PROJEKTDOKUMENTATION ABSCHLUSSPROJEKT KURS RELATIONALE DATENBANKEN IN MS SQL (PATRICK PRIEWE)

10.10.2024

Inhalt

Projektkonzept und Zielsetzung 2
Datenbankstruktur und ER-Diagramm 3
ER-Diagramm 3
Beziehungen 4
Tabellenbeschreibungen, Constraints, Indizes 4
Tabellen für Personengruppen 4
Kunden (Tabelle dbo.customers) 5
Mitarbeiter (Tabelle dbo.employees) 5
Lieferanten (Tabelle dbo.suppliers) 6
Benutzer (Tabelle dbo.users) 6
Tabellen für Waren und Bestände 7
Warengruppen (Tabelle dbo.categories) 7
Produkte (Tabelle dbo.products) 8
Lagerorte (Tabelle dbo.inventory_storagelocations) 8
Lagerbestände (Tabelle dbo.inventory) 8
Tabellen zur Auftragsverarbeitung 9
Bestellstatus (Tabelle dbo.orders_status) 9
Bestellungen (Tabelle dbo.orders) 9
Rechnungsstatus (Tabelle dbo.invoice_status) 10
Rechnungen (Tabelle dbo.invoices) 10
Weitere Parameter (Tabelle dbo.commercial_parameters) 11
Referentielle Integrität 11
Businesslogik und mögliche Workflows 11
Prozeduren und Trigger als Einstiegspunkte in zentrale Prozesskette 13
trgWhenInsertingOrderDoStuff 13
spCheckExistingOrderForCommission 13
spCreateNewOrder 13
Zentrale Prozesskette 14
Zweck der zentralen Prozesskette 14
Bestandteile der Prozesskette 14
spDefineOrderStatusAndProcessIfCommissioned 14
spCreateInvoiceOnOrder 15
Ergebnis der zentralen Prozesskette (Output) 16
Weitere Prozeduren und Trigger 16
spChangeInvoiceStatusAndDoStuff 16
trgDeleteInvoicesOnlyWhenNoOrder 17
Unterstützende Sichten für Workflows und Analysen 17

Generelle Sicht für Aufträge: dbo.vw_0Aufträge 17
Sicht Aufträge überschrittenem Zahltag: dbo.vw_1AufträgeMitÜberfälligerZahlung 18
Sicht für kritische Lagerbestände: dbo.vw_2KritischeLagerbestände 18
Sicht für abgelehnte Bestellungen: dbo.vw_3AbgelehnteBestellungenMitLieferbarkeitsangabe 19
Sicht als Basis für Umsatzanalysen: dbo.vw_4BasisFürUmsatzanalysen 19
Kunden nach Rohgewinn 20
Kunden nach Handelsspanne 20
Produkte und ihre Verkaufsdaten 21
Warengruppen und ihre Verkaufsdaten 21
Weitere nützliche Abfragen und Prozeduraufrufe 21
Schnelles Lagernachfüllen mit Zufallsdaten 21
Gezielt überfällige Rechnungen aufspüren und overdue setzen 22
Abgelehnte Bestellungen erneut in Prozesskette geben 22
Entgangene Umsätze aufgrund von Rabatten ausgeben 22
Zusammenfassung 22
Weiterentwicklungspotenziale 23

Projektkonzept und Zielsetzung

Für die Heimtierbedarfhandlung Bird Paradise sollen in MS SQL Daten zu kommerziellen Personengruppen (Kunde, Mitarbeiter, Lieferanten), Produkten (Produkte, Bestände, Lager) und Bestellvorgängen erfasst und verwaltet werden können. Es sollen die wesentlichen Geschäftsvorfälle über Prozeduren abgebildet werden, und Datenabfragen und Sichten zum Abruf und der Analyse von Daten eingerichtet werden.

Folgendes sind die Kernanforderungen:

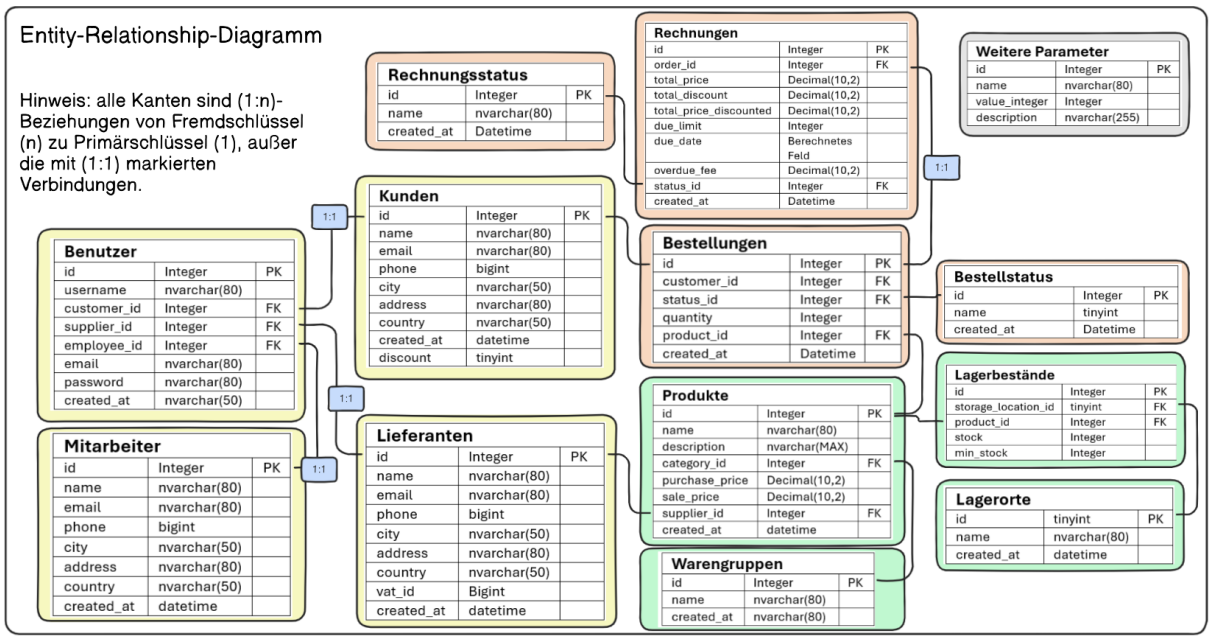
- **Grundsätzliche Datenverwaltung**:
  - Personengruppen (Mitarbeiter, Kunde, Lieferant), Produktdaten (Produkt, Lagerbestand, Lagerort), Auftragsdaten (Bestellung, Rechnung) sollen geführt werden können.
- **Auftragsverwaltung**:
  - Eingegangene Kundenbestellungen von Produkten sollen verarbeitet werden können.
  - Für eingegangene Kundenbestellungen sollen Rechnungen erstellt werden.
  - Für eingegangene Kundenbestellungen sollen verfügbare Produktbestände beachtet werden und ein Lieferbarkeits- und Lieferstatus verfolgt werden können.
  - Kundenbestellungen und Rechnungen werden Aufträgen zusammengefasst.
  - Für Aufträge soll abhängig von der Lieferbarkeit der Bestellung (bestandsabhängig) und dem Rechnungsstatus (offen, bezahlt, überfällig) eine Statuslogik eingerichtet werden.
- **Kommerzielle Anforderungen**:
  - Verschiedene Umsatzanalysen für sollen durchgeführt werden können.
  - Für Rechnungen sollen Mahngebühren erhoben werden können.
  - Stammkunden sollen Rabatte erhalten können.

Für obige Anforderungen sollen möglichst viele Arbeitsschritte teilautomatisiert werden. Dies wird durch Prozeduren, Datenbanktrigger, und Sichten umgesetzt. Es werden außerdem einige Skripte mit weiteren Abfragen bzw. Datenbankoperationen bereitgestellt.

Datenbankstruktur und ER-Diagramm

Zur Umsetzung obiger Anforderungen betrachten wir in diesem Kapitel zunächst unser Datenmodell.

ER-Diagramm



Datentabellen für Personengruppen sind hier gelb markiert, Datentabellen für die Auftragsverwaltung orange, Datentabellen für die Produktdaten grün, weitere Parameter grau.

Beziehungen

Obiges ER-Diagramm gruppiert zur Übersichtlichkeit nach Farben. In der Folge werden die einzelnen Beziehungen zwischen den Tabellen beschrieben:

- **Gelb: Personengruppen und Benutzer.** Hier findet wenig Logik in der später vorgestellten Businesslogik statt.
  - **Ein Mitarbeiter** kann genau **einen Benutzer** haben, **muss dies aber nicht**! (1:1 Mitarbeiter zu Benutzer)
  - **Ein Lieferant**
    - kann genau **einen Benutzer** haben, **muss dies aber nicht**! (1:1 Lieferant zu Benutzer)
    - kann **mehrere Produkte** haben (1:n Lieferant zu Produkt)
  - **Ein Kunde**
    - kann genau **einen Benutzer** haben, **muss dies aber nicht**! (1:n Kunde zu Benutzer)
    - kann **mehrere Bestellungen** haben (1:n Kunde zu Bestellungen)
  - **Ein Benutzer**
    - muss in jedem Fall **genau eine Verbindung** zu **entweder** einem Lieferanten, **oder** einem Kunden, **oder** einem Mitarbeiter haben (**exklusives** „oder"!) (1:1 Benutzer zu (Kunde ODER Lieferant ODER Mitarbeiter))
  - **Hinweis:** Die Bezeichnung „1:1" zwischen den gelben Tabellen ist also eigentlich **nicht ganz korrekt**. Sie wurde hier aber trotzdem gewählt, da für den einzelnen Benutzer genau eine eine eindeutige Verbindung existieren muss, die in eine der drei Tabellen zeigt.
- **Orange: Auftragsverarbeitung**. Hier findet in der später vorgestellten Businesslogik sehr viel statt.
  - **Ein Rechnungsstatus** kann von **mehreren Rechnungen** angenommen werden (1:n Status zu Rechnungen).
  - **Ein Bestellstatus** kann von **mehreren Bestellungen** angenommen werden (1:n Status zu Bestellungen)
  - **Eine Bestellung**
    - hat genau **eine Rechnung** (1:1 Bestellung zu Rechnung)
    - hat genau **einen Status**, der aber wiederum **von mehreren Bestellungen** angenommen werden kann (n:1 Bestellung zu Status)
    - bezieht sich auf **genau ein Produkt**, das aber wiederum **von mehreren Bestellungen** bestellt werden kann (n:1 Bestellung zu Produkt)
    - kommt von **genau einem Kunden**, der aber **mehrere Bestellungen** tätigen kann (n:1 Bestellung zu Kunde)
- **Grün: Waren und Bestände**. Hier findet wiederum weniger Businesslogik statt
  - **Eine Warengruppe** kann **mehrere Produkte** haben (1:n Warengruppe zu Produkte)
  - **Ein Produkt**
    - kann **mehrere Lagerbestände** (an verschiedenen Lagerorten) haben, aber **an jedem Lagerort nur einen!**
    - kann (siehe oben) mehrere Bestellungen haben.
    - kann genau einen Lieferanten haben, der wiederum mehrere Produkte haben kann (n:1 Produkt zu Lieferant)

