合肥大學
HEFEI UNIVERSITY

Datenbanken

36. Logisches Schema: Entitäten zu Tabellen

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2025-11-19

Databases

Dies ist ein Kurs über Datenbanken an der Universität Hefei (合肥大学).

Die Webseite mit dem Lehrmaterial dieses Kurses ist https://thomasweise.github.io/databases (siehe auch den QR-Kode unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielen finden Sie unter https://github.com/thomasWeise/databasesCode.

Outline

Einleitung

Einleitung

- Wir wollen nun unser konzeptuelles Modell des Lehre-Management-Systems als eine relationale Datenbank implementieren.

Einleitung

- Wir wollen nun unser konzeptuelles Modell des Lehre-Management-Systems als eine relationale Datenbank implementieren.
- Das erfordert, dass wir Entitäten, Beziehungen, und Einschränkungen auf Objekte aus der relationalen Welt abbilden.

Einleitung

- Wir wollen nun unser konzeptuelles Modell des Lehre-Management-Systems als eine relationale Datenbank implementieren.
- Das erfordert, dass wir Entitäten, Beziehungen, und Einschränkungen auf Objekte aus der relationalen Welt abbilden.
- Wir werden Sicheten, Anfragen, und Einfüge-Regeln für unsere Daten entwerfen müssen.

Einleitung

- Wir wollen nun unser konzeptuelles Modell des Lehre-Management-Systems als eine relationale Datenbank implementieren.
- Das erfordert, dass wir Entitäten, Beziehungen, und Einschränkungen auf Objekte aus der relationalen Welt abbilden.
- Wir werden Sicheten, Anfragen, und Einfüge-Regeln für unsere Daten entwerfen müssen.
- Natürlich wählen wir PostgreSQL als der DBMS.

Einleitung

- Wir wollen nun unser konzeptuelles Modell des Lehre-Management-Systems als eine relationale Datenbank implementieren.
- Das erfordert, dass wir Entitäten, Beziehungen, und Einschränkungen auf Objekte aus der relationalen Welt abbilden.
- Wir werden Sicheten, Anfragen, und Einfüge-Regeln für unsere Daten entwerfen müssen.
- Natürlich wählen wir PostgreSQL als der DBMS.
- PostgreSQL unterstützt SQL, also können wir die meiste Funktonalität, die wir benutzen, 1:1 auch in Systemen wie MySQL, MariaDB, oder SQLite verwenden.

Übersetzung

- Aber wie übersetzen wir ein konzeptuelles Model in ein logisches?

Übersetzung

- Aber wie übersetzen wir ein konzeptuelles Model in ein logisches?
- Viele Literaturquellen sagen, dass Entity-Relationship-Modelle einfach in logisches Schemas des relationalen Datenmodells übersetzt werden können und das Werkzeuge das automatisch können[83].

Übersetzung

- Aber wie übersetzen wir ein konzeptuelles Model in ein logisches?
- Viele Literaturquellen sagen, dass Entity-Relationship-Modelle einfach in logisches Schemas des relationalen Datenmodells übersetzt werden können und das Werkzeuge das automatisch können[83].
- Ich denke, das hängt schon sehr davon ab, wie abstrakt die ERDs sind und davon, wie genau die Übersetzung seien soll.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.
- Wir benutzen yEd, welcher von Datenbanktechnologien völlig unabhängig ist.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.
- Wir benutzen yEd, welcher von Datenbanktechnologien völlig unabhängig ist.
- Solche Modelle zu logischen Modellen zu übersetzen erfortet schon Denkarbeit, obwohl es oft nicht so schwer ist.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.
- Wir benutzen yEd, welcher von Datenbanktechnologien völlig unabhängig ist.
- Solche Modelle zu logischen Modellen zu übersetzen erfortet schon Denkarbeit, obwohl es oft nicht so schwer ist.
- Wir hätten auch den PgModeler verwenden können, um unsere ERDs zu zeichnen.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.
- Wir benutzen yEd, welcher von Datenbanktechnologien völlig unabhängig ist.
- Solche Modelle zu logischen Modellen zu übersetzen erfortet schon Denkarbeit, obwohl es oft nicht so schwer ist.
- Wir hätten auch den PgModeler verwenden können, um unsere ERDs zu zeichnen.
- Der PgModeler kann SQL produzieren bzw. sogar direkt mit dem PostgreSQL DBMS verbunden werden.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.
- Wir benutzen yEd, welcher von Datenbanktechnologien völlig unabhängig ist.
- Solche Modelle zu logischen Modellen zu übersetzen erfortet schon Denkarbeit, obwohl es oft nicht so schwer ist.
- Wir hätten auch den PgModeler verwenden können, um unsere ERDs zu zeichnen.
- Der PgModeler kann SQL produzieren bzw. sogar direkt mit dem PostgreSQL DBMS verbunden werden.
- Dann hätten wir aber kein abstraktes konzeptuelles Modell gehabt.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.
- Wir benutzen yEd, welcher von Datenbanktechnologien völlig unabhängig ist.
- Solche Modelle zu logischen Modellen zu übersetzen erfortet schon Denkarbeit, obwohl es oft nicht so schwer ist.
- Wir hätten auch den PgModeler verwenden können, um unsere ERDs zu zeichnen.
- Der PgModeler kann SQL produzieren bzw. sogar direkt mit dem PostgreSQL DBMS verbunden werden.
- Dann hätten wir aber kein abstraktes konzeptuelles Modell gehabt.
- Sondern direkt ein logisches Modell.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.
- Wir benutzen yEd, welcher von Datenbanktechnologien völlig unabhängig ist.
- Solche Modelle zu logischen Modellen zu übersetzen erfortet schon Denkarbeit, obwohl es oft nicht so schwer ist.
- Wir hätten auch den PgModeler verwenden können, um unsere ERDs zu zeichnen.
- Der PgModeler kann SQL produzieren bzw. sogar direkt mit dem PostgreSQL DBMS verbunden werden.
- Dann hätten wir aber kein abstraktes konzeptuelles Modell gehabt.
- Sondern direkt ein logisches Modell.
- Aber wir haben ein abstraktes konzeptuelles Modell.

Werkzeuge

- Es gibt verschiedene Werkzeuge, um ERDs zu malen.
- Wir benutzen yEd, welcher von Datenbanktechnologien völlig unabhängig ist.
- Solche Modelle zu logischen Modellen zu übersetzen erfortet schon Denkarbeit, obwohl es oft nicht so schwer ist.
- Wir hätten auch den PgModeler verwenden können, um unsere ERDs zu zeichnen.
- Der PgModeler kann SQL produzieren bzw. sogar direkt mit dem PostgreSQL DBMS verbunden werden.
- Dann hätten wir aber kein abstraktes konzeptuelles Modell gehabt.
- Sondern direkt ein logisches Modell.
- Aber wir haben ein abstraktes konzeptuelles Modell.
- Und nun lernen wir, wie man diese in logische Modelle übersetzt.

Entitäten und Tabellen

Entitäten zu Tabellen

- Entitätstypen vom konzeptuellen Modell in das logische Modell zu übersetzen ist sehr einfach.

Entitäten zu Tabellen

- Entitätstypen vom konzeptuellen Modell in das logische Modell zu übersetzen ist sehr einfach.
- Jeder Entitätstyp wird eine Tabelle.

Entitäten zu Tabellen

- Entitätstypen vom konzeptuellen Modell in das logische Modell zu übersetzen ist sehr einfach.
- Jeder Entitätstyp wird eine Tabelle.
- Der Entitätstyp *Student* wird die Tabelle `student`.

Entitäten zu Tabellen

- Entitätstypen vom konzeptuellen Modell in das logische Modell zu übersetzen ist sehr einfach.
- Jeder Entitätstyp wird eine Tabelle.
- Der Entitätstyp *Student* wird die Tabelle `student`.
- Jedes einfache, ein-wertige Attribut wird eine Spalte in der Tabelle.

Entitäten zu Tabellen

- Entitätstypen vom konzeptuellen Modell in das logische Modell zu übersetzen ist sehr einfach.
- Jeder Entitätstyp wird eine Tabelle.
- Der Entitätstyp *Student* wird die Tabelle `student`.
- Jedes einfache, ein-wertige Attribut wird eine Spalte in der Tabelle.
- Das Attribut *Name* wird zur Spalte `name`.

Entitäten zu Tabellen

- Entitätstypen vom konzeptuellen Modell in das logische Modell zu übersetzen ist sehr einfach.
- Jeder Entitätstyp wird eine Tabelle.
- Der Entitätstyp *Student* wird die Tabelle `student`.
- Jedes einfache, ein-wertige Attribut wird eine Spalte in der Tabelle.
- Das Attribut *Name* wird zur Spalte `name` mit einem SQL-Datentyp für Text.

Entitäten zu Tabellen

- Entitätstypen vom konzeptuellen Modell in das logische Modell zu übersetzen ist sehr einfach.
- Jeder Entitätstyp wird eine Tabelle.
- Der Entitätstyp *Student* wird die Tabelle `student`.
- Jedes einfache, ein-wertige Attribut wird eine Spalte in der Tabelle.
- Das Attribut *Name* wird zur Spalte `name` mit einem SQL-Datentyp für Text, vielleicht mit zusätzlichen Einschränkungen wie z. B. das Namen mit druckbaren Zeichen anfangen und enden müssen.

Entitäten zu Tabellen

- Entitätstypen vom konzeptuellen Modell in das logische Modell zu übersetzen ist sehr einfach.
- Jeder Entitätstyp wird eine Tabelle.
- Der Entitätstyp *Student* wird die Tabelle `student`.
- Jedes einfache, ein-wertige Attribut wird eine Spalte in der Tabelle.
- Das Attribut *Name* wird zur Spalte `name` mit einem SQL-Datentyp für Text, vielleicht mit zusätzlichen Einschränkungen wie z. B. das Namen mit druckbaren Zeichen anfangen und enden müssen.
- Abgeleitete Attribute werden normalerweise gar nicht gespeichert.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.
- Nehmen wir an, *Name* ist kein einfaches Attribut ist, sondern ein zusammengesetztes Attribut.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.
- Nehmen wir an, *Name* ist kein einfaches Attribut ist, sondern ein zusammengesetztes Attribut.
- Sagen wir, es besteht aus den beiden Komponenten *Full Name* und *Salutation*.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.
- Nehmen wir an, *Name* ist kein einfaches Attribut ist, sondern ein zusammengesetztes Attribut.
- Sagen wir, es besteht aus den beiden Komponenten *Full Name* und *Salutation*.
- Dann haben wir die beiden Spalten `full_name` und `saluation`.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.
- Nehmen wir an, *Name* ist kein einfaches Attribut ist, sondern ein zusammengesetztes Attribut.
- Sagen wir, es besteht aus den beiden Komponenten *Full Name* und *Salutation*.
- Dann haben wir die beiden Spalten `full_name` und `saluation`.
- Beide hätten passende Datentypen und Einschränkungen.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.
- Nehmen wir an, *Name* ist kein einfaches Attribut ist, sondern ein zusammengesetztes Attribut.
- Sagen wir, es besteht aus den beiden Komponenten *Full Name* und *Salutation*.
- Dann haben wir die beiden Spalten `full_name` und `saluation`.
- Beide hätten passende Datentypen und Einschränkungen.
- Das zusammengesetzte Attribute *Name* selber hat keine Spalte in der Tabelle.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.
- Nehmen wir an, *Name* ist kein einfaches Attribut ist, sondern ein zusammengesetztes Attribut.
- Sagen wir, es besteht aus den beiden Komponenten *Full Name* und *Salutation*.
- Dann haben wir die beiden Spalten `full_name` und `saluation`.
- Beide hätten passende Datentypen und Einschränkungen.
- Das zusammengesetzte Attribute *Name* selber hat keine Spalte in der Tabelle.
- Stattdessen haben seine Komponenten Spalten.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.
- Nehmen wir an, *Name* ist kein einfaches Attribut ist, sondern ein zusammengesetztes Attribut.
- Sagen wir, es besteht aus den beiden Komponenten *Full Name* und *Salutation*.
- Dann haben wir die beiden Spalten `full_name` und `saluation`.
- Beide hätten passende Datentypen und Einschränkungen.
- Das zusammengesetzte Attribute *Name* selber hat keine Spalte in der Tabelle.
- Stattdessen haben seine Komponenten Spalten.
- Wenn die Komponenten selber zusammengesetzte Attribute sind, dann wird der Prozess rekursiv angewandt.

Zusammengesetzte Attribute (Teil 1)

- Jede Komponente eines zusammengesetzten Attributs wird eine Spalte der Tabelle.
- Nehmen wir an, *Name* ist kein einfaches Attribut ist, sondern ein zusammengesetztes Attribut.
- Sagen wir, es besteht aus den beiden Komponenten *Full Name* und *Salutation*.
- Dann haben wir die beiden Spalten `full_name` und `saluation`.
- Beide hätten passende Datentypen und Einschränkungen.
- Das zusammengesetzte Attribute *Name* selber hat keine Spalte in der Tabelle.
- Stattdessen haben seine Komponenten Spalten.
- Wenn die Komponenten selber zusammengesetzte Attribute sind, dann wird der Prozess rekursiv angewandt.
- Die Kompontenten werden dann heruntergebrochen und zwar so lange, bis wir bei einfachen Attributen ankommen, die dann Spalten bekommen.

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.
- Jede derer Zeilen referenziert eine Zeile in er Tabelle des Entitätstyps (durch Speicheren von deren Primärschlüssel als Fremdschlüssel).

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.
- Jede derer Zeilen referenziert eine Zeile in er Tabelle des Entitätstyps (durch Speicheren von deren Primärschlüssel als Fremdschlüssel).
- Wenn der Entitätstyp *Student* im konzeptuellen Modell ein mehrwertiges Attribut *Mobile Phone* hat, dann kann jede Entität vom Typ *Student* mehrere *Mobile Phone* Werte habe.

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.
- Jede derer Zeilen referenziert eine Zeile in er Tabelle des Entitätstyps (durch Speicheren von deren Primärschlüssel als Fremdschlüssel).
- Wenn der Entitätstyp *Student* im konzeptuellen Modell ein mehrwertiges Attribut *Mobile Phone* hat, dann kann jede Entität vom Typ *Student* mehrere *Mobile Phone* Werte habe.
- Im relationalen Datenmodell sind alle Datentypen atomisch, wir können also keine Spalte vom Typ „Liste von (irgendwas)“ haben.

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.
- Jede derer Zeilen referenziert eine Zeile in er Tabelle des Entitätstyps (durch Speicheren von deren Primärschlüssel als Fremdschlüssel).
- Wenn der Entitätstyp *Student* im konzeptuellen Modell ein mehrwertiges Attribut *Mobile Phone* hat, dann kann jede Entität vom Typ *Student* mehrere *Mobile Phone* Werte habe.
- Im relationalen Datenmodell sind alle Datentypen atomisch, wir können also keine Spalte vom Typ „Liste von (irgendwas)“ haben.
- Jedes Attribut kann nur einen Wert in jedem Datensatz haben.

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.
- Jede derer Zeilen referenziert eine Zeile in er Tabelle des Entitätstyps (durch Speicheren von deren Primärschlüssel als Fremdschlüssel).
- Wenn der Entitätstyp *Student* im konzeptuellen Modell ein mehrwertiges Attribut *Mobile Phone* hat, dann kann jede Entität vom Typ *Student* mehrere *Mobile Phone* Werte habe.
- Im relationalen Datenmodell sind alle Datentypen atomisch, wir können also keine Spalte vom Typ „Liste von (irgendwas)“ haben.
- Jedes Attribut kann nur einen Wert in jedem Datensatz haben.
- Daher müssen mehrwertige Attribute selber Tabellen werden.

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.
- Jede derer Zeilen referenziert eine Zeile in er Tabelle des Entitätstyps (durch Speicheren von deren Primärschlüssel als Fremdschlüssel).
- Wenn der Entitätstyp *Student* im konzeptuellen Modell ein mehrwertiges Attribut *Mobile Phone* hat, dann kann jede Entität vom Typ *Student* mehrere *Mobile Phone* Werte habe.
- Im relationalen Datenmodell sind alle Datentypen atomisch, wir können also keine Spalte vom Typ „Liste von (irgendwas)“ haben.
- Jedes Attribut kann nur einen Wert in jedem Datensatz haben.
- Daher müssen mehrwertige Attribute selber Tabellen werden.
- Also würden wir die Tabelle `mobile` für Mobiltelefonnummern erstellen.

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.
- Jede derer Zeilen referenziert eine Zeile in er Tabelle des Entitätstyps (durch Speicheren von deren Primärschlüssel als Fremdschlüssel).
- Wenn der Entitätstyp *Student* im konzeptuellen Modell ein mehrwertiges Attribut *Mobile Phone* hat, dann kann jede Entität vom Typ *Student* mehrere *Mobile Phone* Werte habe.
- Im relationalen Datenmodell sind alle Datentypen atomisch, wir können also keine Spalte vom Typ „Liste von (irgendwas)“ haben.
- Jedes Attribut kann nur einen Wert in jedem Datensatz haben.
- Daher müssen mehrwertige Attribute selber Tabellen werden.
- Also würden wir die Tabelle `mobile` für Mobiltelefonnummern erstellen.
- Diese Tabelle hätte eine Spalte für die Telefonnummer und eine Spalte die den entsprechenden *Student*-Datensatz über einen Fremdschlüssel referenziert.

Mehrwertige Attribute (Teil 1)

- Mehrwertige Attribute werden separate Tabellen.
- Jede derer Zeilen referenziert eine Zeile in er Tabelle des Entitätstyps (durch Speicheren von deren Primärschlüssel als Fremdschlüssel).
- Wenn der Entitätstyp *Student* im konzeptuellen Modell ein mehrwertiges Attribut *Mobile Phone* hat, dann kann jede Entität vom Typ *Student* mehrere *Mobile Phone* Werte habe.
- Im relationalen Datenmodell sind alle Datentypen atomisch, wir können also keine Spalte vom Typ „Liste von (irgendwas)“ haben.
- Jedes Attribut kann nur einen Wert in jedem Datensatz haben.
- Daher müssen mehrwertige Attribute selber Tabellen werden.
- Also würden wir die Tabelle `mobile` für Mobiltelefonnummern erstellen.
- Diese Tabelle hätte eine Spalte für die Telefonnummer und eine Spalte die den entsprechenden *Student*-Datensatz über einen Fremdschlüssel referenziert.
- Es könnte auch noch einen Ersatzschlüssel geben, aber das lernen wir später.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.
- MySQL Workbench[65], z. B., kann sich mit dem MySQL DBMS verbinden und erlaubt uns, Tabellen in einer ERD-artigen Syntax zu erstellen.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.
- MySQL Workbench[65], z. B., kann sich mit dem MySQL DBMS verbinden und erlaubt uns, Tabellen in einer ERD-artigen Syntax zu erstellen.
- PgModeler[1] erlaubt uns das selbe für das PostgreSQL DBMS.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.
- MySQL Workbench[65], z. B., kann sich mit dem MySQL DBMS verbinden und erlaubt uns, Tabellen in einer ERD-artigen Syntax zu erstellen.
- PgModeler[1] erlaubt uns das selbe für das PostgreSQL DBMS.
- Die Idee ist, dass wir nun eine klarere und eingeschränktere Syntax zur visuellen Repräsentieren einer Datenbank verwenden.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.
- MySQL Workbench[65], z. B., kann sich mit dem MySQL DBMS verbinden und erlaubt uns, Tabellen in einer ERD-artigen Syntax zu erstellen.
- PgModeler[1] erlaubt uns das selbe für das PostgreSQL DBMS.
- Die Idee ist, dass wir nun eine klarere und eingeschränktere Syntax zur visuellen Repräsentieren einer Datenbank verwenden.
- Diese Syntax kann direkt nach SQL übersetzt werden, welches wir dann an den PostgreSQL Server über z. B. den psql-Klienten schicken können.

Konzeptuelles ERD zu Logisches ERD

- Wir haben konzeptuelle Modelle mit yEd gemalt.
- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.
- MySQL Workbench[65], z. B., kann sich mit dem MySQL DBMS verbinden und erlaubt uns, Tabellen in einer ERD-artigen Syntax zu erstellen.
- PgModeler[1] erlaubt uns das selbe für das PostgreSQL DBMS.
- Die Idee ist, dass wir nun eine klarere und eingeschränktere Syntax zur visuellen Repräsentieren einer Datenbank verwenden.
- Diese Syntax kann direkt nach SQL übersetzt werden, welches wir dann an den PostgreSQL Server über z. B. den psql-Klienten schicken können.
- So eine GUI zu verwenden hat zwei große Vorteile.

Konzeptuelles ERD zu Logisches ERD

- Diese sind an keinerlei Technologie gebunden.
- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.
- MySQL Workbench[65], z. B., kann sich mit dem MySQL DBMS verbinden und erlaubt uns, Tabellen in einer ERD-artigen Syntax zu erstellen.
- PgModeler[1] erlaubt uns das selbe für das PostgreSQL DBMS.
- Die Idee ist, dass wir nun eine klarere und eingeschränktere Syntax zur visuellen Repräsentieren einer Datenbank verwenden.
- Diese Syntax kann direkt nach SQL übersetzt werden, welches wir dann an den PostgreSQL Server über z. B. den psql-Klienten schicken können.
- So eine GUI zu verwenden hat zwei große Vorteile
  1. Die Diagramme sind intuitiv und schneller verständlich als SQL-Skripte.

Konzeptuelles ERD zu Logisches ERD

- Logische Modelle sind aber an technologien gebunden.
- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.
- MySQL Workbench[65], z. B., kann sich mit dem MySQL DBMS verbinden und erlaubt uns, Tabellen in einer ERD-artigen Syntax zu erstellen.
- PgModeler[1] erlaubt uns das selbe für das PostgreSQL DBMS.
- Die Idee ist, dass wir nun eine klarere und eingeschränktere Syntax zur visuellen Repräsentieren einer Datenbank verwenden.
- Diese Syntax kann direkt nach SQL übersetzt werden, welches wir dann an den PostgreSQL Server über z. B. den psql-Klienten schicken können.
- So eine GUI zu verwenden hat zwei große Vorteile
  1. Die Diagramme sind intuitiv und schneller verständlich als SQL-Skripte.
  2. Die verschiedenen Dialoge, die wir beim Bauen des ERDs verwenden, helfen uns, korrektes SQL zu erstellen.

Konzeptuelles ERD zu Logisches ERD

- Hier yEd zu verwenden ergibt wenig Sinn.
- Stattdessen nimmt man oft Werkzeuge, die zumindest SQL beherrschen oder direkt an DBMSe gebunden sind.
- MySQL Workbench[65], z. B., kann sich mit dem MySQL DBMS verbinden und erlaubt uns, Tabellen in einer ERD-artigen Syntax zu erstellen.
- PgModeler[1] erlaubt uns das selbe für das PostgreSQL DBMS.
- Die Idee ist, dass wir nun eine klarere und eingeschränktere Syntax zur visuellen Repräsentieren einer Datenbank verwenden.
- Diese Syntax kann direkt nach SQL übersetzt werden, welches wir dann an den PostgreSQL Server über z. B. den psql-Klienten schicken können.
- So eine GUI zu verwenden hat zwei große Vorteile
  1. Die Diagramme sind intuitiv und schneller verständlich als SQL-Skripte.
  2. Die verschiedenen Dialoge, die wir beim Bauen des ERDs verwenden, helfen uns, korrektes SQL zu erstellen.
- Und genau das probieren wir jetzt aus.

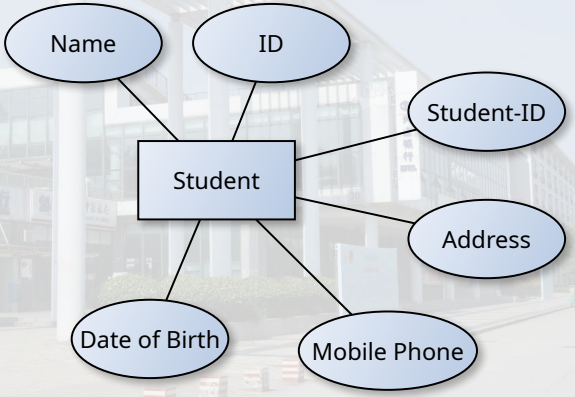
Logisches ERD Erstellen mit PgModeler

Logisches ERD erstellen mit PgModeler

- Wir übersetzen nun ein konzeptuelles ERD in ein logisches Modell.

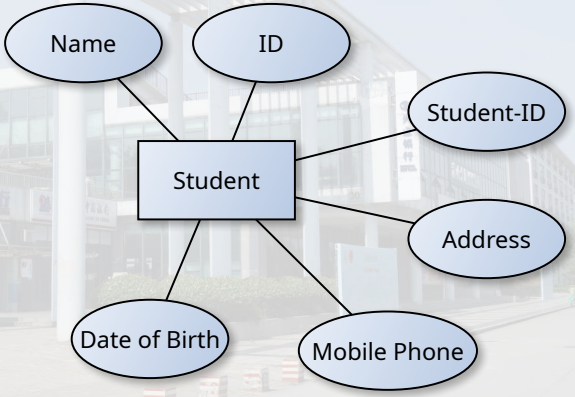
Logisches ERD erstellen mit PgModeler

- Wir übersetzen nun ein konzeptuelles ERD in ein logisches Modell.
- Wir nehmen dafür das allererste ERD was wir gemalt hatten.



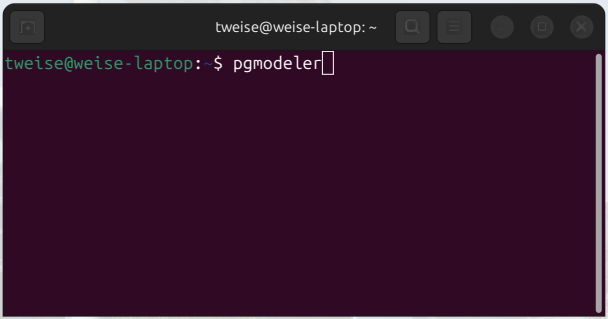
Logisches ERD erstellen mit PgModeler

- Wir übersetzen nun ein konzeptuelles ERD in ein logisches Modell.
- Wir nehmen dafür das allererste ERD was wir gemalt hatten.
- Es hat nur genau einen Entitätstyp.



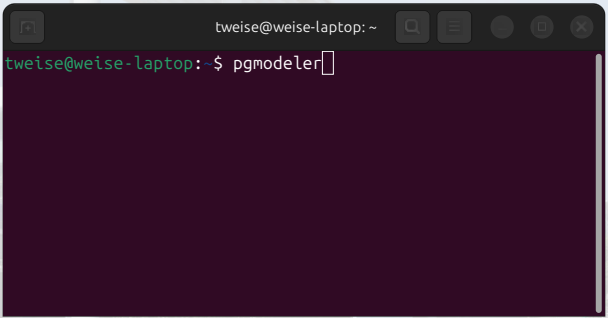
Logisches ERD erstellen mit PgModeler

- Wir starten den PgModeler.



Logisches ERD erstellen mit PgModeler

- Wir starten den PgModeler.
- Unter Ubuntu Linux gegen wir dafür `pgmodler` im Terminal ein drücken ↵.

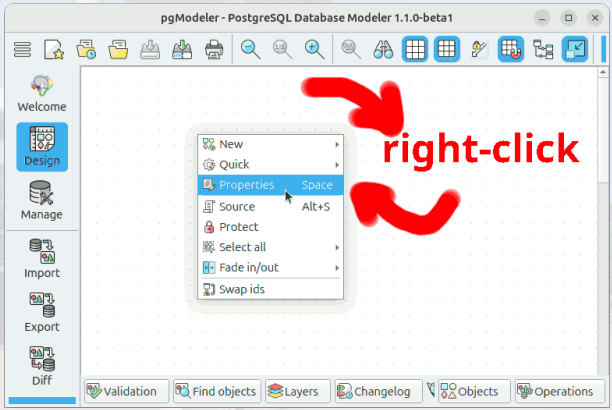


Logisches ERD erstellen mit PgModeler

- Im PgModeler klicken wir auf New Model.

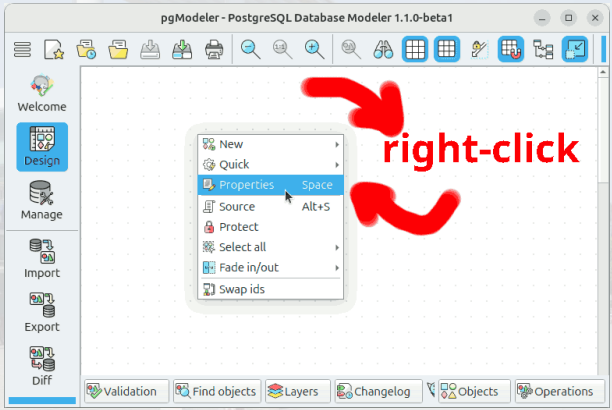
Logisches ERD erstellen mit PgModeler

- Ein leeres ERD öffnet sich.



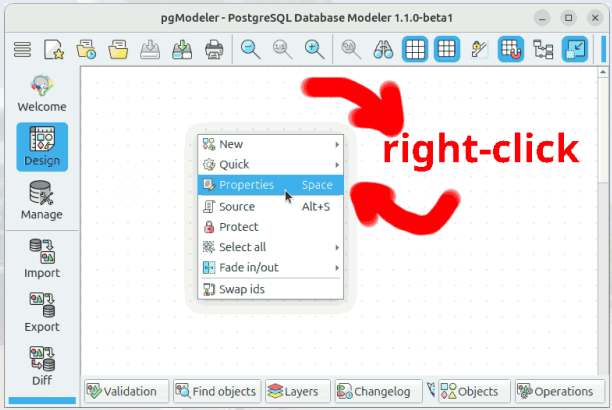
Logisches ERD erstellen mit PgModeler

- Ein leeres ERD öffnet sich.
- Wir klicken irgendwo in es hinein.



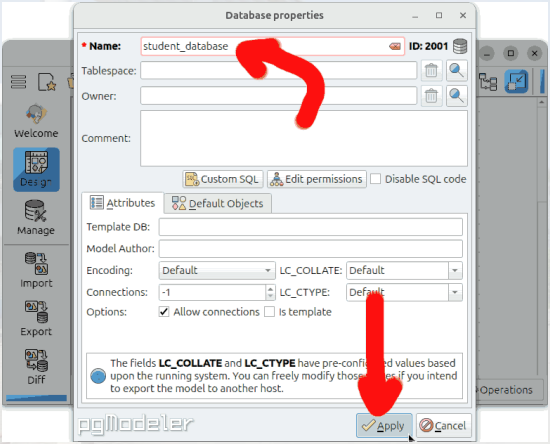
Logisches ERD erstellen mit PgModeler

- Ein leeres ERD öffnet sich.
- Wir klicken irgendwo in es hinein.
- In dem Kontextmenü, dass sich öffnet, klicken wir auf Properties.



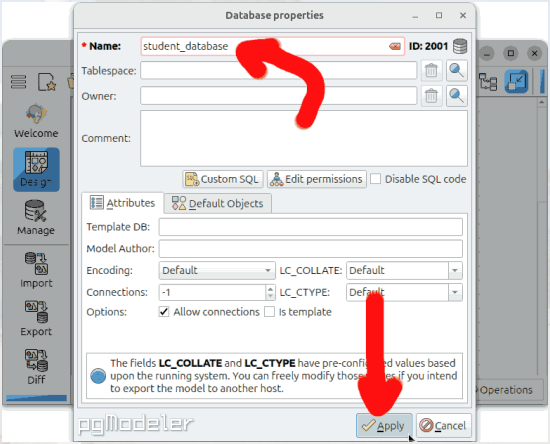
Logisches ERD erstellen mit PgModeler

- Der „Database Properties“-Dialog öffnet sich.



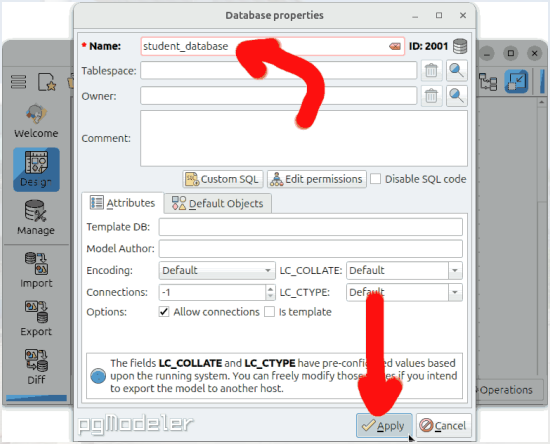
Logisches ERD erstellen mit PgModeler

- Der „Database Properties“-Dialog öffnet sich.
- Wir wollen einen passenden Namen für unsere Datenbank auswählen.



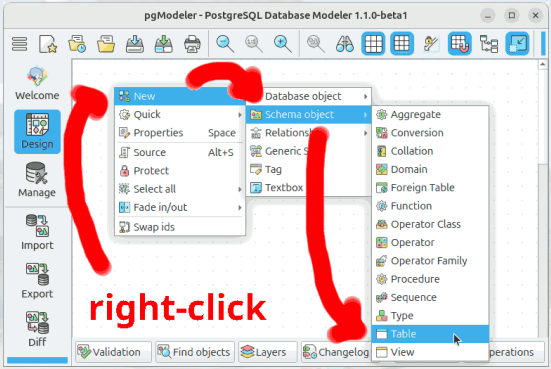
Logisches ERD erstellen mit PgModeler

- Der „Database Properties“-Dialog öffnet sich.
- Wir wollen einen passenden Namen für unsere Datenbank auswählen.
- Wir wählen `student_database` und klicken auf Apply.



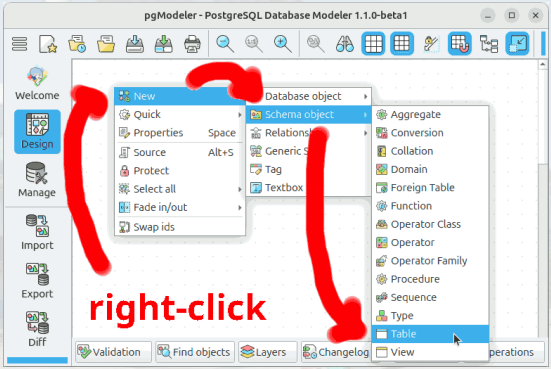
Logisches ERD erstellen mit PgModeler

- Zurück in der ERD-Ansicht rechts-klicken wir nochmal in das leere Diagramm.



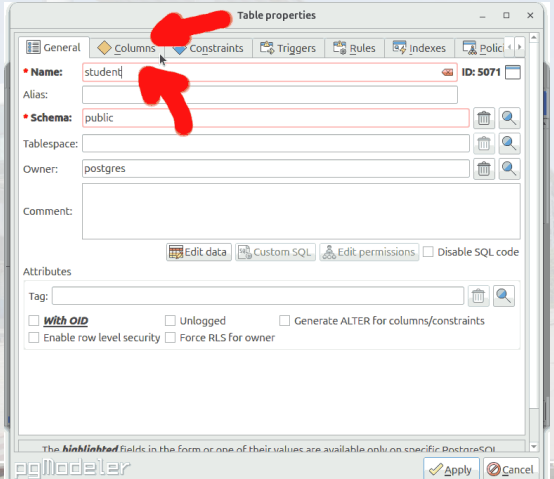
Logisches ERD erstellen mit PgModeler

- Zurück in der ERD-Ansicht rechts-klicken wir nochmal in das leere Diagramm.
- In dem sich öffnenen Popup-Menü klicken wir auf New » Schema Object » Table.



