Modul 153 Mirio Eggmann

Speicherkomponente (2P)	Verwaltungskomponente (2P)
- Physikalisches Abbild der Daten - Speicherung der Nutzerdaten (Tupel, Tabellen, Beziehungen usw.) - Systemtabellen: Speicherung von Metadaten (Tabellenstruktur, Datentypen usw.) - Systemtabellen: Speicherung von Benutzern, Gruppen und Berechtigungen usw. - Führen der Logdatei gegen Datenverlust	- Schnittstelle per SQL Sprache - Datenschutz (Zugangsberechtigungen) - Backup- und Restore - Import und Export

```
USE [master]
CREATE DATABASE [Laden]
GO
```

```
CREATE TABLE [RECHNUNG] (
RechnungID int IDENTITY(1,1) NOT NULL,
KundeID int NOT NULL,
Bestelltext varchar(50),
Datum smalldatetime NOT NULL,
BetragErhalten varchar(50),
CONSTRAINT PK_RechnungID PRIMARY KEY (RechnungID))
```

```
ALTER TABLE [RECHNUNG]
ADD CONSTRAINT FK_KundenID FOREIGN KEY (KundenID)
REFERENCES KUNDEN (KundenID)
```

Konzeptionelles Datenmodell
- Analyse der Realität(Kundengespräche, bestehende Abläufe etc.
-Unabhängig vom verwendeten RDBMS
- Normalisierung geschieht hier

Logisches Datenmodell
-unabhängig vom verwendeten RDBMS
-Darstellung mittels Tabellen
- Aus konzeptionellem entwickelt

Physisches Datenmodell
-Optimierung der Datentypen
- DML Sprache
- Performance Optimierung
- Aus logischem entwickelt

Genaue numerische Werte: bigint, numeric, bit, smallint, decimal, smallmoney, int, tinyint, money
Ungefähre numerische Werte: float, real
Datum und Uhrzeit: date, datetimeoffset, datetime2, smalldatetime, datetime, time
Zeichenfolgen: char, varchar, text, nchar, nvarchar, ntext
Binärdaten: binary, varbinary, image

```
CREATE Table LIEGENSCHAFT (
        LiegenschaftID BIGINT IDENTITY(1,1) NOT NULL,
        KantonID TINYINT,
        Ortschaft varchar(200),
        Strasse varchar(200),
CONSTRAINT PK_LiegenschaftID PRIMARY KEY (LiegenschaftID),
CONSTRAINT FK_KantonID FOREIGN KEY (KantonID) REFERENCES KANTON (KantonID)
)
```

Nr.	Schritt	Hilfsmittel
1	Analyse aus Anwendersicht	Bestehende Dokumente oder Formulare Skizzen, primitive Diagramme, bestehende Daten in Print oder elektronisch usw.
2	Entwurf eines Datenmodells	Entitäten, Attribute, Beziehungen, Schlüssel, Kardinalitäten, Konzeptionelles Datenmodell ER
3	Entwurf des physischen Datenmodells	Abhängig von der verwendeten Datenbanksoftware! Tabellen, Attribute, Datentypen, Einschränkungen, Schlüssel, Indexe, UML
4	Realisierung der Datenbank	Abhängig von der verwendeten Datenbanksoftware! SQL (DDL), Views, (Datenimport, Prototyping für Performancetests)

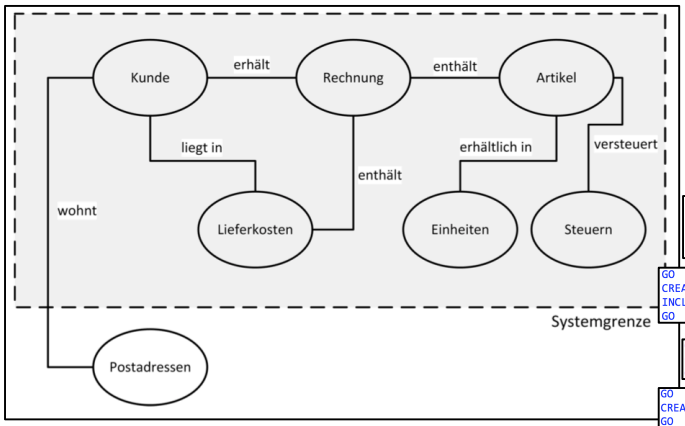
Element	Voranalyse	Konzeptionelles DM	Logisches DM
Entitäten Namen	X	X	
Entitäten Beziehungen	X	X	
Attribute Namen		X	
Primärschlüssel		X	
Fremdschlüssel			X
Tabellen Namen			X
Spalten Namen			X
Datentypen			X

```
CREATE VIEW RechnungKomplett AS
SELECT DISTINCT r.RechnungID, r.Datum, r.Bestelltext, r.BetragErhalten, r.KundeID,
k.Vorname, k.Name
, (SELECT SUM(a.Preis * p.Menge) FROM POSITION AS p LEFT JOIN ARTIKEL AS a ON
p.ArtikelID = a.ArtikelID WHERE p.RechnungID = r.RechnungID) AS Zwischensumme
FROM RECHNUNG AS r
LEFT JOIN KUNDE AS k ON r.KundeID = k.KundeID
```

