programmierung und datenbanken

Joern Ploennigs

Datenbankentwurf

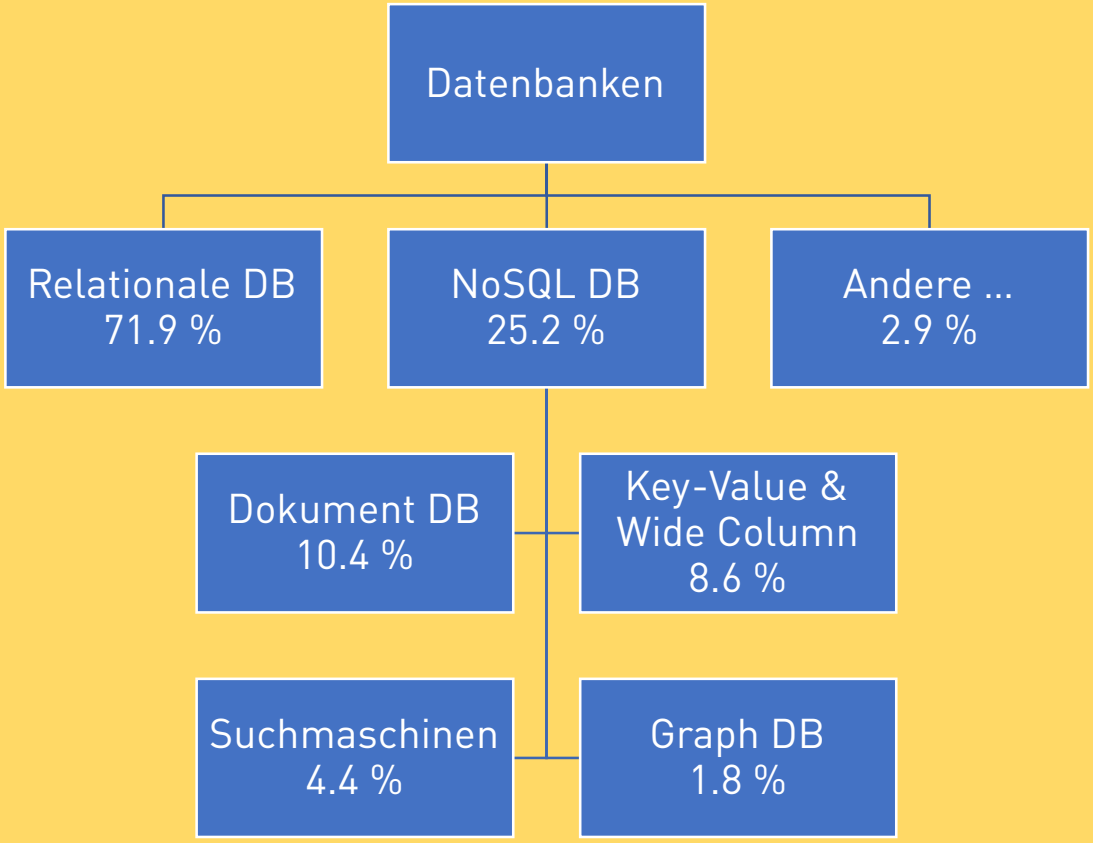
MIDJOURNEY: A WALK IN THE PARK, REF. GEORGES SEURAT

Wiederholung: Hörsaalfrage

Welche Arten von Datenbanken gibt es?

Midjourney: Data Bank

Wiederholung: Datenbanktypen



Wiederholung: Hörsaalfrage

Was sind Relationalen Datenbanken?

Midjourney: Relational Database

Wiederholung: Relationale Datenbanken

RDBMS werden bereits seit Anfang der 1980er Jahre verwendet und basieren auf dem relationalen (=tabellenorientierten) Datenmodell

- Das Schema einer Tabelle (=Relationenschema) ist definiert durch den Tabellennamen und eine fixe Anzahl von Attributen (=Spalten) mit entsprechenden Datentypen
- Da Daten in Tabellen organisiert werden, sind sie stark strukturiert mit einer durch die Tabelle definierten Struktur (Normalisierung)
- Die Standardsprache zum Aufbau/Änderung/Löschen ist *SQL*
- Populäre Systeme: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, IBM Db2

Wiederholung: Hörsaalfrage

Was für Schlüsseltypen gibt es in Relationalen Datenbanken?