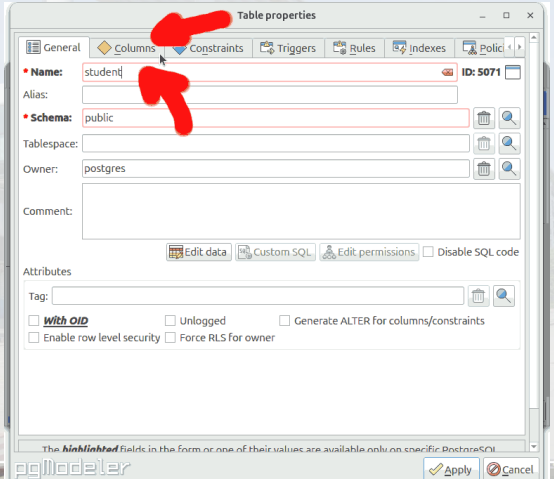
Logisches ERD erstellen mit PgModeler

- Der Dialog „Table properties“ öffnet sich.



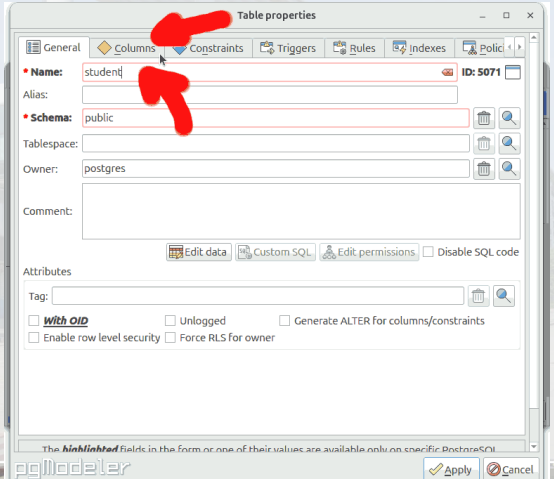
Logisches ERD erstellen mit PgModeler

- Der Dialog „Table properties“ öffnet sich.
- Als Tabellenname geben wir `student` ein.



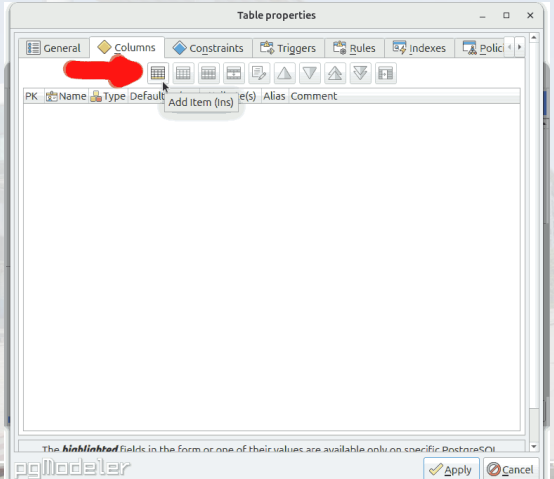
Logisches ERD erstellen mit PgModeler

- Der Dialog „Table properties“ öffnet sich.
- Als Tabellenname geben wir `student` ein.
- Wir klicken auf den Register Columns.



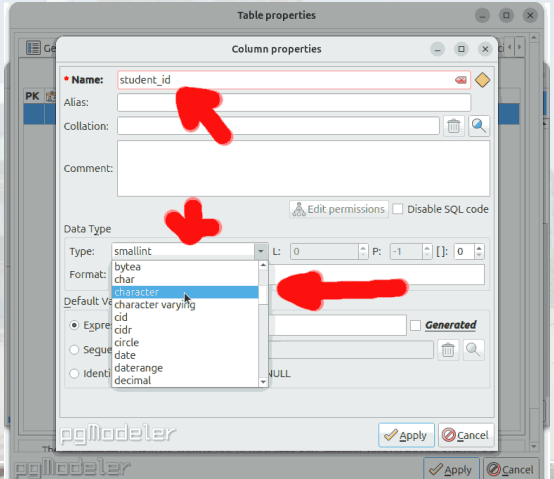
Logisches ERD erstellen mit PgModeler

- Im Register Columns klicken wir auf das Add Item-Symbol .



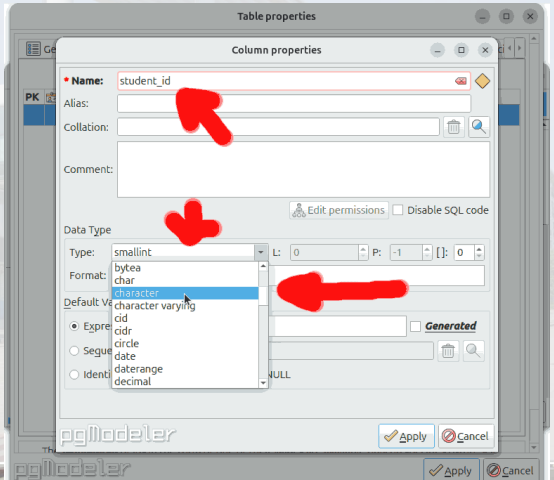
Logisches ERD erstellen mit PgModeler

- Wir wollen eine Spalte einfügen für die Studenten-ID, die von der Uni ausgegeben wird.



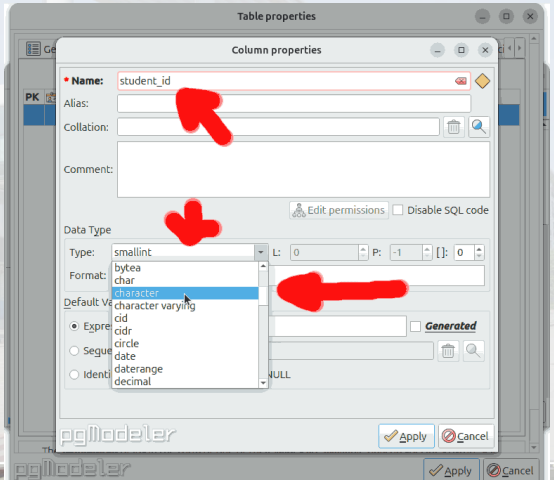
Logisches ERD erstellen mit PgModeler

- Wir wollen eine Spalte einfügen für die Studenten-ID, die von der Uni ausgegeben wird.
- Als Name für die Spalte wählen wir `student_id`.



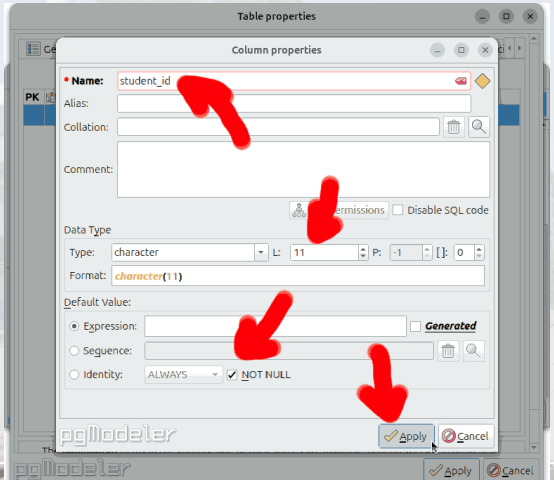
Logisches ERD erstellen mit PgModeler

- Als Name für die Spalte wählen wir `student_id`.
- Als Type wählen wir `character`, also den SQL-Datentyp für Zeichenketten fester Länge (denn alle Studenten-IDs haben die selbe Länge).



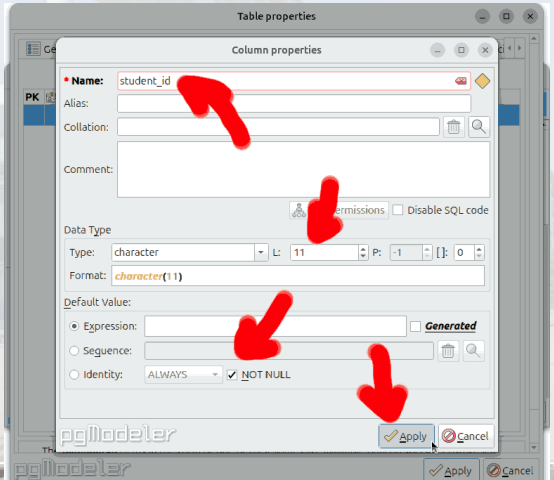
Logisches ERD erstellen mit PgModeler

- Als (feste) Länge geben wir 11 in das L:-Feld ein.



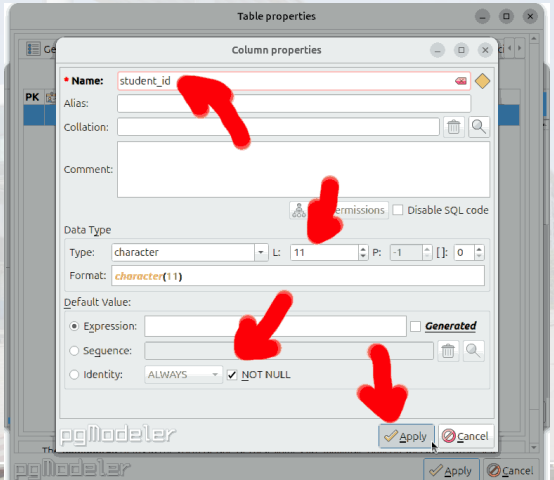
Logisches ERD erstellen mit PgModeler

- Als (feste) Länge geben wir 11 in das L:-Feld ein.
- Wir markieren die Spalte als `NOT NULL`, was bedeutet, dass es niemals einen Studenten-Datensatz ohne Studenten-ID geben kann.



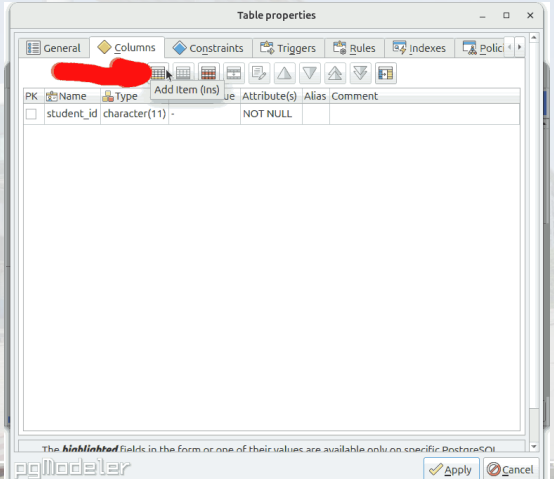
Logisches ERD erstellen mit PgModeler

- Wir klicken auf Apply.



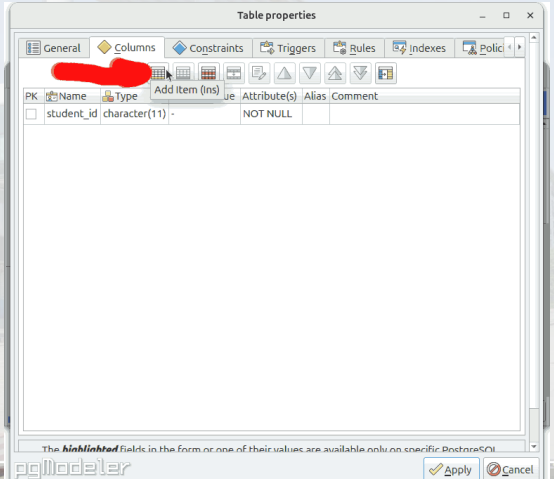
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint im Dialog.



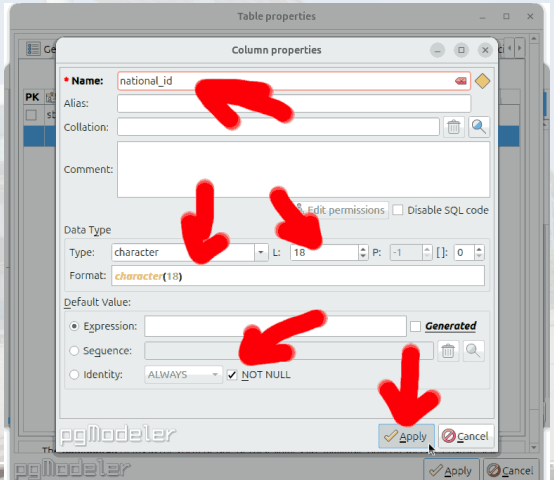
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint im Dialog.
- Wir klicken nochmal auf Add Item .



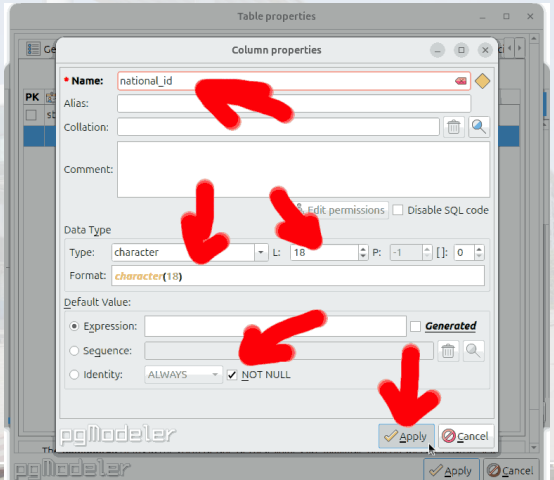
Logisches ERD erstellen mit PgModeler

- Wir fügen die Spalte `national_id` zum Speichern der chinesischen Ausweisnummern (中国公民身份号码) an.



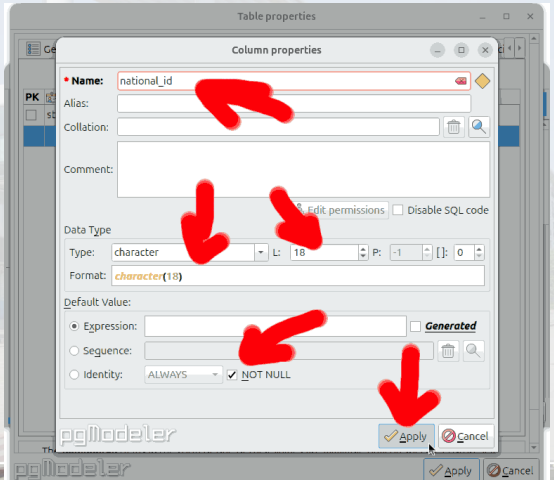
Logisches ERD erstellen mit PgModeler

- Wir fügen die Spalte `national_id` zum Speichern der chinesischen Ausweisnummern (中国公民身份号码) an.
- Solche Nummern sind Zeichenketten (`character`) der festen Länge 18[120].



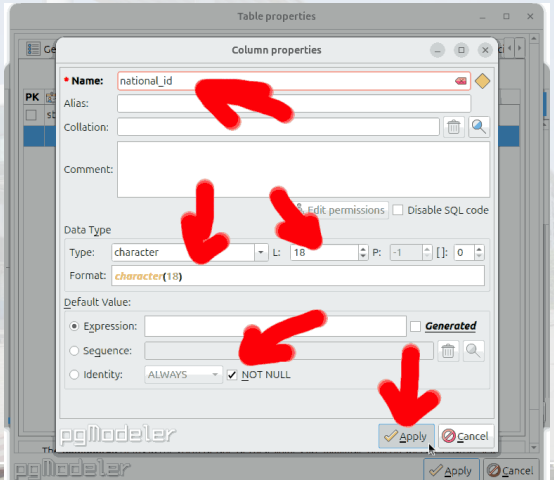
Logisches ERD erstellen mit PgModeler

- Solche Nummern sind Zeichenketten (`character`) der festen Länge 18^{120}.
- Wir markieren auch diese Spalte als `NOT NULL`, denn jeder Datensatz muss so einen Wert speichern.



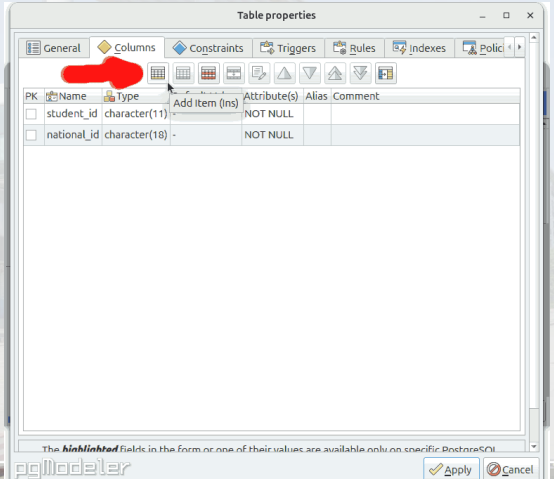
Logisches ERD erstellen mit PgModeler

- Wir markieren auch diese Spalte als `NOT NULL`, denn jeder Datensatz muss so einen Wert speichern.
- Wir klicken auf Apply.



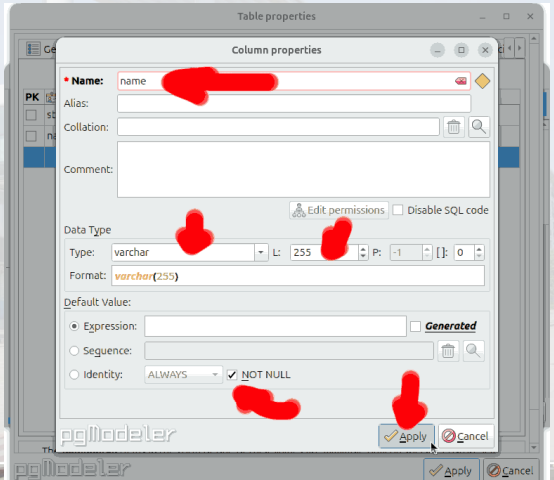
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint und wir klicken auf Add Item .



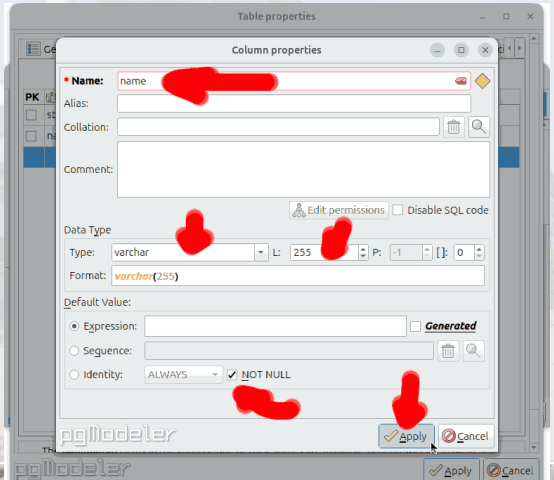
Logisches ERD erstellen mit PgModeler

- Wir definieren die Spalte `name` für Studentennamen.



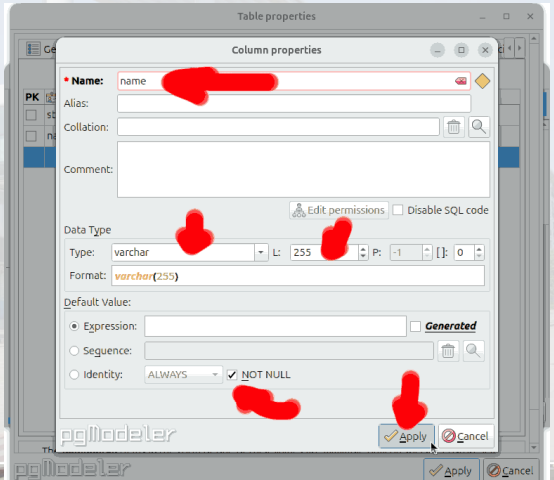
Logisches ERD erstellen mit PgModeler

- Wir definieren die Spalte `name` für Studentennamen.
- Namen haben eine variable Länge (Typ `varchar`).



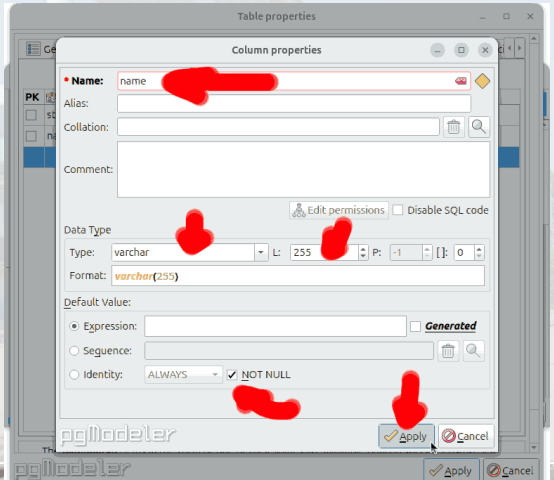
Logisches ERD erstellen mit PgModeler

- Namen haben eine variable Länge (Typ `varchar`).
- Wir setzen die *maximale* Länge auf 255.



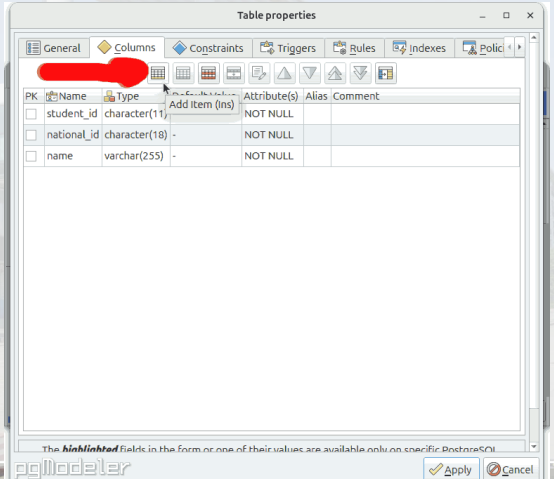
Logisches ERD erstellen mit PgModeler

- Wir setzen die *maximale* Länge auf 255.
- Jeder Student muss einen Namen haben, also spezifizieren wir wieder `NOT NULL` und klicken Apply.



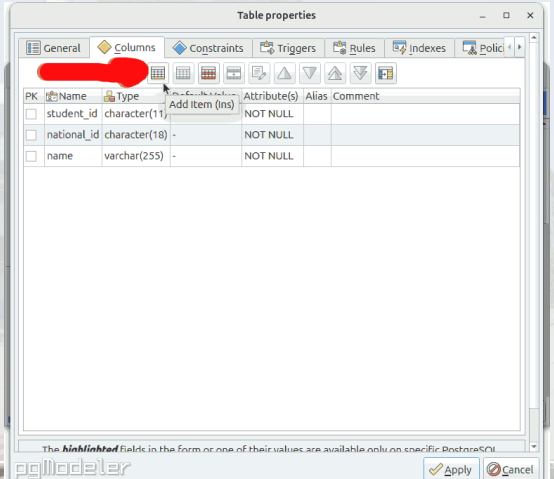
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint im Dialog.



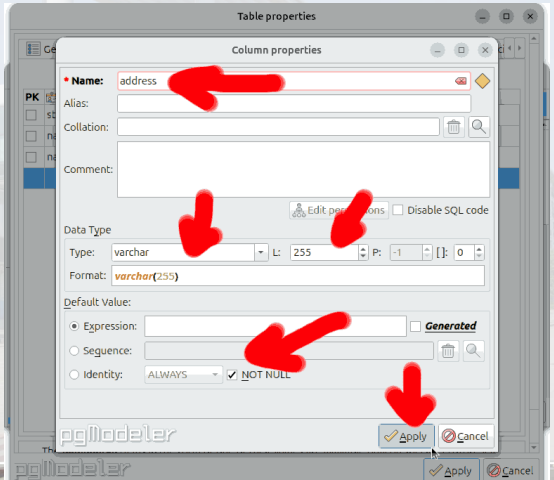
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint im Dialog.
- Wir klicken nochmal auf Add Item .



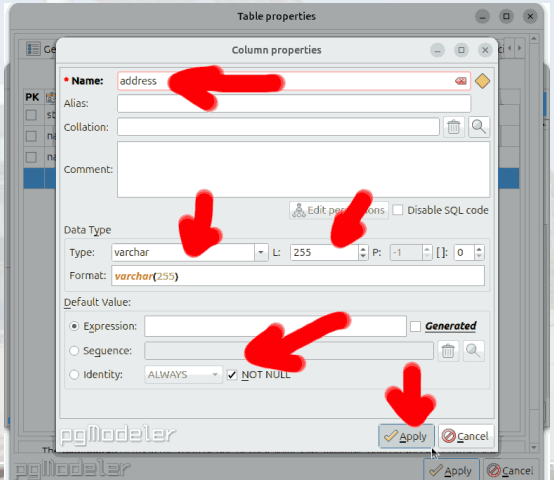
Logisches ERD erstellen mit PgModeler

- Auch Adressen sind Zeichenketten variabler Länge (Typ `varchar`).



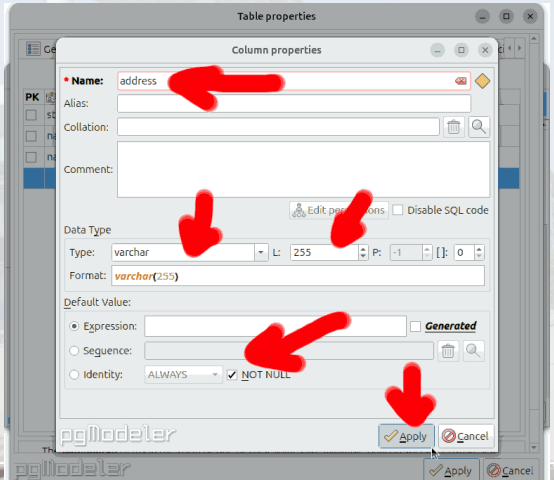
Logisches ERD erstellen mit PgModeler

- Auch Adressen sind Zeichenketten variabler Länge (Typ `varchar`).
- Wir setzen die maximale Länge wieder auf 255 und setzten das Feld auf `NOT NULL`.



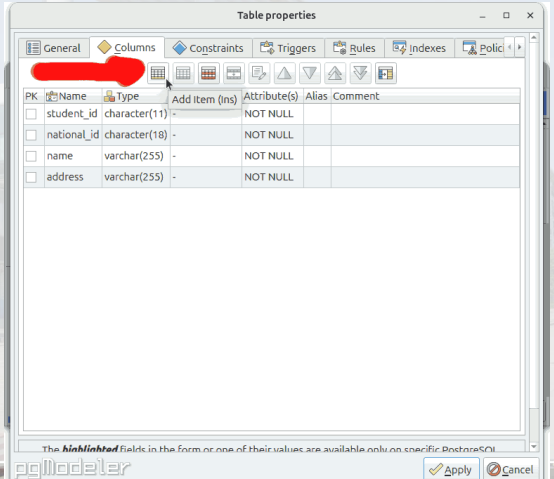
Logisches ERD erstellen mit PgModeler

- Wir setzen die maximale Länge wieder auf 255 und setzten das Feld auf `NOT NULL`.
- Wir klicken Apply.



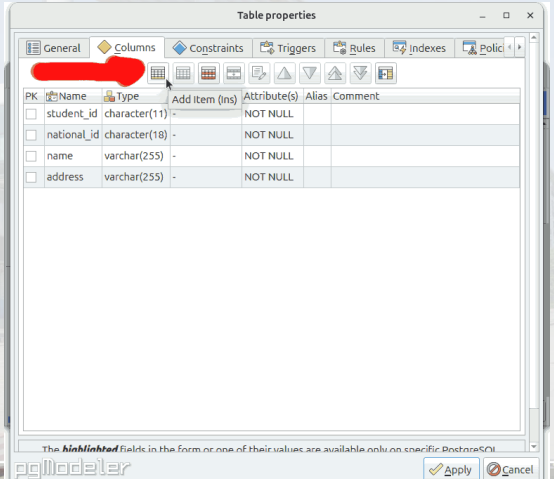
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint im Dialog.



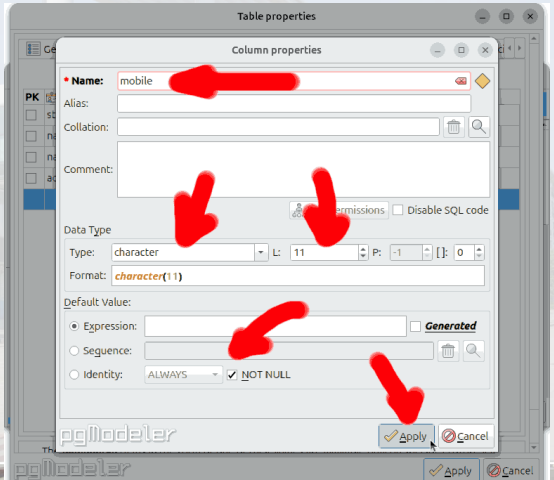
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint im Dialog.
- Wir klicken nochmal auf [Add Item] .



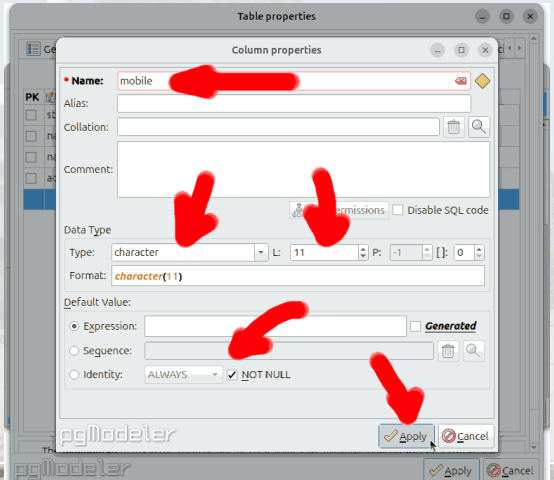
Logisches ERD erstellen mit PgModeler

- Wir fügen eine Spalte `mobile` für Mobiltelefonnummern ein.



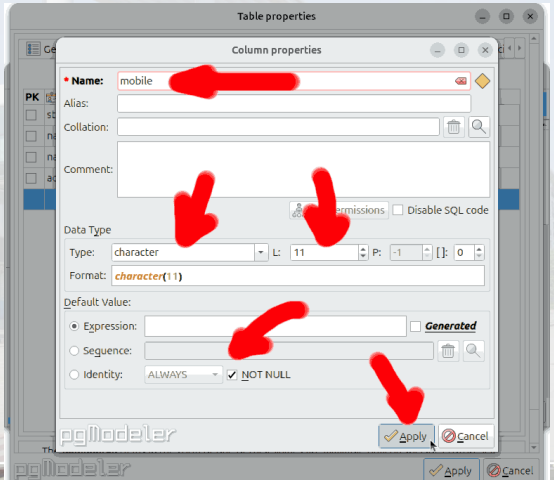
Logisches ERD erstellen mit PgModeler

- Wir fügen eine Spalte `mobile` für Mobiltelefonnummern ein.
- In China sind diese Zeichenketten (`character`) der festen Länge 11[121].



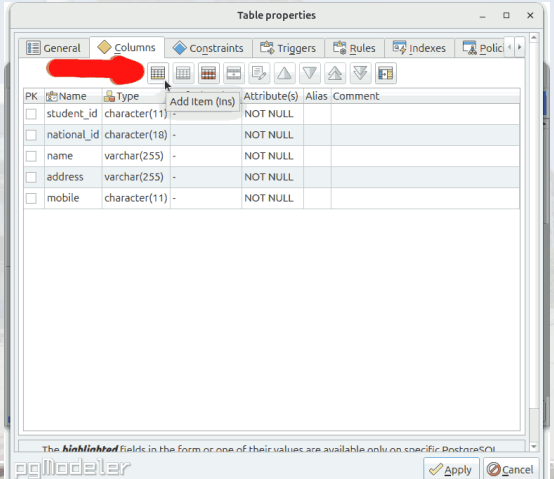
Logisches ERD erstellen mit PgModeler

- In China sind diese Zeichenketten (`character`) der festen Länge 11^{121}.
- Wir verlangen dass diese angegeben werden müssen (`NOT NULL`) und klicken Apply.



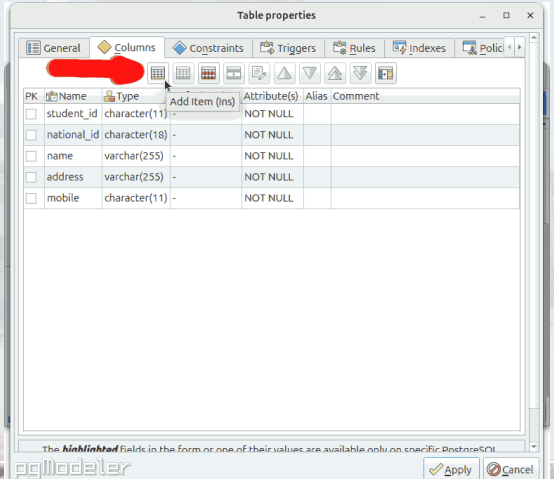
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint im Dialog.



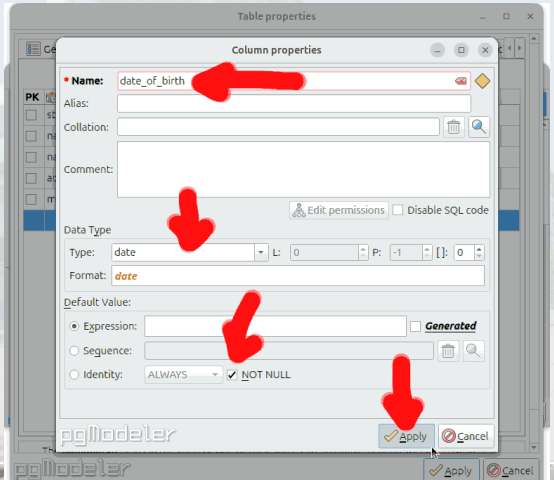
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint im Dialog.
- Wir klicken nochmal auf Add Item .



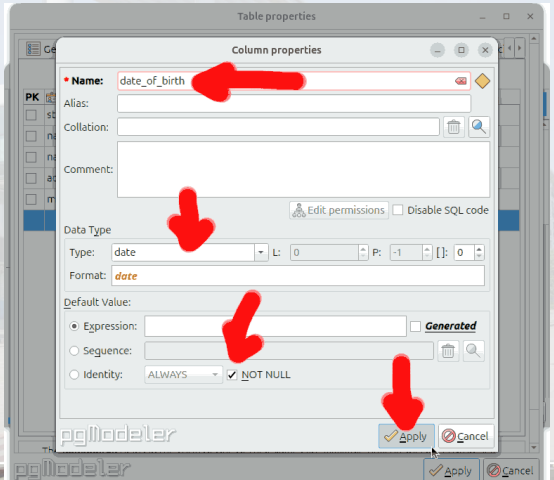
Logisches ERD erstellen mit PgModeler

- Zuletzt fügen wir eine Spalte für das Geburtsdatum, genannt `date_of_birth`, hinzu.



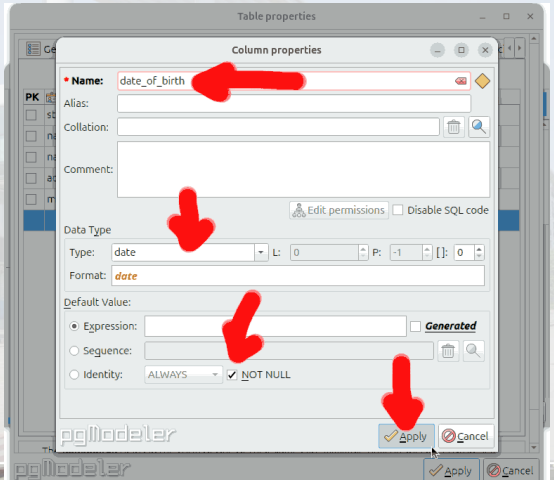
Logisches ERD erstellen mit PgModeler

- Zuletzt fügen wir eine Spalte für das Geburtsdatum, genannt `date_of_birth`, hinzu.
- Der Datentyp ist `date` und wir verlangen, dass es `NOT NULL` ist.



