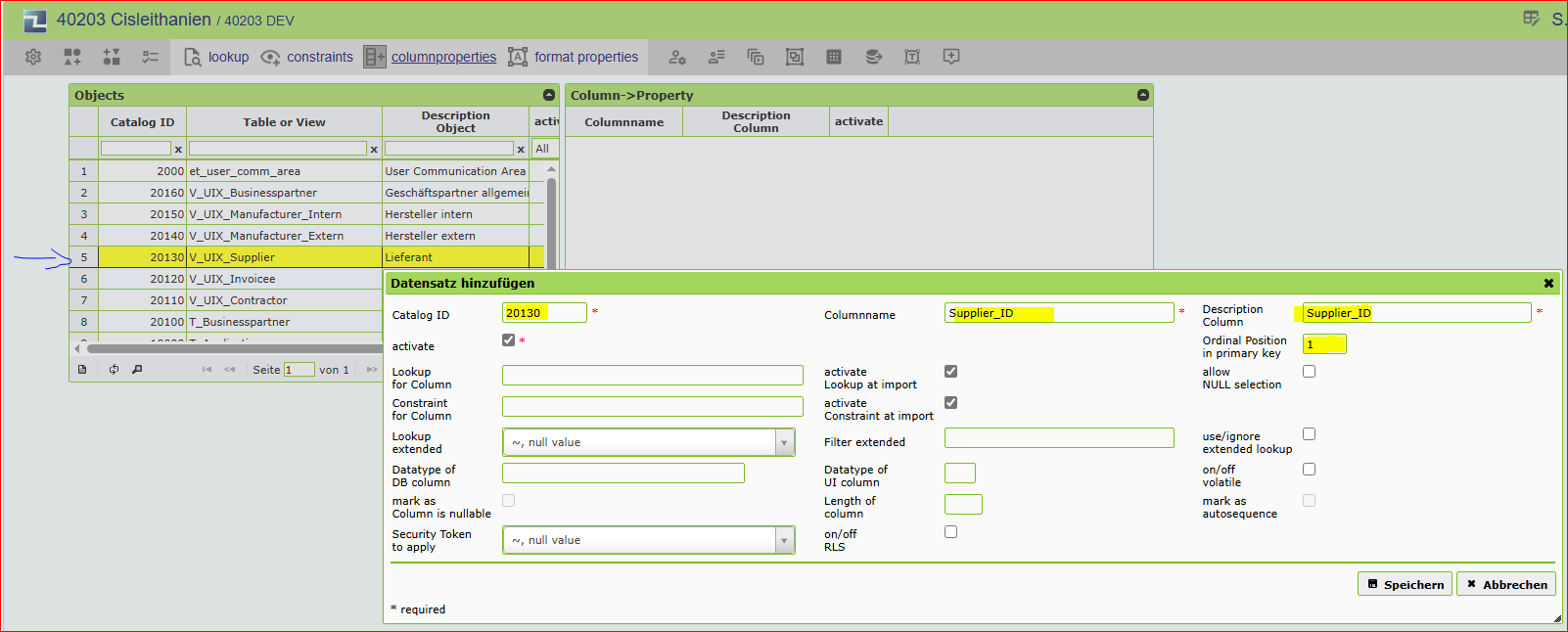
GO

Jede dieser Views wird in niota eingebunden und es wird zur „\*DEFAULT“ – Instanz (die vom System angelegt wird) noch eine LookUp Instanz definiert, die unabhängig von der eigentlichen Struktur der Tabelle nur die ersten beiden Felder selektiert.



In den „column properties“ wird das Feld „Manufacturer ID“ als 1. und einziges Feld im „virtuellen Primary key“ festgelegt.

Entsprechend dieser Systematik bekommen alle Views einen virtuellen Primärschlüssel.



### Instanzen von Tabellen/Views

Um ein- und dieselbe Konfiguration auch in niota mehrfach verwenden zu können und Tabellen nicht doppelt in das System einbinden zu müssen kennt niota „Instanzen“ von Tabellen.