Midjourney: Key on a Keyring

Wiederholung: Verweise über Schlüssel

Gemeinden Tabelle

GemeindeID	Name	Einwohner
1	Dummerstorf	7.329
2	Graal-Müritz	4.278
3	Sanitz	5.831

Primärschlüssel: GemeindeID

Bauwerke Tabelle

BauwerksID	Bauwerkstyp	GemeindeID
5000	Tankstelle	2
5001	Hotel	1
5002	Kirche	2

Fremdschlüssel: GemeindeID

Ablauf


MODELLIERUNG
Fehler und Debugging
Objektorientierung u. Softwareentwurf
Funktionen und Rekursion
DATENBANKEN
Datenhaltung
Datenbanken
Datenbankentwurf
Trends und KI

Entwurfsvorgehen bei Datenbanken

Das klassische Entwurfsvorgehen ähnelt dem Wasserfallmodell beim Softwareentwurf

1. *Anforderungsanalyse*: Welche Anforderungen und Anwendungsfälle stellen sich an die DB?
2. *Konzeptioneller Entwurf*: Grobentwurf im ER-Diagramm mit Entitäten, Attributen und Relationen
3. *Logischer Entwurf*: Detailentwurf des konkreten Datenbankschemas für spezielle DBMS
4. *Physikalischer Entwurf*: Primärindexe und Suchindexe zur Zugriffsoptimierung
5. *Implementation*: Erstellung der Datenbank mit SQL
6. *Wartung*: Verwendung der Datenbank

Entity-Relationship Modell - Einführung

- Entity-Relationship Modelle entwerfen das Datenmodell einer Datenbank
- Legen fest: *was*, *wie* und mit *welchen Zusammenhängen* gespeichert wird
- Häufig in Dokumentationen und Ausschreibungen von Software zu finden
- Entwickelt 1976 von Peter Chen: *"The Entity-Relationship Model"*
- *Verschiedene Varianten* von ER-Diagrammen existieren

ER-Modell - Grundbegriffe

Zentrale Konzepte:

- *Entitätstyp (Entity Type)*: Klasse von Objekten (Beispiel: Punkt, Linie, Polygon)
- *Entität (Entity)*: Einzelnes identifizierbares Objekt (Beispiel: Ein einzelner Punkt)

Weitere Komponenten:

- *Attribute*: Eigenschaften einer Entität (Beispiel: x,y-Koordinaten eines Punktes)
- *Beziehung (Relationship)*: Zusammenhänge zwischen Entitäten (Beispiel: Punkt 0,0 "gehört_zu" Linie 1)

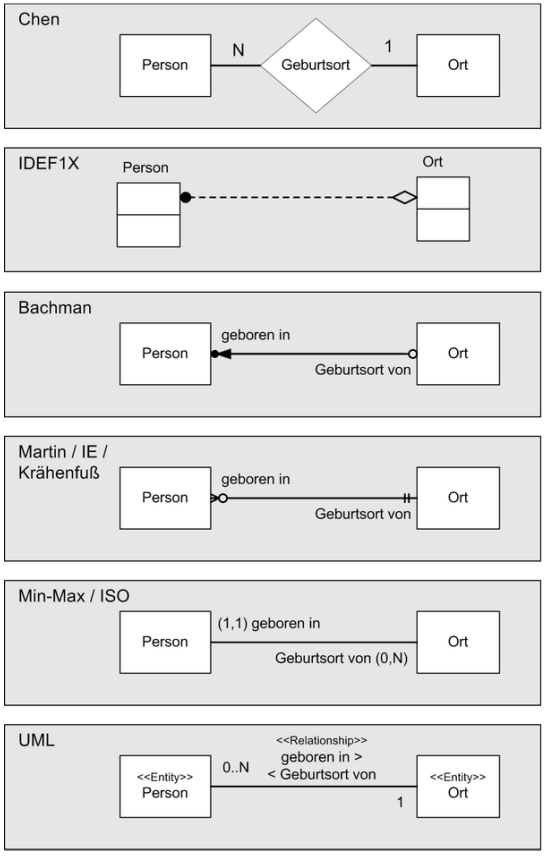
Diagrammarten - Verschiedene Notationen

Historische Entwicklung:

- *Chen-Notation* (Peter Chen, 1976)
- *IDEF1X* (USA Behörden Standard, 1985)
- *Bachman-Notation* (Charles Bachman, 1969)
- *Krähenfuß-Notation* (Gordon Everest, 1976)
- *(min, max)-Notation* (Jean-Raymond Abrial, 1974)

Moderne Ansätze:

- *UML* als ISO-Standard (Ersatz für ER-Diagramme)



https://de.wikipedia.org/wiki/Datei:ERD_Darstellungen.png

Begriffsunterschiede - Terminologie

Objektorientierung	Relationale Datenbank	ER-Diagramme
Objektinstanz	Datentupel	Entität
Klassen	Relationen	Entitätstyp
Klassendefinition	Relationenschema	Entity-Relationship-Modell
Attribute	Attribute	Attribute
Assoziation	Fremdschlüssel	Relationen
Multiplizitäten	-	Kardinalitäten

OOP vs. Relationale Datenbanken - Vergleich

Merkmal	Objektorientierung	Relationale Datenbanken
Modellierung	✅ als Objekte	✅ als Relationen
Attribute	✅	✅
Methoden	✅	❌
Vererbung	✅	❌
Polymorphismus	✅	❌
Generalisierung	✅	❌
Aggregation	✅	❌
Kapselung	✅	✅

Objektorientierter Softwareentwurf

Im objektorientierten Softwareentwurf wird ein *Programm aus Objekten* modelliert:

- *Klassendefinition*: Wie sind Objekte in Form von Klassen definiert?
- *Attribute und Methoden*: Welche Eigenschaften und Verhalten besitzen sie?
- *Vererbung*: Wie bauen Klassen aufeinander auf?
- *Statische Beziehungen*: Wie stehen sie in Referenz zueinander?
- *Dynamische Interaktion*: Wie interagieren sie zur Laufzeit?

Modellierungssprache: UML-Diagramme

Datenbankentwurf

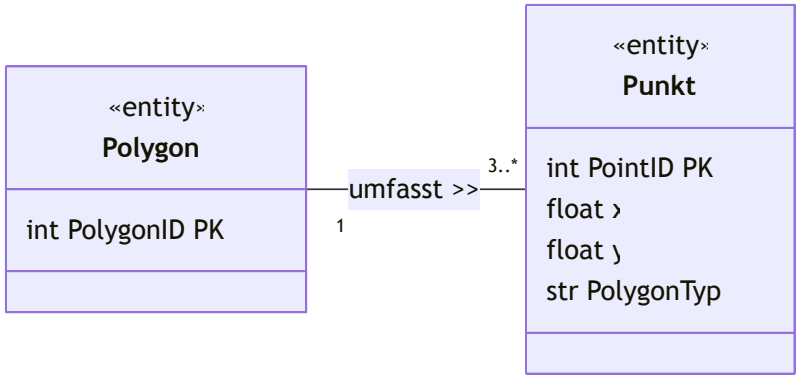
Im Datenbankentwurf wird eine *Datenbank aus Entitäten* modelliert:

- *Entitätstypen*: Wie sind sie in Form von Tabellen definiert?
- *Attribute*: Welche Eigenschaften besitzen Entitäten als Spalten?
- *Statische Beziehungen*: Wie stehen sie in Relation zueinander?

