# Datenbank «Anlagendashboard»

## Leistungsbeurteilung LB2: Datenbank erstellen

| | |
|---|---|
| Modul | 164 Datenbanken erstellen und Daten einfügen |
| Eingereicht von | Simon Leutert, Jonas Vetsch |
| Projektthema | Datenbank erstellen, Daten einfügen und exportieren |
| Eingereicht bei | Michael Abplanalp |
| Datum | 5. Januar 2026 |

# Inhaltsverzeichnis

# 1 Projektbeschreibung

## 1.1 Zweck

Das Projektteam bestehend aus Simon Leutert und Jonas Vetsch wurde von einem Fintech Startup der Universität St.Gallen (HSG) **fiktiv** beauftragt, eine Datenbank zu erstellen. Das Startup «TrackMyAssets» möchte eine mobile App auf den Schweizer Markt bringen, die den Nutzer ermöglicht, ihre Anlagen, die sie bei unterschiedlichen Brokern halten, auf einen Blick zu analysieren.

Die Herausforderung heute ist, dass private Anleger oft bei mehreren Brokern (z.B. PostFinance, BEKB etc.) gleichzeitig ein Portfolio besitzen und deshalb keine Möglichkeit haben, auf einen Blick die gesamte Performance ihrer Anlagen zu sehen. Stand heute müssen Nutzer sich nacheinander bei ihren Brokern einloggen und Werte z.B. händisch in einer Exceltabelle zusammentragen, um die Performance zu sehen.

Für dieses Problem schafft das Startup «TrackMyAssets» mit ihrer mobilen App eine Lösung: Nutzer tragen ihre Anlagen, die sie bei unterschiedlichen Brokern haben, in der App ein und haben fortan immer sofort eine Performanceanalyse ihrer gesamten Anlagen griffbereit, dass ermöglicht den Nutzern fundierter Anlageentscheidungen zu treffen und den Überblick über ihre Anlagen zu behalten. So wird auch die Vermögensdeklaration in der Steuererklärung zum Kinderspiel,

Ein Nutzer trägt in der App ein, bei welchem Broker sie*er welche Anlagen hat, inkl. Kauf- (und ggf. Verkaufs-) Datum und die Performance wird aufgrund von hinterlegten Kursdaten berechnet und angezeigt.

In einer ersten Version soll unsere Datenbank mit fiktiven Kursdaten arbeiten. Später wird «TrackMyAssets» das Entwicklerteam Leutert-Vetsch ggf. in einem weiteren Auftrag mit der Integration von Echtzeit-Marktdaten beauftragen. Dies ist stand Heute aber noch nicht Teil des Projektumfangs.

Die Datenbank für «TrackMyAssets» soll im Rahmen dieses Projekts angelegt und mit ersten Beispieldaten versehen werden. Damit kann das Team der mobilen App, diese dann entwickeln und testen, ehe dann ggf. später echte Daten in der Datenbank eingetragen würden.

Die Datenbank zielt auf ein späteres Backend / Frontend Projekt des Startups «TrackMyAssets» hin, bei dem das Anlagendashboard im Zentrum steht: Der Nutzer soll seine Anlagen (Aktien, Rohstoffe, etc.) aufgelistet sehen (mit Titel, Anzahl, Einkaufswert, Einkaufsdatum, Aktueller Wert, Wertentwicklung).

## 1.2 Abgrenzungen

Im Rahmen dies hier vorliegenden Projekts gibt es noch keine Anbindung an Bankkonten. Der Nutzer sieht also nur die Informationen zu seinen Anlagen, nicht aber zu seinen Konti (Gebühren, Kontostand, Verfügbarer Barbetrag im Konto, Dividendenausschüttung etc.), somit können Gebühren und Dividenden nicht automatisch mit dem Kontostand verrechnet werden. Eine Gesamtübersicht über das Vermögen des Nutzers ist demnach nicht möglich.

Da jede Transaktion manuell erfasst werden muss, ist das Anlagendashboard für Nutzer gemacht, die eine Buy-And-Hold-Strategie haben: Es gibt pro Jahr nur wenige Käufe (ca. 1 bis 10 im Normalfall) und kaum Verkäufe. Entsprechend hält sich die manuelle Arbeit für den Nutzer sehr in Grenzen.