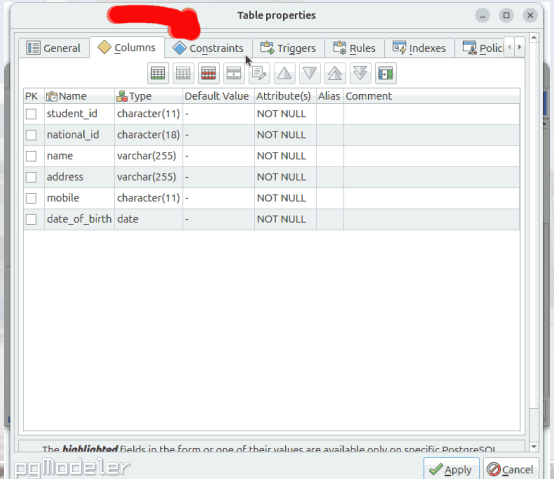
Logisches ERD erstellen mit PgModeler

- Der Datentyp ist `date` und wir verlangen, dass es `NOT NULL` ist.
- Wir klicken auf Apply.



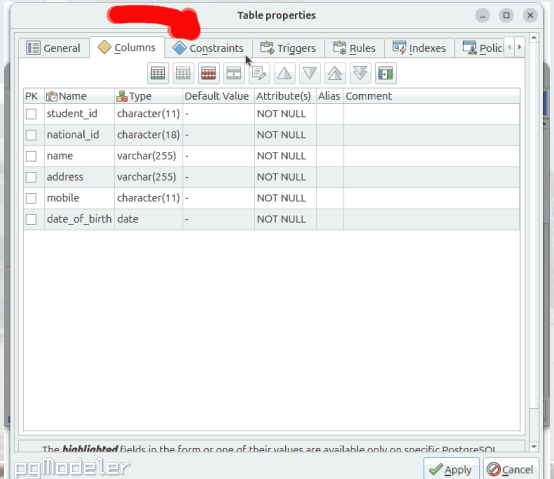
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint.



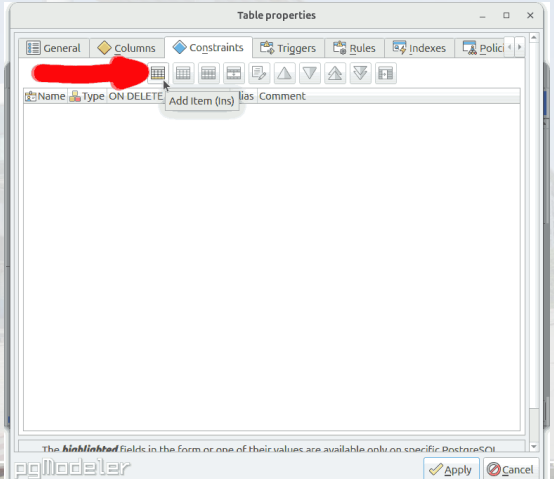
Logisches ERD erstellen mit PgModeler

- Die neue Spalte erscheint.
- Wir klicken auf den Register Constraints, denn wir wollen nun Einschränkungen für die Gültigkeit der Daten definieren.



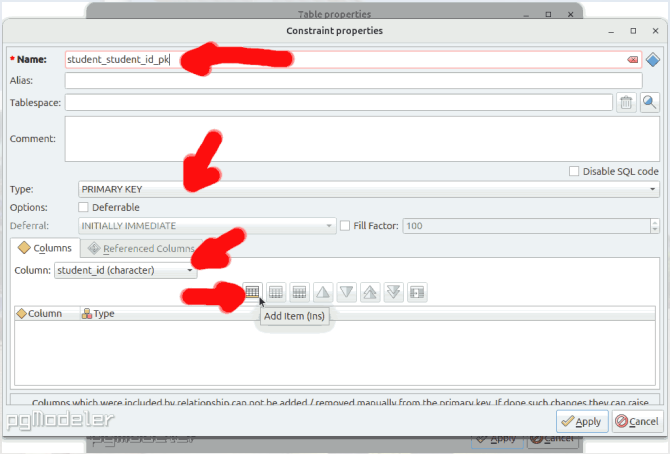
Logisches ERD erstellen mit PgModeler

- Im Register Constraints klicken wir auf Add Item .



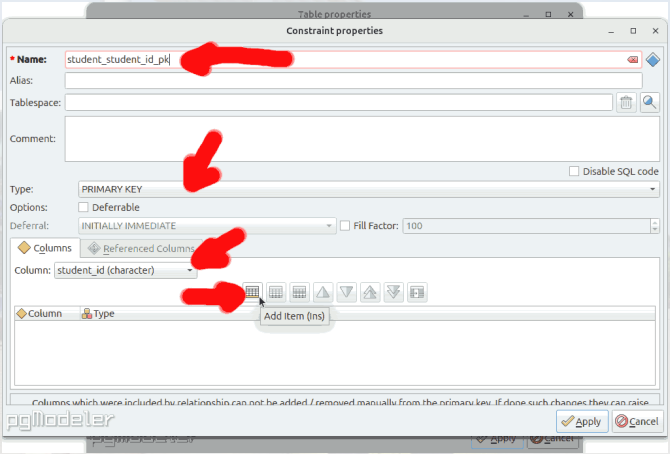
Logisches ERD erstellen mit PgModeler

- Als erste Einschränkung definieren wir, dass `student_id` der Primärschlüssel der Tabelle seien soll.



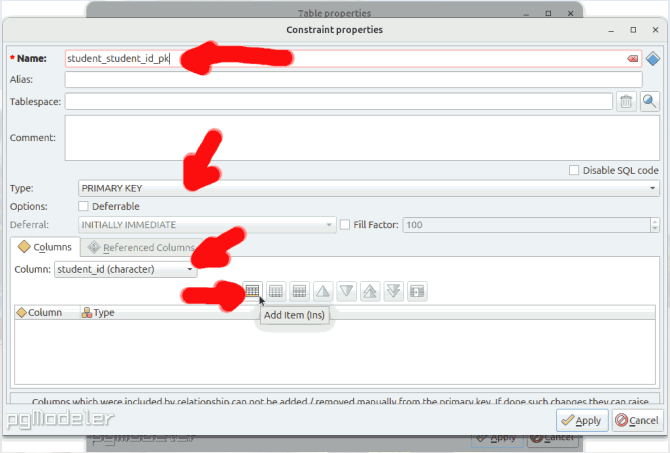
Logisches ERD erstellen mit PgModeler

- Als erste Einschränkung definieren wir, dass `student_id` der Primärschlüssel der Tabelle seien soll.
- Wir nennen diese Einschränkung `student_student_id_pk` und wählen `PRIMARY KEY` als Typ.



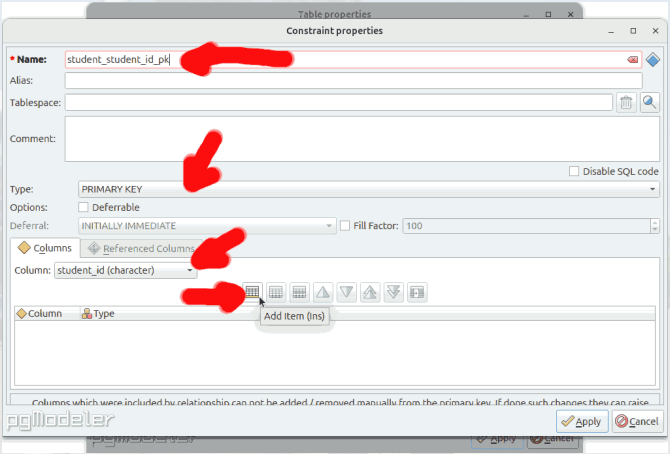
Logisches ERD erstellen mit PgModeler

- Wir nennen diese Einschränkung `student_student_id_pk` und wählen `PRIMARY KEY` als Typ.
- Ein Primärschlüssel-Constraint impliziert immer `NOT NULL` und `UNIQUE`.



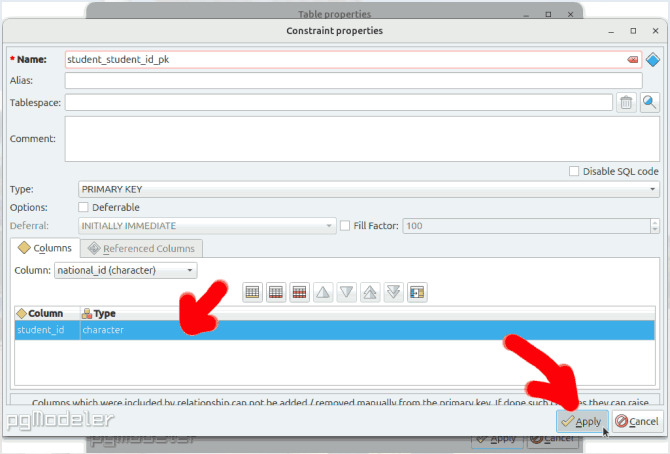
Logisches ERD erstellen mit PgModeler

- Ein Primärschlüssel-Constraint impliziert immer `NOT NULL` und `UNIQUE`.
- Wir wählen die Spalte `student_id` in der Drop-Down-Box Column und klicken auf Add Item .



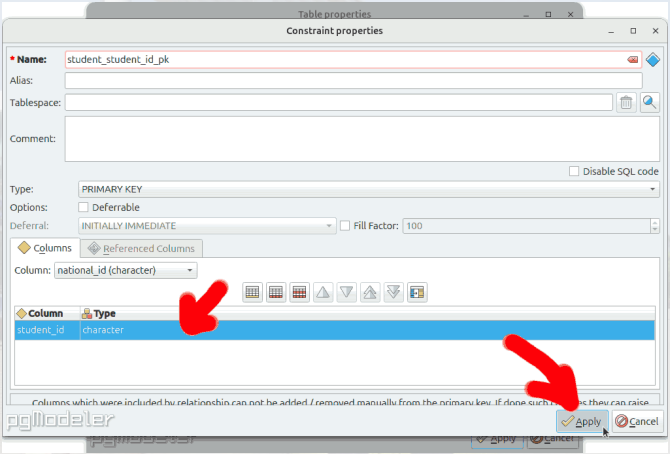
Logisches ERD erstellen mit PgModeler

- Die Spalte `student_id` erscheint in der Liste Columns.



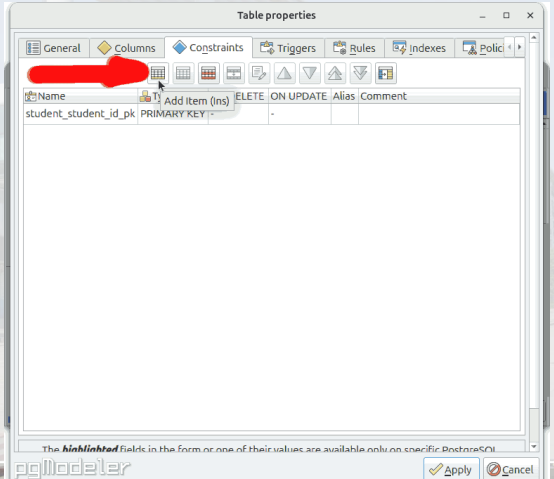
Logisches ERD erstellen mit PgModeler

- Die Spalte `student_id` erscheint in der Liste Columns.
- Wir klicken auf Apply.



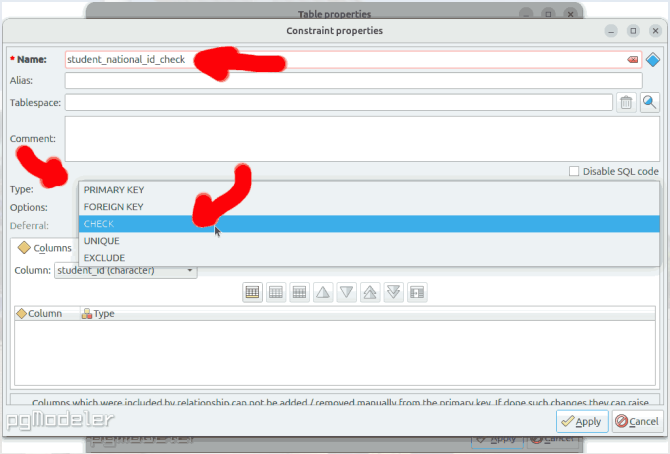
Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint und wir klicken auf Add Item .



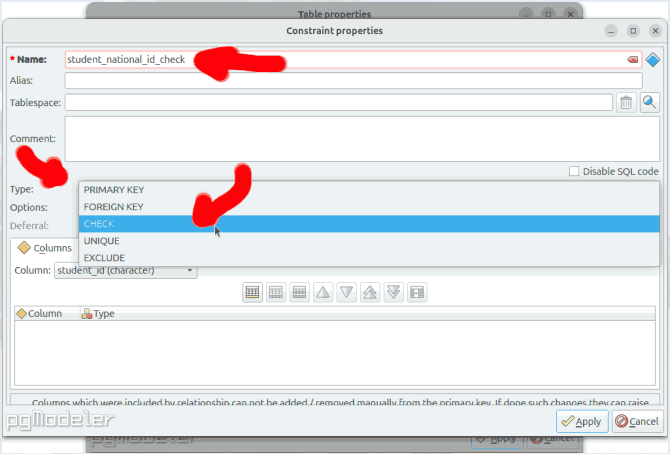
Logisches ERD erstellen mit PgModeler

- Wir wollen nun eine Einschränkung für korrekte Ausweisnummern erstellen.



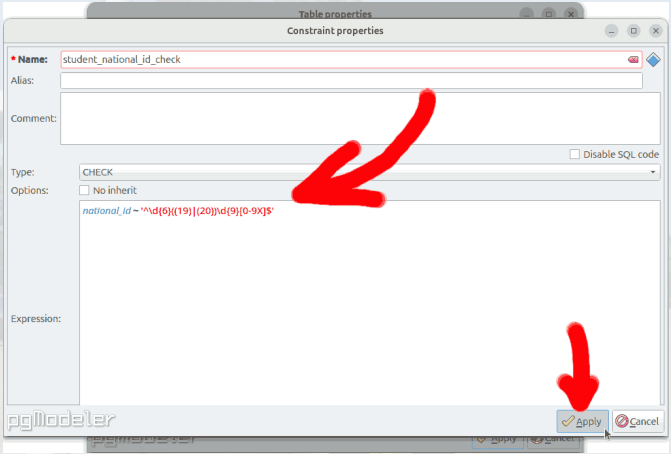
Logisches ERD erstellen mit PgModeler

- Wir wollen nun eine Einschränkung für korrekte Ausweisnummern erstellen.
- Wir nennen sie `student_national_id_check` und wählen `CHECK` in der Drop-Down-Box Type.



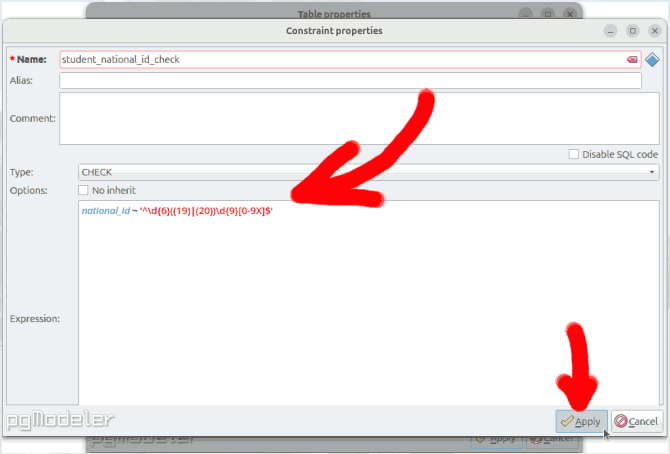
Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.



Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.



Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `national_id ~ xxx` bedeutet, dass der Wert von `national_id` zu der regex `xxx` passen muss.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `national_id ~ xxx` bedeutet, dass der Wert von `national_id` zu der regex `xxx` passen muss.
- In der regex, steht `^` für den Anfang des Textes.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `national_id ~ xxx` bedeutet, dass der Wert von `national_id` zu der regex `xxx` passen muss.
- In der regex, steht `^` für den Anfang des Textes.
- `^\d{6}` bedeutet, das 6 Ziffern sofort auf `^` folgen müssen, also genau am Start des Textes stehen müssen.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `national_id ~ xxx` bedeutet, dass der Wert von `national_id` zu der regex `xxx` passen muss.
- In der regex, steht `^` für den Anfang des Textes.
- `^\d{6}` bedeutet, das 6 Ziffern sofort auf `^` folgen müssen, also genau am Start des Textes stehen müssen.
- Dann kommt `((19)|(20))`, was bedeutet, dass die nächsten beiden Zeichen entweder „19“ oder „20“ seien müssen.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `national_id ~ xxx` bedeutet, dass der Wert von `national_id` zu der regex `xxx` passen muss.
- In der regex, steht `^` für den Anfang des Textes.
- `^\d{6}` bedeutet, das 6 Ziffern sofort auf `^` folgen müssen, also genau am Start des Textes stehen müssen.
- Dann kommt `((19)|(20))`, was bedeutet, dass die nächsten beiden Zeichen entweder „19“ oder „20“ seien müssen.
- Diese beiden Zeichen sind die ersten beiden Ziffern des Geburtsjahrs.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `national_id ~ xxx` bedeutet, dass der Wert von `national_id` zu der regex `xxx` passen muss.
- In der regex, steht `^` für den Anfang des Textes.
- `^\d{6}` bedeutet, das 6 Ziffern sofort auf `^` folgen müssen, also genau am Start des Textes stehen müssen.
- Dann kommt `((19)|(20))`, was bedeutet, dass die nächsten beiden Zeichen entweder „19“ oder „20“ seien müssen.
- Diese beiden Zeichen sind die ersten beiden Ziffern des Geburtsjahrs.
- Wir lassen also keine DOBs vor 1900 oder nach 2099 zu.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `national_id ~ xxx` bedeutet, dass der Wert von `national_id` zu der regex `xxx` passen muss.
- In der regex, steht `^` für den Anfang des Textes.
- `^\d{6}` bedeutet, das 6 Ziffern sofort auf `^` folgen müssen, also genau am Start des Textes stehen müssen.
- Dann kommt `((19)|(20))`, was bedeutet, dass die nächsten beiden Zeichen entweder „19“ oder „20“ seien müssen.
- Diese beiden Zeichen sind die ersten beiden Ziffern des Geburtsjahrs.
- Wir lassen also keine DOBs vor 1900 oder nach 2099 zu.
- Wir verlangen dann mit `\d{9}` dass neun Ziffern folgen.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `national_id ~ xxx` bedeutet, dass der Wert von `national_id` zu der regex `xxx` passen muss.
- In der regex, steht `^` für den Anfang des Textes.
- `^\d{6}` bedeutet, das 6 Ziffern sofort auf `^` folgen müssen, also genau am Start des Textes stehen müssen.
- Dann kommt `((19)|(20))`, was bedeutet, dass die nächsten beiden Zeichen entweder „19“ oder „20“ seien müssen.
- Diese beiden Zeichen sind die ersten beiden Ziffern des Geburtsjahrs.
- Wir lassen also keine DOBs vor 1900 oder nach 2099 zu.
- Wir verlangen dann mit `\d{9}` dass neun Ziffern folgen.
- OK, wir könnten auch prüfen das die nächsten zwei Ziffern gültige Monate und die zwei Ziffern danach gültige Tage sind.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- In der regex, steht `^` für den Anfang des Textes.
- `^\d{6}` bedeutet, das 6 Ziffern sofort auf `^` folgen müssen, also genau am Start des Textes stehen müssen.
- Dann kommt `((19)|(20))`, was bedeutet, dass die nächsten beiden Zeichen entweder „19“ oder „20“ seien müssen.
- Diese beiden Zeichen sind die ersten beiden Ziffern des Geburtsjahrs.
- Wir lassen also keine DOBs vor 1900 oder nach 2099 zu.
- Wir verlangen dann mit `\d{9}` dass neun Ziffern folgen.
- OK, wir könnten auch prüfen das die nächsten zwei Ziffern gültige Monate und die zwei Ziffern danach gültige Tage sind.
- In einer echten Anwendung würde ich das tun, hier lassen wir das als Übung für die interessierten Studenten.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- `^\d{6}` bedeutet, das 6 Ziffern sofort auf `^` folgen müssen, also genau am Start des Textes stehen müssen.
- Dann kommt `((19)|(20))`, was bedeutet, dass die nächsten beiden Zeichen entweder „19“ oder „20“ seien müssen.
- Diese beiden Zeichen sind die ersten beiden Ziffern des Geburtsjahrs.
- Wir lassen also keine DOBs vor 1900 oder nach 2099 zu.
- Wir verlangen dann mit `\d{9}` dass neun Ziffern folgen.
- OK, wir könnten auch prüfen das die nächsten zwei Ziffern gültige Monate und die zwei Ziffern danach gültige Tage sind.
- In einer echten Anwendung würde ich das tun, hier lassen wir das als Übung für die interessierten Studenten.
- So oder so, wir haben jetzt $6 + 2 + 9 = 17$ Ziffern.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- Dann kommt `((19)|(20))`, was bedeutet, dass die nächsten beiden Zeichen entweder „19“ oder „20“ seien müssen.
- Diese beiden Zeichen sind die ersten beiden Ziffern des Geburtsjahrs.
- Wir lassen also keine DOBs vor 1900 oder nach 2099 zu.
- Wir verlangen dann mit `\d{9}` dass neun Ziffern folgen.
- OK, wir könnten auch prüfen das die nächsten zwei Ziffern gültige Monate und die zwei Ziffern danach gültige Tage sind.
- In einer echten Anwendung würde ich das tun, hier lassen wir das als Übung für die interessierten Studenten.
- So oder so, wir haben jetzt $6 + 2 + 9 = 17$ Ziffern.
- Nur das letzte Zeichen – eine Prüfsumme – ist übrig, welche entweder 0 to 9 oder X seien kann.

Logisches ERD erstellen mit PgModeler

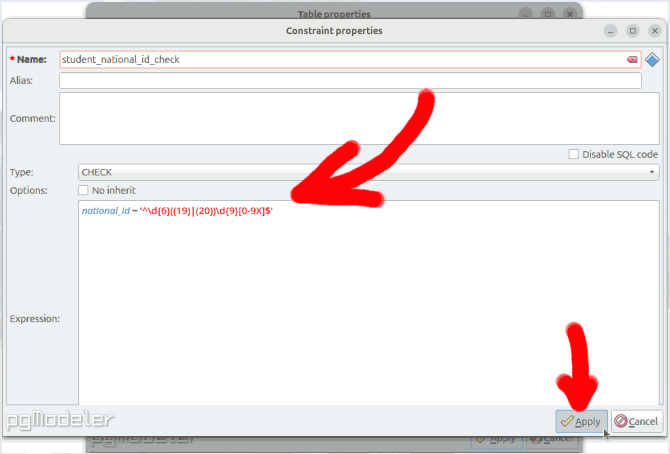
- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- Diese beiden Zeichen sind die ersten beiden Ziffern des Geburtsjahrs.
- Wir lassen also keine DOBs vor 1900 oder nach 2099 zu.
- Wir verlangen dann mit `\d{9}` dass neun Ziffern folgen.
- OK, wir könnten auch prüfen das die nächsten zwei Ziffern gültige Monate und die zwei Ziffern danach gültige Tage sind.
- In einer echten Anwendung würde ich das tun, hier lassen wir das als Übung für die interessierten Studenten.
- So oder so, wir haben jetzt $6 + 2 + 9 = 17$ Ziffern.
- Nur das letzte Zeichen – eine Prüfsumme – ist übrig, welche entweder 0 to 9 oder X seien kann.
- Das wird mit `[0-9X]` ausgedrückt.

Logisches ERD erstellen mit PgModeler

- `CHECK`-Einschränkungen werden in SQL im Expression definiert.
- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- Wir lassen also keine DOBs vor 1900 oder nach 2099 zu.
- Wir verlangen dann mit `\d{9}` dass neun Ziffern folgen.
- OK, wir könnten auch prüfen das die nächsten zwei Ziffern gültige Monate und die zwei Ziffern danach gültige Tage sind.
- In einer echten Anwendung würde ich das tun, hier lassen wir das als Übung für die interessierten Studenten.
- So oder so, wir haben jetzt $6 + 2 + 9 = 17$ Ziffern.
- Nur das letzte Zeichen – eine Prüfsumme – ist übrig, welche entweder 0 to 9 oder X seien kann.
- Das wird mit `[0-9X]` ausgedrückt.
- Das folgende `$` markiert das Ende der Zeichenkette, welche also direkt nach der Prüfsumme enden muss.

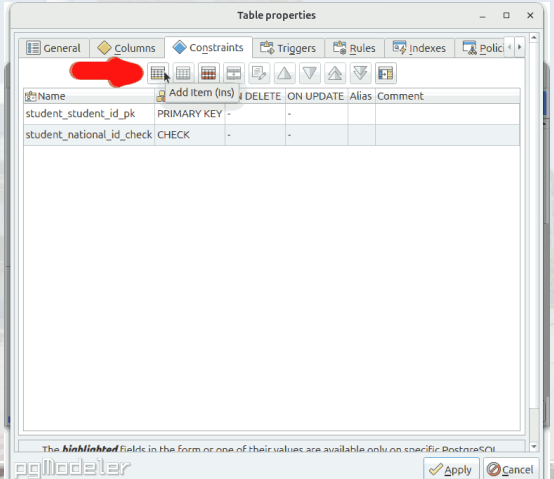
Logisches ERD erstellen mit PgModeler

- Um das Feld `national_id` zu validieren, spezifizieren wir den Regulären Ausdruck (regex) `national_id~'^\d{6}((19)|(20))\d{9}[0-9X]$'`.
- Wir klicken auf Apply.



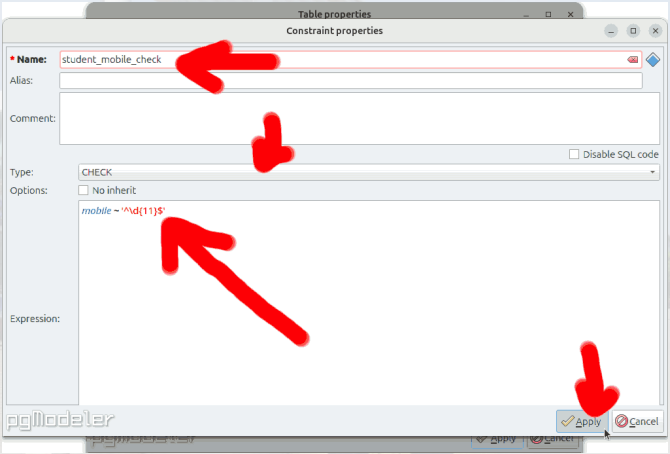
Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint und wir klicken auf Add Item .



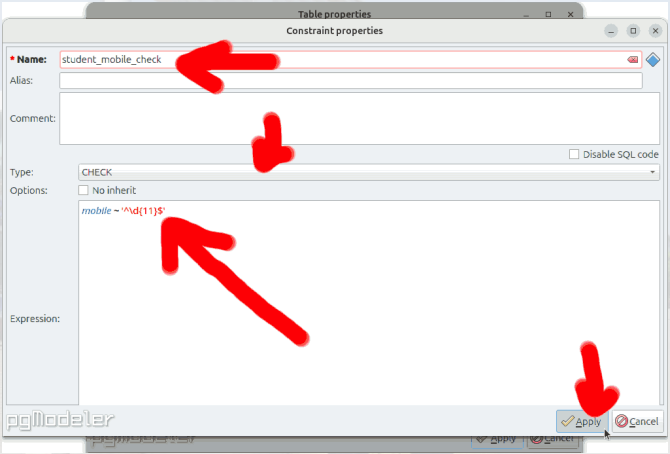
Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `mobile`.



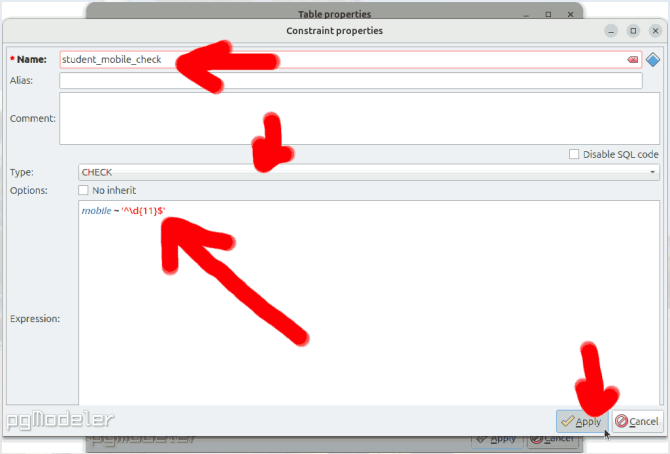
Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `mobile`.
- Als Ausdruck wählen wir `mobile ~ '^\d{11}$'`, was einen Text aus genau 11 Ziffern erfordert.



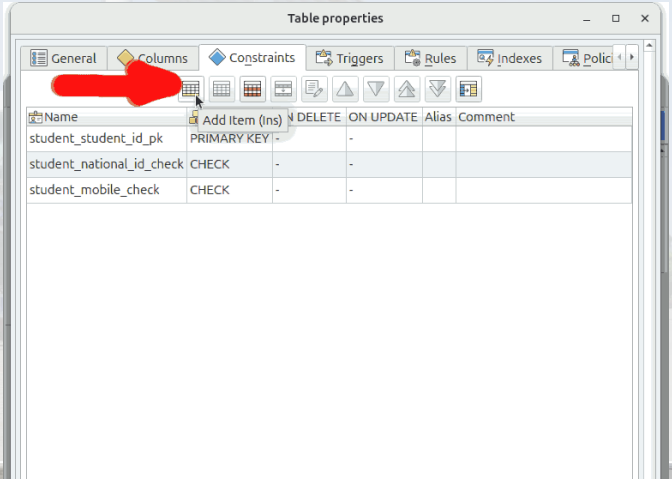
Logisches ERD erstellen mit PgModeler

- Als Ausdruck wählen wir `mobile ~ '^\d{11}$'`, was einen Text aus genau 11 Ziffern erfordert.
- Wir klicken auf Apply.



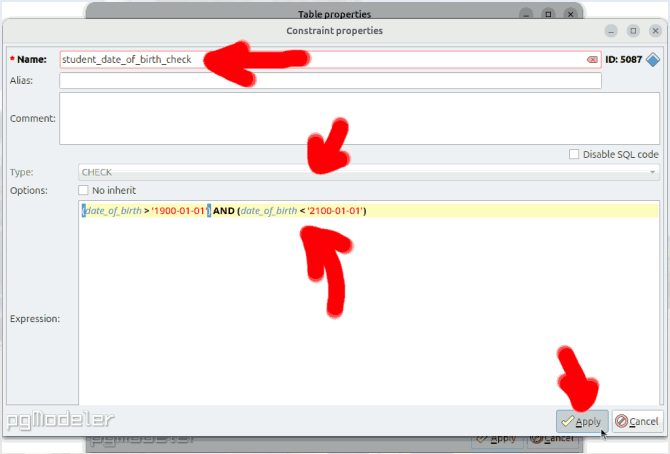
Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint und wir klicken auf Add Item .



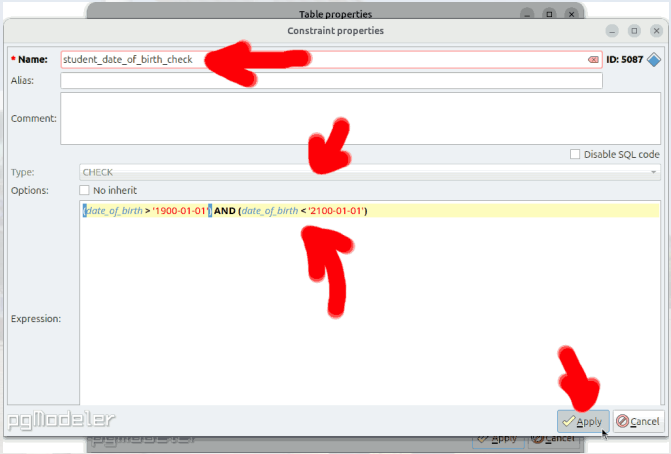
Logisches ERD erstellen mit PgModeler

- Wir definieren die `CHECK`-Einschränkung für das DOB und nennen sie `student_date_of_birth_check`.



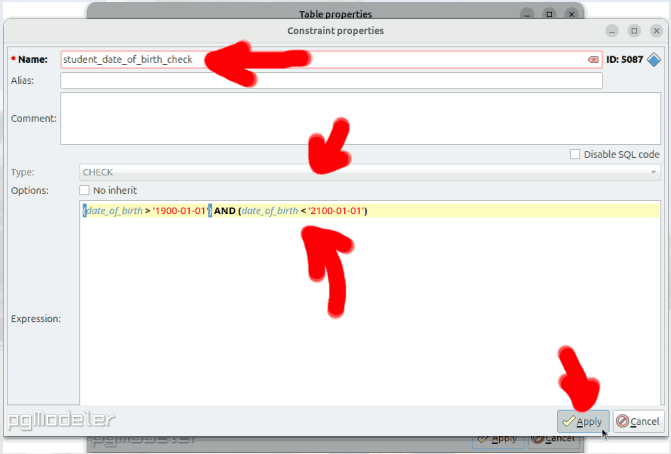
Logisches ERD erstellen mit PgModeler

- Eine Bedingung ist `date_of_birth > '1900-01-01'`, also Studenten können nicht vor 1900 geboren sein.



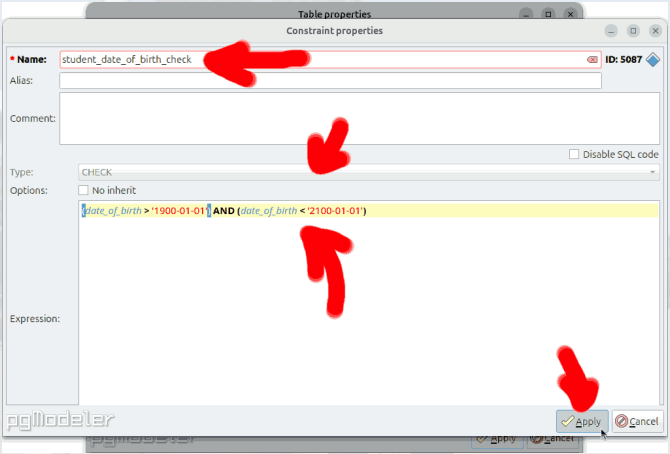
Logisches ERD erstellen mit PgModeler

- Eine zweite Bedingung ist `date_of_birth < '2100-01-01'`, was Studenten aus dem 22. Jahrhundert verbietet.



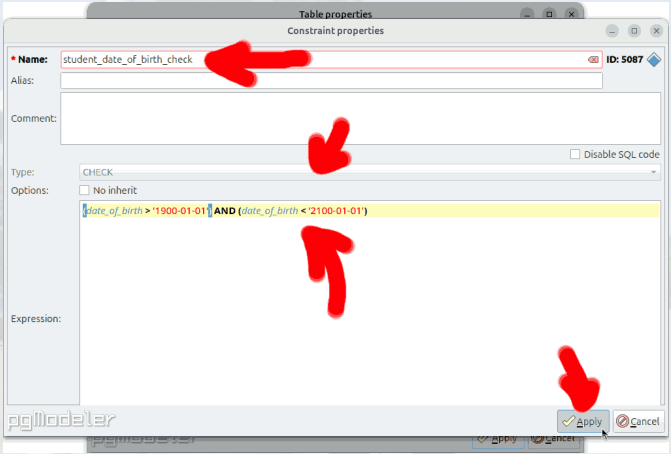
Logisches ERD erstellen mit PgModeler

- Eine zweite Bedingung ist `date_of_birth < '2100-01-01'`, was Studenten aus dem 22. Jahrhundert verbietet.
- Wir verbinden beide Bedingungen mit `AND`.