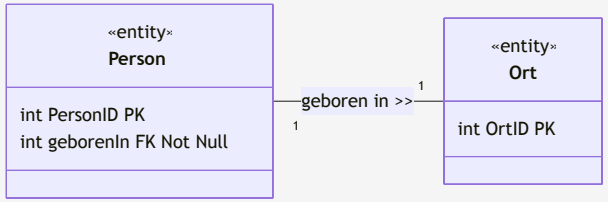
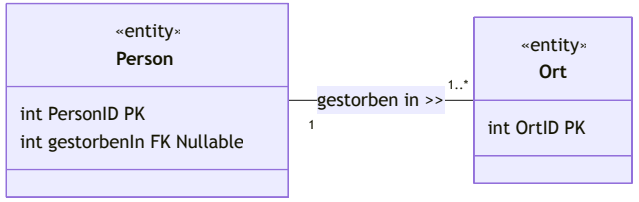
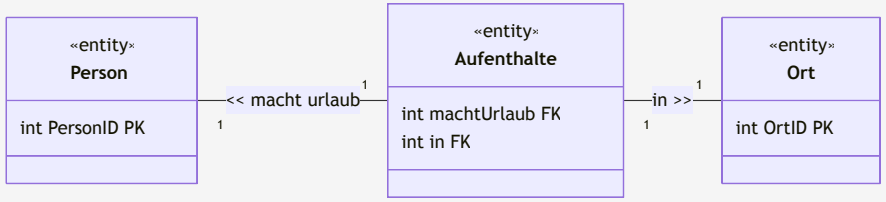
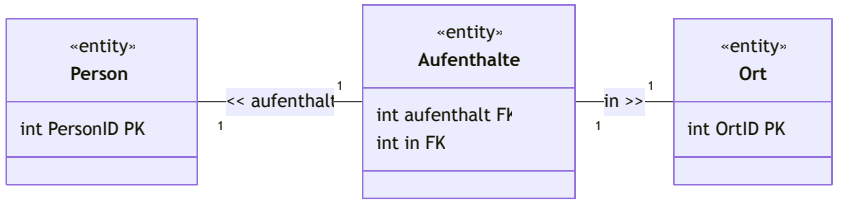
Modellierungssprache: Entity-Relationship Diagramme

UML für ER-Diagramme - Besonderheiten

- Klassen mit `<<Entity>>` annotieren
- Attribute als Klassenattribute mit Datentyp
- *PK* kennzeichnet Primärschlüssel
- Fremdschlüssel als Assoziationen
- Leserichtung: `<<` oder `>>`
- Zahlen geben *Kardinalität* an
- *Keine Methoden!*



Kardinalitäten (Multiplizitäten)

Kardinalität	ER-Diagramm
1 zu 1: Eine Person ist geboren in minimal einem, maximal einem Ort	
1 zu 0..1: Eine Person ist gestorben in minimal Null, maximal einem Ort	
1 zu 0..*: Eine Person macht Ferien in minimal Null, maximal vielen Orten	
1 zu 1..*: Eine Person war bereits in minimal einem, maximal vielen Orten	

Vom Konzeptionellem Modell zum Logischen Modell

- Unterschiedliche DBMS bieten unterschiedliche Datentypen
- Komplexe Attribute (Listen, Dictionary) können nicht direkt abgespeichert werden
- Relationen mit Multiplizitäten >1 können nicht in einer Tabelle mit der Entität abgespeichert werden
- Redundante Daten (z.B. Polygontyp oder Adressen) mit immer den gleichen Werten kosten unnötig Speicher und sind schlecht zu pflegen (Aktualisierung an vielen Stellen)
- Diese Beschränkungen führen dazu dass Konzeptionelle Datenmodelle oft nicht direkt in einer Datenbank abbildbar sind

Normalisierung

📘 Definition: Normalisierung

Die Normalisierung ist ein wichtiger Schritt im Prozess der Abbildung eines Konzeptionellen Datenmodells auf ein Logisches und Physikalisches Datenmodell. Sie hat den Zweck, Redundanzen (mehrfaches Festhalten des gleichen Sachverhalts) zu minimieren, indem:

- komplexe Attribute in neue Tabellen ausgelagert werden
- Relationen mit hoher Kardinalität in neue Tabellen ausgelagert werden
- Redundante Daten (z.B. Polygontyp) in neue Tabellen ausgelagert werden

Normalformen

Verschiedene Normalformen mit fortschreitend strengeren Bedingungen an das Datenbankschema:

- *1. Normalform:* Alle Attributwerte sind atomar (nicht komplex)
- *2. Normalform:* Nicht-Schlüssel Attribute sind von allen Primärschlüsseln voll abhängig
- *3. Normalform:* Nicht-Schlüssel Attribute sind nur von Primärschlüssel abhängig
- *Boyce-Codd-Normalform:* Alle Attribute von denen Attribute abhängen sind Schlüssel
- *4. Normalform:* Es gibt nur noch triviale mehrwertige Abhängigkeiten
- *5. Normalform:* Es gibt keine mehrwertigen Abhängigkeiten, die voneinander abhängig sind

Dies sorgt für sehr viele, sehr stark vereinfachten Tabellen, die wieder zu größeren Relationen zusammengesetzt werden können.
Meistens sind nur die ersten drei Normalformen im Datenbank-Alltag relevant.

Erste Normalform

- *Beispiel:* Erste Normalform verletzt
- *Fehler:* List als Attribut statt Relation

```
-- Falsch: Komplexe Attribute
CREATE TABLE Polygon (
    id INT PRIMARY KEY,
    punkte LIST<Punkt>  -- Verletzt
1NF
);
```

```
-- Richtig: Atomare Attribute
CREATE TABLE Polygon (
    id INT PRIMARY KEY
);

CREATE TABLE PolygonPunkte (
    polygon_id INT,
    punkt_id INT,
    FOREIGN KEY (polygon_id)
REFERENCES Polygon(id),
    FOREIGN KEY (punkt_id) REFERENCES
Punkt(id)
);
```

Zweite Normalform

- *Beispiel:* Zweite Normalform verletzt
- *Fehler:* Punkt nicht ausgelagert

```sql
-- Falsch: Nicht voll abhängig vom Primärschlüssel
CREATE TABLE Linie (
    id INT PRIMARY KEY,
    start_x FLOAT,
    start_y FLOAT,
    end_x FLOAT,
    end_y FLOAT
);
```

```sql
-- Richtig: Punkte ausgelagert
CREATE TABLE Punkt (
    id INT PRIMARY KEY,
    x FLOAT,
    y FLOAT
);

CREATE TABLE Linie (
    id INT PRIMARY KEY,
    start_punkt_id INT NOT NULL,
    end_punkt_id INT NOT NULL,
    FOREIGN KEY (start_punkt_id)
REFERENCES Punkt(id),
    FOREIGN KEY (end_punkt_id)
REFERENCES Punkt(id)
);
```

Dritte Normalform

- *Beispiel:* Dritte Normalform verletzt
- *Fehler:* PolygonTyp von Polygon abhängig

```sql
-- Falsch: Transitive Abhängigkeit
CREATE TABLE Polygon (
    id INT PRIMARY KEY,
    typ_name VARCHAR(50),
    typ_beschreibung VARCHAR(200)
);
```

```sql
-- Richtig: PolygonTyp ausgelagert
CREATE TABLE PolygonTyp (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    beschreibung VARCHAR(200)
);

CREATE TABLE Polygon (
    id INT PRIMARY KEY,
    typ_id INT,
    FOREIGN KEY (typ_id) REFERENCES
PolygonTyp(id)
);
```