## 1.3 Ziele

Die Datenbank muss alle Daten abbilden, die für ein persönliches Anlagendashboard nötig sind. Dazu gehört eine Tabelle mit den Nutzern, eine Tabelle mit den einzelnen Anlagen (z.B. der ETF VWRL.SIX), der zugehörigen Kursdaten sowie aller Transaktionen der Nutzer (speichert Kauf- und Verkauf).

Sie muss ermöglichen, dass ein Nutzer seine gehaltenen Anlagen über verschiedene Broker hinweg erfassen, historisch nachvollziehen und deren aktuellen Wert berechnen kann. Zudem muss sie Anlagetypen und Masseinheiten klar trennen, damit unterschiedliche Asset-Klassen (Aktien, ETFs, Krypto, Rohstoffe) korrekt verarbeitet werden können. Die Datenbank bildet damit die Grundlage für Portfolioübersichten, zeitliche Analysen und Auswertungen nach Broker oder Anlagetyp.

Als Bonus liegt der Datenbank ein Python Skript bei, welches die automatisierte Abfrage aktueller sowie historischer Kursdaten via API ermöglicht. Somit könnten mit wenig Aufwand und direkter Anbindung die Kursdaten stets aktuell gehalten werden. Das Skript ist als Technologiedemonstrator zu verstehen: Die Auswahl der verfügbaren Kursdaten ist im Moment noch begrenzt, zu dem gäbe es Probleme mit API-Limitationen, die man für ein kommerzielles Produkt noch lösen müsste.

# 2 Datenbankmanagementsystem

Für das hier vorgelegte Projekt verwenden wir MySQL 8.0.44 als Datenbankserver.

Für den Zugriff auf die Datenbank arbeiten wir mit

- DataGrip 2025.2.4
    - Build #DB-252.26830.46, built on September 28, 2025
    - Runtime version: 21.0.8+1-b1038.72 amd64 (JCEF 122.1.9)
    - VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.
    - Wir verwenden eine Lizenz für die nicht-kommerzielle Nutzung.

Als Endgeräte verwenden wir je ein

- Lenovo ThinkPad E14 Gen 6
- mit Windows 11 Pro, version 25H2
- und der BIOS-Version R2JET44W(1.21).