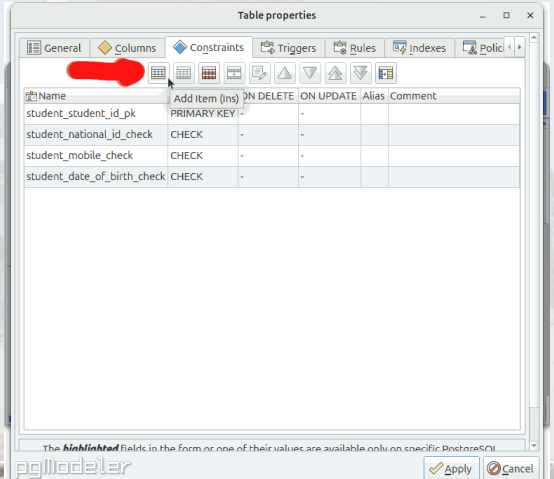
Logisches ERD erstellen mit PgModeler

- Wir verbinden beide Bedingungen mit `AND`.
- Wir klicken auf Apply.



Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint.



Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint.
- Wir *könnten* nun eine Einschränkung definieren, die prüft, ob die acht Ziffern des Geburtsdatums in der Ausweisnummer mit dem Geburtsdatum hier übereinstimmen.

Logisches ERD erstellen mit PgModeler

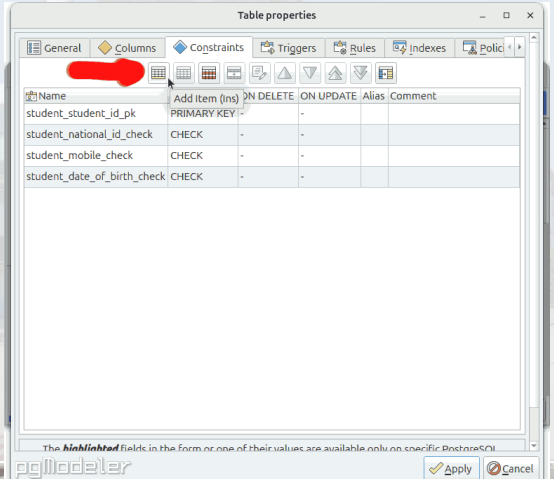
- Die neue Einschränkung erscheint.
- Wir *könnten* nun eine Einschränkung definieren, die prüft, ob die acht Ziffern des Geburtsdatums in der Ausweisnummer mit dem Geburtsdatum hier übereinstimmen.
- Das wäre ein ziemlich cooler Schutz gegen Tippfehler.

Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint.
- Wir *könnten* nun eine Einschränkung definieren, die prüft, ob die acht Ziffern des Geburtsdatums in der Ausweisnummer mit dem Geburtsdatum hier übereinstimmen.
- Das wäre ein ziemlich cooler Schutz gegen Tippfehler.
- Vielleicht wäre es eine gute Übung, das auszuprobieren ... hier machen wir das nicht.

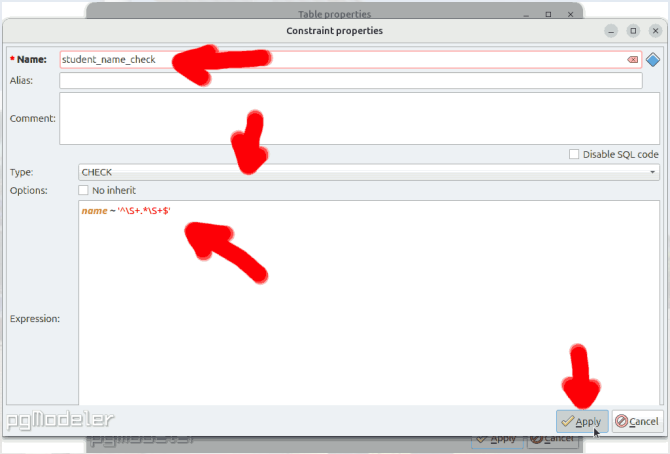
Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint.
- Egal. Wir klicken auf Add Item .



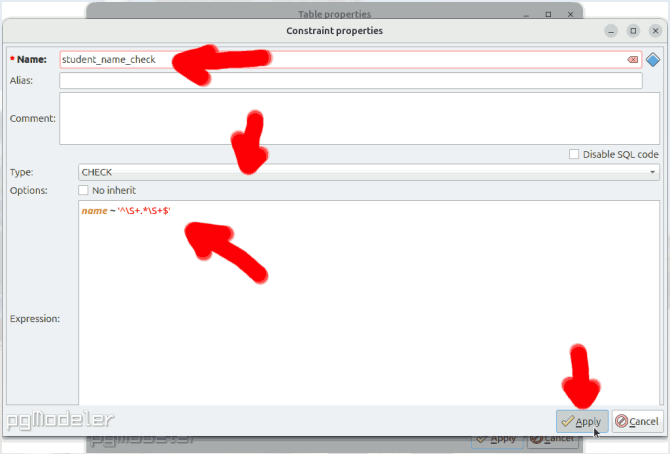
Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.



Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.



Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- Hier ist `^` wieder der Anfang des Strings.

Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- Hier ist `^` wieder der Anfang des Strings.
- `\S` steht für „druckbares Zeichen“, also kein Leerzeichen oder Tabulator.

Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- Hier ist `^` wieder der Anfang des Strings.
- `\S` steht für „druckbares Zeichen“, also kein Leerzeichen oder Tabulator.
- `+` bedeutet „mindestens eins“.

Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- Hier ist `^` wieder der Anfang des Strings.
- `\S` steht für „druckbares Zeichen“, also kein Leerzeichen oder Tabulator.
- `+` bedeutet „mindestens eins“.
- `^\S+` bedeutet also, dass die Zeichenkette mit mindestens einem druckbaren Zeichen anfangen muss.

Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- Hier ist `^` wieder der Anfang des Strings.
- `\S` steht für „druckbares Zeichen“, also kein Leerzeichen oder Tabulator.
- `+` bedeutet „mindestens eins“.
- `^\S+` bedeutet also, dass die Zeichenkette mit mindestens einem druckbaren Zeichen anfangen muss.
- `.` steht für ein beliebiges Zeichen.

Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- Hier ist `^` wieder der Anfang des Strings.
- `\S` steht für „druckbares Zeichen“, also kein Leerzeichen oder Tabulator.
- `+` bedeutet „mindestens eins“.
- `^\S+` bedeutet also, dass die Zeichenkette mit mindestens einem druckbaren Zeichen anfangen muss.
- `.` steht für ein beliebiges Zeichen.
- `*` bedeutet „beliebig viele“.

Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- Hier ist `^` wieder der Anfang des Strings.
- `\S` steht für „druckbares Zeichen“, also kein Leerzeichen oder Tabulator.
- `+` bedeutet „mindestens eins“.
- `^\S+` bedeutet also, dass die Zeichenkette mit mindestens einem druckbaren Zeichen anfangen muss.
- `.` steht für ein beliebiges Zeichen.
- `*` bedeutet „beliebig viele“.
- `.*` bedeutet also, dass beliebig viele beliebige Zeichen auf die druckbaren Zeichen am Anfang folgen können.

Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- Hier ist `^` wieder der Anfang des Strings.
- `\S` steht für „druckbares Zeichen“, also kein Leerzeichen oder Tabulator.
- `+` bedeutet „mindestens eins“.
- `^\S+` bedeutet also, dass die Zeichenkette mit mindestens einem druckbaren Zeichen anfangen muss.
- `.` steht für ein beliebiges Zeichen.
- `*` bedeutet „beliebig viele“.
- `.*` bedeutet also, dass beliebig viele beliebige Zeichen auf die druckbaren Zeichen am Anfang folgen können.
- Dann kommt `\S+$`, wobei `$` für das Ende der Zeichenkette steht.

Logisches ERD erstellen mit PgModeler

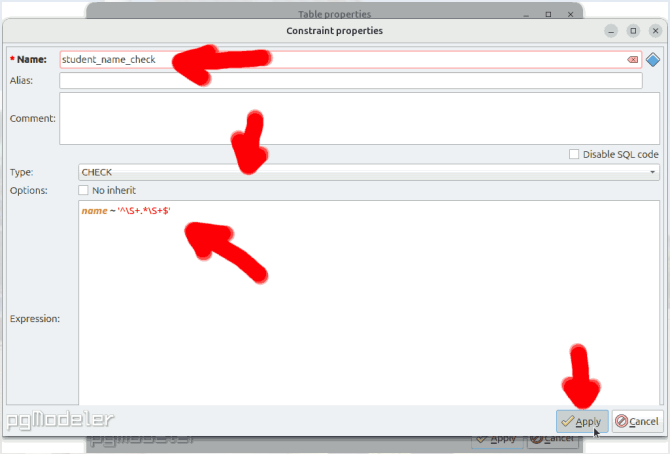
- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- `\S` steht für „druckbares Zeichen“, also kein Leerzeichen oder Tabulator.
- `+` bedeutet „mindestens eins“.
- `^\S+` bedeutet also, dass die Zeichenkette mit mindestens einem druckbaren Zeichen anfangen muss.
- `.` steht für ein beliebiges Zeichen.
- `*` bedeutet „beliebig viele“.
- `.*` bedeutet also, dass beliebig viele beliebige Zeichen auf die druckbaren Zeichen am Anfang folgen können.
- Dann kommt `\S+$`, wobei `$` für das Ende der Zeichenkette steht.
- `\S+$` bedeutet also, dass die Zeichenkette mit mindestens einem durckbaren Zeichen enden muss.

Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name` und nennen sie `student_name_check`.
- Wir verwenden die regex `name ~ '^\S+.*\S+$'`.
- `+` bedeutet „mindestens eins“.
- `^\S+` bedeutet also, dass die Zeichenkette mit mindestens einem druckbaren Zeichen anfangen muss.
- `.` steht für ein beliebiges Zeichen.
- `*` bedeutet „beliebig viele“.
- `.*` bedeutet also, dass beliebig viele beliebige Zeichen auf die druckbaren Zeichen am Anfang folgen können.
- Dann kommt `\S+$`, wobei `$` für das Ende der Zeichenkette steht.
- `\S+$` bedeutet also, dass die Zeichenkette mit mindestens einem durckbaren Zeichen enden muss.
- Jeder Name muss also mit druckbaren Zeichen anfangen und enden.

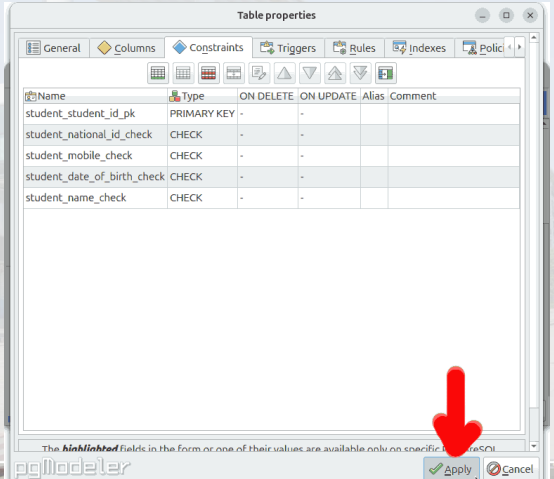
Logisches ERD erstellen mit PgModeler

- Wir erstellen eine `CHECK`-Einschränkung für die Spalte `name`.
- Wir klicken auf Apply.



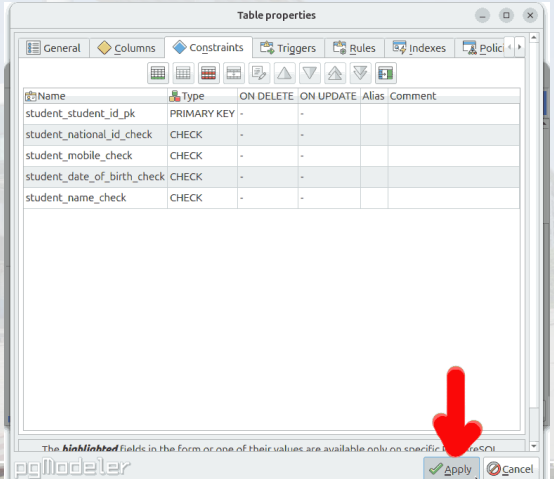
Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint.



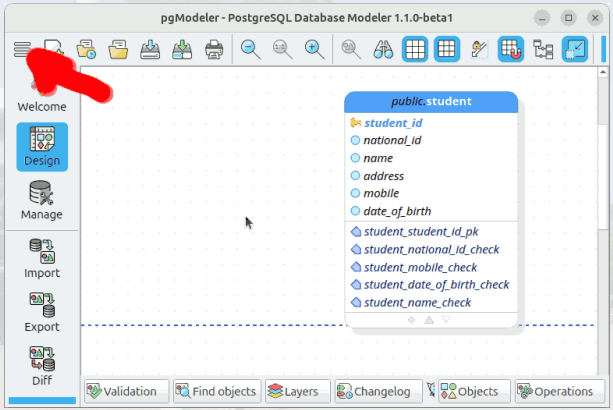
Logisches ERD erstellen mit PgModeler

- Die neue Einschränkung erscheint.
- Wir find soweit fertig und erstellen die Tabelle durch Klick auf Apply.



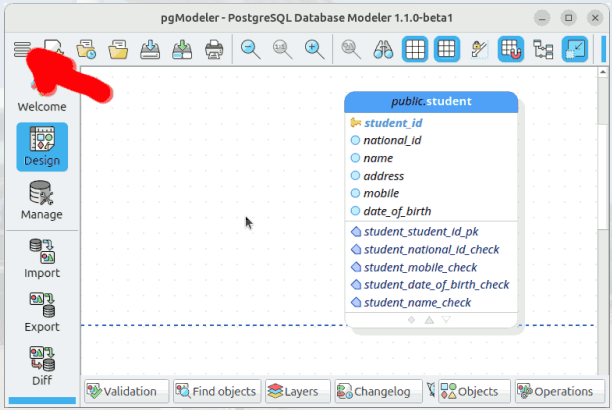
Logisches ERD erstellen mit PgModeler

- Eine neue Tabelle erscheint in unserem ERD fast in unserer kompakten Syntax.



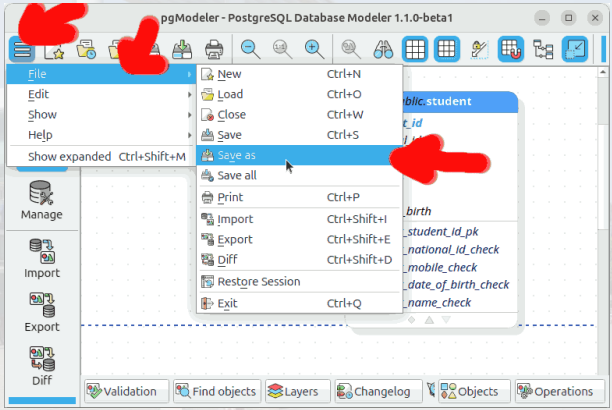
Logisches ERD erstellen mit PgModeler

- Eine neue Tabelle erscheint in unserem ERD fast in unserer kompakten Syntax.
- Wir klicken auf das Hauptmenü ≡.



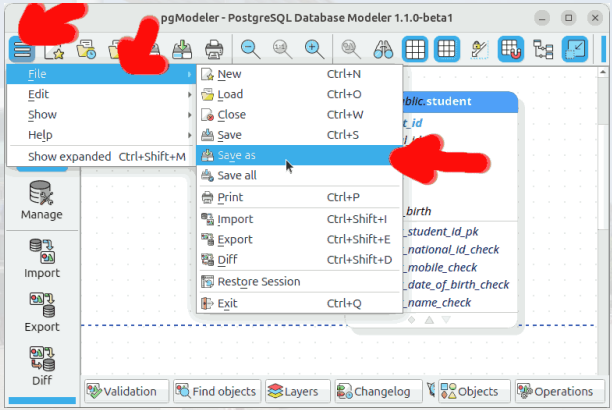
Logisches ERD erstellen mit PgModeler

- Wir wollen das Modell nun in eine Datei speichern.



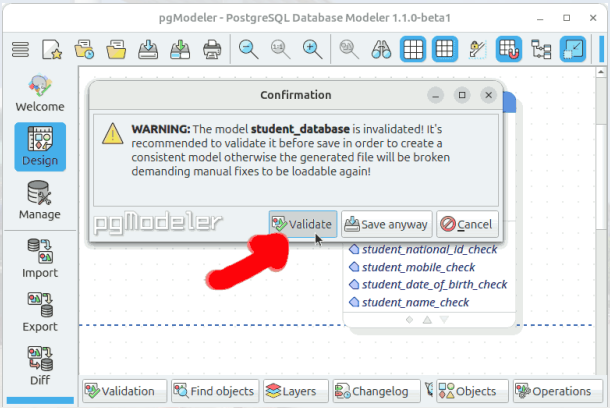
Logisches ERD erstellen mit PgModeler

- Wir wollen das Modell nun in eine Datei speichern.
- Wir klicken auf ≡ > File > Save as.



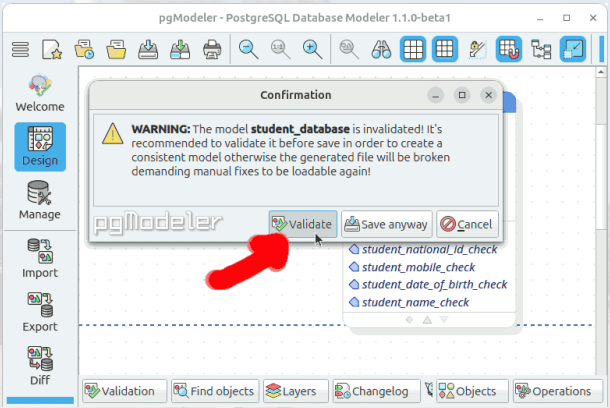
Logisches ERD erstellen mit PgModeler

- Weil das Modell neu und ungeprüft ist, werden wir gefragt, es zu validieren.



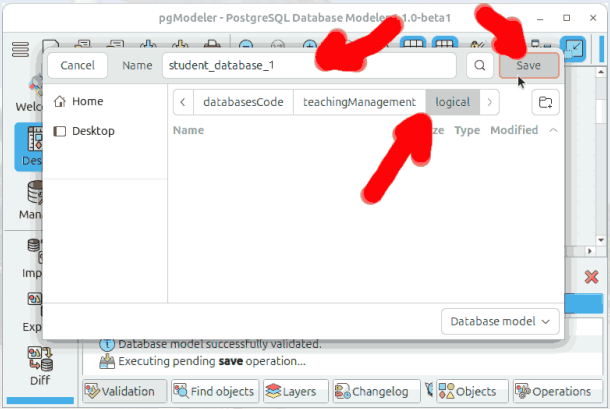
Logisches ERD erstellen mit PgModeler

- Weil das Modell neu und ungeprüft ist, werden wir gefragt, es zu validieren.
- Wir klicken auf Validate.



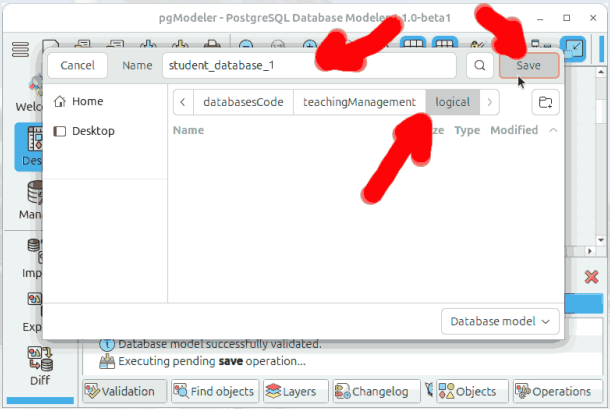
Logisches ERD erstellen mit PgModeler

- Wir können nun einen Dateiname und ein Verzeichnis auswählen, wo das Modell gespeichert werden soll.



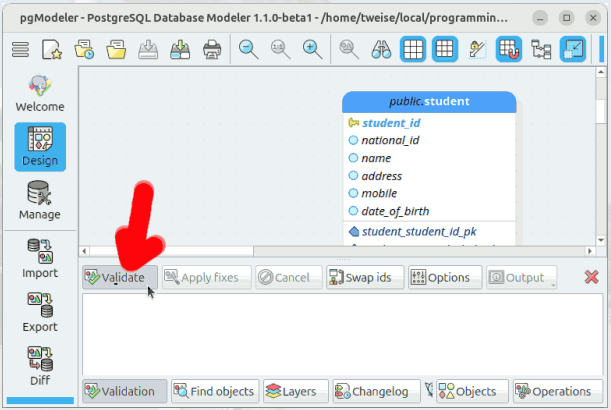
Logisches ERD erstellen mit PgModeler

- Wir können nun einen Dateiname und ein Verzeichnis auswählen, wo das Modell gespeichert werden soll.
- Wir wählen den Name `student_database_1` und klicken auf Save.



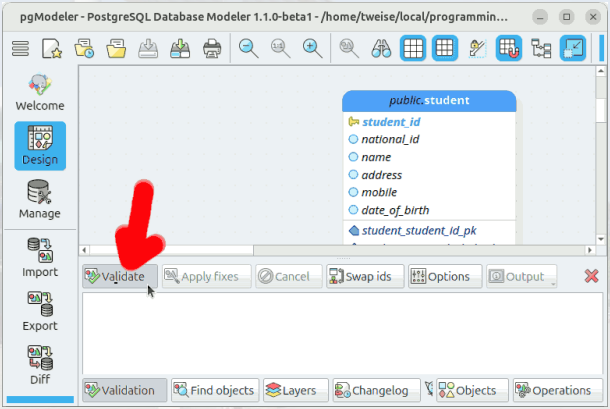
Logisches ERD erstellen mit PgModeler

- Wir sind wieder im Hauptfenster.



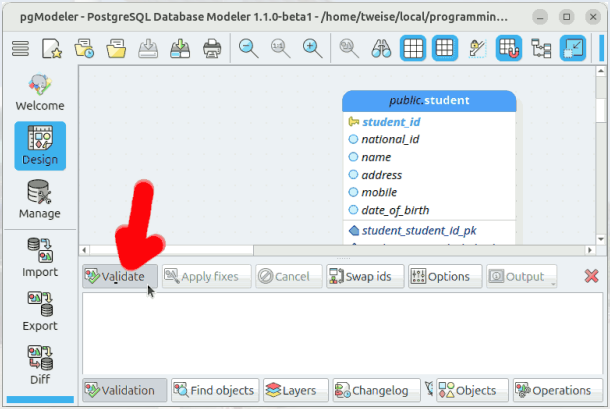
Logisches ERD erstellen mit PgModeler

- Wir sind wieder im Hauptfenster.
- Wir sehen einen Balken mit Knöpfen, einer ist Validate.



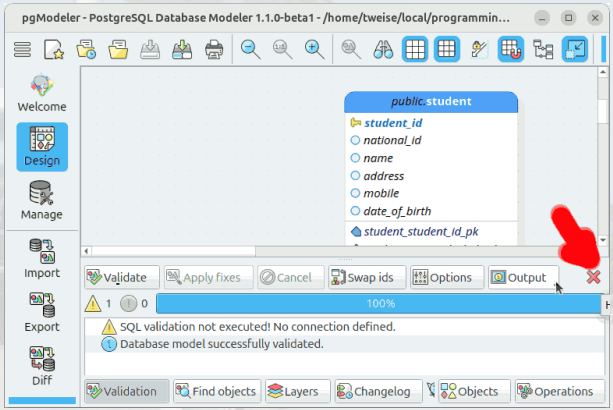
Logisches ERD erstellen mit PgModeler

- Wir sind wieder im Hauptfenster.
- Wir sehen einen Balken mit Knöpfen, einer ist Validate.
- Wir klicken darauf.



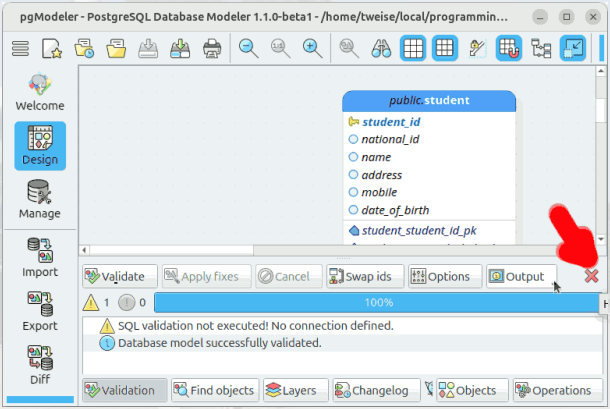
Logisches ERD erstellen mit PgModeler

- Unser Modell wird validiert.



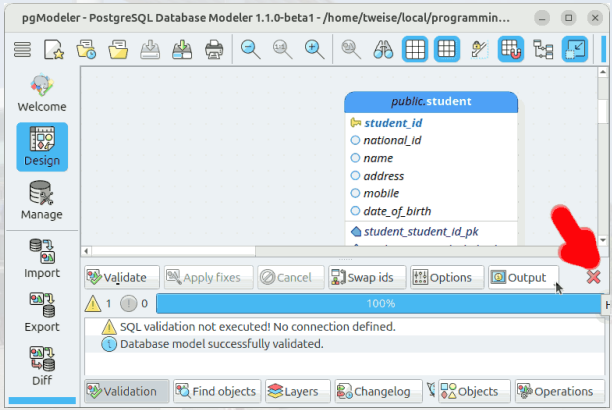
Logisches ERD erstellen mit PgModeler

- Unser Modell wird validiert.
- Es ist OK.



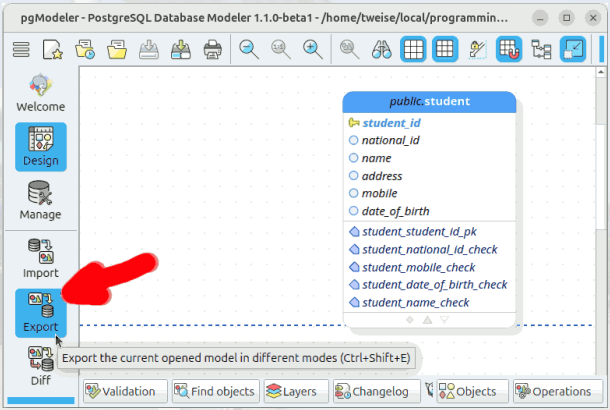
Logisches ERD erstellen mit PgModeler

- Unser Modell wird validiert.
- Es ist OK.
- Wir schließen das Log-Fenster.



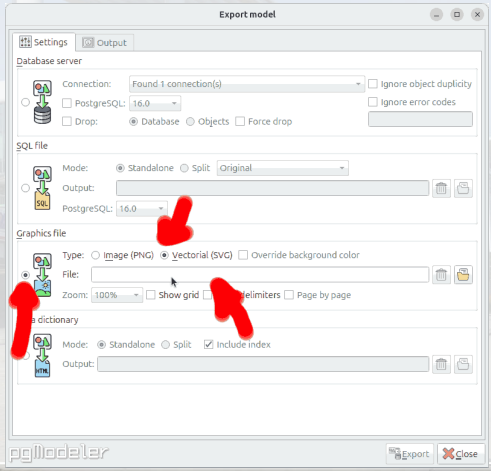
Logisches ERD erstellen mit PgModeler

- Wir wollen das Modell nun exportieren und klicken auf Export.



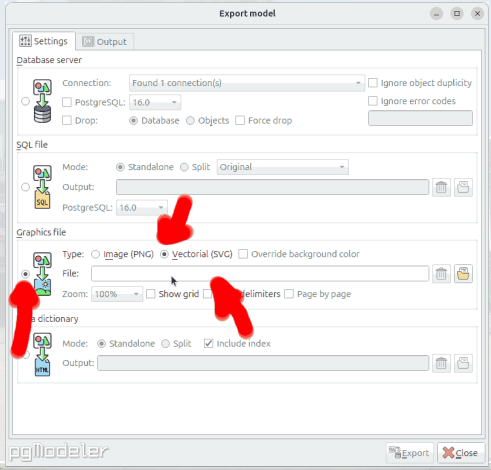
Logisches ERD erstellen mit PgModeler

- Wir wollen es als Grafik speichern.



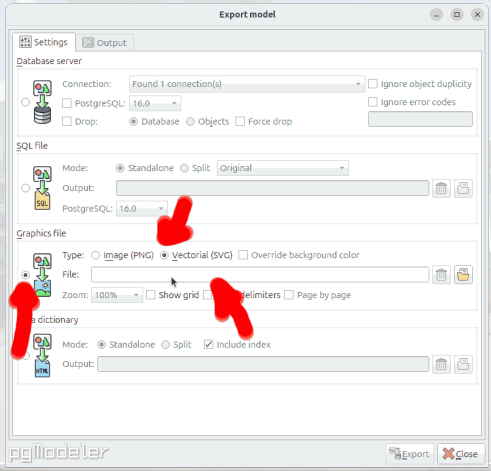
Logisches ERD erstellen mit PgModeler

- Wir wollen es als Grafik speichern.
- Wir klicken auf Graphics file und wählen Vectorial (SVG), wodurch unser Modell im SVG-Format gespeichert werden wird.



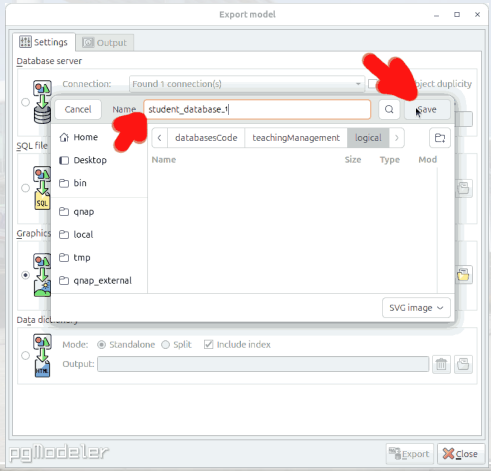
Logisches ERD erstellen mit PgModeler

- Wir klicken auf Graphics file und wählen Vectorial (SVG), wodurch unser Modell im SVG-Format gespeichert werden wird.
- Dann klicken wir auf den File Balken.



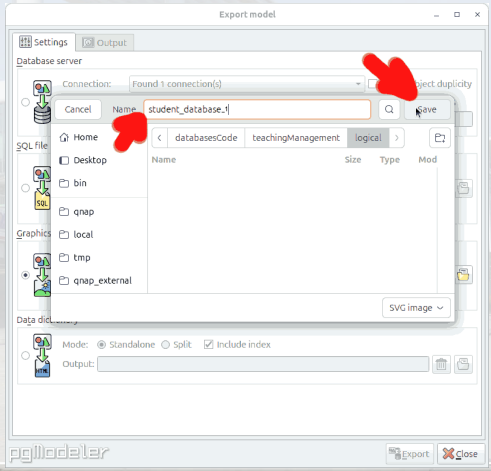
Logisches ERD erstellen mit PgModeler

- Wir wählen wieder einen Dateiname und bleiben bei `student_database_1`.



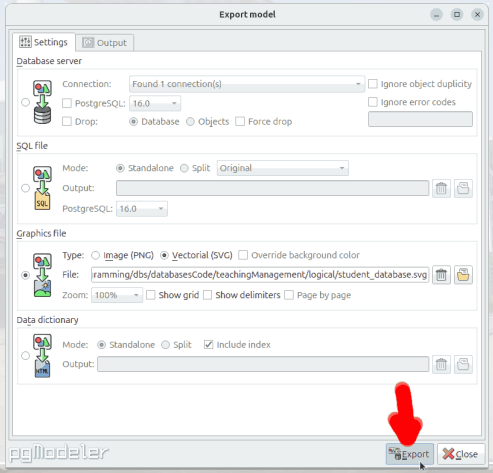
Logisches ERD erstellen mit PgModeler

- Wir wählen wieder einen Dateiname und bleiben bei `student_database_1`.
- Wir klicken auf Save.



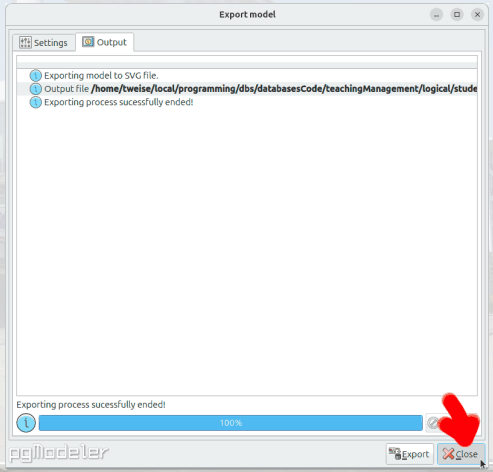
Logisches ERD erstellen mit PgModeler

- Wir können nun auf Export klicken.



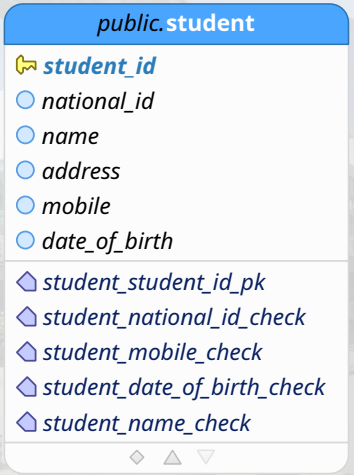
Logisches ERD erstellen mit PgModeler

- Das Modell wird exportiert.



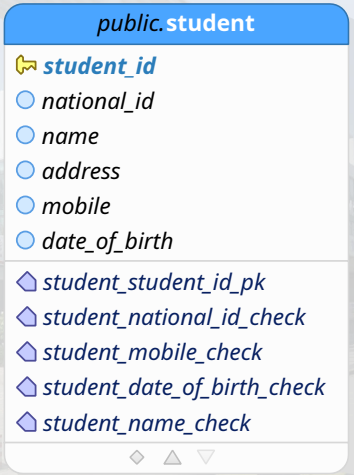
Logisches ERD erstellen mit PgModeler

- Das ist die exportierte Vektorgrafik.



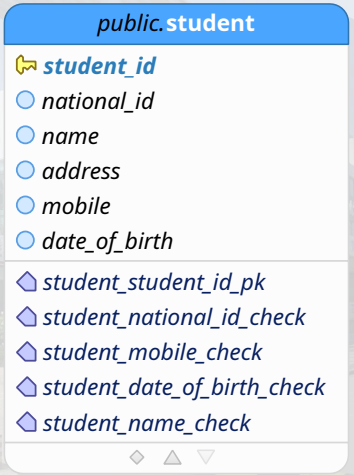
Logisches ERD erstellen mit PgModeler

- Das ist die exportierte Vektorgrafik.
- Sie sieht ganz nett aus.



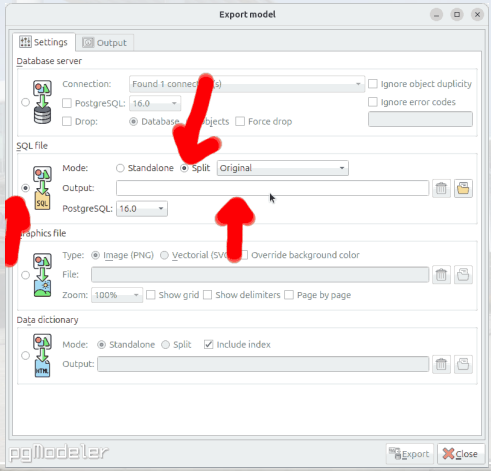
Logisches ERD erstellen mit PgModeler

- Sie sieht ganz nett aus.
- Wenn wir ein Modell mit mehr Tabellen gemacht hätten, würde es cooler aussehen.



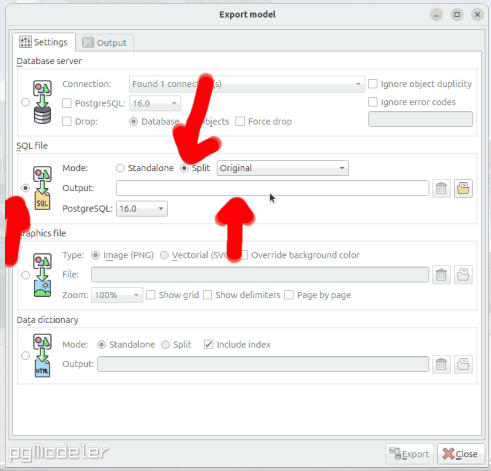
Logisches ERD erstellen mit PgModeler

- Wir öffnen den Export-Dialog nochmal.