Tabellenbeschreibungen, Constraints, Indizes

In diesem Kapitel werden die einzelnen Tabellen genauer beschrieben

Tabellen für Personengruppen

In diesem Kapitel werden die Tabellen zur Abbildung von Personengruppen und der Nutzerverwaltung beschrieben. Personengruppen im Projekt sind die folgenden:

- Kunden (customers):
  - Geben Bestellungen auf, die von von außen ins System gelangen.
  - Können einen Nutzer erhalten (optional) für Bestellung und Lesezugriff auf ausgewählte Inhalte.
- Mitarbeiter (employees):

    - Beeinflussen Daten im System, indem Sie Workflows aufrufen. Erhalten Daten aus dem System, indem Sie Datenabfragen abrufen. Für beides könnte ein Fremdsystem angebunden werden (oder ein erweitertes technisches Rollenkonzept in MS SQL eingerichtet werden).
    - Können einen Nutzer erhalten (optional) mit Berechtigung zum Aufruf von Prozeduren und Datenabfragen.
- Lieferanten (Suppliers):
    - Nehmen keine Eingriffe ins System vor (keine Aufrufe von Prozeduren, keine Datenabfragen).
    - Können einen Nutzer erhalten (optional) für Lesezugriff auf ausgewählte Inhalte.

Kunden (Tabelle dbo.customers)

Die Datenbanktabelle **dbo.customers** bildet Kunden in der Datenbank ab.

Tabellenstruktur

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	nvarchar(80)	Name	NOT NULL
email	nvarchar(80)	Mailadresse	NOT NULL
phone	bigint	Telefonnummer	
city	nvarchar(50)	Stadt	
address	nvarchar(80)	Straße, Hausnummer, Zusatz	
country	nvarchar(50)	Herkunftsland	
created_at	datetime	Zeitstempel für Anlage	NOT NULL, Default getdate()
discount	tinyint	Prozentsatz für Rabattierung	NOT NULL, Default 0

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen.

Hinweise und weitere Erklärungen:

- Das Feld discount wird im Rahmen von Workflows beschrieben und ausgelesen.
- Auf die Trennung allgemeiner Personendaten (Name, Herkunft) von kundenspezifischen Daten (in diesem Fall das Feld discount) wird zugunsten der Einfachheit verzichtet (keine schärfere Normalisierung). Weitere kundenspezifische Daten (z.B. Stammkunde, ...) könnten ergänzt werden.

Mitarbeiter (Tabelle dbo.employees)

Die Datenbanktabelle **dbo.employees** bildet Mitarbeiter in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	nvarchar(80)	Name	NOT NULL
email	nvarchar(80)	Mailadresse	NOT NULL
phone	bigint	Telefonnummer	
city	nvarchar(50)	Stadt	
address	nvarchar(80)	Straße, Hausnummer, Zusatz	
country	nvarchar(50)	Herkunftsland	
created_at	datetime	Zeitstempel für Anlage	NOT NULL, Default getdate()

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen.

Hinweise und weitere Erklärungen:

- Auf die Trennung allgemeiner Personendaten (Name, Herkunft) von mitarbeiterspezifischen Daten wird zugunsten der Einfachheit verzichtet (keine schärfere Normalisierung).

- Mitarbeiterspezifische Daten (z.B. Gehalt, Steuernummer, ..) könnten ergänzt werden.

Lieferanten (Tabelle dbo.suppliers)

Die Datenbanktabelle **dbo.suppliers** bildet Lieferanten in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	nvarchar(80)	Name	NOT NULL
email	nvarchar(80)	Mailadresse	NOT NULL
phone	bigint	Telefonnummer	
city	nvarchar(50)	Stadt	
address	nvarchar(80)	Straße, Hausnummer, Zusatz	
country	nvarchar(50)	Herkunftsland	
vat_id	Bigint	Umsatzsteuernummer	NOT NULL
created_at	datetime	Zeitstempel für Anlage	NOT NULL, Default getdate()

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen.

Hinweise und weitere Erklärungen:

- Auf die Trennung allgemeiner Personendaten (Name, Herkunft) von lieferantenspezifischen Daten (in diesem Fall das Feld vat_id) wird zugunsten der Einfachheit verzichtet (keine schärfere Normalisierung).
- Lieferantenspezifische Daten (z.B. Skonto, Rabattparameter, ..) könnten ergänzt werden.

Benutzer (Tabelle dbo.users)

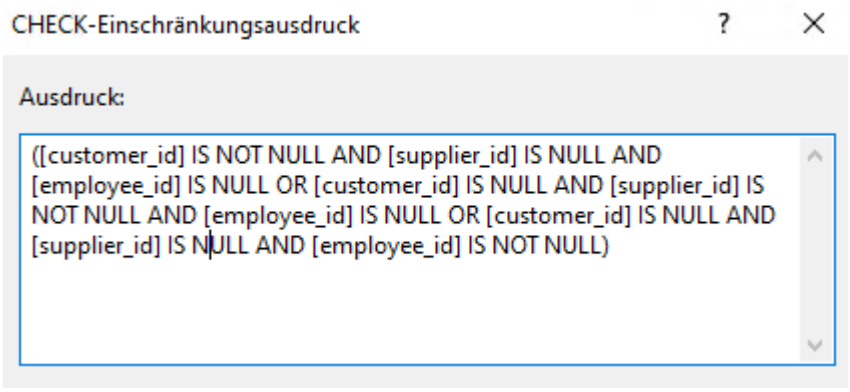
Die Datenbanktabelle **dbo.users** bildet Benutzer in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
username	nvarchar(80)	Für Login zwingend erforderlich	NOT NULL
customer_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.customers.id	Fremdschlüssel auf dbo.customers (siehe unten)
supplier_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.suppliers.id	Fremdschlüssel auf dbo.suppliers (siehe unten)
employee_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.employees.id	Fremdschlüssel auf dbo.employees (siehe unten)
email	nvarchar(80)	Für Login zwingend erforderlich	NOT NULL
password	nvarchar(80)	Für Login zwingend erforderlich	NOT NULL
created_at	nvarchar(50)	Zeitstempel für Anlage	NOT NULL, Default getdate()

Erläuterungen zu Fremdschlüsseln

- **Fremdschlüssel FK_user_cust:**
  - Fremdschlüsselfeld, das auf das Primärschlüsselfeld dbo.customers.id verweist.
  - Optionale Angabe, Erläuterung dazu weiter unten.
- **Fremdschlüssel FK_user_emp:**
  - Fremdschlüsselfeld, das auf das Primärschlüsselfeld dbo.employees.id verweist.
  - Optionale Angabe, Erläuterung dazu weiter unten.
- **Fremdschlüssel FK_user_suppl:**
  - Fremdschlüsselfeld, das auf das Primärschlüsselfeld dbo.suppliers.id verweist.
  - Optionale Angabe, Erläuterung dazu weiter unten.
- **Erzwingung einer Referenz auf Kunde, Lieferant, oder Mitarbeiter:**
  - Durch die Kombination einer Unique-Constraint und einer Check-Constraint wird sichergestellt, dass einem Benutzer immer entweder ein Kunde, ein Mitarbeiter, oder ein Lieferant zugewiesen werden kann. Außerdem können nie zwei verschiedene Benutzer auf einen selben anderen Datensatz verweisen. Dies wird wie folgt umgesetzt:

- **Check-Constraint CK_user_exactlyoneFK:**
  - Stellt sicher, dass ein Benutzer immer entweder einen Fremdschlüsseleintrag für einen Mitarbeiter, einen Kunden, oder einen Lieferanten erhält (**exklusives oder**!), und dass in jedem Fall auf einen dieser drei vergeben wird. Dazu dient folgender Ausdruck:



- **Unique-Constraint UQ_user_uniqueFK:**
  - Stellt sicher, dass keine zwei Benutzer auf denselben Zieldatensatz verweisen: eindeutiger Schlüssel auf die Spaltenkombination (customer_id, supplier_id, employee.id)
- Dadurch wird sichergestellt, dass eine einzige Benutzertabelle über Fremdschlüssel die Beziehung zu drei anderen Tabellen erhalten kann, aber stets eindeutig definiert ist, welchen Kontext der Benutzer hat. Hierdurch kann also anhand des Benutzerdatensatzes eine Rollenzuweisung erfolgen.

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert).

Unique Constraint UQ_user_uniqueFK als weiterer Index (eindeutig, nicht gruppiert) Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen.

Hinweise und weitere Erklärungen:

- Es ist durchaus erwünscht, dass z.B. ein Kunde (oder Lieferant oder Mitarbeiter) nicht zwingend einen Benutzer bekommt. Es kann z.B. Laufkundschaft geben, für die im Laden verkauft wird, oder die telefonisch bestellen. Für diese ist daher in der jeweiligen Tabelle (dbo.customers/dbo.suppliers) zwingend eine Emailangabe nötig.
- Umgekehrt hat ein Benutzer aber immer einen Bezug zu einem Kunden, Mitarbeiter, oder Lieferanten (kann nicht ohne Bezug existieren).
- Dass im Benutzer ein Feld Email existiert, und ebenso in der jeweiligen Referenztabelle, wird in Kauf genommen (ein Kunde kann in seinen Kundenstammdaten also eine andere Emailadresse besitzen, als diejenige, die er für seinen Benutzer verwendet!)