Datensatz=Tupel
Schlüsselmerkmal=Inhalt eindeutig=Name eindeutig
Zelle=Atomare Daten=Kann NULL sein
Tabelle=Enthält Tupel=Entitätsmenge=Name eindeutig
MDB Datei=Datensätze
LDB Datei=Log Level etc.
Normale Entität: Identifizierbar mit Primärschlüssel
Schwache Entität: Mit Schlüssel anderer Entität
Mehrwertige Attribute=Später noch mehr aufgeteilt werden
Zusammengesetzte=Später noch aufteilen
Indexe=Schneller für select, sonst langsamer
U=Vererbung
Benutzerdefinierte, Prädikatsdefinierte, Attributsdefinierte Spezialisierung/Generalisierung

Sicht des Anwenders:
- Hat Spezialistenwissen in seinem Geschäftsfeld
- Kennt den Normalfall gut, tendiert dazu den Spezialfall zu vergessen oder zu wenig Gewicht zu geben
- Möchte ein möglichst einfaches Programm erhalten ohne die bisherigen Abläufe anzupassen

Sicht des Technikers:
- Hat Spezialistenwissen in der Datenarchitektur und -speicherung
- Möchte alles korrekt umsetzen ohne später Korrekturen machen zu müssen
- Muss die Machbarkeit im Auge behalten
- Denkt an die langfristige Speicherung über mehrere Jahre




```
CREATE TABLE KUNDE1(
ID int NOT NULL,
Vorname varchar(50) NOT NULL,
Nachname varchar(50) NOT NULL
)
```

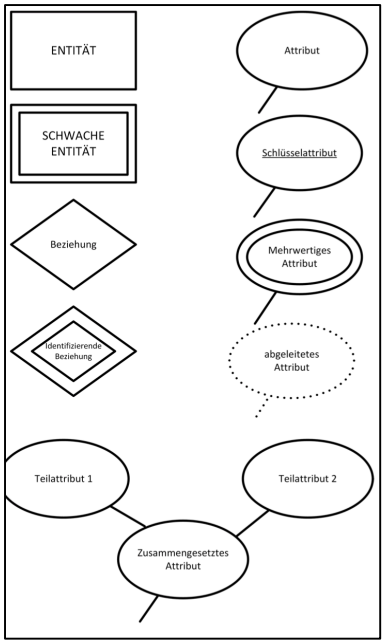
```
Nachname varchar(50) NOT NULL
)
GO
CREATE NONCLUSTERED INDEX idx_nc_Nachname ON KUNDE2 (Nachname)
GO
```

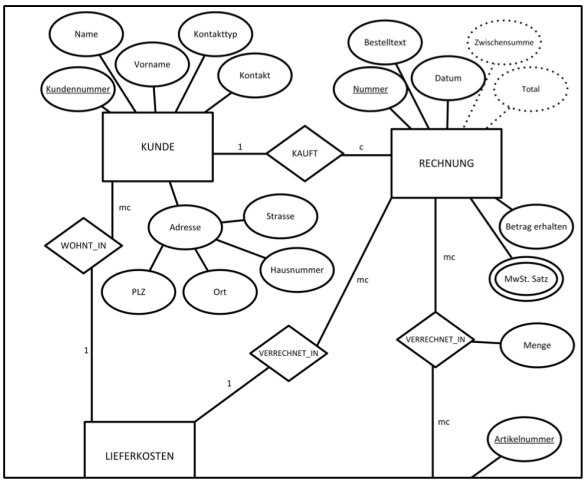
```
GO
CREATE NONCLUSTERED INDEX idx_nc_Nachname ON KUNDE2 (Nachname)
INCLUDE (Vorname);
GO
```

```
GO
CREATE CLUSTERED INDEX idx_cl_Nachname ON KUNDE3 (Nachname)
GO
```

```
GO
CREATE NONCLUSTERED INDEX idx_nc2_Nachname ON KUNDE5 (Nachname)
GO
```