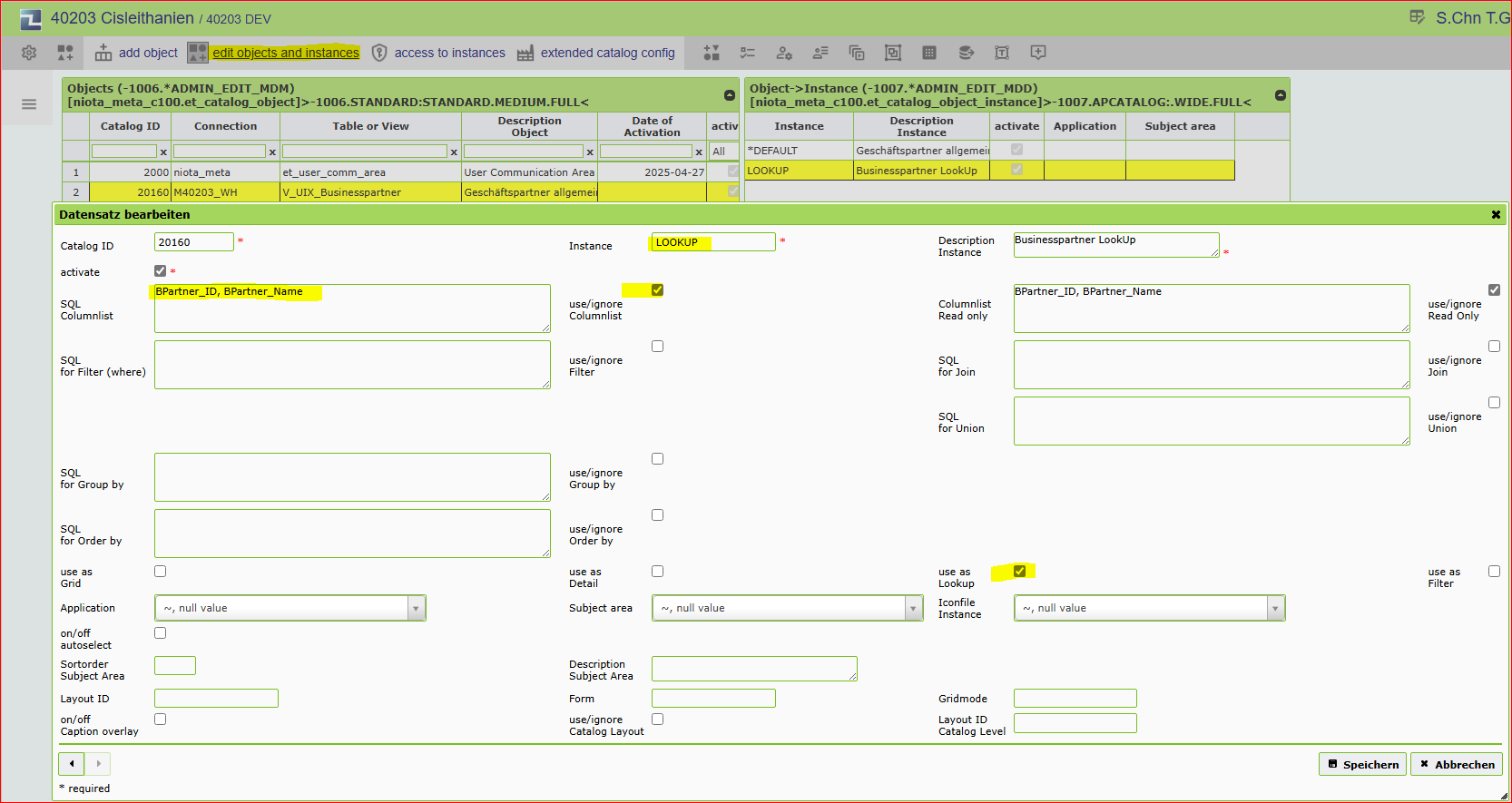
Eine Instanz ist ein „abgeänderte“ form der Basis-Tabelle und kann sich in der Feldlister, der Filterung … unterscheiden.

Vom System wird automatisch immer eine Instanz mit Name „\*DEFAULT“ angelegt. Diese Instanz ist die Haupt-Instanz und darf NIE gelöscht werden.