Tabellen für Waren und Bestände

Relevante Tabellen für Waren und Bestände sind:

- Produkte, die jeweils eindeutig einer Warengruppe zugehörig sind. Dies ermöglicht eindeutige Analysen nach Warengruppen.
- Produktbestände für jedes Produkt und verschiedene Lagerorte für diese Produktbestände (in unserem Fall zwei Lager; Hauptlager und Außenlager)

Warengruppen (Tabelle dbo.categories)

Die Datenbanktabelle **dbo.categories** bildet Warengruppen in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	nvarchar(80)	Bezeichnung	NOT NULL

created_at	nvarchar(80)	Zeitstempel für Anlage	NOT NULL, Default getdate()

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen.

Produkte (Tabelle dbo.products)

Die Datenbanktabelle **dbo.products** bildet Produkte in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	nvarchar(80)	Produktbezeichnung	NOT NULL
description	nvarchar(MAX)	Produktbeschreibung	NOT NULL
category_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.category.id	NOT NULL, Fremdschlüssel
purchase_price	Decimal(10,2)	Einkaufspreis	NOT NULL, Check-Constraint (s.u.)
sale_price	Decimal(10,2)	Verkaufspreis	NOT NULL, Check-Constraint (s.u.)
supplier_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.supplier.id	NOT NULL, Fremdschlüsselfeld
created_at	datetime	Zeitstempel für Anlage	NOT NULL, default getdate()

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen.

Hinweise zu Constraints:

- Check-Constraint **CK_profitable_prices:**
  - Legt fest, dass die Differenz zwischen Verkaufs- und Einkaufspreis mindestens die Mehrwertsteuer (19%) erwirtschaftet:
    ```
    ([sale_price]>=[purchase_price]*(1.19))
    ```
  - (Hinweis: Im Handel zahlt der Händler auf den Einkaufspreis beim Lieferanten keine Mehrwertsteuer, bzw. diese wird zurückerstattet nach HGB. Dies resultiert in obigem Ausdruck.)

Lagerorte (Tabelle dbo.inventory_storagelocations)

Die Datenbanktabelle **dbo.storagelocations** bildet Lagerorte von Produkten in der Datenbank ab (dort werden die physischen Lagerbestände aufbewahrt):

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	tinyint	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	nvarchar(80)	Name/Beschreibung des Lagerorts (in unserem Fall gibt es zwei Einträge: Hauptlager und Außenlager)	NOT NULL
created_at	datetime	Zeitstempel für Anlage	NOT NULL, default getdate()

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen

Lagerbestände (Tabelle dbo.inventory)

Die Datenbanktabelle **dbo.inventory** bildet Lagerbestände von Produkten in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults

id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
storage_location_id	tinyint	Fremdschlüsselfeld, referenziert auf dbo.storagelocations.id	NOT NULL, Fremdschlüsselfeld
product_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.products.id	NOT NULL, Fremdschlüsselfeld
stock	Integer	Lagerbestand eines Produkts in einem Lagerort	NOT NULL
min_stock	Integer	Mindestbestand eines Produkts in einem Lagerort	NOT NULL, default 0

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Unique-Constraint für die Spaltenkombination (product_id, storage_location_id) als weiterer Index (eindeutig, nicht gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen

Hinweise zu Constraints:

- Unique-Constraint **UQ_inventory_product_location**: Stellt sicher, dass es genau einen Bestand (inventory) für ein Produkt pro Lagerort gibt (in unserem Setting gibt es zwei Lagerorte).

Tabellen zur Auftragsverarbeitung

Wir betrachten inhaltlich in unserem Projekt einen „Auftrag“ als die Kombination aus zwei Objekten: Der Bestellung und der Rechnung.

Diese werden in zwei Tabellen geführt und haben jeweils eine weitere Tabelle zum Hinterlegen von möglichen Status-Werten die eine Bestellung und eine Rechnung haben können (allein wegen dieser Ausdifferenzierung des Status werden sie in getrennten Tabellen geführt, obwohl sie in vielen Prozeduren und Sichten gemeinsam betrachtet werden).

Bestellstatus (Tabelle dbo.orders_status)

Die Datenbanktabelle **dbo.orders_status** bildet den Status von Produktbestellungen in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	tinyint	Statusbezeichnung	NOT NULL
created_at	Datetime	Zeitstempel für Anlage	NOT NULL, default getdate()

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen.

Im Projekt verwendete Daten

Wir verwenden im Projekt:

- Status_id 1 = beauftragt
- Status_id 2 = ausgeliefert
- Status_id 3 = abgelehnt

Bestellungen (Tabelle dbo.orders)

Die Datenbanktabelle **dbo.orders** bildet Produktbestellungen von Produkten in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
customer_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.customers.id	NOT NULL, Fremdschlüsselfeld

product_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.products.id	NOT NULL, Fremdschlüsselfeld
quantity	Integer	Bestellte Menge eines Produkts	NOT NULL
status_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.orders_status.id	NOT NULL, Fremdschlüsselfeld
created_at	Datetime	Zeitstempel für Anlage	NOT NULL, default getdate()

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen

Hinweise zu Fremdschlüsseln:

- **FK_orders_customer, FK_orders_product, FK_orders_status** entsprechende Fremdschlüssel

Rechnungsstatus (Tabelle dbo.invoice_status)

Die Datenbanktabelle **dbo.invoice_status** bildet den Status von Rechnungen in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	nvarchar(80)	Bezeichnung des Status	NOT NULL
created_at	Datetime	Zeitstempel für Anlage	NOT NULL

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert). Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen

Im Projekt verwendete Daten

Wir verwenden im Projekt:

- Status_id 1 = unpaid
- Status_id 2 = paid
- Status_id 3 = overdue

Rechnungen (Tabelle dbo.invoices)

Die Datenbanktabelle **dbo.invoices** bildet Rechnungen zu Bestellungen in der Datenbank ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
order_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.orders.id	NOT NULL, Fremdschlüsselfeld. Unique-Constraint
total_price	Decimal(10,2)	Wird Rechnungsanlage durch Prozeduren aus Produktdaten und Bestelldaten befüllt	NOT NULL
total_discount	Decimal(10,2)	Wird von diversen Prozeduren befüllt, falls Kunde Rabatt bekommt	NOT NULL
total_price_discounted	Decimal(10,2)	Wird durch Prozeduren befüllt. total_price mit Rabatt.	NOT NULL
due_limit	Integer	Zahlungsfrist in Tagen. Defaultbelegung 30	
due_date	Berechnetes Feld	Berechnet sich aus created_at und due_limit	
overdue_fee	Decimal(10,2)	Mahngebühr. Wird von Prozeduren befüllt	Default 0
status_id	Integer	Fremdschlüsselfeld, referenziert auf dbo.invoices.id	NOT NULL, Fremdschlüsselfeld
created_at	Datetime	Zeitstempel für Anlage	NOT NULL

Indizes

Primärschlüssel (id) als Clustered Index (eindeutig, gruppiert), sowie eindeutiger Fremdschlüssel (order_id) als Indizes. Keine weiteren Indizes, da weitestgehend alle Abfragen die id abrufen

Hinweise zu Constraints:

- **FK_invoices_orders** und **FK_invoices_status** jeweils Fremdschlüssel.
- Unique-Constraint **UQ_invoice_orderID**: Eindeutiger Fremdschlüssel auf orders

Weitere Parameter (Tabelle dbo.commercial_parameters)

Die Datenbanktabelle **dbo.commercial_parameters** bildet weitere Parameter ab:

Feldname	Datentyp	Beschreibung	Zusätzliche Constraints und Defaults
id	Integer	Identifier	NOT NULL, Primärschlüssel, inkrementelle Wertevergabe
name	nvarchar(80)	Bezeichnung des Parameters	NOT NULL
value_integer	Integer	Wert des Parameters	
description	nvarchar(255)	Beschreibung des Parameters	

Indizes

Primärschlüssel (id) als Index (gruppiert, eindeutig), sonst keine weiteren.

Hinweise:

Für unser Projekt sind hier folgende Parameter als Tabellenzeilen hinterlegt:

- Overdue_fee_percentage (Wert= 10): Prozentsatz Strafgebühr bei überfälligen Rechnungen. Wird auf rabattierten Gesamtpreis der Rechnung aufgeschlagen
- required_revenue_for_discount (Wert = 2000): Sobald ein Kunde diesen Wert mit der Summe seiner bezahlten Rechnungen überschreitet, bekommt er zukünftig Rabatt über die discount_rate
- discount_rate (Wert 5): Rabattrate in Prozent, die Kunde abgezogen bekommt, sobald er mehr Umsätze über bezahlte Rechnungen eingebracht hat als required_revenues_for_discount