Voranalyse (Kreisdiagramm + Tabellenform mit bspw)
Entitäten, Attribute, Beziehungen, Systemgrenze
Konzeptionelles Datenmodell
Art Entität, Art Attribut, Rollen Beziehung, Kardinalität (1,c,m,mc)
Logisches Datenmodell
TADESI, UML, Relationenschreibweise
Physisches Datenmodell
DDL
Überführung Konz in Logisches:
1. Superklasse und Subklasse je eine Tabelle
2. Eine Klasse pro Subklasse (keine Superklasse)
3. Alles eine Tabelle, zusätzliches Attribut
4. Alles eine Tabelle, zusätzlich Attribut pro Subklasse



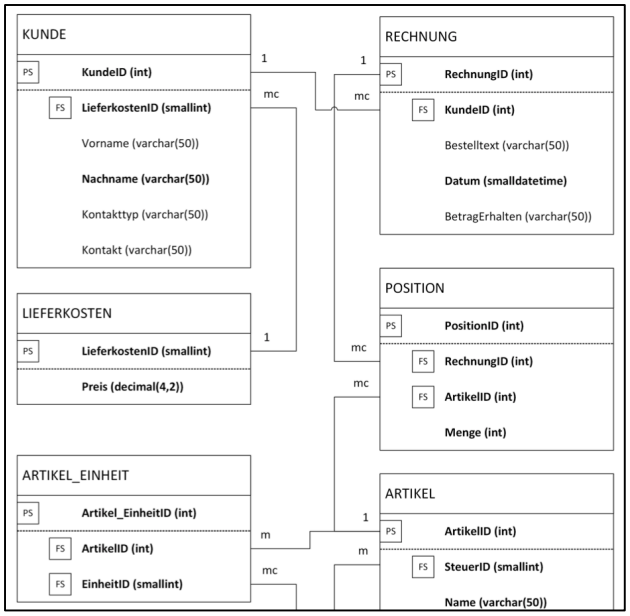


Speicherung von Daten in einem Heap(kein Index)
Vorteil: Schnelle INSERT, schnelle DELETE
Nachteil: Langsame SELECT (alles wird durchsucht)
Speicherung von Daten in einem Heap(mit Index)

Speicherung von Daten in einem Heap(mit Index mit Zusatzattributen)
Vorteil: -Immer wieder abgefragte Attribute können ohne Zusatzschritte im Index abgerufen werden
-Andere Attribute können über Zeiger auf den heap
Nachteil: Bei Insert muss Index nachgeführt werden
-Höherer Speicherbedarf
Speicherung von Daten in einem clustered Index (Primärspeicher) Vorteil: Keine Redundanz, keine weiteren Sprünge auf Heap, Daten werden schnell geladen, wie beim normalen index
Nachteil: Nur ein Attribut (oder zus ges schlüssel mögl)
Speicherung von Daten in einem clustered index plus zusatzindex
Vorteil: mehrere Indizes möglich
Nachteil: beide bäume durchlaufen, da minimal langsamer als über heap.

T	Tabellenname
A	Attribute (inkl. Primär- Fremdschlüssel)
D	Datentypen
E	Einschränkungen (NULL, Gültigkeitsbereiche, Default Werte)
S	Schlüsselarten (Referentielle Integritätsbedingungen)
I	Indexe

1) überlappend total
2) überlappend partiell
3) disjunkt total
4) disjunkt partiell



Relation Kunde mit Beispieldaten:

KUNDE

KundeID	LieferkostenID	Vorname	Nachname	Kontakttyp	Kontakt
1	1	Andi	Aeschbacher	Email	andi@gmx.com
2	1	Beat	Bitterli	Fix	033 999 88 77

RECHNUNG

RechnungID	KundeID	Bestelltext	Datum	BetragErhalten

LIEFERKOSTEN

LieferkostenID	Preis

POSITION

PositionID	RechnungID	Menge

Tabellenname: KUNDE

Attribute	Datentypen	Einschränkungen	Schlüsselarten	Indexierung
KundeID	int	NOT NULL	Primärschlüssel	Ja ohne Duplikate
LieferkostenID	smallint	NOT NULL	Fremdschlüssel	Ja mit Duplikaten
Vorname	varchar(50)			
Nachname	varchar(50)	NOT NULL		
Kontakttyp	varchar(50)	Default: Fix		
Kontakt	varchar(50)			