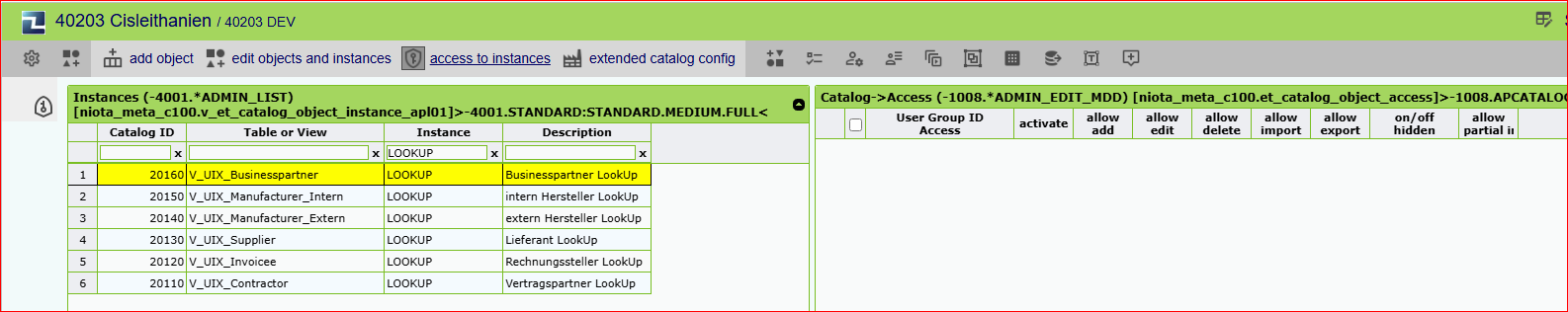
### LookUp Instanzen

Es hat sich bewährt, die Definition von LookUp Objekten nicht auf die Hauptinstanz zu legen, sondern eine separate LookUp-Instanz zu definieren und hier die Feldliste auf 2 Felder zu beschränken.

Die Hauptinstanz/physische Tabelle kann sich nun beliebig verändern, solange die zwei LookUp – Felder (Wert + Text) in der Tabelle sind, ändert sich im Bereich LookUp nichts für den Anwender.



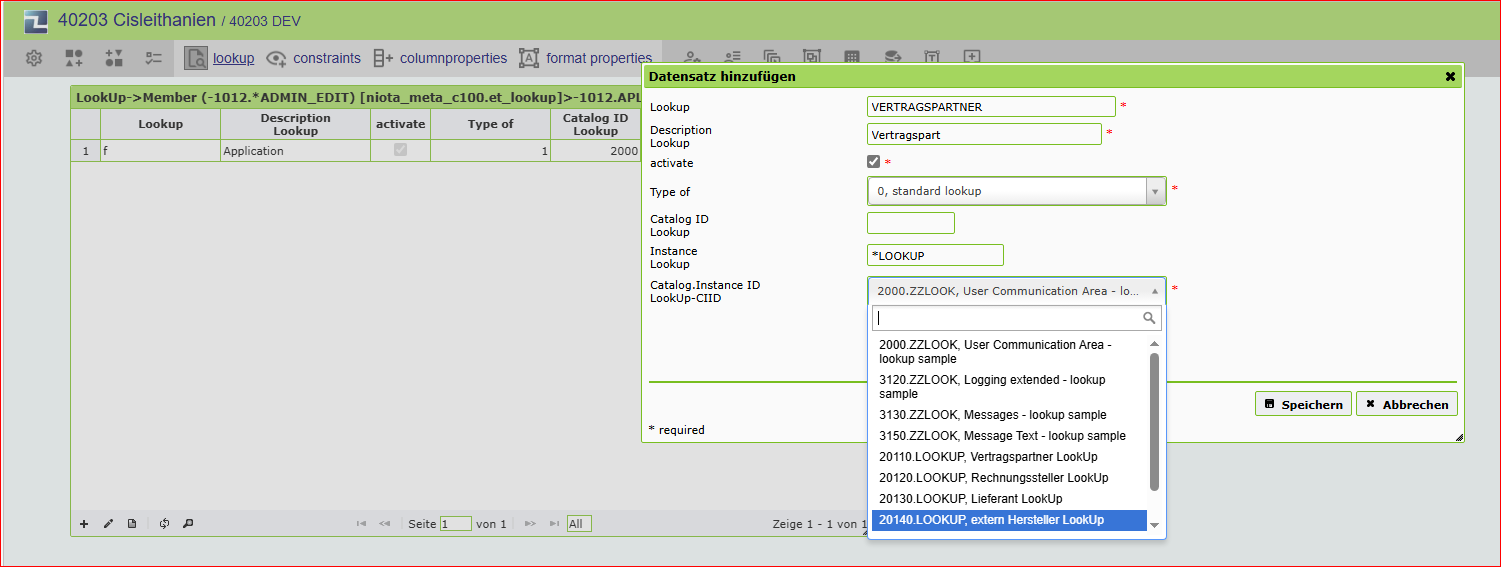
LookUp – Konfigurationen unterliegen keiner Berechtigungsprüfung. Wenn eine Benutzer eine Tabelle sehen darf, dann darf er automatisch auch alle verknüpften LookUp Werte sehen.