Referentielle Integrität

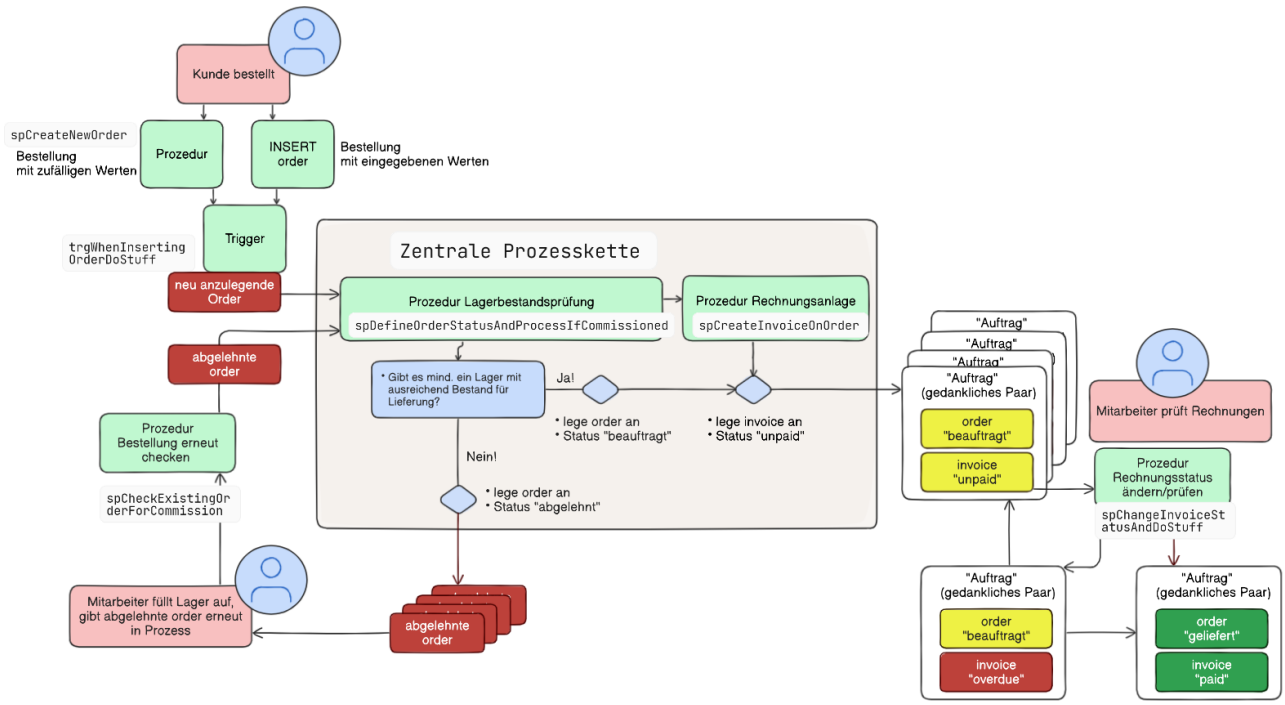
Die referentielle Integrität wird durch die Nutzung von Fremdschlüsseln gewährleistet. Für sämtliche Fremdschlüssel ist ON DELETE CASCADE gesetzt.

Businesslogik und mögliche Workflows

Wir rekapitulieren nochmals unsere Anforderungen an Geschäftsprozesse. Diese waren:

- **Auftragsverwaltung**:
  - Eingegangene Kundenbestellungen von Produkten sollen verarbeitet werden können.
  - Für eingegangene Kundenbestellungen sollen Rechnungen erstellt werden.
  - Für eingegangene Kundenbestellungen sollen verfügbare Produktbestände beachtet werden und ein Lieferbarkeits- und Lieferstatus verfolgt werden können.
  - Kundenbestellungen und Rechnungen werden Aufträgen zusammengefasst.
  - Für Aufträge soll abhängig von der Lieferbarkeit der Bestellung (bestandsabhängig) und dem Rechnungsstatus (offen, bezahlt, überfällig) eine Statuslogik eingerichtet werden.
- **Kommerzielle Anforderungen**:
  - Verschiedene Umsatzanalysen für sollen durchgeführt werden können.
  - Für Rechnungen sollen Mahngebühren erhoben werden können.
  - Stammkunden sollen Rabatte erhalten können.

Um diese abzubilden, etablieren wir die Businesslogik auf folgenden Abbildungen. Im Anschluss werden die Prozeduren genau erklärt.



Obige Logik erlaubt uns also drei hauptsächliche Workflows:

1. **WORKFLOW A: Bestellungen anlegen, Lagerbestandsprüfung, Rechnungserzeugung**.
    a. Der Nutzer muss für den Workflow **lediglich** die Prozedur spCreateNewOrder nutzen, um (eher zu Testzwecken) eine zufallsgenerierte Bestellung zu erzeugen, oder alternativ über INSERT ORDER tatsächliche Werte eingegeben. Weitere Arbeitsschritte muss er nicht vornehmen!
    b. Die zentrale Prozesskette prüft die Lagerbestände
        i. Ist genug in den Lagern, wird ein Auftrag als gedankliches Paar erzeugt, bestehend aus den Objekten order (Status „beauftragt“) und invoice (Status „unpaid“)
        ii. Ist nicht genug in den Lagern, wird lediglich das Objekt order im Status „abgelehnt“ zurückgegeben.
2. **WORKFLOW B: Abgelehnte Bestellungen erneut prüfen lassen**.
    a. Der Nutzer kann sehr bequem einst abgelehnte Bestellungen einfach erneut in den Prozess geben. Dafür muss er **lediglich** die Prozedur spCheckExistingOrderForCommission aufrufen und die Order-ID übergeben. Weitere Arbeitsschritte muss er nicht vornehmen!
    b. Beachte: Wenn die Bestellung nun zum Auftrag wird (und nicht erneut abgelehnt wird), hat die zugehörige Bestellung ein älteres Erzeugungsdatum als die nun später entstandene Rechnung. Das ist wichtig, damit die Zahlungsfrist in der Rechnung korrekt ist! (Würde das Erzeugungsdatum der Bestellung in die später erzeugte Rechnung geschrieben, wäre diese eventuell fälschlicherweise sofort overdue!)
3. **WORKFLOW C: Rechnungen prüfen**.
    a. Der Nutzer kann Rechnungen zu vorliegenden Aufträgen prüfen. Hier gibt es zwei Szenarien:
        i. Der Kunde möchte eine Rechnung („unpaid“ oder „overdue“) auf bezahlt („paid“) setzen. Dafür muss er **lediglich** Rechnung an die Prozedur spChangeIncoiceStatusAndDoStuff übergeben und den gewünschten Zielstatus 2 (=“paid“) mitgeben.
        ii. Der Kunde möchte eine Rechnung auf Fristsäumigkeit (overdue) prüfen. Dazu muss er **lediglich** die Rechnung an die Prozedur spChangeIncoiceStatusAndDoStuff übergeben und den gewünschten Status 3 (=“overdue“) mitgeben. Die Prozedur prüft dann, ob die Rechnung tatsächlich überfällig ist, und setzt den Status entsprechend. Dasselbe gilt, wenn er den Zielstatus 1 (=“unpaid“) mitgibt.
    b. In beiden Szenarien oben wird von der Prozedur immer zusätzlich folgendes vorgenommen:

i. Es wird im Fall des resultierenden Status „overdue“ eine Mahngebühr in die Rechnung geschrieben. Hierfür werden Parameter aus der Tabelle „commercial_parameters“ und defaults aus der Rechnung herangezogen.
ii. Es wird geprüft, ob der Kunde in der Summe aller seiner vergangenen bezahlten Rechnungen einen bestimmten (über die Tabelle commercial_parameters konfigurierbaren) Gesamtumsatz erbracht hat. Falls ja, bekommt er zukünftig einen Rabatt!

Zu allen aufgeführten Workflows gibt es unterstützende Views, die in einem späteren Kapitel erklärt werden.

Zusätzlich gibt es Views für kommerzielle Analysen (Umsatz), die ebenfalls später gezeigt werden.

Prozeduren und Trigger als Einstiegspunkte in zentrale Prozesskette

Folgende Trigger und Prozeduren rufen allesamt dieselbe Folgeprozedur auf, die dieselbe zentrale Prozesskette (siehe entsprechendes Kapitel) einleitet.

trgWhenInsertingOrderDoStuff

Dieser Trigger ist als AFTER-INSERT-Trigger an die Tabelle dbo.orders angelegt. Er übergibt **neu anzulegende Bestellungen** an die zentrale Prozesskette (siehe entsprechendes Kapitel).

Der Trigger ist durch einen zusätzlichen Try-Catch-Block mit Transaktion und Rollback gesichert.

spCheckExistingOrderForCommission

Diese Prozedur übergibt die gewählte Bestellung erneut in die zentrale Prozesskette. Sie ist dafür gedacht, **abgelehnte Bestellungen erneut durchzuschicken**, da in der Zwischenzeit das Produkt lieferbar sein könnte (Bestand wurde aufgefüllt). Es wird dabei geprüft, ob hier wirklich eine Bestellung im Status „abgelehnt“ reingeschickt wird (im Code wird zur Prüfung geschaut, ob es bereits eine Rechnung zur Bestellung gibt, was gleichbedeutend ist).

Der Unterschied zum Einstiegspunkt über den Trigger trgWhenInsertingOrderDoStuff ist also folgender:

- In dieser Prozedur hier wird als Datumswert getdate(), also das Tagesdatum, in die Prozesskette gegeben!
- Resultat ist dann (bei erfolgreichem Durchlauf durch die zentrale Prozesskette) also die Kombination aus **order** (mit ggf. **älterem Anlagedatum**, da die Bestellung ja schon früher angelegt wurde), und **invoice** mit dem **Tagesdatum als Anlagedatum**!
- Dies ist insbesondere wichtig, da die Rechnung ja erst potenziell neu angelegt wird, und der Zahltag entsprechend anhand des tagesaktuellen Anlagedatums derinvoice berechnet werden muss!
- Über den trgWhenInsertingOrderDoStuff und spCreateNewOrder hingegen wird das Datum, das bei der Anlage der Bestellung gewählt wurde, auch in die Rechnung geschrieben!

Da die Prozedur sich (bis auf die andere Übergabe des Tagesdatums) sonst inhaltlich nicht vom Trigger trgWhenInsertingOrderDoStuff unterscheidet, wird auf weitere Erklärungen verzichtet.

spCreateNewOrder

Diese Prozedur kann verwendet werden, um einen "zufälligen" Auftrag zu erstellen, also eine Bestellung (order) für ein zufälliges Produkt (product_id aus Datenbestand zufällig gewählt) mit zufälliger Menge (quantity), zufälligem Bestelleingangsdatum (created_at zwischen dem 1.1.2024 und heute), sowie eine zugehörige Rechnung (invoice). Diese wird dann in die zentrale Prozesskette (siehe entsprechendes Kapitel) geschickt.

Inputparameter:

- @quantity INT: Bestellmenge, default=5 (optionaler Parameter)

Prozedurschritte:

- *Deklariere Hilfsvariablen*
- *Beginne eigentliche Prozedur:*
    - *Erzeuge Zufallsdatum zwischen 1.1.2024 (hart hinterlegt) und Tagesdatum (getdate()).*
    - *Erzeuge Bestellung über INSERT auf Tabelle orders.*
        - *Verwende dabei zufällige Kunden-ID zwischen 1 und 20 (20 Kunden in Testdatenbank), zufällige Produkt-ID zwischen 1 und 30 (30 Produkte in Testdatenbank)*
        - *Verwende Status 1 (=beauftragt) (fest hinterlegt)*
    - *Rufe mit den Ergebnisdaten spDefineOrderStatusAndProcessIfCommissioned auf (und gehe damit in die zentrale Prozesskette)*
    - *Gib zu Kontrollzwecken die erzeugte order_id und invoice_id aus (PRINT)*

Das Ganze wird innerhalb eines Try-Catch-Blocks als Transaktion ausgeführt (Rollback bei Catch).