# 3 Datenmodelle

## 3.1 Konzeptionelles Datenmodell

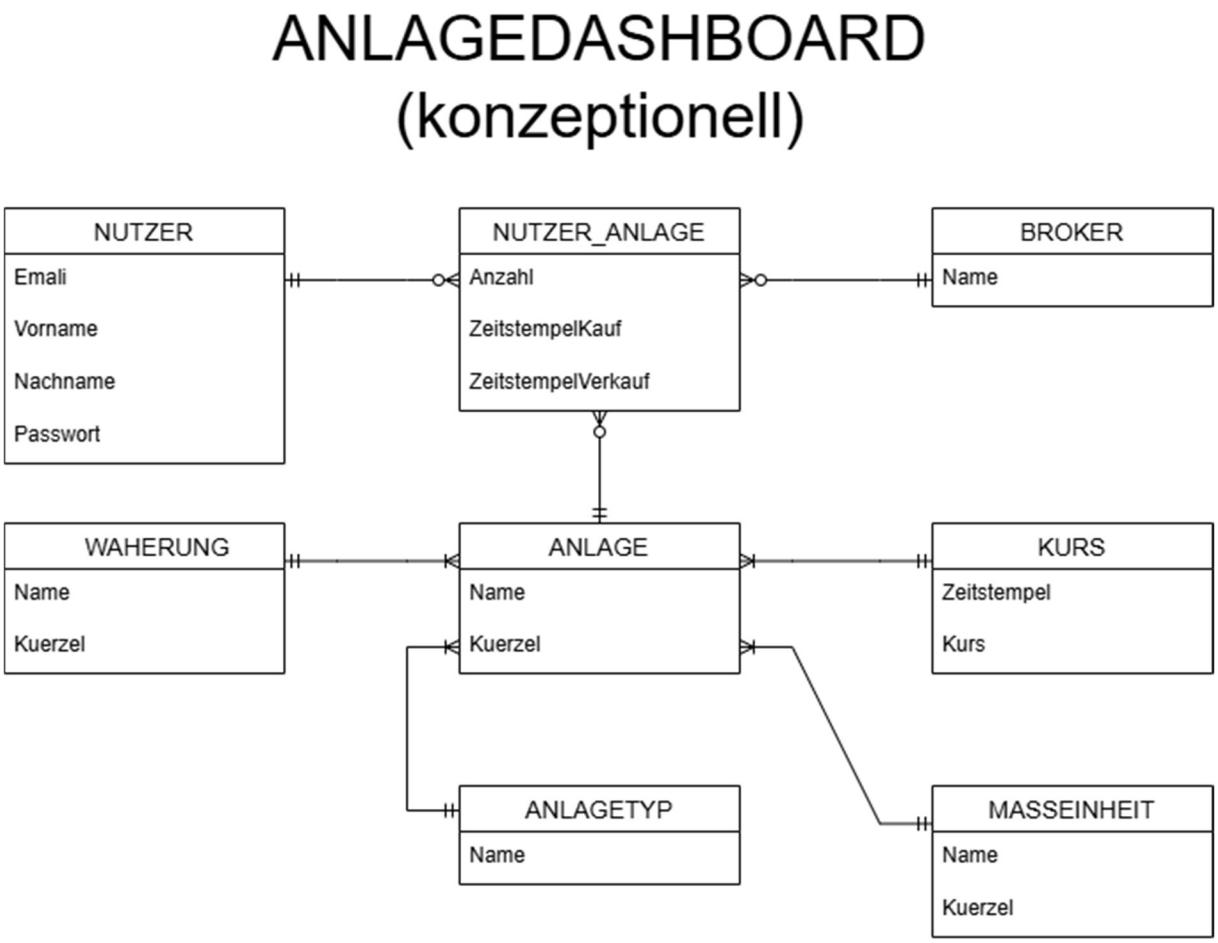


*Abbildung 1 Konzeptionelles Datenmodell des Anlagendashboards*

## 3.2 Beschreibung der Tabellen

| Tabelle | Beschreibung |
|---|---|
| **NUTZER** | Repräsentiert eine reale Person im System. Das Nutzerprofil dient ausschliesslich der Identifikation, Authentifizierung und Trennung von Portfolios. Enthält keine Anlagedaten, sondern ist der Einstiegspunkt für alle nutzerbezogenen Auswertungen. |
| **BROKER** | Beschreibt, über welche Bank oder Plattform eine Anlage gehalten wird. Ermöglicht die Abbildung mehrerer Depots pro Nutzer sowie Auswertungen nach Broker (Bestand, Performance, später Gebühren oder Steuerreports). Zukünftig könnte die Datenbank in dieser Tabelle um weitere Attribute, wie Adresse, Kundenkontakt etc. ergänzt werden. |
| **NUTZER_ANLAGE** | Zentrale Tabelle des Modells. Sie beschreibt eine konkrete Position eines Nutzers in einer Anlage bei einem bestimmten Broker. Enthält Menge und Zeitbezug und ist die Grundlage für Depotbestand, Kauf-/Verkaufshistorie, realisierte und unrealisierte Gewinne. Sie stellt demnach das Bindeglied in der Dreiecksbeziehung zwischen Broker, Nutzer und Anlage dar. |
| **ANLAGE** | Beschreibt das Finanzinstrument selbst, unabhängig vom Nutzer. Dient als gemeinsame Referenz für Kurse, Typ, Währung und Masseinheit. Macht es möglich, dieselbe Anlage mehrfach, von verschiedenen Nutzern und Brokern, konsistent zu verwenden. Vom Nutzer wird die Anlage über ihren Namen und ihr eindeutiges Kürzel identifiziert. |
| **KURS** | Speichert zeitabhängige Preisinformationen einer Anlage. Ermöglicht historische Auswertungen, Performance-Berechnungen, Charts und die Bewertung von Nutzerpositionen zu beliebigen Zeitpunkten. Die Tabelle speichert den Kurswert einer Anlage zu einem bestimmten Zeitpunkt. |
| **WÄHRUNG** | Definiert, in welcher Währung eine Anlage gehandelt und bewertet wird. Grundlage für korrekte Anzeige, Vergleichbarkeit und spätere Umrechnung in eine Referenzwährung auf Portfolio-Ebene. |
| **ANLAGETYP** | Kategorisiert Anlagen nach ihrer Art (z. B. Aktie, ETF, Krypto). Wird für Filter, Gruppierungen und typabhängige Logik im Dashboard verwendet, ohne die Anlage selbst verändern zu müssen. |
| **MASSEINHEITEN** | Legt fest, wie die Menge einer Anlage interpretiert wird (Stück, Anteil, Coin, Gewicht). Verhindert falsche Annahmen bei Berechnungen und sorgt für konsistente Darstellung im Portfolio. |