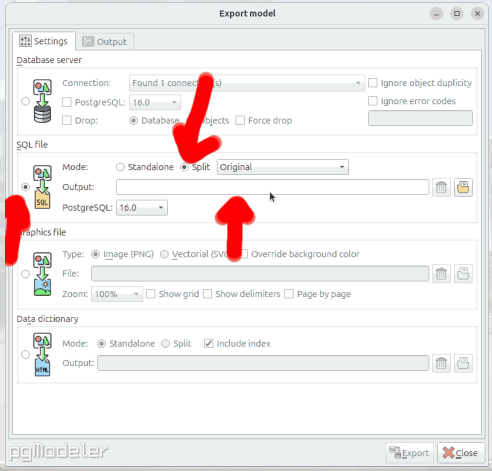
Logisches ERD erstellen mit PgModeler

- Wir öffnen den Export-Dialog nochmal.
- Dieses Mal exportieren wir unser Modell nach SQL.



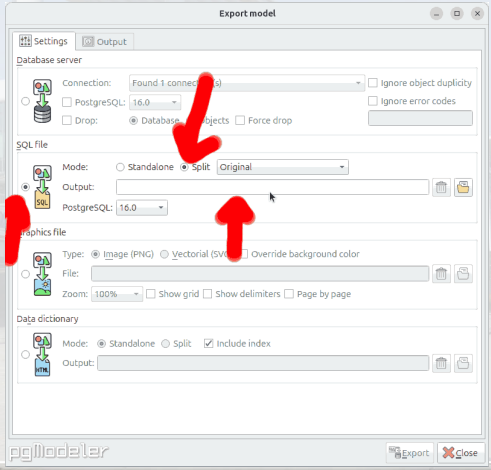
Logisches ERD erstellen mit PgModeler

- Wir öffnen den Export-Dialog nochmal.
- Dieses Mal exportieren wir unser Modell nach SQL.
- Wir klicken auf SQL file.



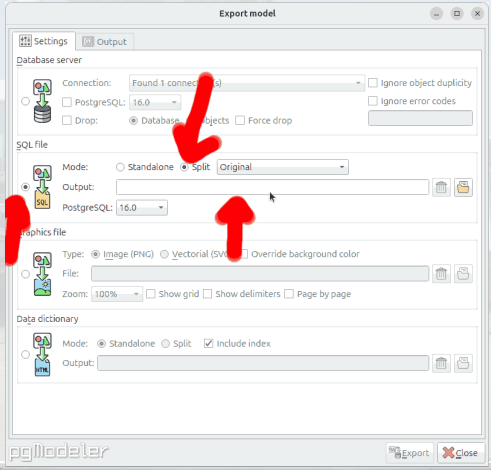
Logisches ERD erstellen mit PgModeler

- Dieses Mal exportieren wir unser Modell nach SQL.
- Wir klicken auf SQL file.
- Wichtig: Markieren Sie die Ausgabe als *Split*!



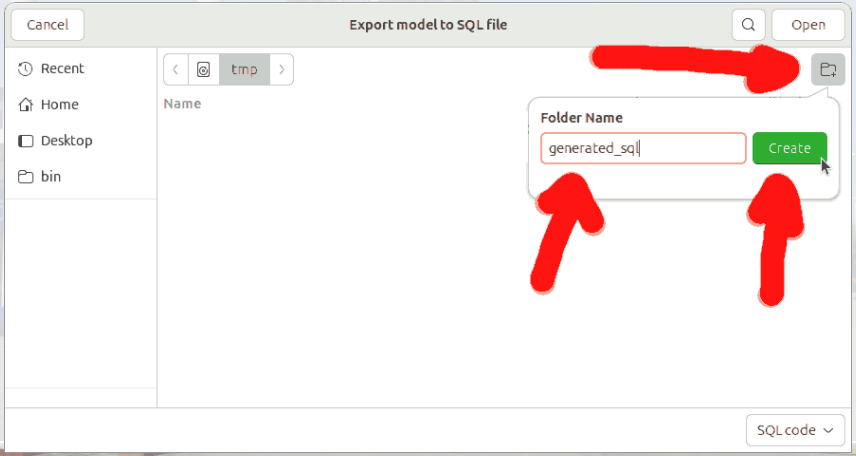
Logisches ERD erstellen mit PgModeler

- Wichtig: Markieren Sie die Ausgabe als *Split*!
- Dann klicken wir auf den File-Balken.



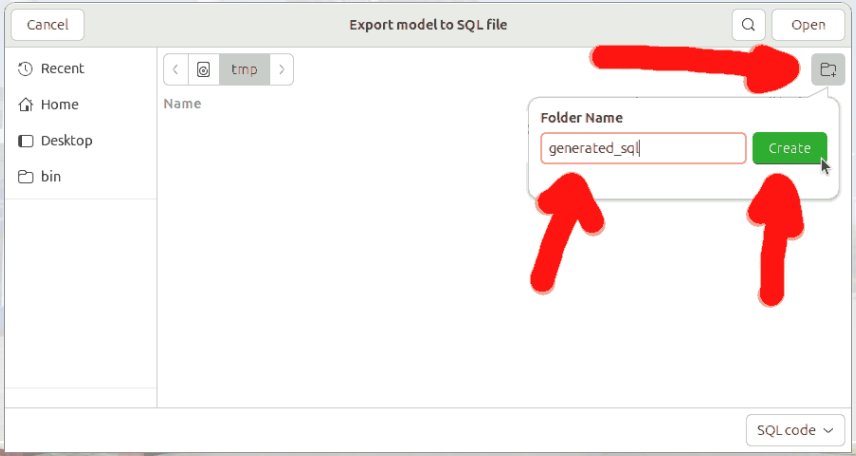
Logisches ERD erstellen mit PgModeler

- Weil wir unser Modell mit der *split*-Methode exportieren, erzeugen wir mehrere SQL-Dateien.



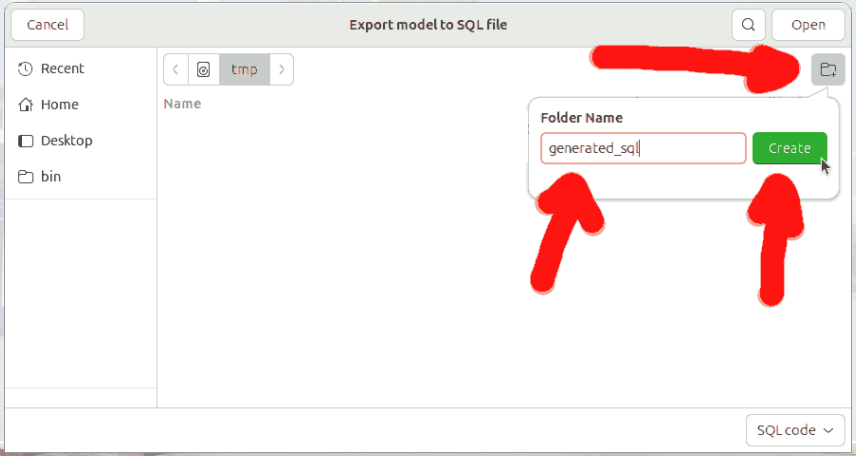
Logisches ERD erstellen mit PgModeler

- Weil wir unser Modell mit der *split*-Methode exportieren, erzeugen wir mehrere SQL-Dateien.
- Anstelle eines Dateinamens geben wir einen Ordnername an.



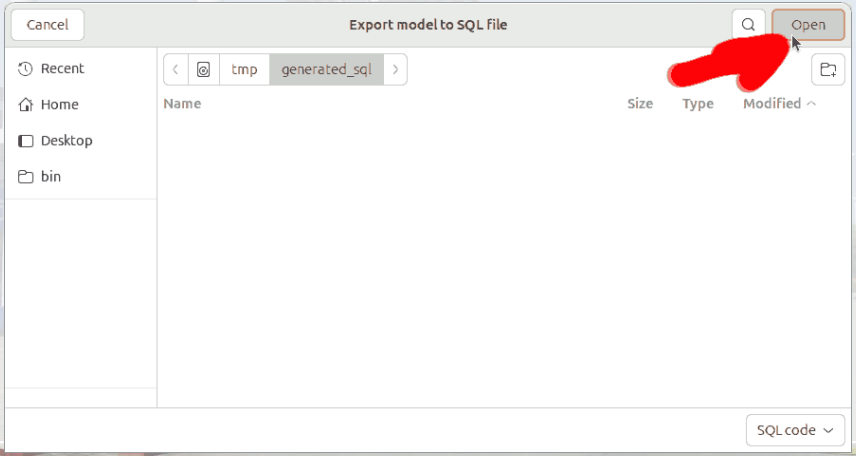
Logisches ERD erstellen mit PgModeler

- Anstelle eines Dateinamens geben wir einen Ordnername an.
- Wir erzeugen einen neuen Ordner und nennen ihn `generated_sql`.



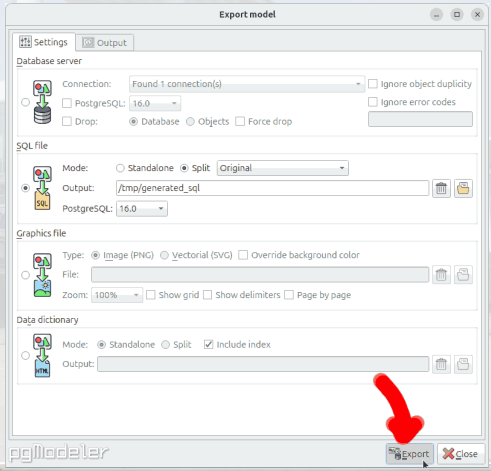
Logisches ERD erstellen mit PgModeler

- Der Ordner wird erstellt. Wir klicken auf Open.



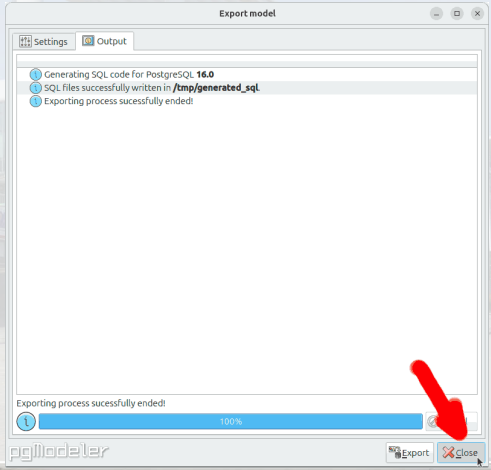
Logisches ERD erstellen mit PgModeler

- Das bringt uns zurück zum *Export model*-Dialog, wo wir auf Export klicken.



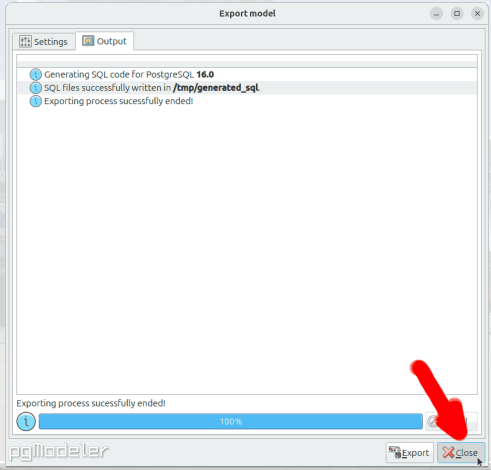
Logisches ERD erstellen mit PgModeler

- Das logische Modell wird nach SQL exportiert.



Logisches ERD erstellen mit PgModeler

- Das logische Modell wird nach SQL exportiert.
- Wir schließen den Dialog mit Klick auf Close.



Logisches ERD erstellen mit PgModeler

- Wir können uns den Ordner nun mit einem Dateibrowser unserers OS anschauen.

Logisches ERD erstellen mit PgModeler

- Wir können uns den Ordner nun mit einem Dateibrowser unserers OS anschauen.
- Wir finden, dass mehrere Dateien drin sind, die wir uns im Folgenden anschauen werden.

Logisches ERD erstellen mit PgModeler

- Wir finden, dass mehrere Dateien drin sind, die wir uns im Folgenden anschauen werden.
- Wir können und werden sie auf dem PostgreSQL Server via psql ausführen.

Datenbank Erstellen

- Es wurde ein Skript `01_student_database_database_2001.sql` zum Erstellen der Datenbank generiert.

```
-- object: student_database | type: DATABASE --
-- DROP DATABASE IF EXISTS student_database;
CREATE DATABASE student_database;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf 01
    ↪ _student_database_database_2001.sql
CREATE DATABASE
# psql 16.11 succeeded with exit code 0.
```

Tabelle

- Das auto-generierte Script `03_public_student_table_5071.sql` erstellt die Tabelle `student`.

```sql
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    student_id character(11) NOT NULL,
    national_id character(18) NOT NULL,
    name varchar(255) NOT NULL,
    address varchar(255) NOT NULL,
    mobile character(11) NOT NULL,
    date_of_birth date NOT NULL,
    CONSTRAINT student_student_id_pk PRIMARY KEY (student_id),
    CONSTRAINT student_national_id_check CHECK (national_id ~ '^\d{6}
        ↪ ((19)|(20))\d{9}[0-9X]$'),
    CONSTRAINT student_mobile_check CHECK (mobile ~ '^\d{11}$'),
    CONSTRAINT student_date_of_birth_check CHECK ((date_of_birth > '
        ↪ 1900-01-01') AND (date_of_birth < '2100-01-01')),
    CONSTRAINT student_name_check CHECK (name ~ '^\S+.*\S+$')
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf 03_public_student_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Wir probieren die Tabelle aus, in dem wir ein paar Daten einfügen.

```
/** Insert some rows into the student database. */

-- Insert records that can be inserted correctly.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'Hefei, China',
     '12345678901', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Chemnitz, Germany',
     '12345678902', '2005-08-07');

-- Print the records that were inserted.
SELECT student_id, name FROM student;

-- Try inserting an invalid record: The date of birth is way too early.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf insert.sql
INSERT 0 2
 student_id  | name
-------------+------
 1234567890  | Bibbo
 1234567891  | Bebbo
(2 rows)

psql:teachingManagement/logical/student_database_1/insert.sql:18: ERROR:
    ↪   new row for relation "student" violates check constraint "
    ↪ student_date_of_birth_check"
DETAIL:  Failing row contains (1111111111 , 123456022501011234, Liu Hui,
    ↪  Zouping, Shandong, 12345678902, 0225-01-01).
psql:teachingManagement/logical/student_database_1/insert.sql:18:
    ↪ STATEMENT:  INSERT INTO student (student_id, national_id, name,
    ↪ address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
# psql 16.11 failed with exit code 3.
```

Daten Einfügen

- Wir probieren die Tabelle aus, in dem wir ein paar Daten einfügen.
- Dazu schreiben wir selbst die SQL-Befehle rechts.

```
/** Insert some rows into the student database. */

-- Insert records that can be inserted correctly.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'Hefei, China',
     '12345678901', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Chemnitz, Germany',
     '12345678902', '2005-08-07');

-- Print the records that were inserted.
SELECT student_id, name FROM student;

-- Try inserting an invalid record: The date of birth is way too early.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf insert.sql
INSERT 0 2
 student_id  | name
-------------+------
 1234567890  | Bibbo
 1234567891  | Bebbo
(2 rows)

psql:teachingManagement/logical/student_database_1/insert.sql:18: ERROR:
    ↪   new row for relation "student" violates check constraint "
    ↪ student_date_of_birth_check"
DETAIL:  Failing row contains (1111111111 , 123456022501011234, Liu Hui,
    ↪ Zouping, Shandong, 12345678902, 0225-01-01).
psql:teachingManagement/logical/student_database_1/insert.sql:18:
    ↪ STATEMENT:  INSERT INTO student (student_id, national_id, name,
    ↪ address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
# psql 16.11 failed with exit code 3.
```

Daten Einfügen

- Wir probieren die Tabelle aus, in dem wir ein paar Daten einfügen.
- Dazu schreiben wir selbst die SQL-Befehle rechts.
- Die ersten beiden Datensätze sind OK.

```sql
/** Insert some rows into the student database. */

-- Insert records that can be inserted correctly.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'Hefei, China',
     '12345678901', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Chemnitz, Germany',
     '12345678902', '2005-08-07');

-- Print the records that were inserted.
SELECT student_id, name FROM student;

-- Try inserting an invalid record: The date of birth is way too early.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf insert.sql
INSERT 0 2
 student_id  | name
-------------+------
 1234567890  | Bibbo
 1234567891  | Bebbo
(2 rows)

psql:teachingManagement/logical/student_database_1/insert.sql:18: ERROR:
    ↪   new row for relation "student" violates check constraint "
    ↪ student_date_of_birth_check"
DETAIL:  Failing row contains (1111111111 , 123456022501011234, Liu Hui,
    ↪  Zouping, Shandong, 12345678902, 0225-01-01).
psql:teachingManagement/logical/student_database_1/insert.sql:18:
    ↪ STATEMENT:  INSERT INTO student (student_id, national_id, name,
    ↪ address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
# psql 16.11 failed with exit code 3.
```

Daten Einfügen

- Wir probieren die Tabelle aus, in dem wir ein paar Daten einfügen.
- Dazu schreiben wir selbst die SQL-Befehle rechts.
- Die ersten beiden Datensätze sind OK.
- Der dritte Datensatz verletzt Einschränkungen und kann nicht eingefügt werden.

```sql
/** Insert some rows into the student database. */

-- Insert records that can be inserted correctly.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'Hefei, China',
     '12345678901', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Chemnitz, Germany',
     '12345678902', '2005-08-07');

-- Print the records that were inserted.
SELECT student_id, name FROM student;

-- Try inserting an invalid record: The date of birth is way too early.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf insert.sql
INSERT 0 2
 student_id  | name
-------------+------
 1234567890  | Bibbo
 1234567891  | Bebbo
(2 rows)

psql:teachingManagement/logical/student_database_1/insert.sql:18: ERROR:
    ↪   new row for relation "student" violates check constraint "
    ↪ student_date_of_birth_check"
DETAIL:  Failing row contains (1111111111 , 123456022501011234, Liu Hui,
    ↪  Zouping, Shandong, 12345678902, 0225-01-01).
psql:teachingManagement/logical/student_database_1/insert.sql:18:
    ↪ STATEMENT:  INSERT INTO student (student_id, national_id, name,
    ↪ address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
# psql 16.11 failed with exit code 3.
```

Daten Einfügen

- Wir probieren die Tabelle aus, in dem wir ein paar Daten einfügen.
- Dazu schreiben wir selbst die SQL-Befehle rechts.
- Die ersten beiden Datensätze sind OK.
- Der dritte Datensatz verletzt Einschränkungen und kann nicht eingefügt werden.
- Im Ergebnis sehen wir genau, was wir erwartet haben: Die ersten beiden Datensätze wurden erfolgreich eingefügt, der dritte nicht.

```sql
/** Insert some rows into the student database. */

-- Insert records that can be inserted correctly.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'Hefei, China',
     '12345678901', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Chemnitz, Germany',
     '12345678902', '2005-08-07');

-- Print the records that were inserted.
SELECT student_id, name FROM student;

-- Try inserting an invalid record: The date of birth is way too early.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
   ↪ ON_ERROR_STOP=1 -ebf insert.sql
INSERT 0 2
 student_id  | name
-------------+------
 1234567890  | Bibbo
 1234567891  | Bebbo
(2 rows)

psql:teachingManagement/logical/student_database_1/insert.sql:18: ERROR:
   ↪   new row for relation "student" violates check constraint "
   ↪ student_date_of_birth_check"
DETAIL:  Failing row contains (1111111111 , 123456022501011234, Liu Hui,
   ↪ Zouping, Shandong, 12345678902, 0225-01-01).
psql:teachingManagement/logical/student_database_1/insert.sql:18:
   ↪ STATEMENT:  INSERT INTO student (student_id, national_id, name,
   ↪ address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
# psql 16.11 failed with exit code 3.
```

Daten Einfügen

- Im Ergebnis sehen wir genau, was wir erwartet haben: Die ersten beiden Datensätze wurden erfolgreich eingefügt, der dritte nicht.
- Der Mathematiker LIU Hui (刘徽), der eine Methode zur Annäherung der Kreiszahl π erfunden und in seinen Kommentaren zum Buch *Jiu Zhang Suanshu* (九章算术)[29,32,55,71,96] veröffentlichte, lebte im 3. Jahrhundert CE[71,117] ... wurde also zu früh geboren, um sich als Student der 合肥大学 einzuschreiben.

```
/** Insert some rows into the student database. */

-- Insert records that can be inserted correctly.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'Hefei, China',
     '12345678901', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Chemnitz, Germany',
     '12345678902', '2005-08-07');

-- Print the records that were inserted.
SELECT student_id, name FROM student;

-- Try inserting an invalid record: The date of birth is way too early.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf insert.sql
INSERT 0 2
 student_id  | name
-------------+------
 1234567890  | Bibbo
 1234567891  | Bebbo
(2 rows)

psql:teachingManagement/logical/student_database_1/insert.sql:18: ERROR:
    ↪   new row for relation "student" violates check constraint "
    ↪ student_date_of_birth_check"
DETAIL:  Failing row contains (1111111111 , 123456022501011234, Liu Hui,
    ↪  Zouping, Shandong, 12345678902, 0225-01-01).
psql:teachingManagement/logical/student_database_1/insert.sql:18:
    ↪ STATEMENT:  INSERT INTO student (student_id, national_id, name,
    ↪ address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
# psql 16.11 failed with exit code 3.
```

Daten Einfügen

- Der Mathematiker LIU Hui (刘徽), der eine Methode zur Annäherung der Kreiszahl π erfunden und in seinen Kommentaren zum Buch *Jiu Zhang Suanshu* (九章算术)[29,32,55,71,96] veröffentlichte, lebte im 3. Jahrhundert CE[71,117] ... wurde also zu früh geboren, um sich als Student der 合肥大学 einzuschreiben.
- (Dieses Skript haben wir selber gemacht, das ist nicht von PgModeler.)

```sql
/** Insert some rows into the student database. */

-- Insert records that can be inserted correctly.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'Hefei, China',
     '12345678901', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Chemnitz, Germany',
     '12345678902', '2005-08-07');

-- Print the records that were inserted.
SELECT student_id, name FROM student;

-- Try inserting an invalid record: The date of birth is way too early.
INSERT INTO student (student_id, national_id, name, address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf insert.sql
INSERT 0 2
 student_id  | name
-------------+------
 1234567890  | Bibbo
 1234567891  | Bebbo
(2 rows)

psql:teachingManagement/logical/student_database_1/insert.sql:18: ERROR:
    ↪   new row for relation "student" violates check constraint "
    ↪ student_date_of_birth_check"
DETAIL:  Failing row contains (1111111111 , 123456022501011234, Liu Hui,
    ↪  Zouping, Shandong, 12345678902, 0225-01-01).
psql:teachingManagement/logical/student_database_1/insert.sql:18:
    ↪ STATEMENT:  INSERT INTO student (student_id, national_id, name,
    ↪ address, mobile,
                     date_of_birth) VALUES
    ('1111111111', '123456022501011234', 'Liu Hui', 'Zouping, Shandong',
     '12345678902', '0225-01-01');
# psql 16.11 failed with exit code 3.
```

Datenbank Löschen

- Zu guter Letzt löschen wir die Datenbank wieder.

```
/* Cleanup after the example: Delete the student database. */

DROP DATABASE IF EXISTS student_database;
```

```
$ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf
    ↪ cleanup.sql
DROP DATABASE
# psql 16.11 succeeded with exit code 0.
```

Datenbank Löschen

- Zu guter Letzt löschen wir die Datenbank wieder.
- Beachten Sie, dass wir uns in der Connection-URI nicht auf die Datenbank verbinden ... sonst könnten wir sie ja nicht löschen.

```
1 /* Cleanup after the example: Delete the student database. */
2
3 DROP DATABASE IF EXISTS student_database;
```

```
1 $ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf
    ↪ cleanup.sql
2 DROP DATABASE
3 # psql 16.11 succeeded with exit code 0.
```

Datenbank Löschen

- Zu guter Letzt löschen wir die Datenbank wieder.
- Beachten Sie, dass wir uns in der Connection-URI nicht auf die Datenbank verbinden ... sonst könnten wir sie ja nicht löschen.
- (Dieses Skript haben wir selber gemacht, das ist nicht von PgModeler.)

```
1 /* Cleanup after the example: Delete the student database. */
2
3 DROP DATABASE IF EXISTS student_database;
```