Keine Outputparameter

Ergebnis

- Es wird eine **neue Bestellung mit zufallsgenerierten Daten in die zentrale Prozesskette** geschickt (für das Ergebnis der zentralen Prozesskette siehe entsprechendes Kapitel bzw. Schaubild zu Businesslogik).

Zentrale Prozesskette

Zweck der zentralen Prozesskette

Zweck der zentralen Prozesskette ist immer derselbe (egal von wo sie aufgerufen wird):

Eine eingespeiste Bestellung soll geprüft werden, ob es genügend Lagerbestand gibt, um die Bestellung zu liefern. Es soll dabei im lieferbaren Fall eine zugehörige Rechnung angelegt werden und der Bestand reduziert, im nicht lieferbaren Fall wird die Bestellung abgelehnt (ohne Rechnung, ohne Bestandsreduzierung).

Bestandteile der Prozesskette

Die zentrale Prozesskette besteht aus der **Verkettung der Prozeduren spDefineOrderStatusAndProcessIfCommisioned und CreateInvoiceOnOrder**. Diese werden in den beiden Folgekapiteln beschrieben.

spDefineOrderStatusAndProcessIfCommissioned

Diese Prozedur **startet die Prozesskette**. Sie wird **automatisch aufgerufen über die Einstiegspunkte, die im entsprechenden Kapitel beschrieben sind:**

- trgWhenInsertingOrderDoStuff: wenn eine Bestellung (order) über Insert angelegt wird, und dieser AFTER-INSERT-Trigger an der orders-Tabelle ausgelöst wird.
- spCreateNewOrder: wenn eine Bestellung über diese Prozedur angelegt wird (was ebenfalls den obigen Trigger auslöst).
- spCheckExistingOrderForCommission: wenn diese Prozedur für eine bestehende Bestellung aufgerufen wird.

Hinweis: Die Prozedur spDefineOrderStatusAndProcessIfCommissioned ist nicht für den manuellen Aufruf gedacht!

Inputparameter (allesamt aus übergebener Bestellung):

- @orderID INT: Order-ID
- @customerID INT: Customer-ID
- @ProductID BIGINT: Produkt-ID
- @OrderQuantity INT: Bestellmenge (quantity)

- @statusID INT: Bestellstatus
- @created_at DATETIME: created_at

Prozedurschritte:

- *Deklariere Hilfsvariablen*
- *Beginne eigentliche Prozedur:*
  - *Lagerbestandsprüfung:*
    - *While-Schleife, die durch alle inventories des Produkts geht (mit Abbruchbedinung)*
      - *Nimm Produktbestand, schaue ob Bestellmenge <= Produktbestand.*
        - *Falls zutreffend, merke dir das Lager, und beende die Schleife (Abbruchbedingung)*
  - *Falls oben kein ausreichender Lagerbestand gefunden wurde:*
    - *Order mit Status 3 = „abgelehnt“ anlegen*
  - *Fall, dass oben ein Lager gefunden wurde:*
    - *Lagerbestand reduzieren, Bestellung auf Status 1 = „beauftragt“ setzen, Prozedur spCreateInvoiceOnOrder starten zum Anlegen einer Rechnung im Status 1 = „unpaid“*

Das Ganze wird innerhalb eines Try-Catch-Blocks als Transaktion ausgeführt (Rollback bei Catch).

Keine Outputparameter

Ergebnis:
- **Entweder** liegt nach der Prozedur eine Bestellung im Status 3 = „abgelehnt“ vor und es wird beendet
- **Oder** es liegt eine Bestellung im Status 1 = „beauftragt“ vor, und es wird die zweite Prozedur spCreateInvoiceOnOrder der zentralen Prozesskette aufgerufen.

spCreateInvoiceOnOrder

Diese Prozedur soll eine Rechnung für eine existierende Bestellung anlegen. Sie wird innerhalb der Prozedur spDefineOrderStatusAndProcessIfCommissioned aufgerufen. Diese checkt vorher die Bestände im Rahmen der Lieferung und passt diese ggf. an. Bei erfolgreicher Bestellung wird (unter anderem) diese Prozedur hier gestartet. Sie legt zur durchgeführten Bestellung die zugehörige Rechnung an und holt dabei auch vom Kunden die Info, ob eventuell rabattiert wird.

Hinweis: Die Prozedur spCreateInvoiceOnOrder ist nicht für den manuellen Aufruf gedacht!

Inputparameter (allesamt aus übergebener Bestellung):

- @order_id_for_invoice INT: Order-ID
- @status_id INT = 1: Status-ID für die Rechnung, default 1 (optional)
- @created_at_for_invoice DATETIME = NULL: Zeitstempel für created_at-Feld der Rechnung

Prozedurschritte:

- *Deklariere Hilfsvariablen*
- *Beginne eigentliche Prozedur:*
  - *Hole vom Kunden die Rabatt-Prozentzahl*
  - *Berechne anhand der Produktdaten (die über die order-ID ermittelt werden) die finalen Preise für die Rechnung*
  - *Lege die Rechnung im entsprechenden Status (default 1 = „unpaid“) an, mit Fremdschlüsselverweis auf die eingespeiste order-ID*
  - *(Hinweis: Das due_date der Rechnung wird bei der Anlage automatisch berechnet, aus created_at und dem Rechnungs-Feld due_limit, das per default 30 Tage ist)*

Das Ganze wird innerhalb eines Try-Catch-Blocks als Transaktion ausgeführt (Rollback bei Catch).

Keine Outputparameter

Ergebnis ist immer dasselbe:

- Rechnung zur eingespeisten Bestellung wurde angelegt im Status 1 = „unpaid“

Ergebnis der zentralen Prozesskette (Output)

Das Ergebnis der zentralen Prozesskette ist **unabhängig vom gewählten Einstiegspunkt** in die zentrale Prozesskette immer dasselbe:

- **Entweder**:
  - Bestellung (order) im Status 1 („beauftragt“) liegt vor
  - Rechnung (invoice) im Status 1 (=unpaid) liegt vor
  - Lagerbestand (inventory) des bestellten Produkts wurde um Bestellmenge (quantity) reduziert.
- **Oder**:
  - Bestellung (order) im Status 3 (=abgelehnt) liegt vor
  - KEINE Rechnung (invoice), KEINE Reduktion des Produktbestands (inventory).

Weitere Prozeduren und Trigger

spChangeInvoiceStatusAndDoStuff

Diese Prozedur ist dafür gedacht, Rechnungen in einen gewünschten Zielstatus zu überführen. Dabei wird insbesondere geprüft, ob die Rechnung den Status 3=overdue haben müsste (also ob der Zahltag - das due_date - in der Vergangenheit liegt). Es wird außerdem nach Statusüberführung geprüft, ob der Kunde für zukünftige Rechnungen einen Rabatt erhält.

Inputparameter (allesamt aus übergebener Bestellung):

- @invoiceID INT: ID der eingespeisten Rechnung
- @newstatusID: ID des gewünschten Zielstatus der Rechnung

Prozedurschritte:

- *Deklariere Hilfsvariablen*
- *Prüfe die Eingangsparameter (liegt Rechnungs-ID vor? Gibt es den Status?)*
- *Hole zugehörige Bestellung (order-ID)*
- *Führe Statusfeststellung und -wechsel durch:*
  - *2 = paid:*
    - *Von diesem Status kommend wird KEIN WECHSEL erlaubt! Bezahlte Rechnungen sollen nicht zurückgesetzt werden können (da unerwünschte Zustände entstehen können, z.B, wenn die Bestellung bereits geliefert ist)*
  - *1 = unpaid oder 3=overdue*
    - *Falls gewünschter Zielstatus 1= unpaid oder 3= overdue:*
      - *Falls Zahltag vergangen (due_date < Tagesdatum getdate()):*
        - *Setze Status 3 = overdue*
        - *Setze overdue_fee (Mahngebühr) als Produkt aus Preis und overdue_fee_percentage (aus Tabelle commercial_parameters).*
      - *Falls Zahltag nicht vergangen (due_date >= Tagesdatum getdate()):*
        - *Setze Status 1 = unpaid*
        - *Setze overdue_fee (Mahngebühr) auf 0.*
- *Prüfe Rabatteigenschaft des Kunden:*
  - *Ziehe Kunden-ID heran*
  - *Ziehe required_revenue_for_discount und discount_rate aus commercial_parameters Tabelle heran*
  - *Prüfe, ob der Kunde in vergangenen bezahlten Rechnungen mehr Umsatz erzielt hat, als required_revenue_for_discount:*

- *Falls ja, setze das Feld discount am Kunden auf den Wert discount_rate aus der commercial_parameters-Tabelle.*
- *Falls nein, setze das Feld discount am Kunden auf 0.*

Keine Outputparameter

Ergebnis:
- Die eingespeiste Rechnung wurde, sofern möglich, in den gewünschten Zielstatus überführt.
- Dabei wurde, sofern der Zahltag verstrichen ist, eine Mahngebühr in die Rechnung geschrieben
- Dabei wurde überprüft, ob der Kunde bereits einen gewissen (über die Tabelle commercial_parameters gesteuerten) Umsatz mit bezahlten Rechnungen erbracht hat. Falls ja, erhält der Kunde auf zukünftige Rechnungen Rabatt.

trgDeleteInvoicesOnlyWhenNoOrder

Dieser Trigger liegt als AFTER-DELETE-Trigger an der Tabelle dbo.invoices.

Er verhindert, dass Rechnungen gelöscht werden können, für die eine Bestellung existiert. Damit verhindern wir unerwünschte Zustände, da wir Rechnungen zu Bestellungen nie entsorgen wollen.