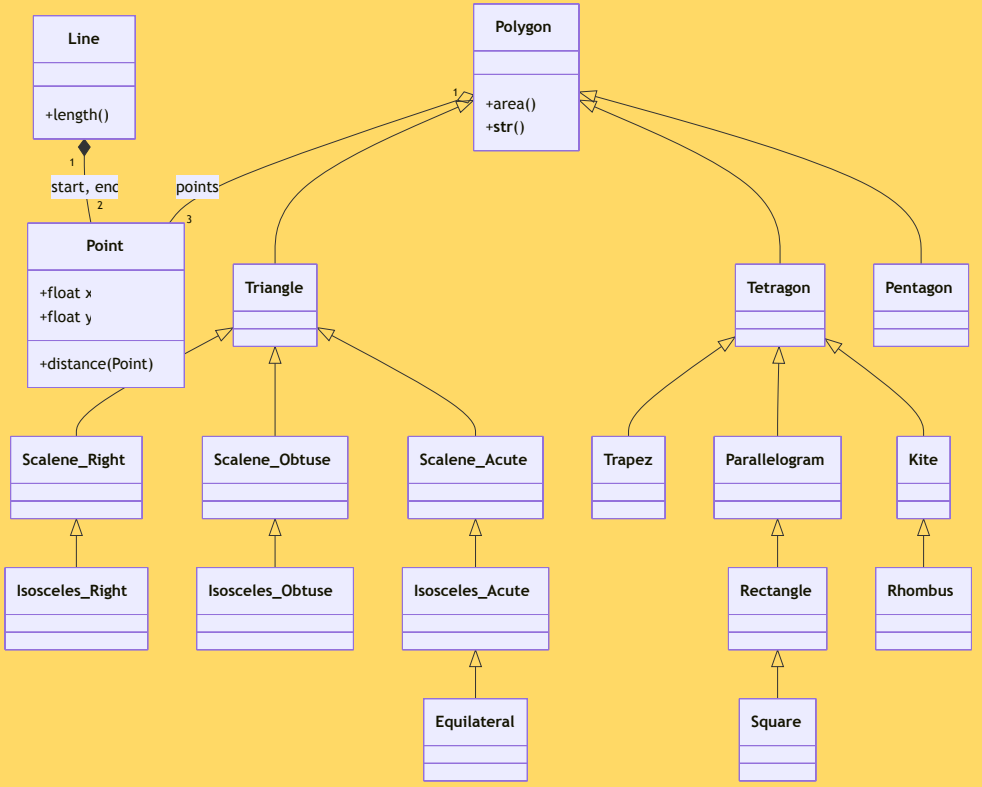
Kardinalitäten (Multiplizitäten)

Kardinalitäten werden unterschiedlich abgebildet:

- *1 zu 1:* Relation wird mit Fremdschlüssel in der Entität abgebildet. Der Fremdschlüssel darf NICHT NULL
- *1 zu 0..1:* Relation wird mit Fremdschlüssel in der Entität abgebildet. Der Fremdschlüssel darf NULL
- *1 zu 0...:** Relation wird als neue Entität mit Fremdschlüssels abgebildet
- *1 zu 1...:** Relation wird als neue Entität mit Fremdschlüssels abgebildet. Mindestens ein Eintrag sollte vorhanden sein