```
1 $ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf
    ↪ cleanup.sql
2 DROP DATABASE
3 # psql 16.11 succeeded with exit code 0.
```

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen. Diese Modelle sind leicht zu verstehende Grafiken in der Krähenfußnotation.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen. Diese Modelle sind leicht zu verstehende Grafiken in der Krähenfußnotation. Der PgModeler kann sich direkt auf den PostgreSQL Server verbinden und die Modelle direkt dorthin schicken.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen. Diese Modelle sind leicht zu verstehende Grafiken in der Krähenfußnotation. Der PgModeler kann sich direkt auf den PostgreSQL Server verbinden und die Modelle direkt dorthin schicken. Es kann die logischen Modelle auch in SQL-Skripte exportieren, die wir dann ausführen können.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen. Diese Modelle sind leicht zu verstehende Grafiken in der Krähenfußnotation. Der PgModeler kann sich direkt auf den PostgreSQL Server verbinden und die Modelle direkt dorthin schicken. Es kann die logischen Modelle auch in SQL-Skripte exportieren, die wir dann ausführen können. Er ist eine nützliche GUI, um logische Schemas für Datenbanken zu erstellen.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen.

Gute Praxis

Daten sollten auf allen Ebenen einer Applikation geprüft werden.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen.

Gute Praxis

Daten sollten auf allen Ebenen einer Applikation geprüft werden: in den Formularen, wo sie eingegeben werden, in der Datenbank mit Einschränkungen, und in den Applikationen, die sie wieder aus der Datenbank laden.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen.

Gute Praxis

Daten sollten auf allen Ebenen einer Applikation geprüft werden: in den Formularen, wo sie eingegeben werden, in der Datenbank mit Einschränkungen, und in den Applikationen, die sie wieder aus der Datenbank laden. Je mehr Verteidigungslinien wir durch Einschränkungen sowie statische und dynamische Überprüfungen erzeugen, desto größer ist unsere Chance, Fehler früh zu entdecken, ihren Grund zu finden, und zu verhindern, dass sie sich weiter fortpflanzen.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen.

Gute Praxis

Daten sollten auf allen Ebenen einer Applikation geprüft werden: in den Formularen, wo sie eingegeben werden, in der Datenbank mit Einschränkungen, und in den Applikationen, die sie wieder aus der Datenbank laden. Je mehr Verteidigungslinien wir durch Einschränkungen sowie statische und dynamische Überprüfungen erzeugen, desto größer ist unsere Chance, Fehler früh zu entdecken, ihren Grund zu finden, und zu verhindern, dass sie sich weiter fortpflanzen. Dadurch bekommen wir die beste Chance, Fehler zu korrigieren und zu verhindern, dass Fehler andere Daten beschmutzen.

Was wir gelernt haben

Nützliches Werkzeug

Mit dem PgModeler haben wir ein Werkzeug, das es uns erlaubt, logische Modelle für Datenbanken im Grunde als ERDs zu malen.

Gute Praxis

Daten sollten auf allen Ebenen einer Applikation geprüft werden: in den Formularen, wo sie eingegeben werden, in der Datenbank mit Einschränkungen, und in den Applikationen, die sie wieder aus der Datenbank laden. Je mehr Verteidigungslinien wir durch Einschränkungen sowie statische und dynamische Überprüfungen erzeugen, desto größer ist unsere Chance, Fehler früh zu entdecken, ihren Grund zu finden, und zu verhindern, dass sie sich weiter fortpflanzen. Dadurch bekommen wir die beste Chance, Fehler zu korrigieren und zu verhindern, dass Fehler andere Daten beschmutzen. Daher sind auch einfache Einschränkungen immer sinnvoll.

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

- Wir sind dabei, Entitätstypen zum relatinalen Datenmodell zu übersetzen.

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

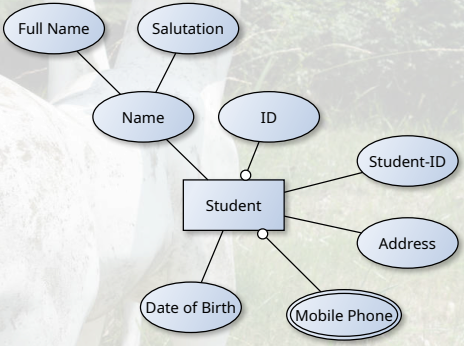
- Wir sind dabei, Entitätstypen zum relatinalen Datenmodell zu übersetzen.
- Dabei werden zusammengesetzte Attribute in ihre Komponenten zerlegt, die dann einzelne Spalten werden.

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

- Wir sind dabei, Entitätstypen zum relatinalen Datenmodell zu übersetzen.
- Dabei werden zusammengesetzte Attribute in ihre Komponenten zerlegt, die dann einzelne Spalten werden.
- Mehrwertige Attribute werden eigene Tabellen.

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

- Dabei werden zusammengesetzte Attribute in ihre Komponenten zerlegt, die dann einzelne Spalten werden.
- Mehrwertige Attribute werden eigene Tabellen.
- Als wir angefangen hatten, konzeptuelle Modelle zu erstellen, hatten wir eine Variante des Studenten-Entitätstyps, die sowohl zusammengesetzte als auch mehrwertige Attribute hatte.



Zusammengesetzte und Mehrwertige Attribute (Teil 2)

- Dabei werden zusammengesetzte Attribute in ihre Komponenten zerlegt, die dann einzelne Spalten werden.
- Mehrwertige Attribute werden eigene Tabellen.
- Als wir angefangen hatten, konzeptuelle Modelle zu erstellen, hatten wir eine Variante des Studenten-Entitätstyps, die sowohl zusammengesetzte als auch mehrwertige Attribute hatte.
- Wir benutzen nun dieses Beispiel und erstellen ein passendes logisches Modell.

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

- Dabei werden zusammengesetzte Attribute in ihre Komponenten zerlegt, die dann einzelne Spalten werden.
- Mehrwertige Attribute werden eigene Tabellen.
- Als wir angefangen hatten, konzeptuelle Modelle zu erstellen, hatten wir eine Variante des Studenten-Entitätstyps, die sowohl zusammengesetzte als auch mehrwertige Attribute hatte.
- Wir benutzen nun dieses Beispiel und erstellen ein passendes logisches Modell.
- Wir haben gerade erst ein logisches Modell für einen (anderen) *Student*-Entitätstyp erstellt.

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

- Dabei werden zusammengesetzte Attribute in ihre Komponenten zerlegt, die dann einzelne Spalten werden.
- Mehrwertige Attribute werden eigene Tabellen.
- Als wir angefangen hatten, konzeptuelle Modelle zu erstellen, hatten wir eine Variante des Studenten-Entitätstyps, die sowohl zusammengesetzte als auch mehrwertige Attribute hatte.
- Wir benutzen nun dieses Beispiel und erstellen ein passendes logisches Modell.
- Wir haben gerade erst ein logisches Modell für einen (anderen) *Student*-Entitätstyp erstellt.
- Der hatte ein einfaches Attribut `name` und ein einwertiges `mobile`-Attribut.

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

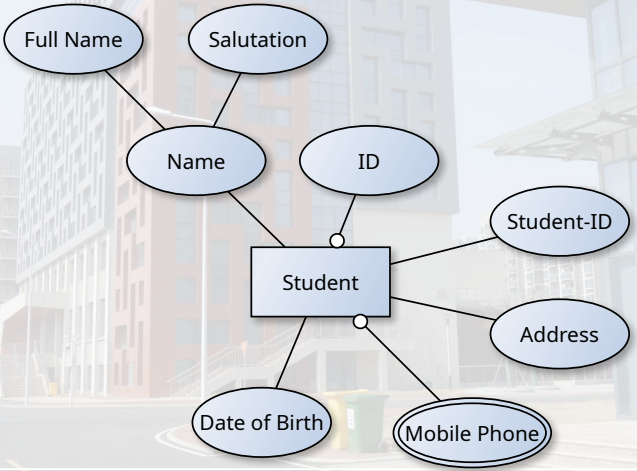
- Dabei werden zusammengesetzte Attribute in ihre Komponenten zerlegt, die dann einzelne Spalten werden.
- Mehrwertige Attribute werden eigene Tabellen.
- Als wir angefangen hatten, konzeptuelle Modelle zu erstellen, hatten wir eine Variante des Studenten-Entitätstyps, die sowohl zusammengesetzte als auch mehrwertige Attribute hatte.
- Wir benutzen nun dieses Beispiel und erstellen ein passendes logisches Modell.
- Wir haben gerade erst ein logisches Modell für einen (anderen) *Student*-Entitätstyp erstellt.
- Der hatte ein einfaches Attribut `name` und ein einwertiges `mobile`-Attribut.
- Wenn wir diese beiden Spalten und entsprechenden Einschränkungen weglassen, dann ist dieses alte Modell ein guter Startpunkt.

Zusammengesetzte und Mehrwertige Attribute (Teil 2)

- Dabei werden zusammengesetzte Attribute in ihre Komponenten zerlegt, die dann einzelne Spalten werden.
- Mehrwertige Attribute werden eigene Tabellen.
- Als wir angefangen hatten, konzeptuelle Modelle zu erstellen, hatten wir eine Variante des Studenten-Entitätstyps, die sowohl zusammengesetzte als auch mehrwertige Attribute hatte.
- Wir benutzen nun dieses Beispiel und erstellen ein passendes logisches Modell.
- Wir haben gerade erst ein logisches Modell für einen (anderen) *Student*-Entitätstyp erstellt.
- Der hatte ein einfaches Attribut `name` und ein einwertiges `mobile`-Attribut.
- Wenn wir diese beiden Spalten und entsprechenden Einschränkungen weglassen, dann ist dieses alte Modell ein guter Startpunkt.
- Sie können es entweder nochmal malen oder die entsprechenden Spalten/Einschränkungen löschen.

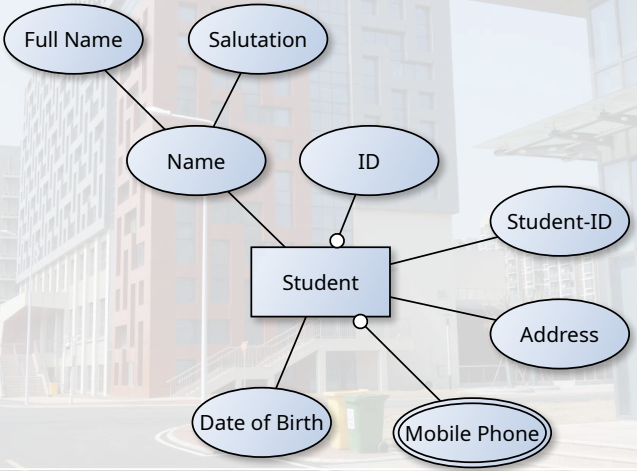
Student Entitätstyp 2

- Wir wollen nun ein Modell erstellen, in dem es sowohl zusammengesetzte als auch mehrwertige Attribute gibt.



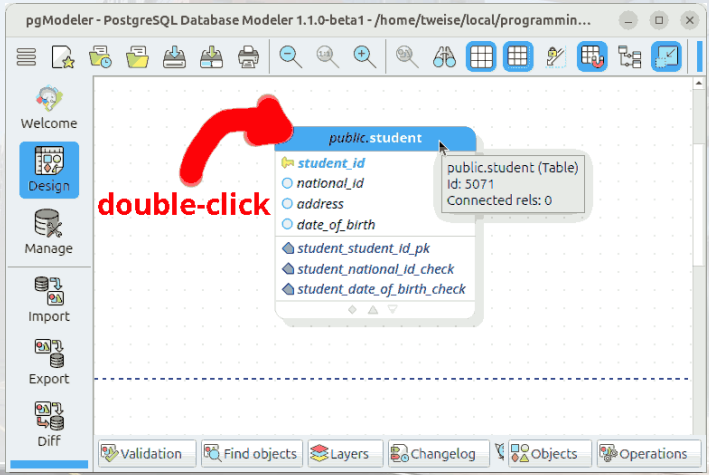
Student Entitätstyp 2

- Hier ist ein entsprechendes konzeptuelles Modell, dass wir ein ein logisches Modell basierend auf dem relationalen Datenmodell umsetzen wollen.



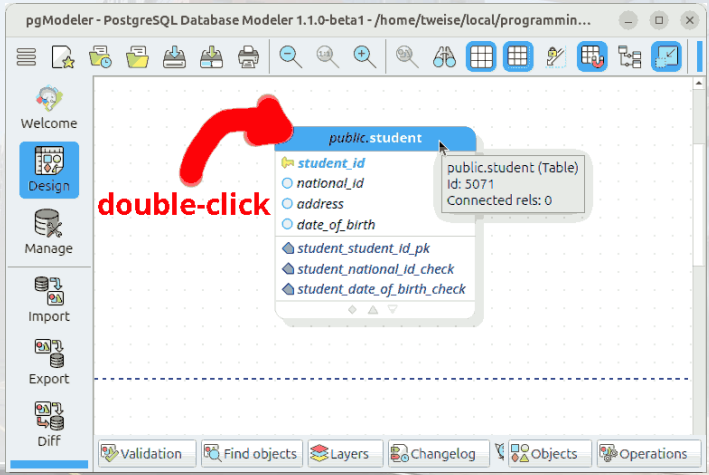
Student Entitätstyp 2

- Im PgModeler fangen wir mit dem selben Modell wie vorhin an.



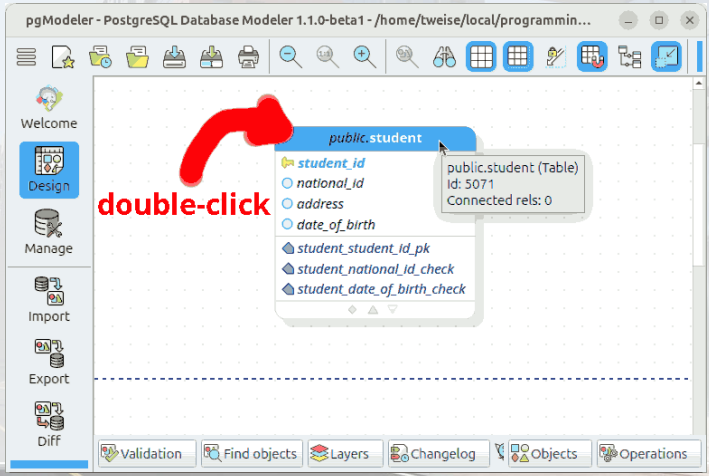
Student Entitätstyp 2

- Im PgModeler fangen wir mit dem selben Modell wie vorhin an.
- Allerdings ohne die Spalten `name` und `mobile` und ihre entsprechenden Einschränkungen.



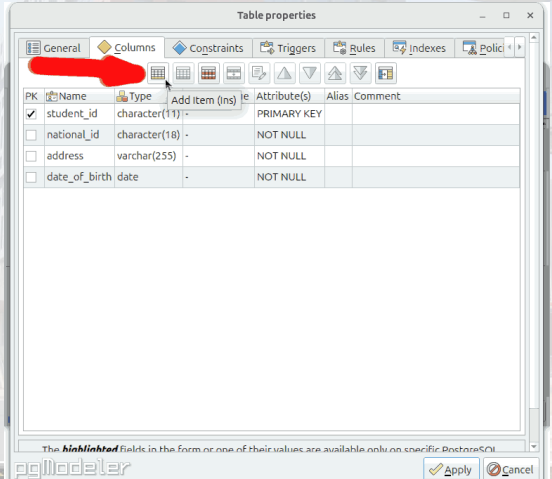
Student Entitätstyp 2

- Im PgModeler fangen wir mit dem selben Modell wie vorhin an.
- Allerdings ohne die Spalten `name` und `mobile` und ihre entsprechenden Einschränkungen.
- Wir doppel-klicken auf die Tabelle `student`, um sie zu bearbeiten.



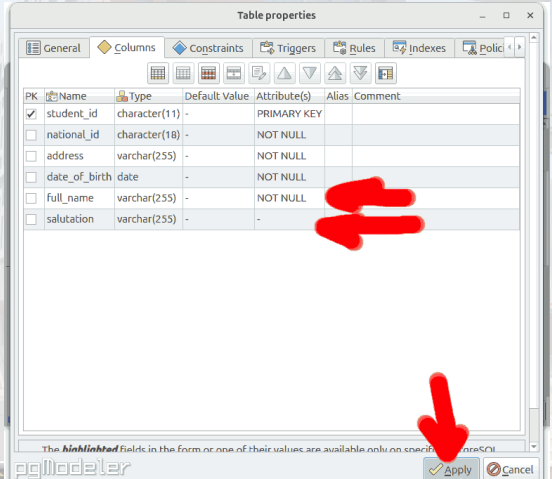
Student Entitätstyp 2

- Wir wollen zuerst die Spalten `full_name` und `saltulation` einfügen, die das zusammengesetzte Attribut `name` repräsentieren.



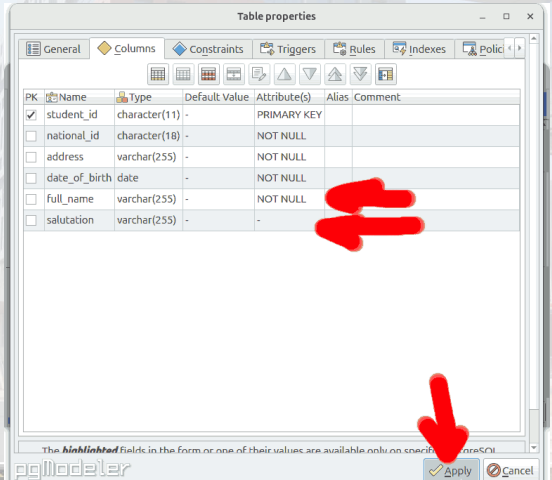
Student Entitätstyp 2

- Wir fügen eine Spalte `full_name` hinzu, die ein beliebig-langer String mit der Maximallänge 255 ist und `NOT NULL` seien muss.



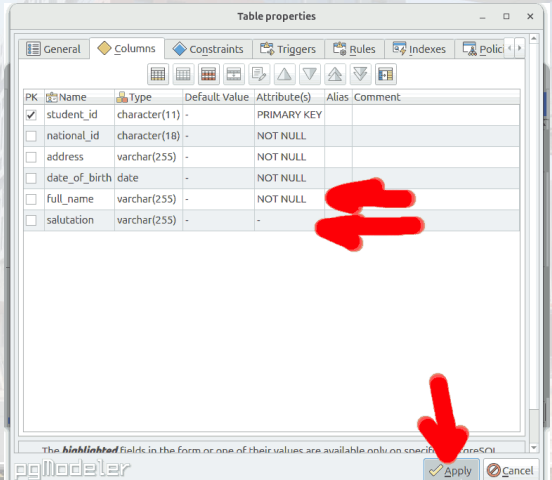
Student Entitätstyp 2

- Wir fügen eine Spalte `full_name` hinzu.
- Wir fügen die Spalte `saltulation` hinzu, die ein beliebig-langer String mit der Maximallänge 255 ist.



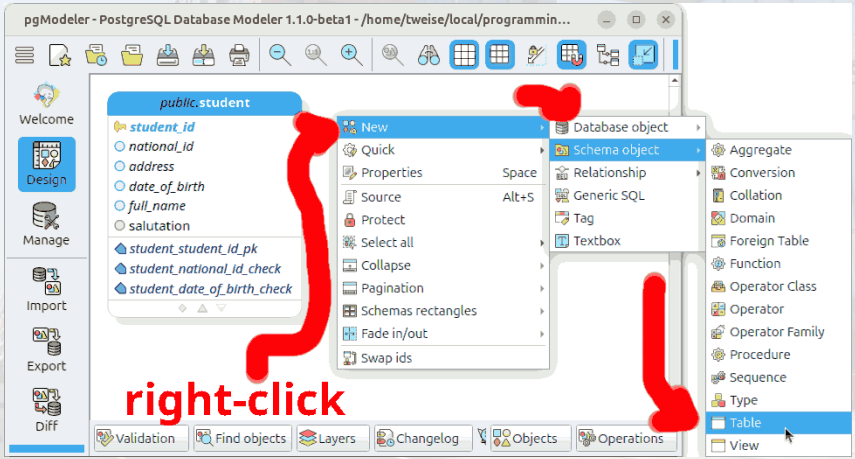
Student Entitätstyp 2

- Wir fügen eine Spalte `full_name` hinzu.
- Wir fügen die Spalte `saltulation` hinzu.
- Wir klicken auf Apply.



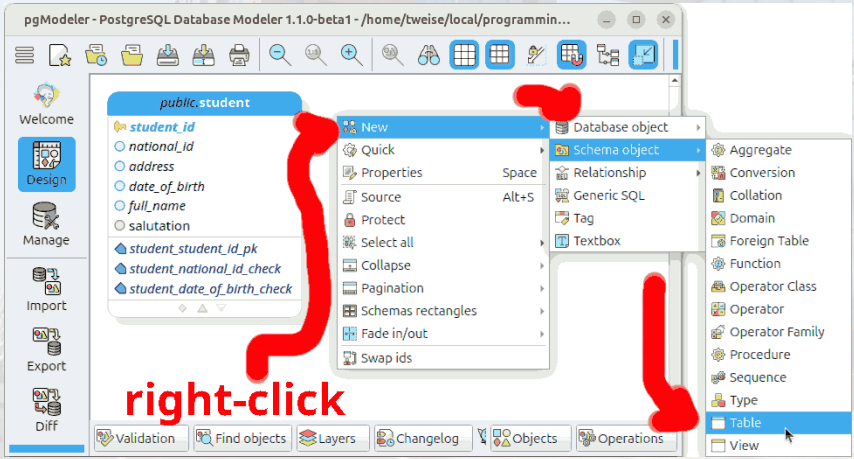
Student Entitätstyp 2

- Jedes mehrwertige Attribut wandert in eine eigene Tabelle.



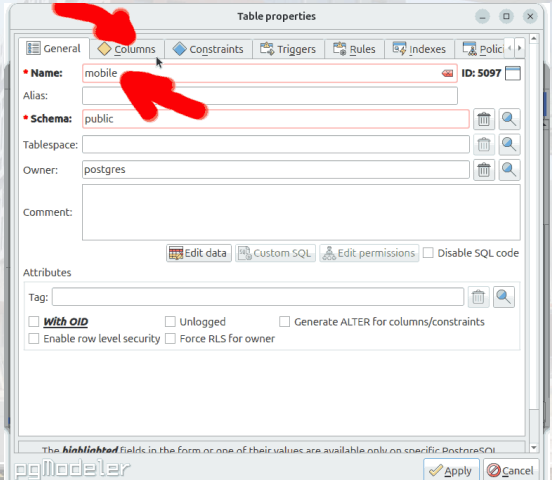
Student Entitätstyp 2

- Jedes mehrwertige Attribut wandert in eine eigene Tabelle.
- Also erstellen wir eine neue Tabelle mit rechts-Klick in unser Modell und Auswahl von New > Schema object > Table.



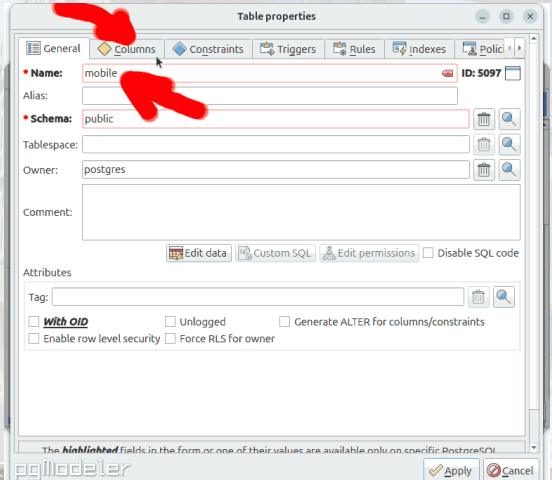
Student Entitätstyp 2

- Wir nennen die Tabelle `mobile`.



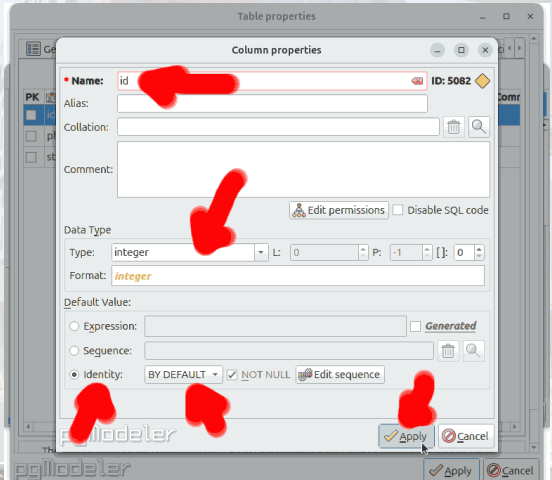
Student Entitätstyp 2

- Wir nennen die Tabelle `mobile`.
- Wir klicken auf Columns um Spalten zu definieren.



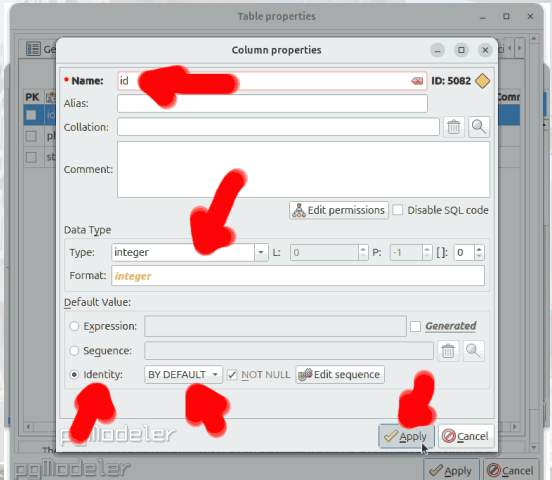
Student Entitätstyp 2

- Wir wollen jeweils eine Mobiltelefonnummer und eine Referenz auf den Studenten-Datensatz speichern, zu dem sie gehört.



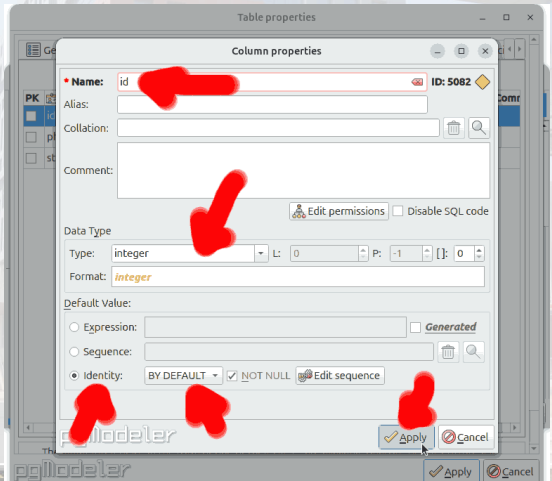
Student Entitätstyp 2

- Keines von beiden muss unbedingt einmalig (`UNIQUE`) sein, weil ja die selbe Person sich nacheinander in mehrere Studiengänge einschreiben kann.



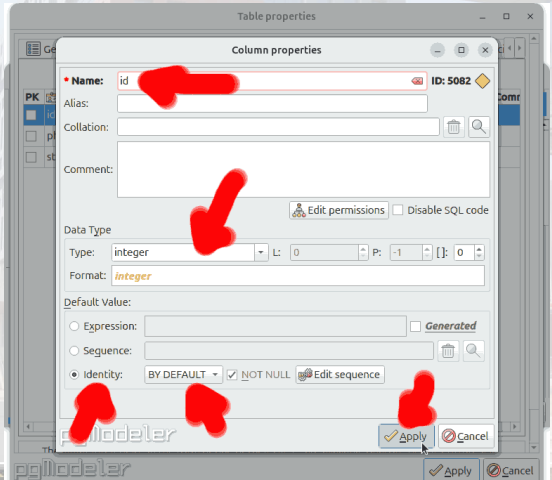
Student Entitätstyp 2

- Keines von beiden muss unbedingt einmalig (`UNIQUE`) sein, weil ja die selbe Person sich nacheinander in mehrere Studiengänge einschreiben kann.
- Wir brauchen also einen Ersatzschlüssel.



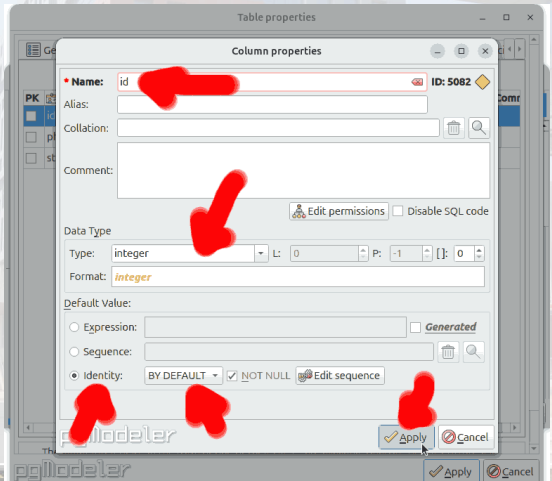
Student Entitätstyp 2

- Wir brauchen also einen Ersatzschlüssel.
- Wr erstellen die Spalte `id` vom Typ `integer` und markieren sie als Identiy, die BY DEFAULT generiert wird.



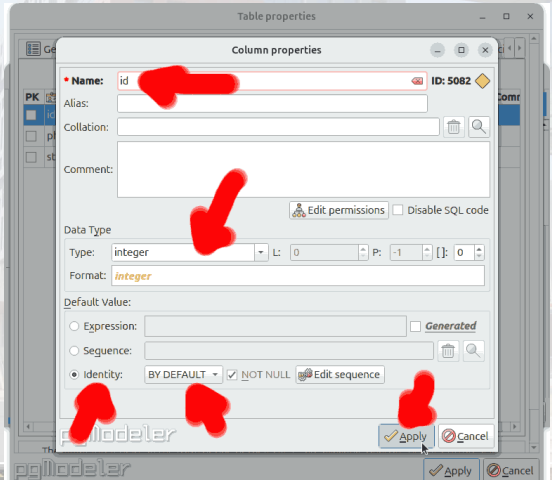
Student Entitätstyp 2

- Wr erstellen die Spalte `id` vom Typ `integer` und markieren sie als Identiy, die BY DEFAULT generiert wird.
- Das ist in etwa das gleiche, wie was wir im Fabrik-Beispiel gemacht haben.



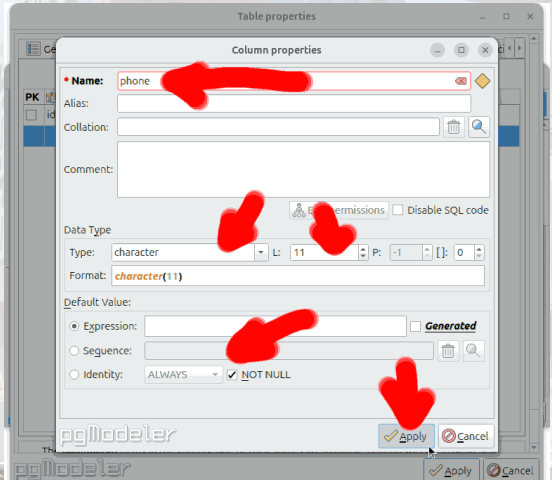
Student Entitätstyp 2

- Das ist in etwa das gleiche, wie was wir im Fabrik-Beispiel gemacht haben.
- Wir klicken auf Apply.



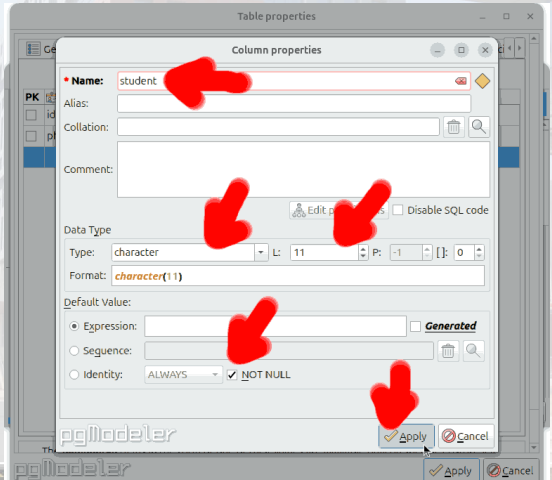
Student Entitätstyp 2

- Wir erstellen eine Spalte `phone` genauso, wie wir das vorhin auch schon gemacht haben.



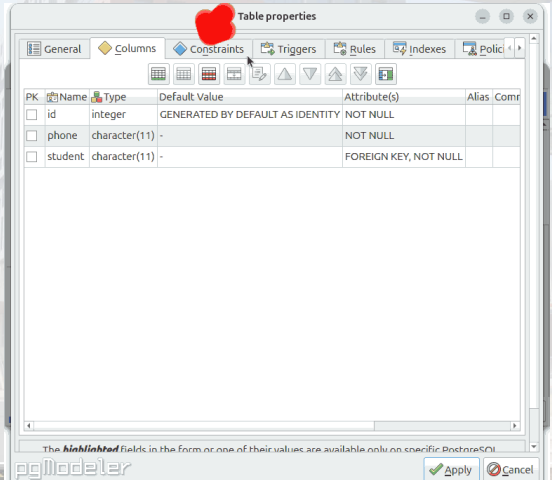
Student Entitätstyp 2

- Wir erstellen eine Spalte `student` die genau den selben Datentyp wie die `student_id`-Spalte mit dem Primärschlüssel der Tabelle `student` haben muss.



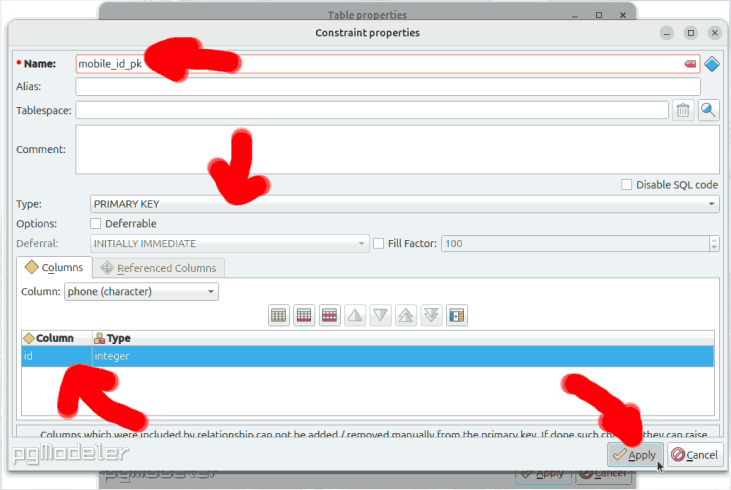
Student Entitätstyp 2

- Wir haben unsere drei Spalten erstellt und machen jetzt mit den Constraints weiter.



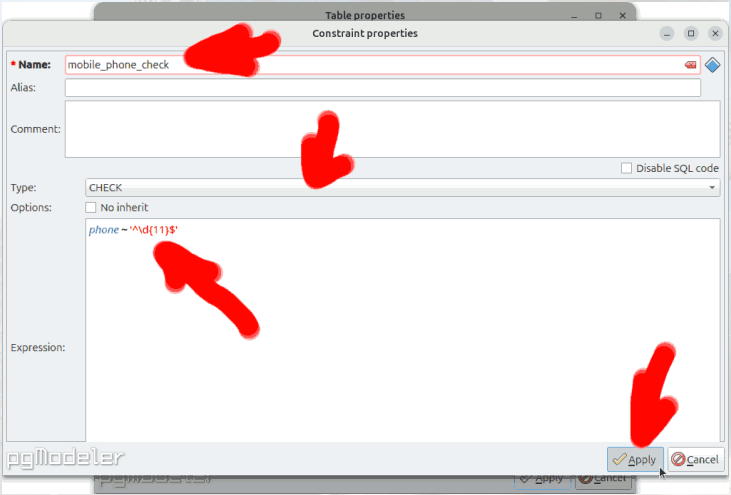
Student Entitätstyp 2

- Zuerst erstellen wir ein Primärschlüssel-Constraint für die Spalte `id`.



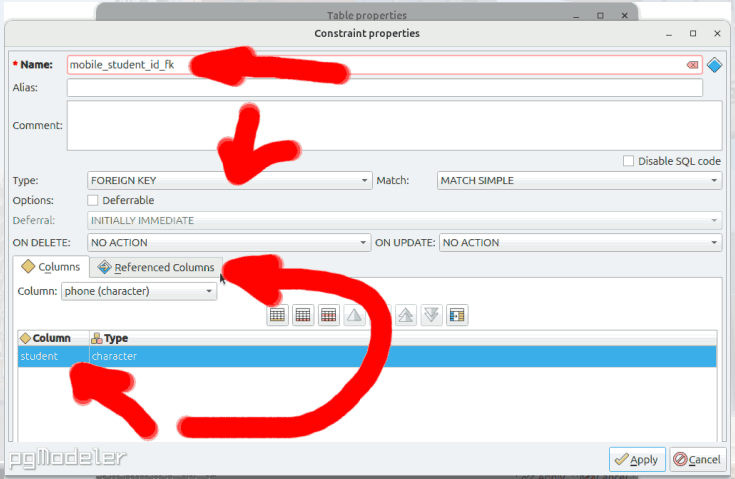
Student Entitätstyp 2

- Dann erstellen wir nochmal die selbe Mobiltelefonnummer-Überprüf-Einschrängung wie vorhin.



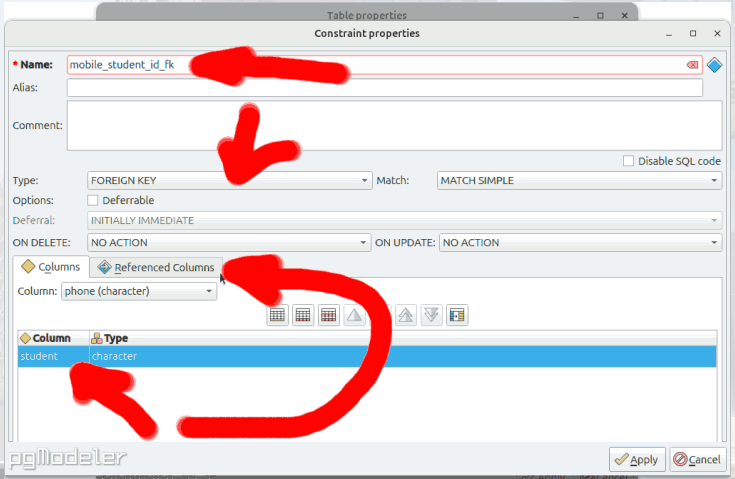
Student Entitätstyp 2

- Nun wollen wir die Zeilen dieser Tabelle mit den Zeilen der Tabelle `student` verbinden.



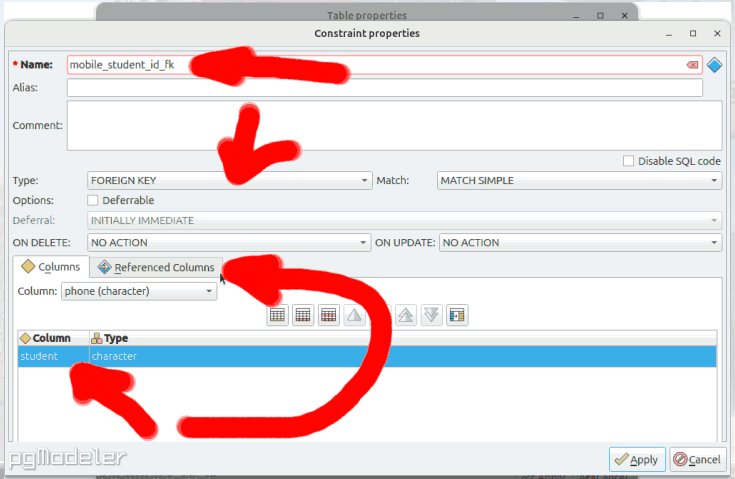
Student Entitätstyp 2

- Nun wollen wir die Zeilen dieser Tabelle mit den Zeilen der Tabelle `student` verbinden.
- Wir erstellen eine `FOREIGN KEY`-Einschränkung und nennen sie `mobile_student_id_fk`.



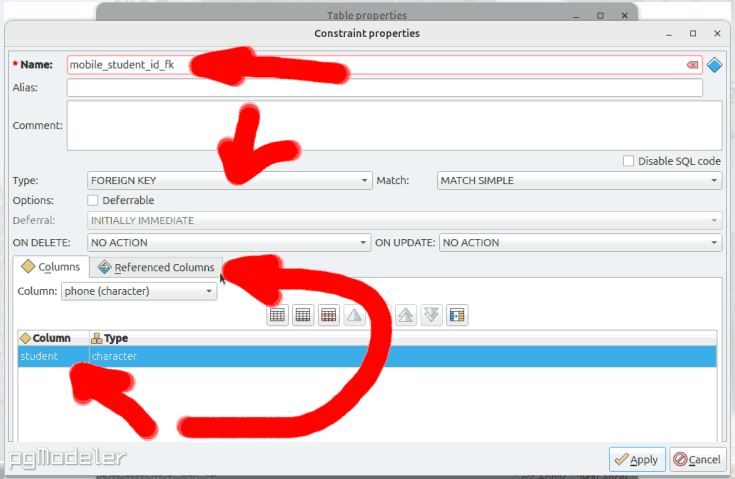
Student Entitätstyp 2

- Wir erstellen eine `FOREIGN KEY`-Einschränkung und nennen sie `mobile_student_id_fk`.
- Wir wählen `FOREIGN KEY` als Type:.



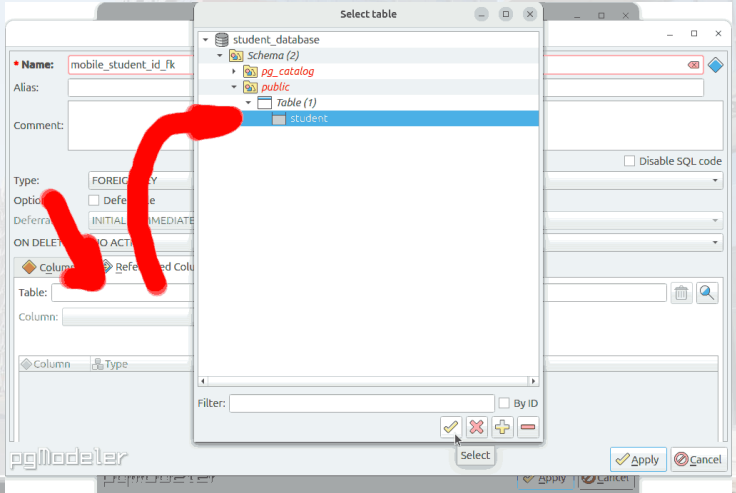
Student Entitätstyp 2

- Wir wählen `FOREIGN KEY` als Type:.
- Dann fügen wir die Spalte `student` unter Columns hinzu und klicken auf Referenced Columns.



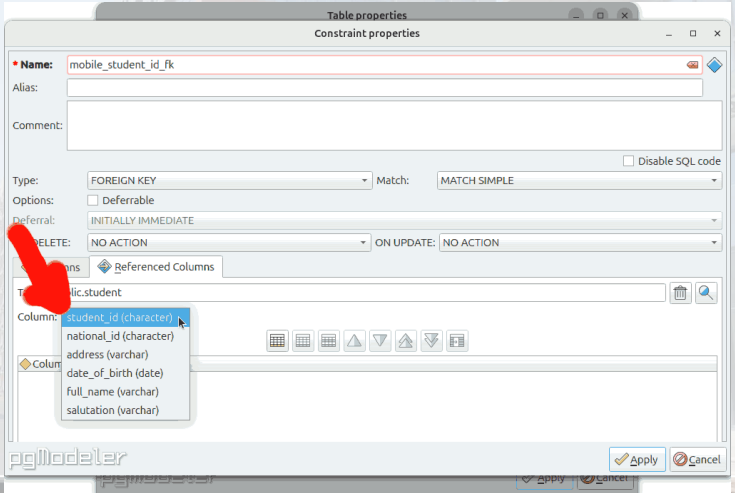
Student Entitätstyp 2

- Dort klicken wir auf Table und wählen die Tabelle `student` in dem Dialog, der sich öffnet, aus.



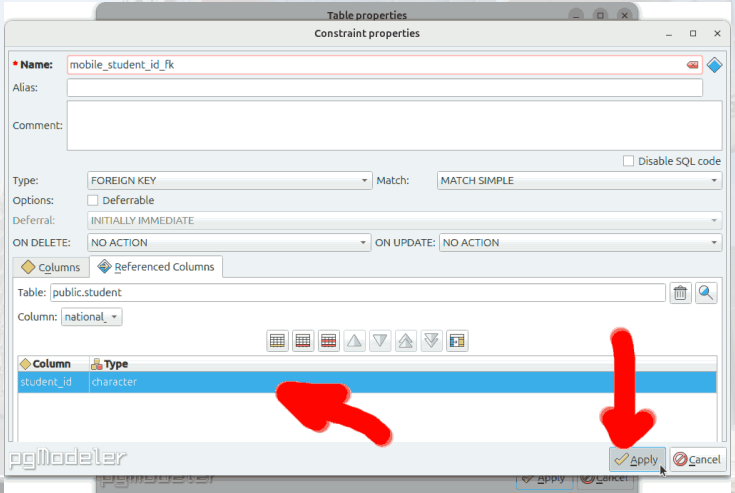
Student Entitätstyp 2

- Dann klicken wir auf Column:, wählen `student_id`, und fügen sie über ▦ hinzu.



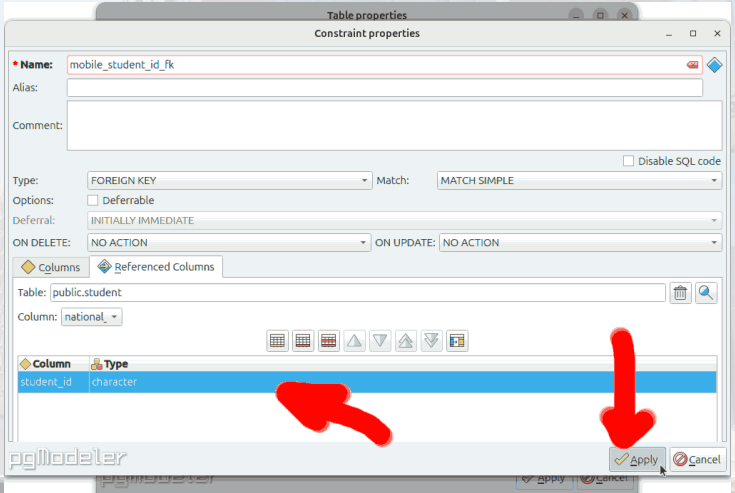
Student Entitätstyp 2

- Sie erscheint in der Spalten-Liste.



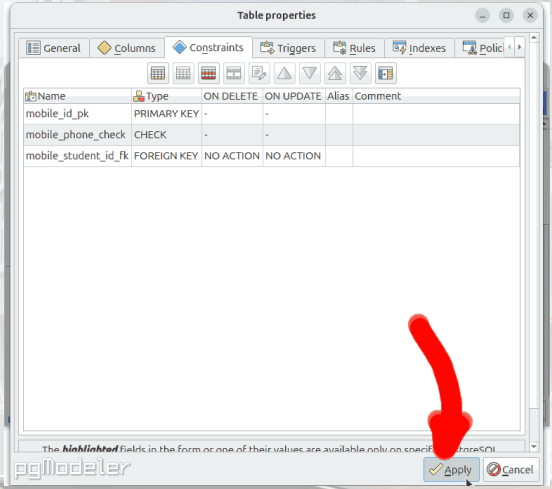
Student Entitätstyp 2

- Sie erscheint in der Spalten-Liste.
- Wir klicken auf Apply.



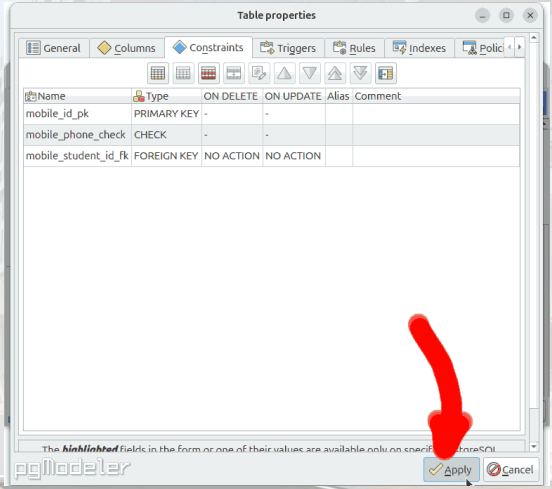
Student Entitätstyp 2

- Wir haben drei Spalten und drei Einschränkungen erstellt.



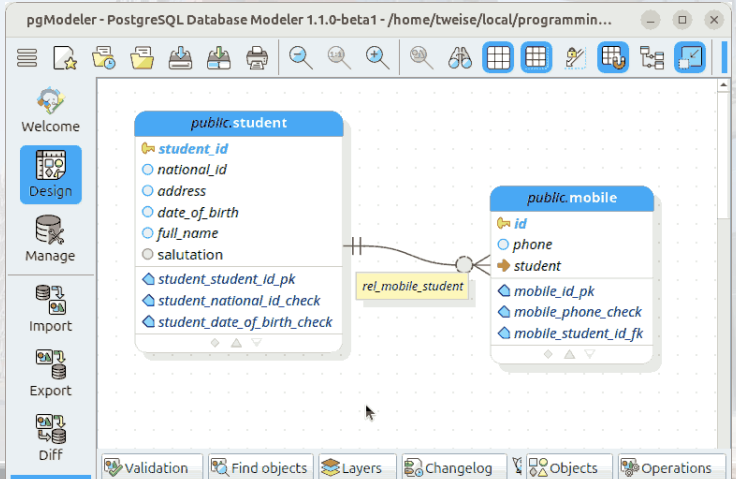
Student Entitätstyp 2

- Wir haben drei Spalten und drei Einschränkungen erstellt.
- Wir klicken auf Apply um die Tabelle zu erstellen.



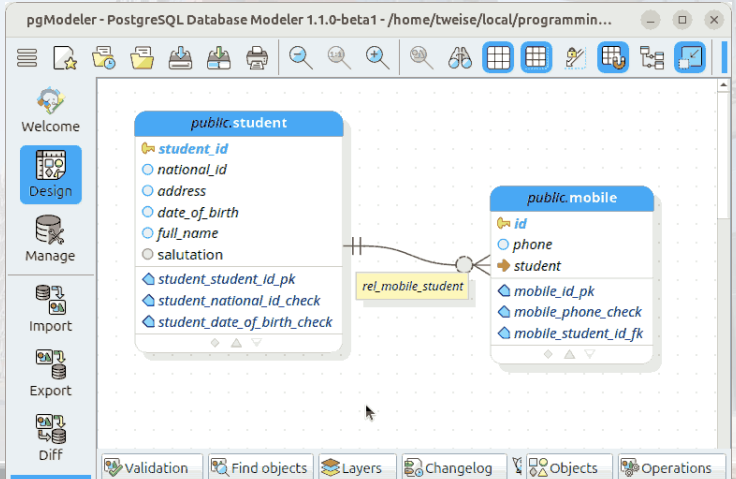
Student Entitätstyp 2

- Wir sehen das Modell in der Krähenfußnotation.



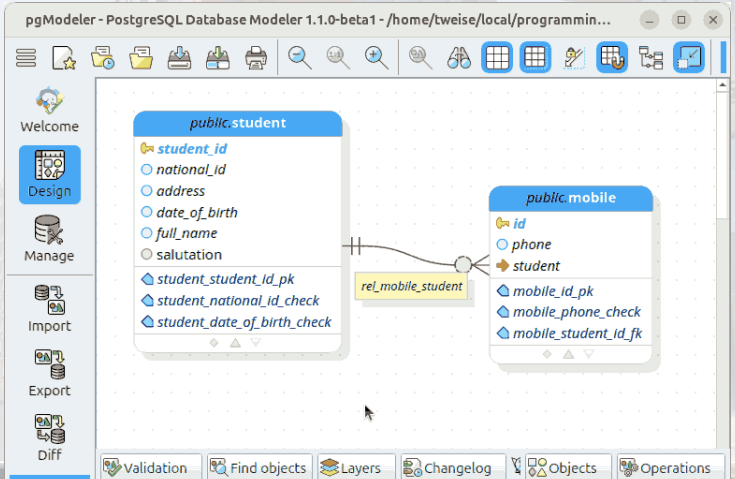
Student Entitätstyp 2

- Wir sehen das Modell in der Krähenfußnotation.
- Wie sehen beide Tabellen.



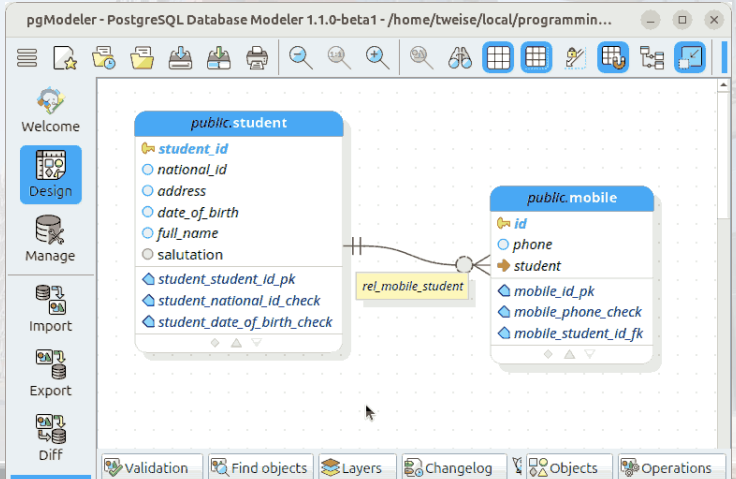
Student Entitätstyp 2

- Wie sehen beide Tabellen.
- Wir sehen, dass jede Zeile der Tabelle `mobile` mit genau einer Zeile der Tabelle `student` verbunden sein muss.



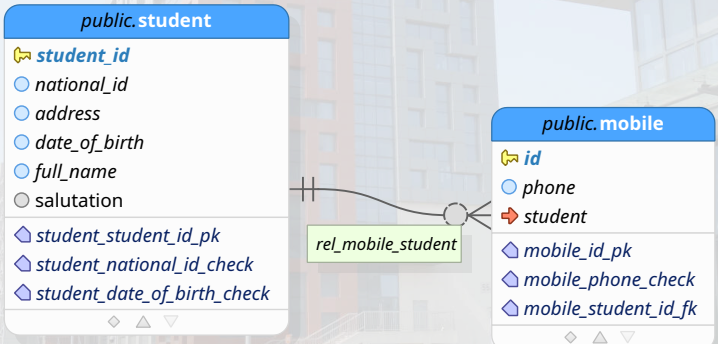
Student Entitätstyp 2

- Wir sehen, dass jede Zeile der Tabelle `student` mit beliebig vielen Zeilen der Tabelle `mobile` verbunden sein kann.



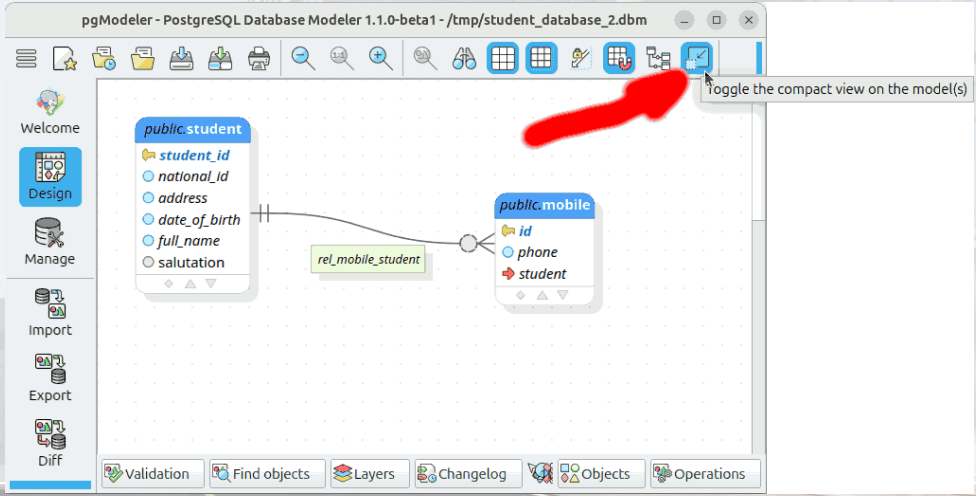
Student Entitätstyp 2

- Wenn wir das Modell als SVG-Grafik exportieren, dann sieht das so aus.



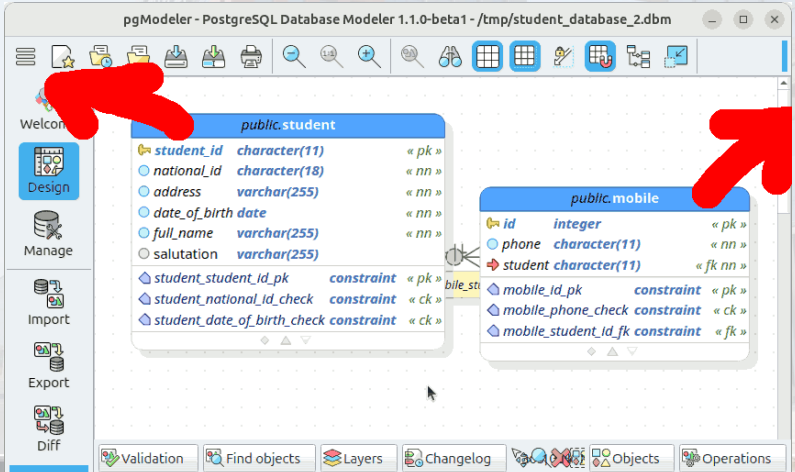
Student Entitätstyp 2

- Um mehr Details sehen zu können, drücken wir auf den -Button in der Werkzeugleiste oben rechts.



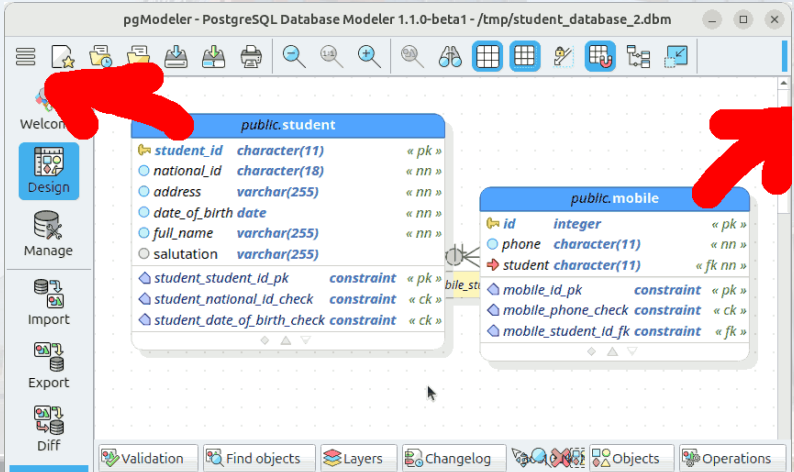
Student Entitätstyp 2

- Mehr Details, wie die Spaltentypen, tauchen auf, so dass die Tabellen sich jetzt teilweise verdecken.



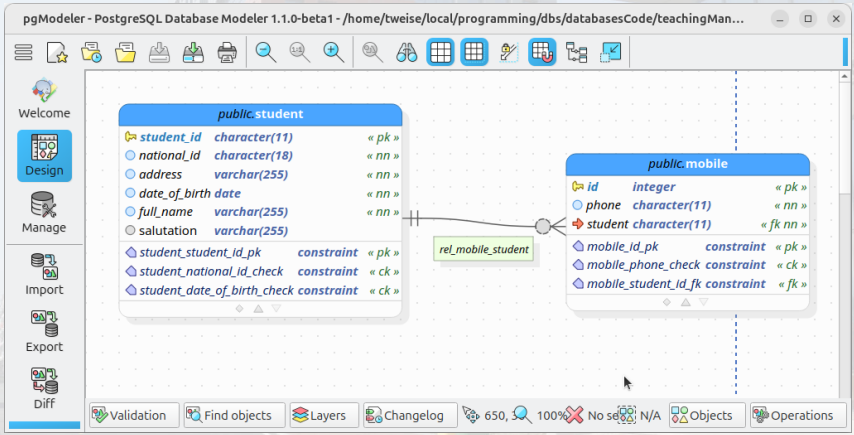
Student Entitätstyp 2

- Mehr Details, wie die Spaltentypen, tauchen auf, so dass die Tabellen sich jetzt teilweise verdecken.
- Wir ziehen sie mit der Maus ein wenig auseinander.



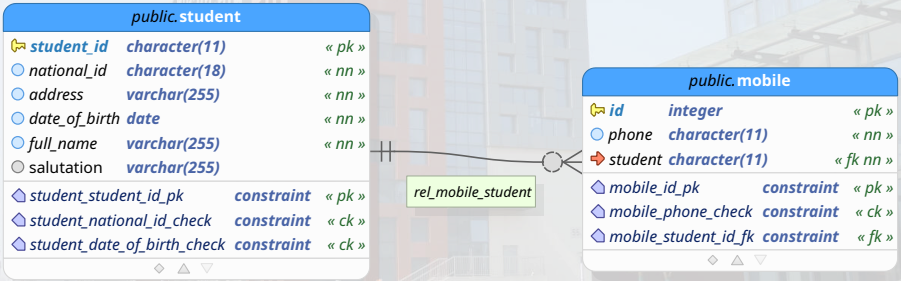
Student Entitätstyp 2

- Das neue Layout sieht viel klarer aus.



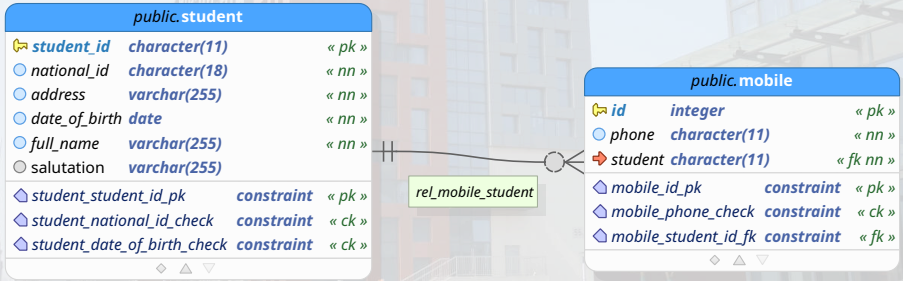
Student Entitätstyp 2

- Wir exportieren es wieder nach SVG exportieren.



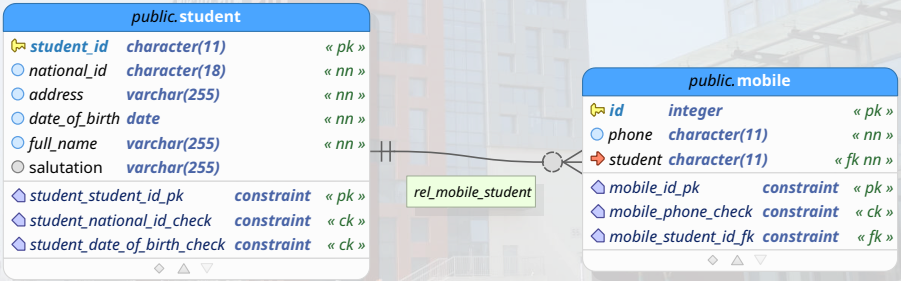
Student Entitätstyp 2

- Wir exportieren es wieder nach SVG exportieren.
- Diese Grafik hat dann auch mehr Details.



Student Entitätstyp 2

- Wir exportieren es wieder nach SVG exportieren.
- Diese Grafik hat dann auch mehr Details.
- Nun exportieren wir das Modell wieder in mehrere SQL-Dateien.



Datenbank Erstellen

- Es wurde ein Skript `01_student_database_database_2001.sql` zum Erstellen der Datenbank generiert.