Unterstützende Sichten für Workflows und Analysen

Generelle Sicht für Aufträge: dbo.vw_0Aufträge

Diese Sicht fasst Bestellungen und Rechnungen zu einer Sicht zusammen, da diese beiden Objekte gedanklich als "Auftrag" behandelt werden. Außerdem einige viele Prozeduren stets auf beide Objekte Auswirkungen. In der Sicht wird entsprechend ein Auftragsstatus "Status_Auftrag" als "künstlicher" Status angezeigt:

- Für den Auftragsstatus werden etliche Fälle unterschieden.
- Fälle, die mit "Prüfen!" angezeigt werden, dürften bei normaler Nutzung nicht vorkommen und sind als Sicherheitsnetz eingebaut.

Der Auftragsstatus als Anzeigespalte wird dabei wie folgt für die Anzeige generiert:

```
CASE
    -- Fall, Bestellung beauftragt, Rechnung unpaid:
    WHEN o.status_id = 1 AND i.status_id = 1            THEN 'Beauftragt, Zahlung offen'
    -- Fall, Bestellung beauftragt, Rechnung paid:
    WHEN o.status_id = 1 AND i.status_id = 2            THEN 'Inkonsistenter Zustand!'                    -- tritt bei normaler Verwendung nicht auf. Zur Sicherheit eingebaut
    -- Fall, Bestellung beauftragt, Rechnung overdue:
    WHEN o.status_id = 1 AND i.status_id = 3            THEN 'Zahlung überfällig!'
    -- Fall Bestellung geliefert, Rechnung unpaid:
    WHEN o.status_id = 2 AND i.status_id = 1            THEN 'Prüfen!! Geliefert, nicht gezahlt!'         -- tritt bei normaler Verwendung nicht auf. Zur Sicherheit eingebaut
    -- Fall Bestellung geliefert, Rechnung paid:
    WHEN o.status_id = 2 AND i.status_id = 2            THEN 'Bezahlt & geliefert!!'
    -- Fall Bestellung geliefert, Rechnung overdue:
    WHEN o.status_id = 2 AND i.status_id = 3            THEN 'Prüfen!! Geliefert, nicht gezahlt!'         -- tritt bei normaler Verwendung nicht auf. Zur Sicherheit eingebaut
    -- Fall, Bestellung geliefert, keine Rechnung:
    WHEN o.status_id = 2 AND i.status_id IS NULL        THEN'Prüfen!! Geliefert, Rechnung fehlt!'         -- tritt bei normaler Verwendung nicht auf. Zur Sicherheit eingebaut
    -- Fall Bestellung abgelehnt, keine Rechnung
    WHEN o.status_id = 3 AND i.status_id IS NULL        THEN 'Abgelehnt!'
    -- Fall, dass Bestellung abgelehnt, aber trotzdem Rechnung vorhanden:
    WHEN o.status_id = 3 AND i.status_id IS NOT NULL    THEN 'Prüfen!! Abgelehnt, aber Rechnung gestellt!'  -- tritt bei normaler Verwendung nicht auf. Zur Sicherheit eingebaut
    -- Fall, dass keine Bestellung vorhanden, aber Rechnung vorhanden:
    When o.status_ID = NULL AND i.status_id IS NOT NULL THEN 'Prüfen!! Bestellung fehlt, Rechnung da!'     -- tritt bei normaler Verwendung nicht auf. Zur Sicherheit eingebaut
END AS Status_Auftrag
```

Die Sicht sieht wie folgt aus (Auszug aus Testdaten):

	Bestell_ID	Bestellstatus	Auftragseingang	Kunden_ID	Produkt_ID	Bestellmenge	Rechnungs_ID	Umsatz	rabattierter_Umsatz	Mahngebühr	Zahlungsfrist	Zahltag	Rechnungsstatus	Status_Auftrag
168	168	id = 1 , beauftragt	2024-02-18	11	2	5	168	180.00	180.00	18.00	30	2024-03-19	id = 3 , overdue	Zahlung überfällig!
169	169	id = 1 , beauftragt	2024-06-07	7	18	5	169	630.00	630.00	63.00	30	2024-07-07	id = 3 , overdue	Zahlung überfällig!
170	170	id = 1 , beauftragt	2024-03-24	13	23	5	170	287.50	287.50	28.75	30	2024-04-23	id = 3 , overdue	Zahlung überfällig!
171	171	id = 1 , beauftragt	2024-07-01	9	26	5	171	187.50	187.50	18.75	30	2024-07-31	id = 3 , overdue	Zahlung überfällig!
172	172	id = 2 , ausgeliefert	2024-08-28	3	28	5	172	375.00	375.00	37.50	30	2024-09-27	id = 2 , paid	Bezahlt & geliefert!!
173	173	id = 2 , ausgeliefert	2024-05-06	14	1	5	173	120.00	120.00	0.00	30	2024-06-05	id = 2 , paid	Bezahlt & geliefert!!
174	174	id = 2 , ausgeliefert	2024-06-08	9	21	5	174	230.00	230.00	0.00	30	2024-07-08	id = 2 , paid	Bezahlt & geliefert!!
175	175	id = 2 , ausgeliefert	2024-08-26	6	23	5	175	287.50	287.50	0.00	30	2024-09-25	id = 2 , paid	Bezahlt & geliefert!!
176	176	id = 2 , ausgeliefert	2024-10-07	17	19	5	176	575.00	575.00	0.00	30	2024-11-06	id = 2 , paid	Bezahlt & geliefert!!
177	177	id = 1 , beauftragt	2024-01-07	16	24	5	177	57.50	57.50	0.00	30	2024-02-06	id = 1 , unpaid	Beauftragt, Zahlung offen
178	178	id = 1 , beauftragt	2024-05-14	7	15	5	178	2625.00	2625.00	0.00	30	2024-06-13	id = 1 , unpaid	Beauftragt, Zahlung offen
179	179	id = 1 , beauftragt	2024-02-18	17	11	5	179	112.50	112.50	0.00	30	2024-03-19	id = 1 , unpaid	Beauftragt, Zahlung offen
180	180	id = 1 , beauftragt	2024-01-16	20	9	5	180	360.00	360.00	0.00	30	2024-02-15	id = 1 , unpaid	Beauftragt, Zahlung offen
181	181	id = 1 , beauftragt	2024-04-19	4	9	5	198	360.00	342.00	0.00	30	2024-11-08	id = 1 , unpaid	Beauftragt, Zahlung offen
182	182	id = 3 , abgelehnt	2024-02-01	11	4	5	NULL	NULL	NULL	NULL	NULL	NULL	NULL	Abgelehnt!
183	183	id = 3 , abgelehnt	2024-05-15	10	6	5	NULL	NULL	NULL	NULL	NULL	NULL	NULL	Abgelehnt!

Sie ist speziell nützlich, um nach Ausführung von **WORKFLOW (A Bestellungen anlegen, Lagerbestandsprüfung, Rechnungserzeugung)** die Ergebnisse der erzeugten Aufträge einzusehen.

Sicht Aufträge überschrittenem Zahltag: dbo.vw_1AufträgeMitÜberfälligerZahlung

Diese Sicht zeigt alle Aufträge mit überfälliger Zahlung (genauer: **mit überschrittenem Zahltag**), die noch nicht bezahlt sind.

Beachte:

- Es wird hier nach dem Zahltag eingeschränkt, lediglich bezahlte Rechnungen werden ausgeschlossen.
- Es können also auch Aufträge auftauchen, bei denen der Zahltag in der Vergangenheit liegt, und bei denen die Rechnung noch nicht auf "overdue" gesetzt worden ist!
- Die View hat also zwei Zwecke:
- Unterstützung bei der Identifikation von Aufträgen, die bereits den entsprechenden "überfällig" Status haben (und z.B. auf abgeschlossen, also bezahlt gesetzt werden sollen)
- Unterstützung bei der Identifikation von Aufträgen, deren Zahltag verstrichen sind, die aber noch nicht den "überfällig" Status haben.

Die Ergebnismenge sieht wie folgt aus (Auszug aus Testdaten):

	Bestell_ID	Bestellstatus	Auftragseingang	Kunden_ID	Produkt_ID	Bestellmenge	Umsatz	Mahngebühr	Zahltag	Rechnungsstatus	Rechnungs_ID	Status_Auftrag
3	169	id = 1 , beauftragt	2024-06-07	7	18	5	630.00	63.00	2024-07-07	id = 3 , overdue	169	Zahlung überfällig!
4	170	id = 1 , beauftragt	2024-03-24	13	23	5	287.50	28.75	2024-04-23	id = 3 , overdue	170	Zahlung überfällig!
5	171	id = 1 , beauftragt	2024-07-01	9	26	5	187.50	18.75	2024-07-31	id = 3 , overdue	171	Zahlung überfällig!
6	162	id = 1 , beauftragt	2024-05-12	5	13	5	945.00	94.50	2024-06-11	id = 3 , overdue	162	Zahlung überfällig!
7	163	id = 1 , beauftragt	2024-06-20	9	28	5	375.00	37.50	2024-07-20	id = 3 , overdue	163	Zahlung überfällig!
8	164	id = 1 , beauftragt	2024-04-02	14	7	5	90.00	9.00	2024-05-02	id = 3 , overdue	164	Zahlung überfällig!
9	165	id = 1 , beauftragt	2024-06-29	4	29	5	225.00	22.50	2024-07-29	id = 3 , overdue	165	Zahlung überfällig!
10	166	id = 1 , beauftragt	2024-03-10	15	29	5	225.00	0.00	2024-12-11	id = 1 , unpaid	166	Beauftragt, Zahlung offen
11	177	id = 1 , beauftragt	2024-01-07	16	24	5	57.50	0.00	2024-02-06	id = 1 , unpaid	177	Beauftragt, Zahlung offen
12	178	id = 1 , beauftragt	2024-05-14	7	15	5	2625.00	0.00	2024-06-13	id = 1 , unpaid	178	Beauftragt, Zahlung offen
13	179	id = 1 , beauftragt	2024-02-18	17	11	5	112.50	0.00	2024-03-19	id = 1 , unpaid	179	Beauftragt, Zahlung offen
14	180	id = 1 , beauftragt	2024-01-16	20	9	5	360.00	0.00	2024-02-15	id = 1 , unpaid	180	Beauftragt, Zahlung offen

Die Sicht ist vor allem für **WORKFLOW C (Rechnungen prüfen)** nützlich, speziell um overdue-Rechnungen abzuarbeiten.