LookUp Objekte

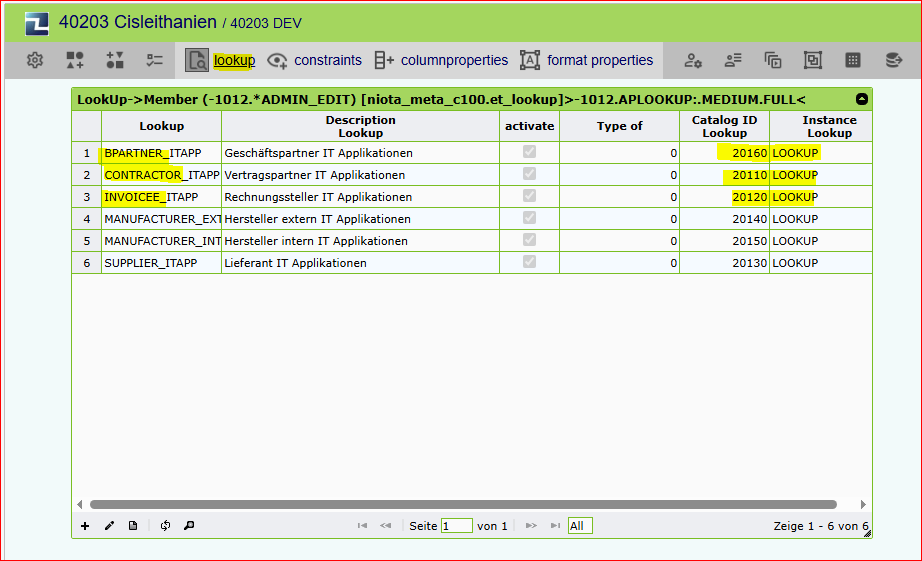
Ein LookUp Objekt ist technisch gesehen ein „Alias“ auf eine LookUp Instanz.

Wenn man separate LookUp Instanzen definiert und diese auch als solche qualifiziert (use as Lookup), dann werden diese (und nur diese) auch bei der Definition von LookUp Objekten angezeigt.

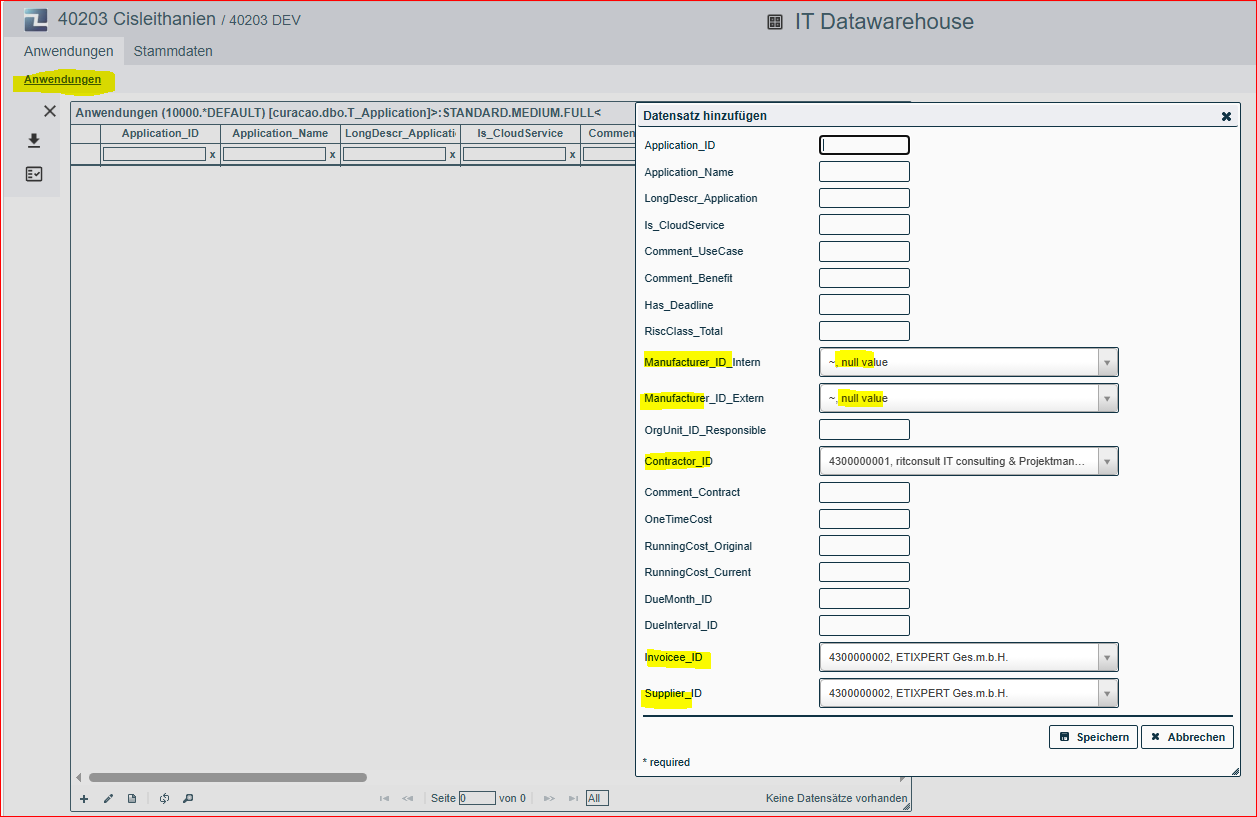


Die nun definierten Lookup Objekte sind in diesem Mandanten nun für alle Objekte verfügbar.