## 3.3 Logisches Datenmodell

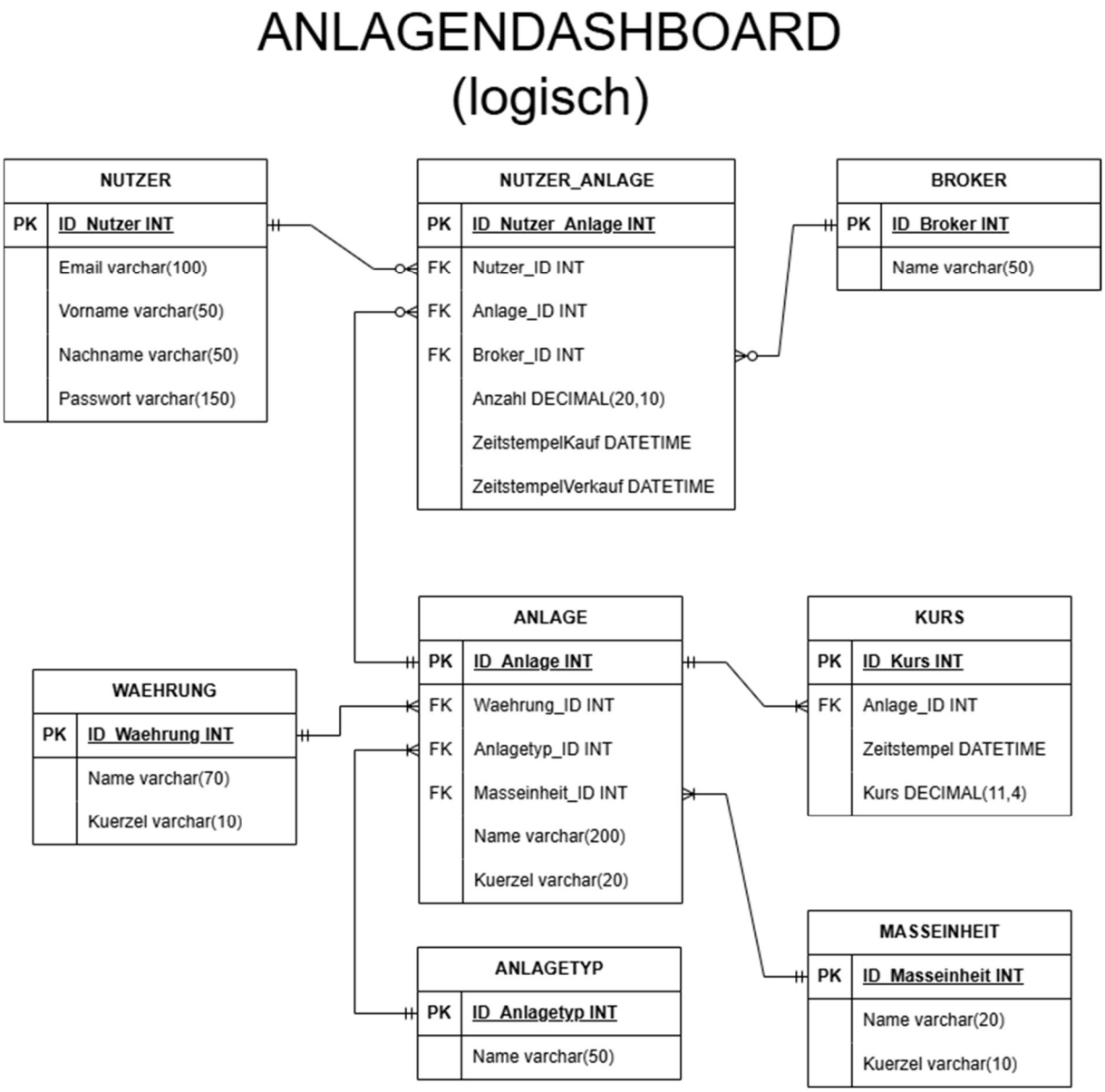


*Abbildung 2 Logisches Datenmodell des Anlagendashboards*

## 3.4 Beschreibung des logischen Datenmodells

| Aspekt | Beschreibung |
|---|---|
| **Primärschlüssel (PK)** | Jede Tabelle besitzt einen Primärschlüssel vom Typ INT (ID_…). Dies stellt eine eindeutige Identifikation sicher, entkoppelt fachliche Attribute von der Identität und vereinfacht Joins sowie spätere Änderungen an Stammdaten. |
| **Eigenständiger PK in NUTZER_ANLAGE** | Die Tabelle **NUTZER_ANLAGE** verfügt über einen eigenen Primärschlüssel (ID_Nutzer_Anlage). Dadurch ist jede Position eindeutig referenzierbar und nicht nur durch eine Kombination von Fremdschlüsseln identifiziert. |
| **Fremdschlüssel (FK)** | Beziehungen zwischen Tabellen werden explizit über Fremdschlüsselspalten abgebildet (z. B. Nutzer_ID, Anlage_ID, Broker_ID). Dadurch wird die referenzielle Integrität technisch erzwungen und die Fachlogik eindeutig umgesetzt. |
| **Datentypen für Mengen** | Mengenangaben (Anzahl) werden als DECIMAL(20,10) gespeichert. Dies ermöglicht die präzise Abbildung von Bruchteilen sowie sehr grossen Werten und ist damit für unterschiedliche Anlagearten geeignet. Für Finanzdaten empfiehlt sich die Verwendung von Zahlenwerten im DECIMAL Format. |
| **Datentypen für Kurse** | Kurswerte werden als DECIMAL(11,4) gespeichert. Die Genauigkeit ist bewusst auf vier Nachkommastellen begrenzt und definiert den maximalen Präzisionsgrad für Preisberechnungen. Für Finanzdaten empfiehlt sich die Verwendung von Zahlenwerten im DECIMAL Format. |