Beachte: Für die Abarbeitung von unpaid-Rechnungen, die noch nicht säumig sind, ist die Sicht nicht geeignet (solche werden hier nicht angezeigt). Für diesen Zweck kann aber jederzeit auf die Sicht dbo.vw_0Aufträge zurückgegriffen werden.

Sicht für kritische Lagerbestände: dbo.vw_2KritischeLagerbestände

Diese Sicht zeigt alle Lagerbestände (inventories), die aktuell unter dem Mindestbestand liegen.

Beachte:

- Es ist hier keine Info enthalten, wie Aufträge mit abgelehnter Bestellung je angezeigtem Lager vorliegen.
- Das wäre eventuell auch irreführend, denn:

- Es kann Bestände mit unterschrittenem Mindestbestand geben, für die keine abgelehnten Bestellungen vorliegen! Beispiel: Mindestbestand 20, Restbestand 5 (also Unterschreitung), und es wurden nur Bestellmengen 1 bestellt (und nicht abgelehnt).
- Es kann abgelehnte Bestellungen geben, bei denen der Lager nicht unterschritten ist! Beispiel: Bestellung mit Menge 10000 wird abgelehnt, da „nur“ 100 im Bestand (und Mindestbestand 10 ist nicht unterschritten).
- Für abgelehnte Bestellungen gibt es daher eine separate Sicht (nächstes Teilkapitel)
- Diese Sicht hier ist aber trotzdem sinnvoll, da leere Bestände zwangsweise gefüllt werden müssen für Neubestellungen! Über einen Mindestbestand bekommen wir frühzeitig mit, dass sich Bestände leeren, und können abgelehnte Bestellungen verhindern!

Hier eine beispielhafte Ansicht mit Testdaten:

	Inventory_ID	Produkt_ID	Lagerort	Bestand	Mindestbestand
1	3	2	Lager-ID = 1; Hauptlager	5	15
2	4	2	Lager-ID = 2; Außenlager	4	5
3	5	3	Lager-ID = 1; Hauptlager	2	20
4	6	3	Lager-ID = 2; Außenlager	3	5
5	7	4	Lager-ID = 1; Hauptlager	3	25
6	8	4	Lager-ID = 2; Außenlager	2	10
7	9	5	Lager-ID = 1; Hauptlager	2	12
8	10	5	Lager-ID = 2; Außenlager	3	10

Sicht für abgelehnte Bestellungen: dbo.vw_3AbgelehnteBestellungenMitLieferbarkeitsangabe

Diese Sicht zeigt abgelehnte Bestellungen an, ihren aktuellen Bestand im jeweiligen Lager, und ob sie mittlerweile aus dem entsprechenden Lager lieferbar wären.

Hier ein Auszug aus den Testdaten:

	Abgelehnte_Bestellung	Bestellstatus	Bestellte_Produkt_ID	Bestellmenge	Inventory_ID	Lagerort	Bestand	Mindestbestand	Lieferbarkeitsangabe
9	186	abgelehnt	11	5	21	Lager-ID = 1; Hauptlager	5	14	lieferbar
10	186	abgelehnt	11	5	22	Lager-ID = 2; Außenlager	2	5	..
11	187	abgelehnt	25	5	49	Lager-ID = 1; Hauptlager	5	13	lieferbar
12	187	abgelehnt	25	5	50	Lager-ID = 2; Außenlager	4	14	..
13	188	abgelehnt	6	5	11	Lager-ID = 1; Hauptlager	4	8	..
14	188	abgelehnt	6	5	12	Lager-ID = 2; Außenlager	5	5	lieferbar
15	189	abgelehnt	28	5	55	Lager-ID = 1; Hauptlager	5	9	lieferbar
16	189	abgelehnt	28	5	56	Lager-ID = 2; Außenlager	4	15	..
17	190	abgelehnt	22	5	43	Lager-ID = 1; Hauptlager	4	17	..
18	190	abgelehnt	22	5	44	Lager-ID = 2; Außenlager	4	5	..
19	191	abgelehnt	13	5	25	Lager-ID = 1; Hauptlager	3	8	..

Diese Sicht ist also insbesondere für **WORKFLOW B (Abgelehnte Bestellungen erneut prüfen lassen)** sehr nützlich.

Beachte: Da der Bestand in jedem Lager für ein bestelltes Produkt angezeigt wird, gibt es ggf. mehrfaches Auftauchen desselben Auftrags. Siehe zum Beispiel Spalte „Abgelehnte_Bestellung“, es taucht die ID 186 zweimal auf (da wir zwei Lager haben). Man kann hier also erkennen, dass die Bestellung aus dem Hauptlager lieferbar ist, nicht aber aus dem Außenlager! Wird die Bestellung geliefert, verschwinden hier beide Zeilen!

Sicht als Basis für Umsatzanalysen: dbo.vw_4BasisFürUmsatzanalysen

Diese Sicht dient als Vorlage für Umsatzanalysen.

Beachte: Umsätze sind **eingegangene Zahlungen**! Wir betrachten hier also **nur bezahlte Aufträge!**

Da sie berechnete Anzeigespalten enthält, wird hier die Spaltenstruktur abgebildet:

```sql
SELECT  i.id AS Rechnungs_ID,
        i.total_price AS Umsatz,
        p.category_id AS Warengruppe,
        o.customer_id AS Kunden_ID,
        o.quantity AS Bestellmenge,
        p.id AS Produkt_ID,
        p.purchase_price AS Einkaufspreis,
        p.sale_price AS Verkaufspreis,
        p.sale_price / 1.19 AS Verkaufspreis_ohne_MWSt,
        p.sale_price / 1.19 * o.quantity - p.purchase_price * o.quantity AS Rohgewinn,
        p.supplier_id AS Lieferanten_ID
FROM dbo.invoices i
JOIN dbo.orders o ON i.order_id = o.id
JOIN dbo.products p ON o.product_id = p.id
WHERE i.status_id = 2
```

Beachte zur Erklärung für die Spalten „Verkaufspreis_ohne_MWSt“ und „Rohgewinn“:

- Gemäß kommerziellem Standard zahlen Händler keine MWSt an Lieferanten, bzw. diese wird umgelegt. Da die MWSt jedoch im Verkaufspreis enthalten sind, muss diese aus dem Verkaufspreis herausdividiert werden. Das erklärt die Berechnung.

Hier ein Auszug aus den Testdaten:

	Rechnungs_ID	Umsatz	Warengruppe	Kunden_ID	Bestellmenge	Produkt_ID	Einkaufspreis	Verkaufspreis	Verkaufspreis_ohne_MWSt	Rohgewinn	Lieferanten_ID
1	1	575.00	3	16	5	19	50.00	115.00	96.638655	233.193275	1
2	2	112.50	1	17	5	11	9.00	22.50	18.907563	49.537815	3
3	3	115.00	3	11	5	20	10.00	23.00	19.327731	46.638655	1
4	4	187.50	4	4	5	26	15.00	37.50	31.512605	82.563025	1
5	5	945.00	2	7	5	13	90.00	189.00	158.823529	344.117645	1
6	6	90.00	1	15	5	7	6.00	18.00	15.126050	45.630250	4
7	7	180.00	1	15	5	2	15.00	36.00	30.252100	76.260500	1
8	8	312.50	4	7	5	25	25.00	62.50	52.521008	137.605040	1
9	9	375.00	4	3	5	28	30.00	75.00	63.025210	165.126050	1
10	10	275.00	4	13	5	30	22.00	55.00	46.218487	121.092435	1
11	11	115.00	3	1	5	20	10.00	23.00	19.327731	46.638655	1
12	12	172.50	3	11	5	22	15.00	34.50	28.991596	69.957980	1
13	13	180.00	2	2	5	16	15.00	36.00	30.252100	76.260500	2

Basierend auf dieser Sicht können weitere nützliche Abfragen für kommerzielle Analysen angewendet werden, wie zum Beispiel die folgenden:

Kunden nach Rohgewinn

```sql
------------------Kunde nach Rohgewinn ------------
-- Wir geben uns nun eine Übersicht aus, die für jeden Kunden alle erzielten Umsätze (bez. Rechnungen) summiert, und den Rohgewinn ausgibt.
-- Der Kunde mit dem meisten Rohgewinn steht ganz oben
    SELECT    Kunden_ID,
              COUNT(*) AS 'Anzahl bezahlte Rechnungen',
              SUM(Bestellmenge) AS 'Anzahl gekaufte Artikel',
              SUM(Umsatz) AS 'Gesammelter Umsatz',
              CAST(SUM(Rohgewinn) AS decimal(10,2)) AS 'erzielter Rohgewinn'
    FROM dbo.vw_4BasisFürUmsatzanalysen
    GROUP BY Kunden_ID
    ORDER BY CAST(SUM(Rohgewinn) AS decimal(10,2)) DESC
```

Kunden nach Handelsspanne

```sql
    ------------------Kunde nach Handelsspanne ------------
 -- Wir geben uns nun eine Übersicht aus, die für jeden Kunden alle erzielten Umsätze (bez. Rechnungen) summiert, zusätzlich die durchschnittl.
   Handelsspanne ausgibt.
 -- Der Kunde mit der größten Handelsspanne steht ganz oben
     SELECT    Kunden_ID,
               COUNT(*) AS 'Anzahl bezahlte Rechnungen',
               SUM(Bestellmenge) AS 'Anzahl gekaufte Artikel',
               SUM(Umsatz) AS 'Gesammelter Umsatz',
               CAST(SUM(Rohgewinn) AS decimal(10,2)) AS 'erzielter Rohgewinn',
               CAST(CAST(SUM(Rohgewinn) / SUM(Umsatz) * 100 AS decimal(10,2)) AS nvarchar(80)) + ' %' AS 'Durchschnittl. Handelsspanne'

     FROM dbo.vw_4BasisFürUmsatzanalysen
     GROUP BY Kunden_ID
     ORDER BY CAST(CAST(SUM(Rohgewinn) / SUM(Umsatz) * 100 AS decimal(10,2)) AS nvarchar(80)) + ' %' DESC
```