„Jedem“ Feld kann nur dieses LookUp als (Feld-) Eigenschaft zugewiesen werden und am FrontEnd werden somit die möglichen Werte als DropDown angeboten.

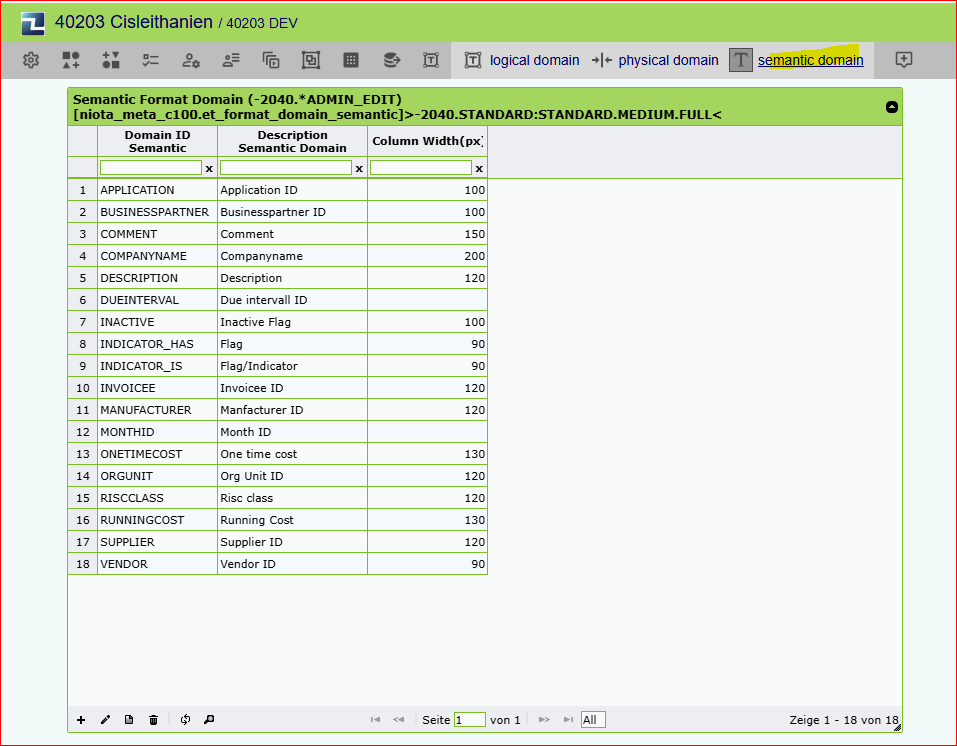


Wenn diese Lookup zugeordnet sind, dann sieht die Erfassungsmaske im ersten Schritt so aus.