| **Zeitliche Attribute** | Zeitbezogene Informationen (Kauf, Verkauf, Kurszeitpunkt) werden als DATETIME modelliert. Dadurch sind zeitbasierte Auswertungen, historische Analysen und Performance-Berechnungen möglich. |
| **Zeitreihenmodellierung** | Die Tabelle **KURS** besitzt einen eigenen Primärschlüssel und ist über Anlage_ID eindeutig einer Anlage zugeordnet. Jeder Datensatz repräsentiert einen Kurs zu einem bestimmten Zeitpunkt. |
| **Feldlängen (VARCHAR)** | Für Textfelder wurden konkrete Längen definiert (z. B. Namen, Kürzel, E-Mail). Diese Begrenzungen dienen der Datenvalidierung, Performance und Konsistenz der gespeicherten Werte. |

# 4 Automatisierter Kursdatendownload

Als Zusatzleistung haben wir ein Python-Skript entwickelt, das für Aktien echte Marktdaten von der Yahoo-Finance API beziehen kann. Das Skript nutzt die yfinance-Bibliothek, um aktuelle Marktdaten von Yahoo Finance abzurufen. Ausgegeben werden die heruntergeladenen Daten als .csv-Datei, welche dann bequem in SQL importiert werden kann.

Die technische Umsetzung erfolgt über eine strukturierte Pipeline: Zuerst werden die Kürzel (Ticker-Symbole) identifiziert (müssen zuvor aus der Datenbank exportiert werden). Für jedes dieser Symbole fragt das Skript die historischen Kursdaten sowie den zugehörigen Zeitstempel ab.

Damit konnten wir für die Tabelle **kurs** bei drei Aktien einen grossen Umfang an echten Marktdaten in unsere Datenbank importieren, die ansonsten fiktive Daten enthält.