Hörsaalfrage

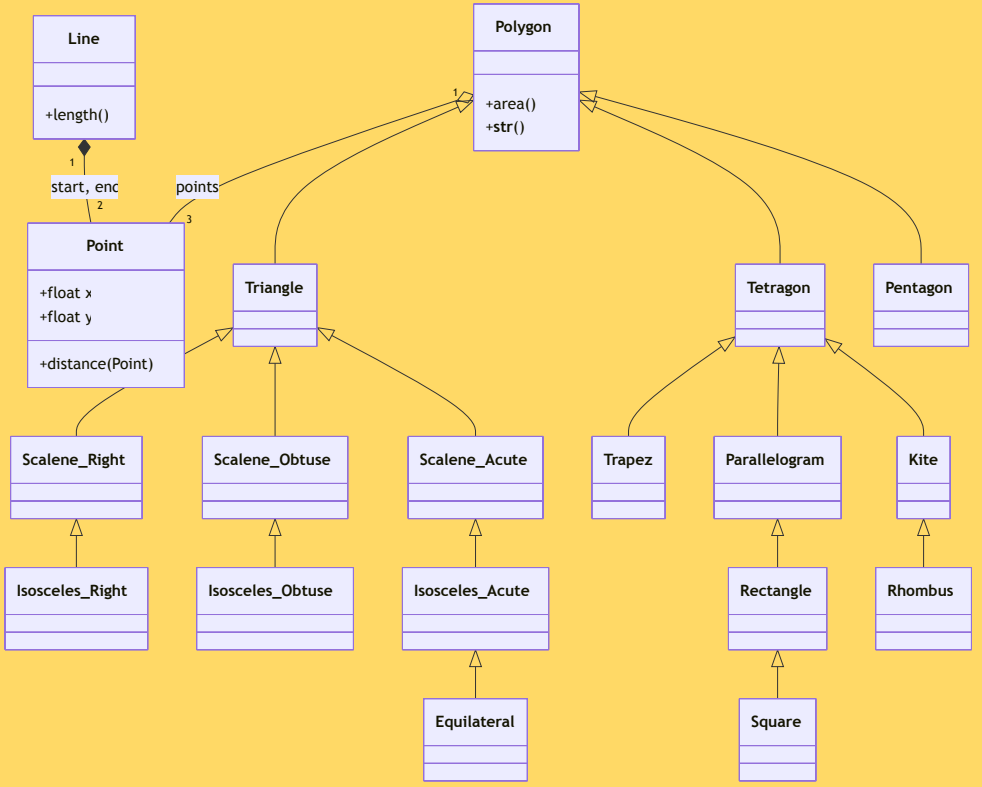
Welche Tabellen brauchen wir für unser Geometriebeispiel?



Hörsaalfrage

Durch Normalisierung erhalten wir *fünf Tabellen:*

- *Punkt:* Besitzt die Attribute x,y-Koordinate, Datentyp float
- *Linie:* Start- und End-Punkt sind Fremdschlüssel die NichtNull sind da eine 1-1-Relation vorliegt
- *Polygon:* Eine Tabelle mit PK und FK auf den PolynomTypen, die Punkte sind extern
- *PolygonTyp:* Eine Tabelle der redundanten Polygontypen mit Name (Dreieck, Viereck, etc.) und PK
- *PolygonPunkte:* Eine Tabelle die für jeden Polygon FK, die zugehörigen Punkte FK listet, da eine 3..* Relation vorlag
- *Für alle fünf* brauchen wir einen Primärschlüssel (Immer Not Null)



Lesson Learned

Midjourney: planning a trip trough the jungle with the panic monkey

Lessons Learned - Prokrastination & Prüfungsplanung

- Man schätzt die Zeit für die Prüfungsvorbereitung gerne optimistisch
- Bei der Vorbereitung direkt vor der Prüfung merkt man, dass man mehr Zeit braucht -> Panik
- Man geht mit Panik in die Prüfung -> Schlechtes Karma

Lessons Learned - Prüfungsplanung & Timemanagement

- Pro: Man nutzt das Unterbewusstsein zur Bearbeitung
- Pro: Durch die Planungs- und Informationsphase kann man den Aufwand besser einschätzen
- Con: Man kann sich dennoch verschätzen

Lessons Learned - Prüfungsplanung & Timemanagement

Schätzt erst den Aufwand und schaut erst dann auf das Datum (plant nicht danach wie viel Zeit ihr habt, sondern wie viel Zeit ihr braucht)

Fach	Prüfungsart	Stoff	Lernaufwand	Datum
Mathematik	Schriftlich	100% Übungen	2+1 Tage	
Technische Mechanik	Schriftlich	80% Übungen / 20% Lernen	4+1 Tage	
Informatik	Schriftlich	50% Übungen / 50% Lernen	3+1 Tage	
Bauchemie	Schriftlich	40% Übungen / 60% Lernen	3+2 Tage	
Baustoffe	Schriftlich	100% Lernen	2+1 Tage	

fragen?

PROGRAMMIERUNG UND DATENBANKEN