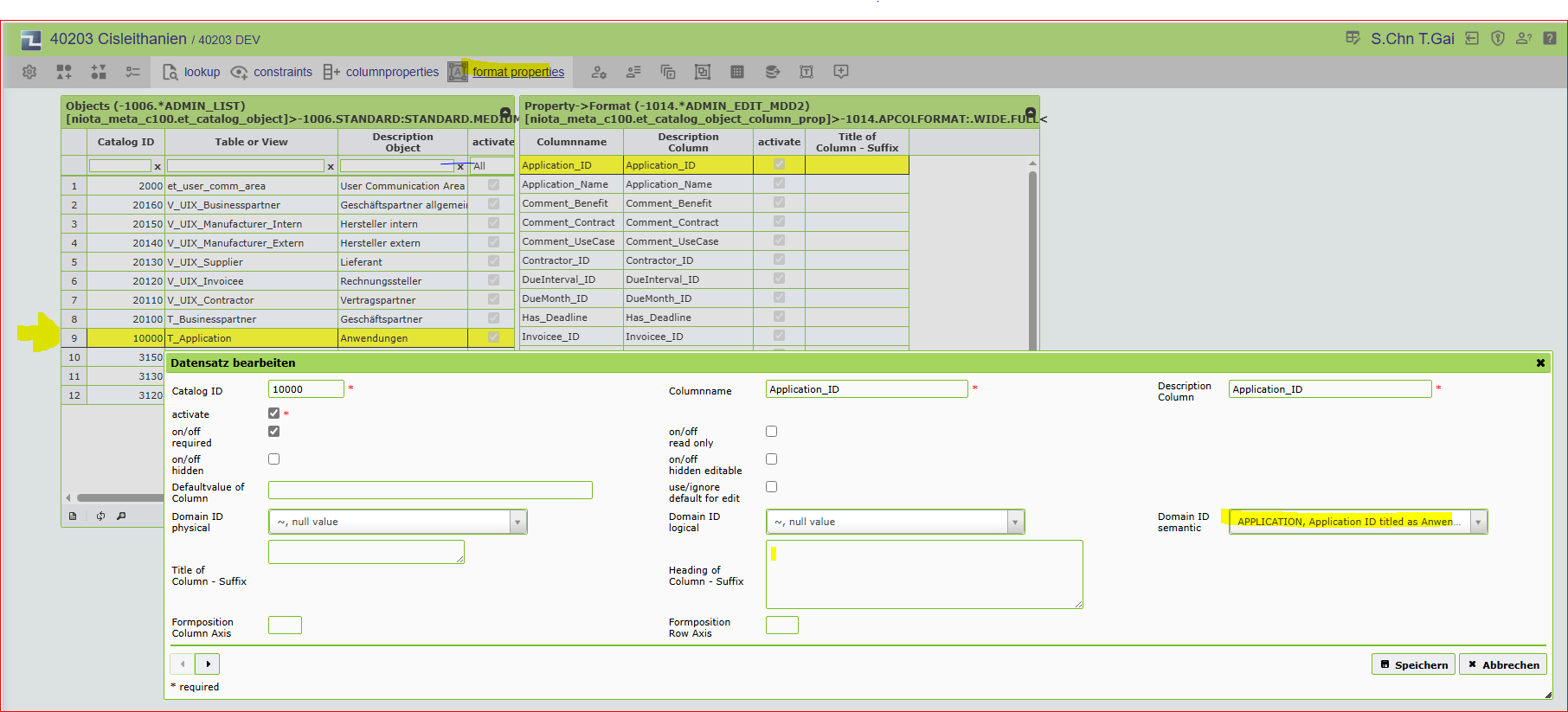
Es auch Möglich NULL-Werte zuzulassen. Da ist auf Datenbank-Ebene mit referentieller Integrität allerdings NICHT abbildbar.

Für die restlichen/alle Felder der Tabellen die semantischen Domänen definieren und anschliessend diese auch als Feld-Eigenschaft zuweisen.

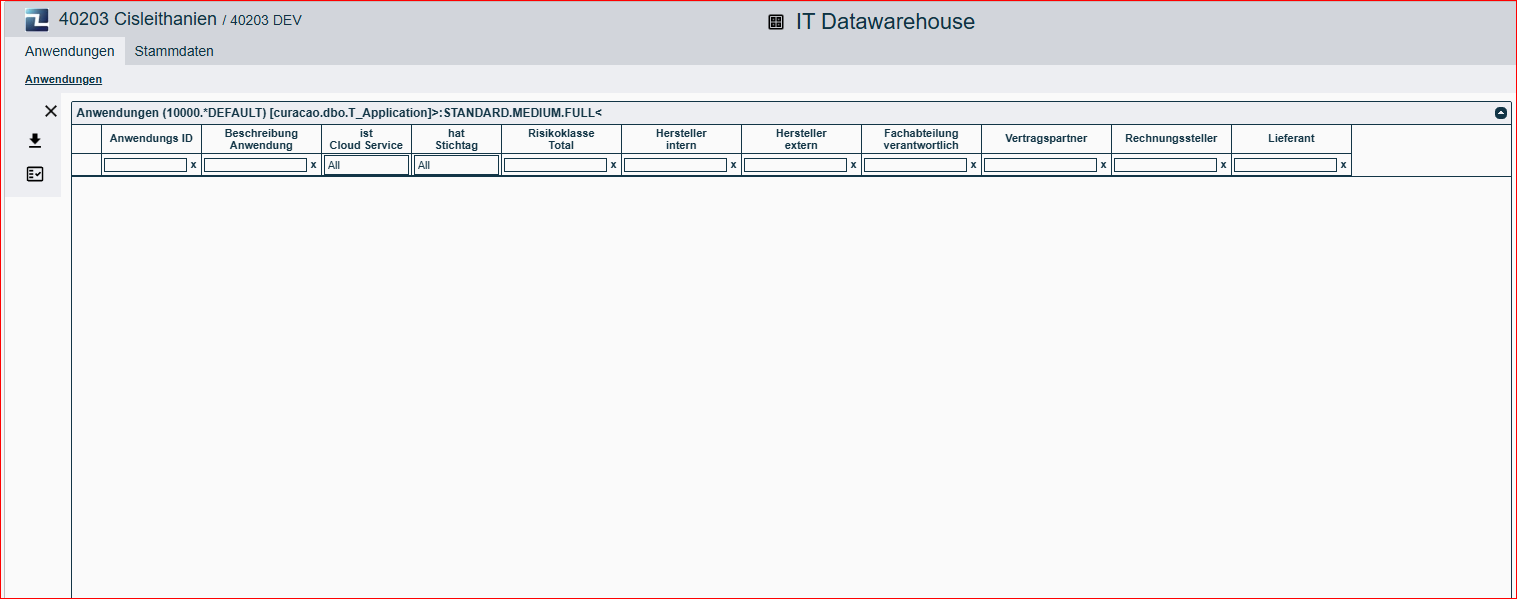


Der Text aus „Title of Column Suffix“ und “Heading of Column Suffix” warden zusätzlich zu den Texten aus der semantischen Domäne angezeigt.