```
-- object: student_database | type: DATABASE --
-- DROP DATABASE IF EXISTS student_database;
CREATE DATABASE student_database;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf 01
    ↪ _student_database_database_2001.sql
CREATE DATABASE
# psql 16.11 succeeded with exit code 0.
```

Tabelle

- Das auto-generierte Script `03_public_student_table_5071.sql` erstellt die Tabelle `student`.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    student_id character(11) NOT NULL,
    national_id character(18) NOT NULL,
    address varchar(255) NOT NULL,
    date_of_birth date NOT NULL,
    full_name varchar(255) NOT NULL,
    salutation varchar(255),
    CONSTRAINT student_student_id_pk PRIMARY KEY (student_id),
    CONSTRAINT student_national_id_check CHECK (national_id ~ '^\d{6}
        ↪ ((19)|(20))\d{9}[0-9X]$'),
    CONSTRAINT student_date_of_birth_check CHECK ((date_of_birth > '
        ↪ 1900-01-01') AND (date_of_birth < '2100-01-01'))
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf 03_public_student_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle

- Das auto-generierte Script `04_public_mobile_table_5081.sql` erstellt die Tabelle `mobile`.

```
-- object: public.mobile | type: TABLE --
-- DROP TABLE IF EXISTS public.mobile CASCADE;
CREATE TABLE public.mobile (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    phone character(11) NOT NULL,
    student character(11) NOT NULL,
    CONSTRAINT mobile_id_pk PRIMARY KEY (id),
    CONSTRAINT mobile_phone_check CHECK (phone ~ '^\d{11}$')
);
-- ddl-end --
ALTER TABLE public.mobile OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
   ↪ ON_ERROR_STOP=1 -ebf 04_public_mobile_table_5081.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

References Einschränkung

- Es wurde ein Skript 05_public_mobile_mobile_student_id_fk_constraint_5087.sql zum Erstellen der Fremdschlüssel-REFERENCES-Einschränkung generiert.

```
1 -- object: mobile_student_id_fk | type: CONSTRAINT --
2 -- ALTER TABLE public.mobile DROP CONSTRAINT IF EXISTS
    ↪ mobile_student_id_fk CASCADE;
3 ALTER TABLE public.mobile ADD CONSTRAINT mobile_student_id_fk FOREIGN
    ↪ KEY (student)
4 REFERENCES public.student (student_id) MATCH SIMPLE
5 ON DELETE NO ACTION ON UPDATE NO ACTION;
6 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/student_database" -v
    ↪ ON_ERROR_STOP=1 -ebf 05
    ↪ _public_mobile_mobile_student_id_fk_constraint_5087.sql
2 ALTER TABLE
3 # psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Wir probieren die Tabellen aus, in dem wir ein paar Daten einfügen.

```sql
/** Insert data into the student database and join the two tables. */

-- Insert several student records.
INSERT INTO student (student_id, national_id, full_name, salutation,
                     address, date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'The Bib-Man',
     'Hefei, China', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Bebbo Machine',
     'Chemnitz, Germany', '2005-08-07');

-- Insert several mobile phone numbers
INSERT INTO mobile (phone, student) VALUES
    ('11111111111', '1234567890'), ('22222222222', '1234567891'),
    ('33333333333', '1234567890');

-- Print the mobile phone numbers of the students.
SELECT student.full_name, mobile.phone, mobile.id FROM mobile
    INNER JOIN student ON mobile.student = student.student_id;
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
   ↪ ON_ERROR_STOP=1 -ebf insert_and_select.sql
INSERT 0 2
INSERT 0 3
 full_name |    phone    | id
-----------+-------------+----
 Bibbo     | 11111111111 |  1
 Bebbo     | 22222222222 |  2
 Bibbo     | 33333333333 |  3
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Wir probieren die Tabellen aus, in dem wir ein paar Daten einfügen.
- Zuerst fügen wir zwei Datensätze in die Tabelle `student` ein.

```
/** Insert data into the student database and join the two tables. */

-- Insert several student records.
INSERT INTO student (student_id, national_id, full_name, salutation,
                     address, date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'The Bib-Man',
     'Hefei, China', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Bebbo Machine',
     'Chemnitz, Germany', '2005-08-07');

-- Insert several mobile phone numbers
INSERT INTO mobile (phone, student) VALUES
    ('11111111111', '1234567890'), ('22222222222', '1234567891'),
    ('33333333333', '1234567890');

-- Print the mobile phone numbers of the students.
SELECT student.full_name, mobile.phone, mobile.id FROM mobile
    INNER JOIN student ON mobile.student = student.student_id;
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
   ↪ ON_ERROR_STOP=1 -ebf insert_and_select.sql
INSERT 0 2
INSERT 0 3
 full_name |    phone    | id
-----------+-------------+----
 Bibbo     | 11111111111 |  1
 Bebbo     | 22222222222 |  2
 Bibbo     | 33333333333 |  3
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Wir probieren die Tabellen aus, in dem wir ein paar Daten einfügen.
- Zuerst fügen wir zwei Datensätze in die Tabelle `student` ein.
- Danach fügen wir drei Telefonnummern in die Tabelle `mobile` ein.

```
/** Insert data into the student database and join the two tables. */

-- Insert several student records.
INSERT INTO student (student_id, national_id, full_name, salutation,
                     address, date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'The Bib-Man',
     'Hefei, China', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Bebbo Machine',
     'Chemnitz, Germany', '2005-08-07');

-- Insert several mobile phone numbers
INSERT INTO mobile (phone, student) VALUES
    ('11111111111', '1234567890'), ('22222222222', '1234567891'),
    ('33333333333', '1234567890');

-- Print the mobile phone numbers of the students.
SELECT student.full_name, mobile.phone, mobile.id FROM mobile
    INNER JOIN student ON mobile.student = student.student_id;
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
   ↪ ON_ERROR_STOP=1 -ebf insert_and_select.sql
INSERT 0 2
INSERT 0 3
 full_name |    phone    | id
-----------+-------------+----
 Bibbo     | 11111111111 |  1
 Bebbo     | 22222222222 |  2
 Bibbo     | 33333333333 |  3
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Wir probieren die Tabellen aus, in dem wir ein paar Daten einfügen.
- Zuerst fügen wir zwei Datensätze in die Tabelle `student` ein.
- Danach fügen wir drei Telefonnummern in die Tabelle `mobile` ein.
- Danach holen lesen wir die Daten wieder aus

```
/** Insert data into the student database and join the two tables. */

-- Insert several student records.
INSERT INTO student (student_id, national_id, full_name, salutation,
                     address, date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'The Bib-Man',
     'Hefei, China', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Bebbo Machine',
     'Chemnitz, Germany', '2005-08-07');

-- Insert several mobile phone numbers
INSERT INTO mobile (phone, student) VALUES
    ('11111111111', '1234567890'), ('22222222222', '1234567891'),
    ('33333333333', '1234567890');

-- Print the mobile phone numbers of the students.
SELECT student.full_name, mobile.phone, mobile.id FROM mobile
    INNER JOIN student ON mobile.student = student.student_id;
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
   ↪ ON_ERROR_STOP=1 -ebf insert_and_select.sql
INSERT 0 2
INSERT 0 3
 full_name |    phone    | id
-----------+-------------+----
 Bibbo     | 11111111111 |  1
 Bebbo     | 22222222222 |  2
 Bibbo     | 33333333333 |  3
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Wir probieren die Tabellen aus, in dem wir ein paar Daten einfügen.
- Zuerst fügen wir zwei Datensätze in die Tabelle `student` ein.
- Danach fügen wir drei Telefonnummern in die Tabelle `mobile` ein.
- Danach holen lesen wir die Daten wieder aus
- Wir verknüfen beide Tabellen über `INNER JOIN`.

```
/** Insert data into the student database and join the two tables. */

-- Insert several student records.
INSERT INTO student (student_id, national_id, full_name, salutation,
                     address, date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'The Bib-Man',
     'Hefei, China', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Bebbo Machine',
     'Chemnitz, Germany', '2005-08-07');

-- Insert several mobile phone numbers
INSERT INTO mobile (phone, student) VALUES
    ('11111111111', '1234567890'), ('22222222222', '1234567891'),
    ('33333333333', '1234567890');

-- Print the mobile phone numbers of the students.
SELECT student.full_name, mobile.phone, mobile.id FROM mobile
    INNER JOIN student ON mobile.student = student.student_id;
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
   ↪ ON_ERROR_STOP=1 -ebf insert_and_select.sql
INSERT 0 2
INSERT 0 3
 full_name |    phone    | id
-----------+-------------+----
 Bibbo     | 11111111111 |  1
 Bebbo     | 22222222222 |  2
 Bibbo     | 33333333333 |  3
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Wir probieren die Tabellen aus, in dem wir ein paar Daten einfügen.
- Zuerst fügen wir zwei Datensätze in die Tabelle `student` ein.
- Danach fügen wir drei Telefonnummern in die Tabelle `mobile` ein.
- Danach holen lesen wir die Daten wieder aus
- Wir verknüfen beide Tabellen über `INNER JOIN`.
- (Dieses Skript haben wir selber gemacht, das ist nicht von PgModeler.)

```
/** Insert data into the student database and join the two tables. */

-- Insert several student records.
INSERT INTO student (student_id, national_id, full_name, salutation,
                     address, date_of_birth) VALUES
    ('1234567890', '123456199501021234', 'Bibbo', 'The Bib-Man',
     'Hefei, China', '1995-01-02'),
    ('1234567891', '123456200508071234', 'Bebbo', 'Bebbo Machine',
     'Chemnitz, Germany', '2005-08-07');

-- Insert several mobile phone numbers
INSERT INTO mobile (phone, student) VALUES
    ('11111111111', '1234567890'), ('22222222222', '1234567891'),
    ('33333333333', '1234567890');

-- Print the mobile phone numbers of the students.
SELECT student.full_name, mobile.phone, mobile.id FROM mobile
    INNER JOIN student ON mobile.student = student.student_id;
```

```
$ psql "postgres://postgres:XXX@localhost/student_database" -v
   ↪ ON_ERROR_STOP=1 -ebf insert_and_select.sql
INSERT 0 2
INSERT 0 3
 full_name |    phone    | id
-----------+-------------+----
 Bibbo     | 11111111111 |  1
 Bebbo     | 22222222222 |  2
 Bibbo     | 33333333333 |  3
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Datenbank Löschen

- Zu guter Letzt löschen wir die Datenbank wieder.

```
/* Cleanup after the example: Delete the student database. */

DROP DATABASE IF EXISTS student_database;
```

```
$ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf
    ↪ cleanup.sql
DROP DATABASE
# psql 16.11 succeeded with exit code 0.
```

Datenbank Löschen

- Zu guter Letzt löschen wir die Datenbank wieder.
- Beachten Sie, dass wir uns in der Connection-URI nicht auf die Datenbank verbinden ... sonst könnten wir sie ja nicht löschen.

```
1 /* Cleanup after the example: Delete the student database. */
2
3 DROP DATABASE IF EXISTS student_database;
```

```
1 $ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf
    ↪ cleanup.sql
2 DROP DATABASE
3 # psql 16.11 succeeded with exit code 0.
```

Datenbank Löschen

- Zu guter Letzt löschen wir die Datenbank wieder.
- Beachten Sie, dass wir uns in der Connection-URI nicht auf die Datenbank verbinden ... sonst könnten wir sie ja nicht löschen.
- (Dieses Skript haben wir selber gemacht, das ist nicht von PgModeler.)

```
1 /* Cleanup after the example: Delete the student database. */
2
3 DROP DATABASE IF EXISTS student_database;
```