# 5 SELECT-Abfragen

Um die Kernfunktionalitäten der «TrackMyAssets»-App zu realisieren, wurden verschiedene SQL-Abfragen entwickelt, die über einfache Datenabrufe hinausgehen. Da das Ziel der App die brokerübergreifende Analyse von Vermögenswerten ist, müssen die Abfragen komplexe Verknüpfungen zwischen Nutzerdaten, Stammdaten der Anlagen und zeitabhängigen Kursen herstellen. Ein zentrales Hilfsmittel war uns dabei die Common Table Expression (CTE), die wir **latest_kurs**, die sicherstellt, dass für jede Bewertung stets der aktuelle verfügbare Marktpreis herangezogen wird.

**Portfolio-Management und KPI-Berechnung**

Die wichtigste Abfrage für den Endnutzer ist die Portfolioübersicht (Abfragen 1 und 2). Hierbei werden alle noch nicht verkauften Positionen (*ZeitstempelVerkauf IS NULL*) mit den aktuellen Kursdaten verknüpft. Dies ermöglicht die Berechnung des aktuellen Gesamtwerts sowie der Anzahl offener Positionen in Echtzeit. Für «TrackMyAssets» ist dies essenziell, da das die Kernfunktionalität der App darstellt: Den Überblick über das Gesamtvermögen zu behalten, ohne sich bei mehreren Brokern einzeln anmelden zu müssen.

**Visualisierung der Assets**

Ein wesentlicher Bestandteil moderner Anlage-Apps ist es, dass man eine visuelle Darstellung der Risikoverteilung hat. Durch die Abfragen zur Portfolio-Verteilung nach Anlagetyp (Abfrage 3) und nach Broker (Abfrage 4) liefert die Datenbank die notwendigen Rohdaten für Diagramme (z. B. «Aktien vs. ETFs»). Diese Aggregationen sind für Nutzer relevant, um Risiken zu identifizieren. So können sie erkennen, ob sie beispielsweise zu stark bei einem einzelnen Broker investiert sind oder eine Asset-Klasse im Gesamtportfolio übergewichtet ist.

**Marktübersicht und Kursverläufe**

Neben der persönlichen Bestandsaufnahme bietet die Datenbank Funktionen zur Marktbeobachtung. Die Abfragen 5 bis 7 ermöglichen die Erstellung von Preislisten und historischen Kurscharts. Insbesondere die Abfrage des Kursverlaufs über einen bestimmten Zeitraum ist für die App-Entwicklung kritisch, da sie die Basis für die Performance-Graphen bildet. Hierbei werden die historischen Daten aus der Tabelle KURS genutzt, um die Wertentwicklung einer Anlage (z.B. der Apple-Aktie) visuell nachvollziehbar zu machen.

**Transaktionshistorie und Nutzeranalyse**

Die Abfragen 8 bis 11 vereinfachen die Analyse und Administration für Nutzer von «TrackMyAssets». Die Transaktionsübersicht (Abfrage 8) hilft dem Nutzer bei der Steuererklärung, indem sie realisierte Gewinne und Haltedauern aufzeigt. Die Nutzeranalysen (Abfrage 10 und 11) dienen dem Startup zur statistischen Auswertung. So lässt sich ermitteln, welche Anlagetypen oder spezifischen Instrumente bei den Nutzern besonders beliebt sind. Den Abschluss bilden Stammdaten-Listen (Abfrage 12), die mittels UNION ALL z.B. für Menüs im Front-End verwendet werden können.