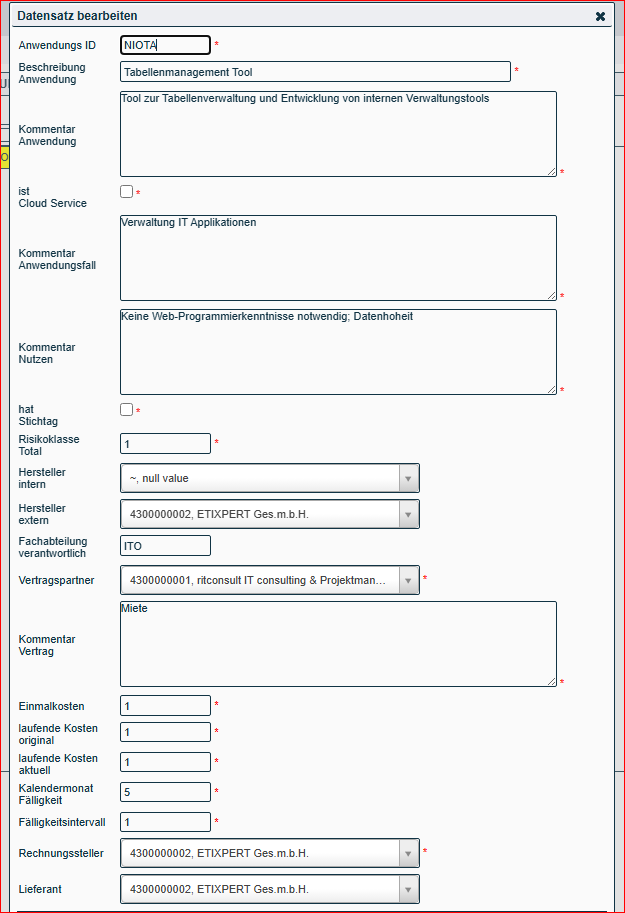
Es ist daher sinnvoll, hier nur „Zusatzinformationen“ anzuführen und die allgemeinen Werte auf der Domäne zu hinterlegen.



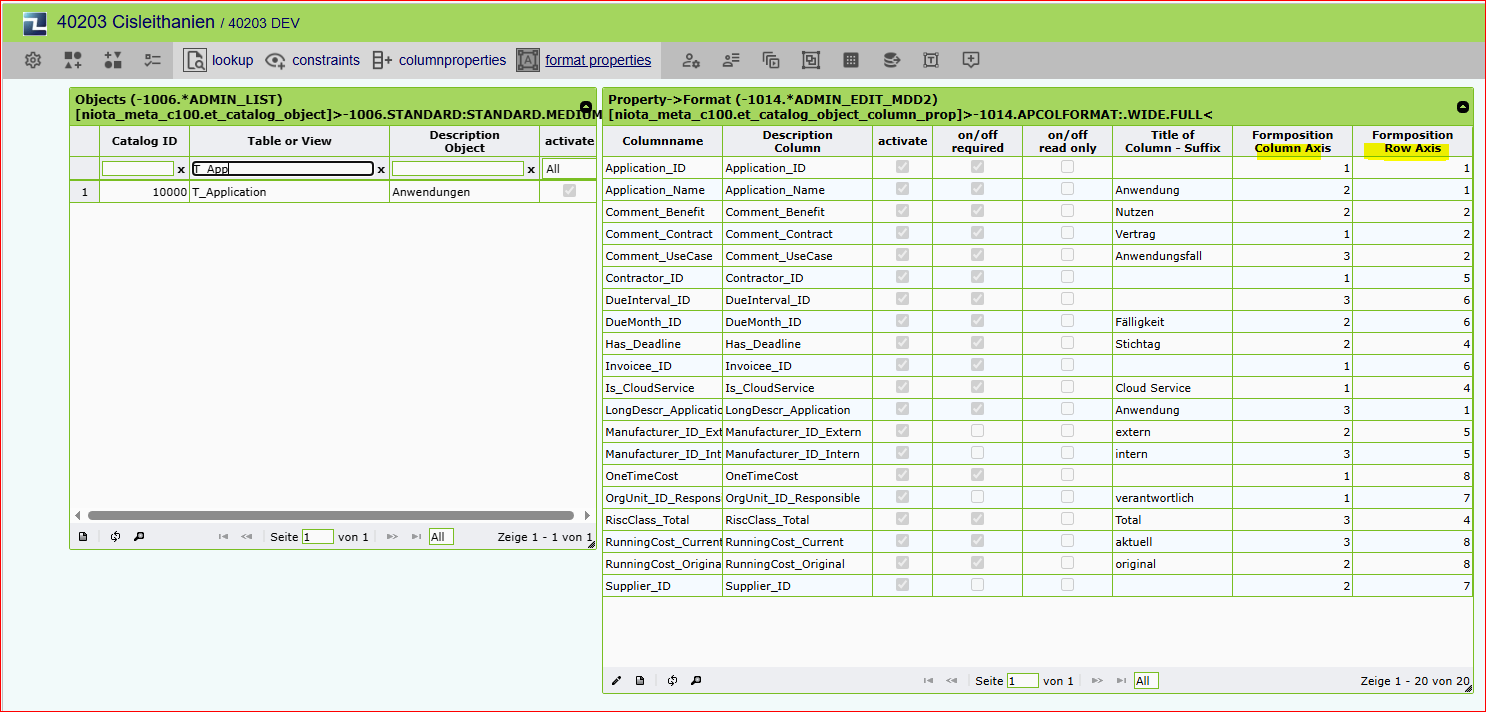
Die Liste der Anwendungen kann man nun zum Beispiel nur mit diesen Feldern darstellen.