Produkte und ihre Verkaufsdaten

```
------------------Produkte und ihre Verkaufsdaten-----------
-- Hier nun eine Übersicht für Produkte, gewinnstärkste Produkte oben
       SELECT    Produkt_ID,
                 SUM(Bestellmenge) AS 'Verkaufte Stückzahl',
                 SUM(Umsatz) AS 'Gesammelter Umsatz',
                 CAST(SUM(Rohgewinn) AS decimal(10,2)) AS 'erzielter Rohgewinn'
       FROM dbo.vw_4BasisFürUmsatzanalysen
       GROUP BY Produkt_ID
       ORDER BY CAST(SUM(Rohgewinn) AS decimal(10,2)) DESC
```

Warengruppen und ihre Verkaufsdaten

```
 ------------------- Warengruppen und ihre Verkaufsdaten------------
  -- Dasselbe nochmal für Warengruppen:
        SELECT     Warengruppe,
                   SUM(Bestellmenge) AS 'Anzahl verkaufte Produkte',
                   SUM(Umsatz) AS 'Gesammelter Umsatz',
                   CAST(SUM(Rohgewinn) AS decimal(10,2)) AS 'erzielter Rohgewinn'
         FROM dbo.vw_4BasisFürUmsatzanalysen
         GROUP BY Warengruppe
         ORDER BY CAST(SUM(Rohgewinn) AS decimal(10,2)) DESC
```

Weitere nützliche Abfragen und Prozeduraufrufe

Schnelles Lagernachfüllen mit Zufallsdaten

```
--------------------Schnelles Lagernachfüllen mit Zufallswerten

       -- Beachte: es gibt 60 inventory-IDs (für jedes der 30 Produkte 2 Lager)!
       -- Hilfsvariablen:
       DECLARE @Counter1 INT = 0;
       DECLARE @renewStock INT;
       -- Schleife zum Befüllen aller Bestände:
       WHILE @Counter1 < 60 + 1 -- alle 60 inventory-IDs durchgehen und mit
       BEGIN
              --SET @renewStock = 8
              SET @renewstock = FLOOR(10 + (RAND() * (30 - 10 + 1)))      ---- Füllt Bestände mit Zufallswerten zwischen 10 und 30
              UPDATE  inventory
              SET     stock = @renewStock
              WHERE id = @Counter1

              SET @Counter1 = @Counter1 + 1;
       END;

       -- Stolz aufgefüllte Bestände betrachten
       select * from inventory
```

Gezielt überfällige Rechnungen aufspüren und overdue setzen

```
------------------------- Rechnungen mit Zahltag < Tagesdatum auf "overdue" setzen.
   ------ niedrigste ID aus Unpaid-Rechnungen (status_id 1) ermitteln:
       DECLARE @lowestOverdueID INT;
       SET @lowestOverdueID = (        SELECT TOP 1 id
                                  FROM invoices
                                  WHERE status_id = 1 AND due_date < getdate()
                                  ORDER BY id ASC)
       PRINT @lowestOverdueID
   ---- Für diese ID Unpaid-Rechnung auf overdue setzen!
           EXEC spChangeInvoiceStatusAndDoStuff
               @invoiceID      = @lowestOverdueID,
               @newStatusID    = 3
   ---- Stolz Ergebnisse anschauen
       select * from dbo.vw_0Aufträge WHERE Rechnungs_ID = @lowestUnpaidID
```

Abgelehnte Bestellungen erneut in Prozesskette geben

```
------------------------------ Bestellungen nachträglich durchschieben
------ niedrigste ID aus abgelehnt-Bestellungen (status_id 3) ermitteln:
           DECLARE @lowestDeclinedID INT;
           SET @lowestDeclinedID = (       SELECT TOP 1 id
                                      FROM orders
                                      WHERE status_id = 3
                                      ORDER BY id ASC)
           PRINT @lowestDeclinedID
         ---- Für diese ID die abgelehnt-Bestellung auf paid (@newstatus_id 2 als input) setzen, ODER auf overdue
           (@newstatus_id = 3 als input)!
         EXEC spCheckExistingOrderForCommission @orderID_checkforcomm = @lowestDeclinedID
```

Entgangene Umsätze aufgrund von Rabatten ausgeben

```
------------------------- Rabattkunden finden und "entgangene Umsätze" aufgrund der Rabatte anzeigen
      --Rabattkunden
      SELECT  COUNT(*)
      FROM customers c
      WHERE discount > 0
      -- Rabattfreie Kunden
      SELECT COUNT(*)
      FROM customers c
      WHERE discount = 0
      -- entgangene Umsätze
      SELECT  SUM(total_price) AS gemachter_Umsatz,
              SUM(total_discount) AS entgangener_Umsatz
      FROM customers c
      JOIN orders o ON o.customer_id = c.id
      JOIN invoices i ON i.order_id = o.id
```

Zusammenfassung

Zur Zusammenfassung rekapitulieren wir lediglich nochmals die Anforderungen und stellen sie den möglich gemachten Workflows gegenüber:

Anforderungen:

- **Grundsätzliche Datenverwaltung**:
  - Personengruppen (Mitarbeiter, Kunde, Lieferant), Produktdaten (Produkt, Lagerbestand, Lagerort), Auftragsdaten (Bestellung, Rechnung) sollen geführt werden können.
- **Auftragsverwaltung**:
  - Eingegangene Kundenbestellungen von Produkten sollen verarbeitet werden können.

    - Für eingegangene Kundenbestellungen sollen Rechnungen erstellt werden.
    - Für eingegangene Kundenbestellungen sollen verfügbare Produktbestände beachtet werden und ein Lieferbarkeits- und Lieferstatus verfolgt werden können.
    - Kundenbestellungen und Rechnungen werden Aufträgen zusammengefasst.
    - Für Aufträge soll abhängig von der Lieferbarkeit der Bestellung (bestandsabhängig) und dem Rechnungsstatus (offen, bezahlt, überfällig) eine Statuslogik eingerichtet werden.
- **Kommerzielle Anforderungen**:
    - Verschiedene Umsatzanalysen für sollen durchgeführt werden können.
    - Für Rechnungen sollen Mahngebühren erhoben werden können.
    - Stammkunden sollen Rabatte erhalten können.

Obige Anforderungen wurden durch ein Zusammenspiel mehrerer Prozeduren möglich gemacht. Es wurden folgende Workflows ermöglicht:

1. **WORKFLOW A: Bestellungen anlegen, Lagerbestandsprüfung, Rechnungserzeugung**.
    a. Der Nutzer muss für den Workflow **lediglich** die Prozedur spCreateNewOrder nutzen, um (eher zu Testzwecken) eine zufallsgenerierte Bestellung zu erzeugen, oder alternativ über INSERT ORDER tatsächliche Werte eingegeben. Weitere Arbeitsschritte muss er nicht vornehmen!
    b. Die zentrale Prozesskette prüft die Lagerbestände
        i. Ist genug in den Lagern, wird ein Auftrag als gedankliches Paar erzeugt, bestehend aus den Objekten order (Status „beauftragt“) und invoice (Status „unpaid“)
        ii. Ist nicht genug in den Lagern, wird lediglich das Objekt order im Status „abgelehnt“ zurückgegeben.
2. **WORKFLOW B: Abgelehnte Bestellungen erneut prüfen lassen**.
    a. Der Nutzer kann sehr bequem einst abgelehnte Bestellungen einfach erneut in den Prozess geben. Dafür muss er **lediglich** die Prozedur spCheckExistingOrderForCommission aufrufen und die Order-ID übergeben. Weitere Arbeitsschritte muss er nicht vornehmen!
    b. Beachte: Wenn die Bestellung nun zum Auftrag wird (und nicht erneut abgelehnt wird), hat die zugehörige Bestellung ein älteres Erzeugungsdatum als die nun später entstandene Rechnung. Das ist wichtig, damit die Zahlungsfrist in der Rechnung korrekt ist! (Würde das Erzeugungsdatum der Bestellung in die später erzeugte Rechnung geschrieben, wäre diese eventuell fälschlicherweise sofort overdue!)
3. **WORKFLOW C: Rechnungen prüfen**.
    a. Der Nutzer kann Rechnungen zu vorliegenden Aufträgen prüfen. Hier gibt es zwei Szenarien:
        i. Der Kunde möchte eine Rechnung („unpaid“ oder „overdue“) auf bezahlt („paid“) setzen. Dafür muss er **lediglich** Rechnung an die Prozedur spChangeIncoiceStatusAndDoStuff übergeben und den gewünschten Zielstatus 2 (=“paid“) mitgeben.
        ii. Der Kunde möchte eine Rechnung auf Fristsäumigkeit (overdue) prüfen. Dazu muss er **lediglich** die Rechnung an die Prozedur spChangeIncoiceStatusAndDoStuff übergeben und den gewünschten Status 3 (=“overdue“) mitgeben. Die Prozedur prüft dann, ob die Rechnung tatsächlich überfällig ist, und setzt den Status entsprechend. Dasselbe gilt, wenn er den Zielstatus 1 (=“unpaid“) mitgibt.

Entsprechend haben wir alle Anforderungen in unseren Workflows abgedeckt. Das Datenmodell deckt ebenfalls die Anforderung an die Datenverwaltung ab.

Weiterentwicklungspotenziale

Folgende Weiterentwicklungen würden einen Mehrwert bieten:

- Rückabwicklung offener Aufträge (Stornieren)
- Abbildung von Nachbestellungen bei Lieferanten
- Automatische Nachbestellung bei Lieferanten, wenn Kundenbestellungen aufgrund geringer Produktbestände abgelehnt werden
- Automatisches Setzen der overdue-Eigenschaft überfälliger Rechnungen

    - Dies wäre mit wenig Mehraufwand möglich. Eine Prozedur zur Statusprüfung gibt es bereits. Diese könnte über einen Serverdienst in MS SQL z.B. nächtlich aufgerufen werden, alle unbezahlten Rechnungen prüfen, und die Eigenschaft setzen. Dazu müsste die Prozedur lediglich mit der entsprechenden Rechnungs-ID gefüttert werden.
- Bestellung mehrerer Produkte in einer einzigen Bestellung (Bestellpositionen)