# 6 Beschreibung der Beziehungen

| Beziehung (A ↔ B) | Beziehungstyp | Realisierung | Zweck / fachliche Bedeutung | Wichtige Regeln / Integrität |
|---|---|---|---|---|
| **nutzer ↔ anlage** | m:m (Ereignis-Beziehung) | Zwischentabelle **nutzer_anlage** (Nutzer_ID, Anlage_ID) | Zentrale Geschäftsbeziehung: Nutzer kaufen, halten und verkaufen Anlagen. Eine Anlage kann von vielen Nutzern gehandelt werden. nutzer_anlage repräsentiert eine Position bzw. ein Kauf-/Verkaufsereignis. | Ereignisattribute: Anzahl, ZeitstempelKauf, optional ZeitstempelVerkauf, Broker_ID. CHECK (Verkauf IS NULL OR Kauf < Verkauf) verhindert logische Fehler. |
| **broker ↔ anlage** | m:m (indirekt) | Ebenfalls über **nutzer_anlage** (Broker_ID, Anlage_ID) | Eine Anlage kann über verschiedene Broker gehandelt werden; ein Broker bietet viele Anlagen an. Die Beziehung entsteht nur durch reale Positionen von Nutzern. | Fachlich ist ein Broker immer vorhanden. Technisch sollte Broker_ID NOT NULL gesetzt werden, um dies zu erzwingen. |
| **anlage → kurs** | 1:m | FK kurs.Anlage_ID → anlage.ID_Anlage | Eine Anlage besitzt eine Kurs-Zeitreihe. Grundlage für Charts, Performance-Berechnungen und den aktuellen Marktwert. | ON DELETE CASCADE: Beim Löschen einer Anlage werden alle zugehörigen Kurse automatisch entfernt. |
| **waehrung → anlage** | 1:m (c:m) | FK anlage.Waehrung_ID → waehrung.ID_Waehrung | Währung ist ein Stammdatum (Kategorie). Jede Anlage ist genau einer Währung zugeordnet, viele Anlagen teilen sich dieselbe Währung. | Name und Kuerzel der Währung sind eindeutig (UNIQUE). |
| **anlagetyp → anlage** | 1:m (c:m) | FK anlage.Anlagetyp_ID → anlagetyp.ID_Anlagetyp | Anlagetyp klassifiziert Anlagen (z. B. Aktie, ETF, Krypto). Ein Typ umfasst viele Anlagen. | anlagetyp.Name ist eindeutig. Ermöglicht Auswertungen nach Kategorie. |
| **masseinheit → anlage** | 1:m (c:m) | FK anlage.Masseinheit_ID → masseinheit.ID_Masseinheit | Masseinheit beschreibt, in welcher Einheit eine Anlage gehandelt wird (z. B. Stück, Anteil). | masseinheit.Name und Kuerzel sind eindeutig. |

| | | | | |
|---|---|---|---|---|
| **nutzer → nutzer_anlage** | 1:m | FK nutzer_anlage.Nutzer_ID → nutzer.ID_Nutzer | Ein Nutzer kann mehrere Positionen besitzen (auch dieselbe Anlage mehrfach). Abbildung der individuellen Handels- und Besitzhistorie. | ON DELETE CASCADE: Beim Löschen eines Nutzers werden seine Positionen entfernt. |
| **anlage → nutzer_anlage** | 1:m | FK nutzer_anlage.Anlage_ID → anlage.ID_Anlage | Eine Anlage kann in vielen Nutzerpositionen vorkommen. Grundlage für Auswertungen wie „Wie oft wurde eine Anlage gekauft?“. | ON DELETE NO ACTION: Anlagen mit bestehenden Positionen dürfen nicht gelöscht werden. |
| **broker → nutzer_anlage** | 1:m | FK nutzer_anlage.Broker_ID → broker.ID_Broker | Ein Broker ist der Ausführungs- bzw. Verwahrungskanal einer Position. | Fachlich verpflichtend; technisch empfohlen: Broker_ID NOT NULL. |

# 7 Lieferobjekte

Das hier vorgelegte Projekt umfasst folgende Lieferobjekte.

- **INFW2025a_01_Skripts.zip**
  - SQL-Skript zum Erstellen der Datenbank und der Tabellen
    - CREATE DATABASE und CREATE TABLE
  - SQL-Skript zum Einfügen der Daten
    - Import mit CSV-Dateien, mit Importverzeichnis C:\Daten\CSV\
  - SQL-Skript mit wichtigen Abfragen von Daten (SELECT-Befehle)
  - Die beiden Datenmodelle (konzeptionell und logisch)
- **INFW2025a_01_CSV.zip**
  - Für jede Tabelle der Datenbank eine .csv-Datei mit den Daten (für den Import)
- **INFW2025a_01_Zusatzleistung.zip**
  - Das Python-Skript für automatisiertes Herunterladen von Aktienkursdaten
- **INFW2025a_01_Dokumentation.pdf**
  - Die hier vorliegende Projektdokumentation