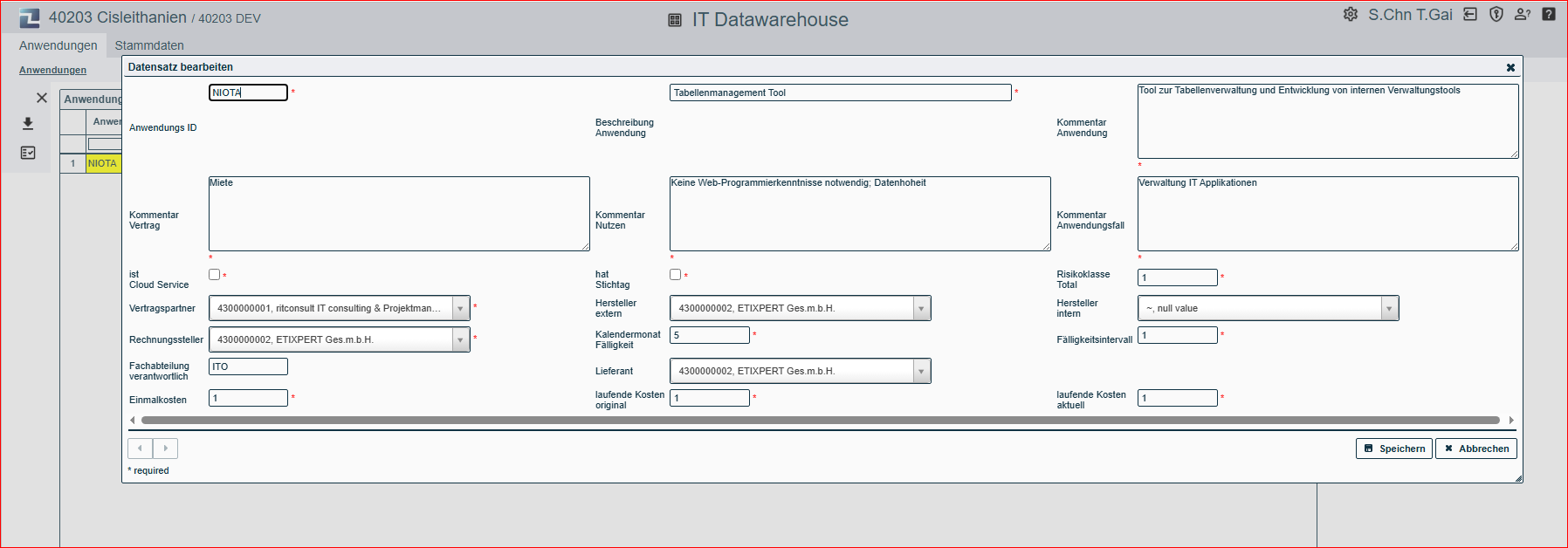


In der Bearbeitungsmaske sind allerding auch Kommentare und Kosten sichtbar.



Durch Zuordnung der Tabellenfelder in einen Zeilen-/Spaltenraster, kann man die Reihenfolge der Felder im Formular umgestalten.





Beziehungen von Tabellen

# Anhang