```
1 $ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf
    ↪ cleanup.sql
2 DROP DATABASE
3 # psql 16.11 succeeded with exit code 0.
```

Zusammenfassung

Zusammenfassung: Modellieren

- Wow, das war viel.

Zusammenfassung: Modellieren

- Wow, das war viel.
- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.

Zusammenfassung: Modellieren

- Wow, das war viel.
- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.
- Jeder Entitätstyp wird eine Relation, also eine Tabelle.

Zusammenfassung: Modellieren

- Wow, das war viel.
- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.
- Jeder Entitätstyp wird eine Relation, also eine Tabelle.
- Seine Attribute werden Spalten dieser Tabelle.

Zusammenfassung: Modellieren

- Wow, das war viel.
- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.
- Jeder Entitätstyp wird eine Relation, also eine Tabelle.
- Seine Attribute werden Spalten dieser Tabelle.
- Das gilt allerdings nur für einfache ein-wertige Attribute.

Zusammenfassung: Modellieren

- Wow, das war viel.
- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.
- Jeder Entitätstyp wird eine Relation, also eine Tabelle.
- Seine Attribute werden Spalten dieser Tabelle.
- Das gilt allerdings nur für einfache ein-wertige Attribute.
- Zusammengesetzte Attribute werden auf ihre unteilbaren Komponenten heruntergebrochen und diese Werden dann Spalten.

Zusammenfassung: Modellieren

- Wow, das war viel.
- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.
- Jeder Entitätstyp wird eine Relation, also eine Tabelle.
- Seine Attribute werden Spalten dieser Tabelle.
- Das gilt allerdings nur für einfache ein-wertige Attribute.
- Zusammengesetzte Attribute werden auf ihre unteilbaren Komponenten heruntergebrochen und diese Werden dann Spalten.
- Mehrwertige Attribute müssen ganz aus der Tabelle herausgezogen werden.

Zusammenfassung: Modellieren

- Wow, das war viel.
- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.
- Jeder Entitätstyp wird eine Relation, also eine Tabelle.
- Seine Attribute werden Spalten dieser Tabelle.
- Das gilt allerdings nur für einfache ein-wertige Attribute.
- Zusammengesetzte Attribute werden auf ihre unteilbaren Komponenten heruntergebrochen und diese Werden dann Spalten.
- Mehrwertige Attribute müssen ganz aus der Tabelle herausgezogen werden.
- In relationalen Datenbanken sind alle Attribute einwertig.

Zusammenfassung: Modellieren

- Wow, das war viel.
- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.
- Jeder Entitätstyp wird eine Relation, also eine Tabelle.
- Seine Attribute werden Spalten dieser Tabelle.
- Das gilt allerdings nur für einfache ein-wertige Attribute.
- Zusammengesetzte Attribute werden auf ihre unteilbaren Komponenten heruntergebrochen und diese Werden dann Spalten.
- Mehrwertige Attribute müssen ganz aus der Tabelle herausgezogen werden.
- In relationalen Datenbanken sind alle Attribute einwertig.
- Wir ziehen die mehrwertigen Attribute also in jeweils eine eigene Tabelle.

Zusammenfassung: Modellieren

- Wir haben gelernt, wie wir Entitätstypen vom konzeptuellen Modell in logische Schemas des relationalen Datenmodells übersetzen können.
- Jeder Entitätstyp wird eine Relation, also eine Tabelle.
- Seine Attribute werden Spalten dieser Tabelle.
- Das gilt allerdings nur für einfache ein-wertige Attribute.
- Zusammengesetzte Attribute werden auf ihre unteilbaren Komponenten heruntergebrochen und diese Werden dann Spalten.
- Mehrwertige Attribute müssen ganz aus der Tabelle herausgezogen werden.
- In relationalen Datenbanken sind alle Attribute einwertig.
- Wir ziehen die mehrwertigen Attribute also in jeweils eine eigene Tabelle.
- Diese Tabellen sind dann über Fremdschlüssel mit der „Haupttabelle“ des Entitätstyps verbunden.

Zusammenfassung: Modellieren

- Jeder Entitätstyp wird eine Relation, also eine Tabelle.
- Seine Attribute werden Spalten dieser Tabelle.
- Das gilt allerdings nur für einfache ein-wertige Attribute.
- Zusammengesetzte Attribute werden auf ihre unteilbaren Komponenten heruntergebrochen und diese Werden dann Spalten.
- Mehrwertige Attribute müssen ganz aus der Tabelle herausgezogen werden.
- In relationalen Datenbanken sind alle Attribute einwertig.
- Wir ziehen die mehrwertigen Attribute also in jeweils eine eigene Tabelle.
- Diese Tabellen sind dann über Fremdschlüssel mit der „Haupttabelle“ des Entitätstyps verbunden.
- Mehrere Zeilen dieser Tabelle können auf eine Zeile in der Haupttabelle verweise, wodurch die mehrwertigkeit realisiert wird.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.
- Es erlaubt es uns, logische Modelle für das PostgreSQL DBMS fast genauso zu malen, wie wir es aus yEd gewohnt sind.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.
- Es erlaubt es uns, logische Modelle für das PostgreSQL DBMS fast genauso zu malen, wie wir es aus yEd gewohnt sind.
- Die Modelle, die wir mit yEd gemacht haben, waren Technologie-unabhängige konzeptuelle Modelle.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.
- Es erlaubt es uns, logische Modelle für das PostgreSQL DBMS fast genauso zu malen, wie wir es aus yEd gewohnt sind.
- Die Modelle, die wir mit yEd gemacht haben, waren Technologie-unabhängige konzeptuelle Modelle.
- Mit PgModeler machen wir logische Modelle, die an SQL und das PostgreSQL DBMS gebunden sind.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.
- Es erlaubt es uns, logische Modelle für das PostgreSQL DBMS fast genauso zu malen, wie wir es aus yEd gewohnt sind.
- Die Modelle, die wir mit yEd gemacht haben, waren Technologie-unabhängige konzeptuelle Modelle.
- Mit PgModeler machen wir logische Modelle, die an SQL und das PostgreSQL DBMS gebunden sind.
- Wir können sie wieder als SVG-Grafiken exportieren, genauso, wie wir das mit yEd gemacht haben.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.
- Es erlaubt es uns, logische Modelle für das PostgreSQL DBMS fast genauso zu malen, wie wir es aus yEd gewohnt sind.
- Die Modelle, die wir mit yEd gemacht haben, waren Technologie-unabhängige konzeptuelle Modelle.
- Mit PgModeler machen wir logische Modelle, die an SQL und das PostgreSQL DBMS gebunden sind.
- Wir können sie wieder als SVG-Grafiken exportieren, genauso, wie wir das mit yEd gemacht haben.
- Wir können sie aber auch nach SQL exportieren.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.
- Es erlaubt es uns, logische Modelle für das PostgreSQL DBMS fast genauso zu malen, wie wir es aus yEd gewohnt sind.
- Die Modelle, die wir mit yEd gemacht haben, waren Technologie-unabhängige konzeptuelle Modelle.
- Mit PgModeler machen wir logische Modelle, die an SQL und das PostgreSQL DBMS gebunden sind.
- Wir können sie wieder als SVG-Grafiken exportieren, genauso, wie wir das mit yEd gemacht haben.
- Wir können sie aber auch nach SQL exportieren.
- Und das ist total cool.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.
- Es erlaubt es uns, logische Modelle für das PostgreSQL DBMS fast genauso zu malen, wie wir es aus yEd gewohnt sind.
- Die Modelle, die wir mit yEd gemacht haben, waren Technologie-unabhängige konzeptuelle Modelle.
- Mit PgModeler machen wir logische Modelle, die an SQL und das PostgreSQL DBMS gebunden sind.
- Wir können sie wieder als SVG-Grafiken exportieren, genauso, wie wir das mit yEd gemacht haben.
- Wir können sie aber auch nach SQL exportieren.
- Und das ist total cool.
- Wir können unser Datenbankschema mit einer komfortablen GUI zeichnen.

PgModeler

- Zum Glück haben wir das neue Werkzeug PgModeler kennen gelernt.
- Es erlaubt es uns, logische Modelle für das PostgreSQL DBMS fast genauso zu malen, wie wir es aus yEd gewohnt sind.
- Die Modelle, die wir mit yEd gemacht haben, waren Technologie-unabhängige konzeptuelle Modelle.
- Mit PgModeler machen wir logische Modelle, die an SQL und das PostgreSQL DBMS gebunden sind.
- Wir können sie wieder als SVG-Grafiken exportieren, genauso, wie wir das mit yEd gemacht haben.
- Wir können sie aber auch nach SQL exportieren.
- Und das ist total cool.
- Wir können unser Datenbankschema mit einer komfortablen GUI zeichnen.
- Und dann laden wir es in PostgreSQL.

谢谢您们！

Thank you!

Vielen Dank!

References I

[1] Raphael „rkhaotix“ Araújo e Silva. *pgModeler – PostgreSQL Database Modeler*. Palmas, Tocantins, Brazil, 2006–2025. URL: https://pgmodeler.io (besucht am 2025-04-12) (siehe S. 49–61, 340).

[2] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also[3] (siehe S. 324, 340).

[3] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also[2] (siehe S. 324, 340).

[4] Richard Barker. *Case*Method: Entity Relationship Modelling (Oracle)*. 1. Aufl. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., Jan. 1990. ISBN: 978-0-201-41696-1 (siehe S. 339).

[5] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 339, 341).

[6] Daniel Bartholomew. *Learning the MariaDB Ecosystem: Enterprise-level Features for Scalability and Availability*. New York, NY, USA: Apress Media, LLC, Okt. 2019. ISBN: 978-1-4842-5514-8 (siehe S. 340).

[7] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt (besucht am 2025-02-05) (siehe S. 342).

[8] Tim Berners-Lee, Larry Masinter und Mark P. McCahill. *Uniform Resource Locators (URL)*. Request for Comments (RFC) 1738. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Dez. 1994. URL: https://www.ietf.org/rfc/rfc1738.txt (besucht am 2025-02-05) (siehe S. 342).

[9] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 338).

[10] Grady Booch, James Rumbaugh und Ivar Jacobson. *The Unified Modeling Language Reference Manual*. 1. Aufl. Reading, MA, USA: Addison-Wesley Professional, Jan. 1999. ISBN: 978-0-201-57168-4 (siehe S. 342).

References II

[11] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 340).

[12] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 340).

[13] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 338).

[14] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. Request for Comments (RFC) 8259. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Dez. 2017. URL: https://www.ietf.org/rfc/rfc8259.txt (besucht am 2025-02-05) (siehe S. 339).

[15] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen und Eve Maler, Hrsg. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. Wakefield, MA, USA: World Wide Web Consortium (W3C), 26. Nov. 2008–7. Feb. 2013. URL: http://www.w3.org/TR/2008/REC-xml-20081126 (besucht am 2024-12-15) (siehe S. 342).

[16] Ben Brumm. "A Guide to the Entity Relationship Diagram (ERD)". In: *Database Star*. Armadale, VIC, Australia: Elevated Online Services PTY Ltd., 30. Juli 2019–23. Dez. 2023. URL: https://www.databasestar.com/entity-relationship-diagram (besucht am 2025-03-29) (siehe S. 339).

[17] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 339, 341).

[18] Antonio Cavacini. "Is the CE/BCE notation becoming a standard in scholarly literature?" *Scientometrics* 102(2):1661–1668, Juli 2015. London, England, UK: Springer Nature Limited. ISSN: 0138-9130. doi:10.1007/s11192-014-1352-1 (siehe S. 338).

[19] Josh Centers. *Take Control of iOS 18 and iPadOS 18*. San Diego, CA, USA: Take Control Books, Dez. 2024. ISBN: 978-1-990783-55-5 (siehe S. 339).

[20] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: https://cacm.acm.org/research/50-years-of-queries (besucht am 2025-01-09) (siehe S. 341).

References III

[21] Peter Pin-Shan Chen. "Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned". In: *Software Pioneers: Contributions to Software Engineering*. Hrsg. von Manfred Broy und Ernst Denert. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Feb. 2002, S. 296–310. doi:10.1007/978-3-642-59412-0_17. URL: http://bit.csc.lsu.edu/%7Echen/pdf/Chen_Pioneers.pdf (besucht am 2025-03-06) (siehe S. 339).

[22] Peter Pin-Shan Chen. "The Entity-Relationship Model – Toward a Unified View of Data". *ACM Transactions on Database Systems (TODS)* 1(1):9–36, März 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0362-5915. doi:10.1145/320434.320440 (siehe S. 326, 339).

[23] Peter Pin-Shan Chen. "The Entity-Relationship Model: Toward a Unified View of Data". In: *1st International Conference on Very Large Data Bases (VLDB'1975)*. 22.–24. Sep. 1975, Framingham, MA, USA. Hrsg. von Douglas S. Kerr. New York, NY, USA: Association for Computing Machinery (ACM), 1975, S. 173. ISBN: 978-1-4503-3920-9. doi:10.1145/1282480.1282492. See[22] for a more comprehensive introduction. (Siehe S. 339).

[24] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 341).

[25] Edgar Frank „Ted" Codd. "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM (CACM)* 13(6):377–387, Juni 1970. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/362384.362685. URL: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf (besucht am 2025-01-05) (siehe S. 340).

[26] Coding Gears und Train Your Brain. *YAML Fundamentals for DevOps, Cloud and IaC Engineers*. Birmingham, England, UK: Packt Publishing Ltd, März 2022. ISBN: 978-1-80324-243-9 (siehe S. 342).

[27] Timothy W. Cole und Myung-Ja K. Han. *XML for Catalogers and Metadata Librarians (Third Millennium Cataloging)*. 1. Aufl. Dublin, OH, USA: Libraries Unlimited, 23. Mai 2013. ISBN: 978-1-59884-519-8 (siehe S. 342).

[28] "`csv` – CSV File Reading and Writing". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/csv.html (besucht am 2024-11-14) (siehe S. 338).

References IV

[29] Christopher Cullen. "Learning from Liu Hui? A Different Way to Do Mathematics". *Notices of the American Mathematical Society* 49(7):783–790, Aug. 2002. Providence, RI, USA: American Mathematical Society (AMS). ISSN: 1088-9477. URL: https://www.ams.org/notices/200207/comm-cullen.pdf (besucht am 2024-08-09) (siehe S. 213–219).

[30] Erik Dahlström, Patrick Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, Jonathan Watt, Jon Ferraiolo, Jun Fujisawa und Dean Jackson, Hrsg. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation. Wakefield, MA, USA: World Wide Web Consortium (W3C), 16. Aug. 2011. URL: http://www.w3.org/TR/2011/REC-SVG11-20110816 (besucht am 2024-12-17) (siehe S. 341).

[31] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 341).

[32] Joseph W. Dauben. "Archimedes and Liu Hui on Circles and Spheres". *Ontology Studies (Cuadernos de Ontología)* 10:21–38, 2010. Leioa, Bizkaia, Spain: Universidad del País Vasco / Euskal Herriko Unibertsitatea. ISSN: 1576-2270. URL: https://ddd.uab.cat/pub/ontstu/15762270n10/15762270n10p21.pdf (besucht am 2024-08-10) (siehe S. 213–219).

[33] Matt David und Blake Barnhill. *How to Teach People SQL*. San Francisco, CA, USA: The Data School, Chart.io, Inc., 10. Dez. 2019–10. Apr. 2023. URL: https://dataschool.com/how-to-teach-people-sql (besucht am 2025-02-27) (siehe S. 341).

[34] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 341).

[35] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebW[: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 342).

[36] Ingy döt Net, Tina Müller, Pantelis Antoniou, Eemeli Aro, Thomas Smith, Oren Ben-Kiki und Clark C. Evans. *YAML Ain't Markup Language (YAML™) version 1.2*. Revision 1.2.2. Seattle, WA, USA: YAML Language Development Team, 1. Okt. 2021. URL: https://yaml.org/spec/1.2.2 (besucht am 2025-01-05) (siehe S. 342).

[37] Russell J.T. Dyer. *Learning MySQL and MariaDB*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2015. ISBN: 978-1-4493-6290-4 (siehe S. 340).

References V

[38] *ECMAScript Language Specification*. Standard ECMA-262, 3rd Edition. Geneva, Switzerland: Ecma International, Dez. 1999. URL: https://ecma-international.org/wp-content/uploads/ECMA-262_3rd_edition_december_1999.pdf (besucht am 2024-12-15) (siehe S. 339).

[39] Leonhard Euler. "An Essay on Continued Fractions". Übers. von Myra F. Wyman und Bostwick F. Wyman. *Mathematical Systems Theory* 18(1):295–328, Dez. 1985. New York, NY, USA: Springer Science+Business Media, LLC. ISSN: 1432-4350. doi:10.1007/BF01699475. URL: https://www.researchgate.net/publication/301720080 (besucht am 2024-09-24). Translation of[40]. (Siehe S. 328).

[40] Leonhard Euler. "De Fractionibus Continuis Dissertation". *Commentarii Academiae Scientiarum Petropolitanae* 9:98–137, 1737–1744. Petropolis (St. Petersburg), Russia: Typis Academiae. URL: https://scholarlycommons.pacific.edu/cgi/viewcontent.cgi?article=1070 (besucht am 2024-09-24). See[39] for a translation. (Siehe S. 328, 343).

[41] Luca Ferrari und Enrico Pirozzi. *Learn PostgreSQL*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: 978-1-83763-564-1 (siehe S. 340).

[42] Michael Filaseta. "The Transcendence of e and π". In: *Math 785: Transcendental Number Theory*. Columbia, SC, USA: University of South Carolina, Frühling 2011. Kap. 6. URL: https://people.math.sc.edu/filaseta/gradcourses/Math785/Math785Notes6.pdf (besucht am 2024-07-05) (siehe S. 343).

[43] Kevin P. Gaffney, Martin Prammer, Laurence C. Brasfield, D. Richard Hipp, Dan R. Kennedy und Jignesh M. Patel. "SQLite: Past, Present, and Future". *Proceedings of the VLDB Endowment (PVLDB)* 15(12):3535–3547, Aug. 2022. Irvine, CA, USA: Very Large Data Bases Endowment Inc. ISSN: 2150-8097. doi:10.14778/3554821.3554842. URL: https://www.vldb.org/pvldb/vol15/p3535-gaffney.pdf (besucht am 2025-01-12). All papers in this issue were presented at the *48th International Conference on Very Large Data Bases (VLDB 2022)*, 9 5-9, 2022, hybrid/Sydney, NSW, Australia (siehe S. 341).

[44] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. 3. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juli 2024. ISBN: 978-0-443-23791-1 (siehe S. 340).

[45] Jan L. Harrington. *Relational Database Design and Implementation*. 4. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Apr. 2016. ISBN: 978-0-12-849902-3 (siehe S. 340).

References VI

[46] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 339).

[47] Christian Heimes. "`defusedxml` 0.7.1: XML Bomb Protection for Python stdlib Modules". In: 8. März 2021. URL: https://pypi.org/project/defusedxml (besucht am 2024-12-15) (siehe S. 342).

[48] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: 978-0-13-668539-5 (siehe S. 339, 341).

[49] D. Richard Hipp u. a. "Well-Known Users of SQLite". In: *SQLite*. Charlotte, NC, USA: Hipp, Wyrick & Company, Inc. (Hwaci), 2. Jan. 2023. URL: https://www.sqlite.org/famous.html (besucht am 2025-01-12) (siehe S. 341).

[50] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 340).

[51] *IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX(TM))--Part 2: Shell and Utilities*. IEEE Std 1003.2-1992. New York, NY, USA: Institute of Electrical and Electronics Engineers (IEEE), 23. Juni 1993. URL: https://mirror.math.princeton.edu/pub/oldlinux/Linux.old/Ref-docs/POSIX/all.pdf (besucht am 2025-03-27). Board Approved: 1992-09-17, ANSI Approved: 1993-04-05. See unapproved draft IEEE P1003.2 Draft 11.2 of 9 1991 at the url (siehe S. 340).

[52] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip (besucht am 2025-01-08). Consists of several parts, see https://modern-sql.com/standard for information where to obtain them. (Siehe S. 341).

[53] *Information Technology – Universal Coded Character Set (UCS)*. International Standard ISO/IEC 10646:2020. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Dez. 2020 (siehe S. 336, 342).

References VII

[54] Arthur Jones, Kenneth R. Pearson und Sidney A. Morris. "Transcendence of e and π". In: *Abstract Algebra and Famous Impossibilities*. Universitext (UTX). New York, NY, USA: Springer New York, 1991. Kap. 9, S. 115–161. ISSN: 0172-5939. ISBN: 978-1-4419-8552-1. doi:10.1007/978-1-4419-8552-1_8 (siehe S. 343).

[55] Shen Kangshen, John Newsome Crossley und Anthony W.-C. Lun. *The Nine Chapters on the Mathematical Art: Companion and Commentary*. Oxford, Oxfordshire, England, UK: Oxford University Press, 7. Okt. 1999. ISBN: 978-0-19-853936-0. doi:10.1093/oso/9780198539360.001.0001 (siehe S. 213–219).

[56] Shannon Kempe und Paul Williams. *A Short History of the ER Diagram and Information Modeling*. Studio City, CA, USA: Dataversity Digital LLC, 25. Sep. 2012. URL: https://www.dataversity.net/a-short-history-of-the-er-diagram-and-information-modeling (besucht am 2025-03-06) (siehe S. 339).

[57] Katie Kodes. *Intro to XML, JSON, & YAML*. London, England, UK: Payhip, 2019–4. Sep. 2020 (siehe S. 342).

[58] Andrew M. Kuchling. *Python 3 Documentation. Regular Expression HOWTO*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/howto/regex.html (besucht am 2024-11-01) (siehe S. 340).

[59] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 341).

[60] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 340).

[61] Gloria Lotha, Aakanksha Gaur, Erik Gregersen, Swati Chopra und William L. Hosch. "Client-Server Architecture". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 3. Jan. 2025. URL: https://www.britannica.com/technology/client-server-architecture (besucht am 2025-01-20) (siehe S. 338).

[62] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 340).

[63] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: https://mariadb.com/kb/en/documentation (besucht am 2025-04-24) (siehe S. 340).

References VIII

[64] "Mathematical Functions and Operators". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.3. URL: https://www.postgresql.org/docs/17/functions-math.html (besucht am 2025-02-27) (siehe S. 343).

[65] Michael McLaughlin. *MySQL Workbench: Data Modeling & Development*. Oracle Press. New York, NY, USA: McGraw-Hill, 9. Apr. 2013. ISBN: 978-0-07-179188-5 (siehe S. 49–61, 340).

[66] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISBN: 978-1-55860-456-8 (siehe S. 341).

[67] Zsolt Nagy. *Regex Quick Syntax Reference: Understanding and Using Regular Expressions*. New York, NY, USA: Apress Media, LLC, Aug. 2018. ISBN: 978-1-4842-3876-9 (siehe S. 340).

[68] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 338).

[69] Thomas Nield. *An Introduction to Regular Expressions*. Sebastopol, CA, USA: O'Reilly Media, Inc., Juni 2019. ISBN: 978-1-4920-8255-2 (siehe S. 340).

[70] Ivan Niven. "The Transcendence of π". *The American Mathematical Monthly* 46(8):469–471, Okt. 1939. London, England, UK: Taylor and Francis Ltd. ISSN: 1930-0972. doi:10.2307/2302515 (siehe S. 343).

[71] John J. O'Connor und Edmund F. Robertson. *Liu Hui*. St Andrews, Scotland, UK: University of St Andrews, School of Mathematics and Statistics, Dez. 2003. URL: https://mathshistory.st-andrews.ac.uk/Biographies/Liu_Hui (besucht am 2024-08-10) (siehe S. 213–219).

[72] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 340).

[73] *OMG® Unified Modeling Language® (OMG UML®). Version 2.5.1*. OMG Document formal/2017-12-05. Milford, MA, USA: Object Management Group, Inc. (OMG), Dez. 2017. URL: https://www.omg.org/spec/UML/2.5.1/PDF (besucht am 2025-03-30) (siehe S. 342).

[74] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 338).

References IX

[75] "POSIX Regular Expressions". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.7.3. URL: https://www.postgresql.org/docs/17/functions-matching.html#FUNCTIONS-POSIX-REGEXP (besucht am 2025-02-27) (siehe S. 340).

[76] *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), Feb. 2025. URL: https://www.postgresql.org/docs/17/index.html (besucht am 2025-02-25).

[77] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 340).

[78] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: 978-1-78883-546-6 (siehe S. 338).

[79] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 340).

[80] "`re` – Regular Expression Operations". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/re.html#module-re (besucht am 2024-11-01) (siehe S. 340).

[81] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: 2577-1647. ISBN: 978-1-6654-3554-3. doi:10.1109/IECON48115.2021.9589382 (siehe S. 340).

[82] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: 978-1-4920-4345-4 (siehe S. 338).

[83] Heinz Schweppe und Manuel Scholz. "Schema Design: Logical Design using the Relational Data Model". In: *Einführung in die Datenbanksysteme. Datenbanken für die Bioinformatik*. Berlin, Germany: Freie Universität Berlin, Apr.–Okt. 2005. Kap. 3. URL: https://www.inf.fu-berlin.de/lehre/SS05/19517-V/FolienEtc/dbs05-04-RDM1-2.pdf (besucht am 2025-04-04) (siehe S. 10–12).

References X

[84] Matthias Sedlmeier und Martin Gogolla. "Model Driven ActiveRecord with yEd". In: *25th International Conference on Information Modelling and Knowledge Bases XXVII (EJC'2015)*. 8.–12. Juni 2015, Maribor, Štajerska, Podravska, Slovenia. Hrsg. von Tatjana Welzer, Hannu Jaakkola, Bernhard Thalheim, Yasushi Kiyoki und Naofumi Yoshida. Bd. 280 der Reihe Frontiers in Artificial Intelligence and Applications. Amsterdam, The Netherlands: IOS Press BV, 2015, S. 65–76. ISSN: 0922-6389. ISBN: 978-1-61499-610-1. doi:10.3233/978-1-61499-611-8-65 (siehe S. 343).

[85] Syamal K. Sen und Ravi P. Agarwal. "Existence of year zero in astronomical counting is advantageous and preserves compatibility with significance of AD, BC, CE, and BCE". In: *Zero – A Landmark Discovery, the Dreadful Void, and the Ultimate Mind*. Amsterdam, The Netherlands: Elsevier B.V., 2016. Kap. 5.5, S. 94–95. ISBN: 978-0-08-100774-7. doi:10.1016/C2015-0-02299-7 (siehe S. 338).

[86] Yakov Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. Request for Comments (RFC) 4180. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Okt. 2005. URL: https://www.ietf.org/rfc/rfc4180.txt (besucht am 2025-02-05) (siehe S. 338).

[87] Yuriy Shamshin. "Conceptual Database Model. Entity Relationship Diagram (ERD)". In: *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. Kap. 04. URL: https://dbs.academy.lv/lection/dbs_LS04EN_erd.pdf (besucht am 2025-03-29) (siehe S. 339).

[88] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 339).

[89] Bryan Sills, Brian Gardner, Kristin Marsicano und Chris Stewart. *Android Programming: The Big Nerd Ranch Guide*. 5. Aufl. Reading, MA, USA: Addison-Wesley Professional, Mai 2022. ISBN: 978-0-13-764579-4 (siehe S. 338).

[90] Drew Smith. *Modern Apple Platform Administration – macOS and iOS Essentials (2025)*. Birmingham, England, UK: Packt Publishing Ltd, Feb. 2025. ISBN: 978-1-80580-309-6 (siehe S. 339, 340).

[91] John Miles Smith und Philip Yen-Tang Chang. "Optimizing the Performance of a Relational Algebra Database Interface". *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 340).

References XI

[92] *SQLite*. Charlotte, NC, USA: Hipp, Wyrick & Company, Inc. (Hwaci), 2025. URL: https://sqlite.org (besucht am 2025-04-24) (siehe S. 341).

[93] "SQL Commands". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: https://www.postgresql.org/docs/17/sql-commands.html (besucht am 2025-02-25) (siehe S. 341).

[94] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Tech Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 334, 341).

[95] Ryan K. Stephens, Ronald R. Plew, Bryan Morgan und Jeff Perkins. *SQL in 21 Tagen. Die Datenbank-Abfragesprache SQL vollständig erklärt (in 14/21 Tagen)*. 6. Aufl. Burgthann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of[94] (siehe S. 341).

[96] Philip D. Straffin Jr. "Liu Hui and the First Golden Age of Chinese Mathematics". *Mathematics Magazine* 71(3):163–181, Juni 1998. London, England, UK: Taylor and Francis Ltd. ISSN: 0025-570X. doi:10.2307/2691200. URL: https://www.researchgate.net/publication/237334342 (besucht am 2024-08-10) (siehe S. 213–219).

[97] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 340, 341).

[98] Alkin Tezuysal und Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2024. ISBN: 978-1-80323-347-5 (siehe S. 340).

[99] The Editors of Encyclopaedia Britannica, Hrsg. *Encyclopaedia Britannica*. Chicago, IL, USA: Encyclopædia Britannica, Inc.

[100] *The JSON Data Interchange Syntax*. Standard ECMA-404, 2nd Edition. Geneva, Switzerland: Ecma International, Dez. 2017. URL: https://ecma-international.org/publications-and-standards/standards/ecma-404 (besucht am 2024-12-15) (siehe S. 339).

[101] *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library (besucht am 2025-04-27).

References XII

[102] *The Unicode Standard, Version 15.1: Archived Code Charts*. South San Francisco, CA, USA: The Unicode Consortium, 25. Aug. 2023. URL: https://www.unicode.org/Public/15.1.0/charts/CodeCharts.pdf (besucht am 2024-07-26) (siehe S. 342).

[103] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 339).

[104] *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. Aufl. The Addison-Wesley Object Technology Series. Reading, MA, USA: Addison-Wesley Professional, Sep. 2003. ISBN: 978-0-321-19368-1 (siehe S. 342).

[105] *UML Notation Guide*. Version 1.1. Santa Clara, CA, USA: Rational Software Corporation, Redmond, WA, USA: Microsoft Corporation, Palo Alto, CA, USA: Hewlett-Packard Company, Redwood Shores, CA, USA: Oracle Corporation, Dallas, TX, USA: Sterling Software, Ottawa, ON, Canada: MCI Systemhouse Corporation, Blue Bell, PA, USA: Unisys Corporation, Blue Bell, PA, USA: ICON Computing, Santa Clara, CA, USA: IntelliCorp, Burlington, MA, USA: i-Logix, Armonk, NY, USA: International Business Machines Corporation (IBM), Kanata, ON, Canada: ObjecTime Limited, Chicago, IL, USA: Platinum Technology Inc., Boston, MA, USA: Ptech Inc., Orlando, FL, USA: Taskon A/S, Paoli, PA, USA: Reich Technologies und Paris, Île-de-France, France: Softeam, 1. Sep. 1997. URL: https://web.cse.msu.edu/~cse870/Materials/uml-notation-guide-9-97.pdf (besucht am 2025-03-30) (siehe S. 342).

[106] *Unicode®15.1.0*. South San Francisco, CA, USA: The Unicode Consortium, 12. Sep. 2023. ISBN: 978-1-936213-33-7. URL: https://www.unicode.org/versions/Unicode15.1.0 (besucht am 2024-07-26) (siehe S. 342).

[107] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 339).

[108] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2025. URL: https://thomasweise.github.io/databases (besucht am 2025-01-05) (siehe S. 338, 340).

[109] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 340).

References XIII

[110] Matthew West. *Developing High Quality Data Models*. Version: 2.0, Issue: 2.1. London, England, UK: Shell International Limited und European Process Industries STEP Technical Liaison Executive (EPISTLE); Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 8. Dez. 1995–Dez. 2010. ISBN: 978-0-12-375107-2. URL: https://www.researchgate.net/publication/286610894 (besucht am 2025-03-24). Edited by Julian Fowler (siehe S. 339).

[111] *What is a Relational Database?* Armonk, NY, USA: International Business Machines Corporation (IBM), 20. Okt. 2021–12. Dez. 2024. URL: https://www.ibm.com/think/topics/relational-databases (besucht am 2025-01-05) (siehe S. 340).

[112] Ulf Michael „Monty" Widenius, David Axmark und Uppsala, Sweden: MySQL AB. *MySQL Reference Manual – Documentation from the Source*. Sebastopol, CA, USA: O'Reilly Media, Inc., 9. Juli 2002. ISBN: 978-0-596-00265-7 (siehe S. 340).

[113] Marianne Winslett und Vanessa Braganholo. "Richard Hipp Speaks Out on SQLite". *ACM SIGMOD Record* 48(2):39–46, Juni 2019. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0163-5808. doi:10.1145/3377330.3377338 (siehe S. 341).

[114] Kinza Yasar und Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., Juni 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (besucht am 2025-01-11) (siehe S. 339).

[115] *yEd Graph Editor Manual*. Tübingen, Baden-Württemberg, Germany: yWorks GmbH, 2011–2025. URL: https://yed.yworks.com/support/manual/index.html (besucht am 2025-03-31) (siehe S. 343).

[116] François Yergeau. *UTF-8, A Transformation Format of ISO 10646*. Request for Comments (RFC) 3629. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Nov. 2003. URL: https://www.ietf.org/rfc/rfc3629.txt (besucht am 2025-02-05). See Unicode and[53] (siehe S. 342).

[117] Wenqi Ying (应雯棋), Hrsg. *Commemoration of Ancient Chinese Mathematical Master Liu Hui for his Timeless Influence on Mathematics and Civilizational Exchange*. Bd. 48 (Special Issue) der Reihe CAST Newsletter. China, Beijing (中国北京市): 中国科学技术协会 (China Association for Science and Technology, CAST), Nov. 2024. URL: https://english.cast.org.cn/cms_files/filemanager/1941250207/attach/202412/8f23655a82364d19ad7874eb37b23035.pdf (besucht am 2025-08-24). Proofreader: Yumeng Wei (魏雨萌), Designer: Shan Zhang (张珊) (siehe S. 213–219).

[118] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 338).

References XIV

[119] Nicola Abdo Ziadeh, Michael B. Rowton, A. Geoffrey Woodhead, Wolfgang Helck, Jean L.A. Filliozat, Hiroyuki Momo, Eric Thompson, E.J. Wiesenberg und Shih-ch'ang Wu. "Chronology – Christian History, Dates, Events". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 26. Juli 1999–20. März 2024. URL: https://www.britannica.com/topic/chronology/Christian (besucht am 2025-08-27) (siehe S. 338).

[120] 公民身份号码 *(Citizen Identification Number)*. 中华人民共和国国家标准 (National Standard of the People's Republic of China, GB) GB11643-1999. China, Beijing (中国北京市): 中华人民共和国国家质量监督检验检疫总局 (General Administration of Quality Supervision, Inspection and Quarantine of the People's Republic of China), 中国国家标准化管理委员会 (Standardization Administration of the People's Republic of China, SAC) und 中国标准出版社 (Standards Press of China), 19. Jan.–3. Nov. 1999. URL: https://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=080D6FBF2BB468F9007657F26D60013E (besucht am 2024-07-26) (siehe S. 63–91).

[121] "手机号码：电话管理部门为手机设定的号码 (Mobile Phone Number: A Number Set by the Telephone Management Department for Mobile Phones)". In: 百度百科 *(Baidu Baike)*. China, Beijing (中国北京市): Baidu (百度公司), 6. Jan. 2025. URL: https://baike.baidu.com/item/%E6%89%8B%E6%9C%BA%E5%8F%B7%E7%A0%81 (besucht am 2025-04-17) (siehe S. 63–107).

Glossary (in English) I

Android is a common operating system for mobile phones[89].

Bash is a the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs[13,68,118]. Learn more at `https://www.gnu.org/software/bash`.

BCE The time notation *before Common Era* is a non-religious but chronological equivalent alternative to the traditional *Before Christ (BC)* notation, which refers to the years *before* the birth of Jesus Christ[18]. The years BCE are counted down, i.e., the larger the year, the farther in the past. The year 1 BCE comes directly before the year 1 Common Era (CE)[85,119].

CE The time notation *Common Era* is a non-religious but chronological equivalent alternative to the traditional *Anno Domini (AD)* notation, which refers to the years *after* the birth of Jesus Christ[18]. The years CE are counted upwards, i.e., the smaller they are, the farther they are in the past. The year 1 CE comes directly after the year 1 before Common Era (BCE)[85,119].

client In a client-server architecture, the client is a device or process that requests a service from the server. It initiates the communication with the server, sends a request, and receives the response with the result of the request. Typical examples for clients are web browsers in the internet as well as clients for database management systems (DBMSes), such as psql.

client-server architecture is a system design where a central server receives requests from one or multiple clients[9,61,74,78,82]. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of database (DB) software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

CSV *Comma-Separated Values* is a very common and simple text format for exchanging tabular or matrix data[86]. Each row in the text file represents one row in the table or matrix. The elements in the row are separated by a fixed delimiter, usually a comma („,“), sometimes a semicolon („;“). Python offers some out-of-the-box CSV support in the `csv` module[28].

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*[108].

Glossary (in English) II

DBMS	A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB[114].
DBS	A *database system* is the combination of a DB and a the corresponding DBMS, i.e., basically, an installation of a DBMS on a computer together with one or multiple DBs. DBS = DB + DBMS.
DOB	Date of Birth
ERD	Entity relationship diagrams show the relationships between objects, e.g., between the tables in a DB and how they reference each other[4,16,21–23,56,87,110]
GUI	graphical user interface
iOS	is the operating system that powers Apple iPhones[19,90]. Learn more at https://www.apple.com/ios.
iPadOS	is the operating system that powers Apple iPads[19]. Learn more at https://www.apple.com/ipados.
IT	information technology
JavaScript	JavaScript is the predominant programming language used in websites to develop interactive contents for display in browsers[38].
JSON	*JavaScript Object Notation* is a data interchange format[14,100] based on JavaScript[38] syntax.
LAMP Stack	A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP[17,48].
Linux	is the leading open source operating system, i.e., a free alternative for Microsoft Windows[5,46,88,103,107]. We recommend using it for this course, for software development, and for research. Learn more at https://www.linux.org. Its variant Ubuntu is particularly easy to use and install.

Glossary (in English) III

macOS	or Mac OS is the operating system that powers Apple Mac(intosh) computers[90]. Learn more at https://www.apple.com/macos.
MariaDB	An open source relational database management system that has forked off from MySQL[2,3,6,37,63,79]. See https://mariadb.org for more information.
Microsoft Windows	is a commercial proprietary operating system[12]. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at https://www.microsoft.com/windows.
MySQL	An open source relational database management system[11,37,81,98,112]. MySQL is famous for its use in the LAMP Stack. See https://www.mysql.com for more information.
MySQL Workbench	is a visual tool for DB designers that offers tools ranging from graphical modeling to performance analysis[65]. Learn more at https://www.mysql.com/products/workbench.
OS	Operating System, the system that runs your computer, see, e.g., Linux, Microsoft Windows, macOS, and Android.
PgModeler	the PostgreSQL DB modeler is a tool that allows for graphical modeling of logical schemas for DBs using an entity relationship diagram (ERD)-like notation[1]. Learn more at https://pgmodeler.io.
PostgreSQL	An open source object-relational DBMS[41,72,77,98]. See https://postgresql.org for more information.
psql	is the client program used to access the PostgreSQL DBMS server.
Python	The Python programming language[50,60,62,109], i.e., what you will learn about in our book[109]. Learn more at https://python.org.
regex	A *Regular Expression*, often called „regex“ for short, is a sequence of characters that defines a search pattern for text strings[51,58,67,69]. In Python, the `re` module offers functionality work with regular expressions[58,80]. In PostgreSQL, regex-based pattern matching is supported as well[75].
relational database	A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other[25,44,45,91,97,108,111].

Glossary (in English) IV

server In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carring the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples for servers are web servers[17] in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“[59].

SQL The *Structured Query Language* is basically a programming language for querying and manipulating relational databases[20,31,33,34,52,66,93–95,97]. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference[93].

SQLite is an relational DBMS which runs as in-process library that works directly on files as opposed to the client-server architecture used by other common DBMSes. It is the most wide-spread SQL-based DB in use today, installed in nearly every smartphone, computer, web browser, television, and automobile[20,43,49,113]. Learn more at https://sqlite.org[92].

SVG The Scalable Vector Graphics (SVG) format is an Extensible Markup Language (XML)-based format for vector graphics[30]. Vector graphics are composed of geometric shapes like lines, rectangles, circles, and text. As opposed to raster / pixel graphics, they can be scaled seamlessly and without artifacts. They are stored losslessly.

terminal A terminal is a text-based window where you can enter commands and execute them[5,24]. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf ⊞+R, dann Schreiben von `cmd`, dann Druck auf ↵. Under Ubuntu Linux, Ctrl+Alt+T opens a terminal, which then runs a Bash shell inside.

Ubuntu is a variant of the open source operating system Linux[24,48]. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at https://ubuntu.com. If you are in China, you can download it from https://mirrors.ustc.edu.cn/ubuntu-releases.

UCS Universal Coded Character Set, see Unicode

Glossary (in English) V

UML The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems[10,73,104,105]

Unicode A standard for assigning characters to numbers[53,102,106]. The Unicode standard supports basically all characters from all languages that are currently in use, as well as many special symbols. It is the predominantly used way to represent characters in computers and is regularly updated and improved.

URI A *Uniform Resource Identifier* is an identifier for an abstract or physical resource in the internet[116]. It can be a Uniform Resource Locator (URL), a name, or both. URIs are supersets of URLs. The connection strings of the PostgreSQL DBMS are examples for URIs.

URL A *Uniform Resource Locator* identifies a resource in the WWW and a way to obtain it by describing a network access mechanism. The most notable example of URLs is the text you write into web browsers to visit websites[8]. URLs are subsets of Uniform Resource Identifiers (URIs).

UTF-8 The *UCS Transformation Format 8* is one standard for encoding Unicode characters into a binary format that can be stored in files[53,116]. It is the world wide web's most commonly used character encoding, where each character is represented by one to four bytes. It is backwards compatible with ASCII.

WWW World Wide Web[7,35]

XML The *Extensible Markup Language* is a text-based language for storing and transporting of data[15,27,57]. It allows you to define elements in the form `<myElement myAttr="x">...text..</myElement>`. Different from comma-separated values (CSV), elements in XML can be hierarchically nested, like `<a><b><c>test</c></b><b>bla</b></a>`, and thus easily represent tree structures. XML is one of most-used data interchange formats. To process XML in Python, use the `defusedxml` library[47], as it protects against several security issues.

YAML *YAML Ain't Markup Language™* is a human-friendly data serialization language for all programming languages[26,36,57]. It is widely used for configuration files in the DevOps environment. See `https://yaml.org` for more information.

Glossary (in English) VI

yEd is a graph editor for high-quality graph-based diagrams[84,115], suitable to draw, e.g., technology-independent ERDs, control flow charts, or Unified Modeling Language (UML) class diagrams. An online version of the editor is available at https://www.yworks.com/yed-live. Learn more at https://www.yworks.com/products/yed.

π is the ratio of the circumference U of a circle and its diameter d, i.e., $\pi = U/d$. $\pi \in \mathbb{R}$ is an irrational and transcendental number[42,54,70], which is approximately $\pi \approx 3.141\,592\,653\,589\,793\,238\,462\,643$. In Python, it is provided by the `math` module as constant `pi` with value `3.141592653589793`. In PostgreSQL, it is provided by the SQL function `pi()` with value `3.141592653589793`[64].

e is Euler's number[40], the base of the natural logarithm. $e \in \mathbb{R}$ is an irrational and transcendental number[42,54], which is approximately $e \approx 2.718\,281\,828\,459\,045\,235\,360$. In Python, it is provided by the `math` module as constant `e` with value `2.718281828459045`. In PostgreSQL, you can obtain it via the SQL function `exp(1)` as value `2.718281828459045`[64].

$\mathbb{R}$ the set of the real numbers.