Tabellenname: RECHNUNG

Attribute	Datentypen	Einschränkungen	Schlüsselarten	Indexierung
RechnungID	int	NOT NULL	Primärschlüssel	Ja ohne Duplikate
KundeID	int	NOT NULL	Fremdschlüssel	Ja mit Duplikaten
Bestelltext	varchar(50)			
Datum	smalldatetime	NOT NULL		
BetragErhalten	varchar(50)			

Vorteile:
- Sehr schnell erstellt ohne „Designaufwand“
- Anzeige der Beispieldaten kann Fragen/Unklarheiten beantworten oder neue Fragen aufwerfen
- Schnell anpassbar

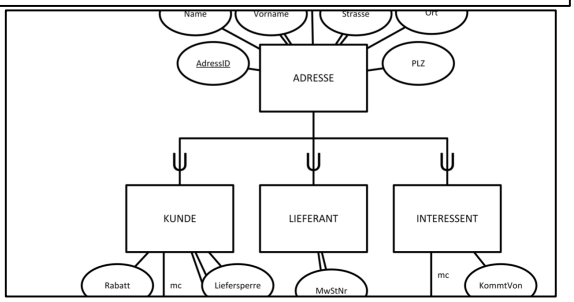
Nachteile:
- Fremdschlüssel nicht sichtbar (könnte mittels # ergänzt werden)
- Datentypen nicht sichtbar
- Kardinalität nicht sichtbar
- Einschränkungen nicht sichtbar
- NULL / NOT NULL nicht sichtbar (könnte mittels **fett** ergänzt werden)

Spezialisierung

Tante Emmas Adressen sind bisher alle in der Tabelle ADRESSE gespeichert gewesen. Neu möchte Sie diese aufteilen können in die 3 oben erwähnten Gruppen.
→ Aufteilen, Top Down Sicht

Generalisierung

Tante Emma hat bis jetzt 3 unterschiedliche Tabellen für die Adressen geführt und möchte diese nun wie gleichwertige „Adressen“ behandeln können.
→ Zusammenfassen, Bottom Up Sicht



Attributdefinierte Spezialisierung / Generalisierung

ADRESSE

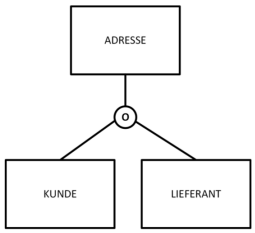
AdresseID	Adresstyp	Vorname	Nachname	Kontakttyp	Kontakt

KUNDE

AdresseID	LieferkostenID	Rabatt	Liefersperre	MwStNr	Ertragskonto

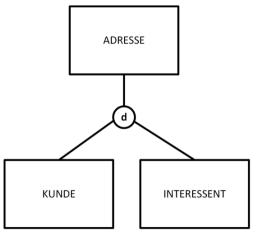
Die Subklassen Kunde und Lieferant sind **überlappend**, da ein Kunde auch Lieferant sein kann und umgekehrt.

Darstellung im konzeptionellen Datenmodell
(o = overlap)



Die Subklassen Kunde und Interessent sind **disjunkt**, da eine Adresse nicht beides gleichzeitig sein kann.

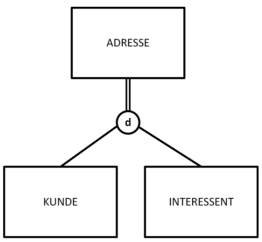
(d = disjoint)



Suchen Sie selber je 2 Beispiele, welche überlappend oder disjunkt sind und notieren Sie diese stichwortartig!

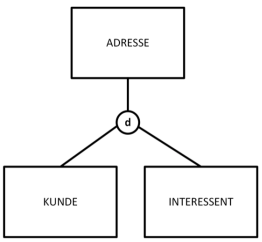
Es handelt sich um eine **totale Spezialisierung**, da jede Adresse einer Subklasse angehören muss.

Darstellung im konzeptionellen Datenmodell
(doppelte Linie)

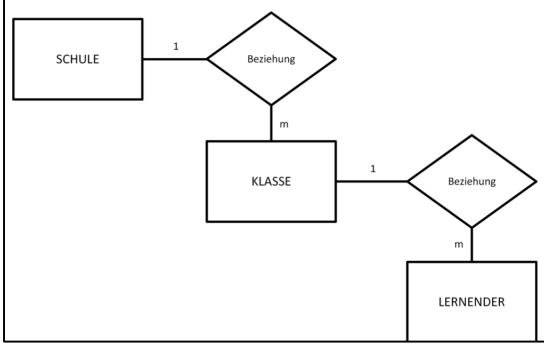


Es handelt sich um eine **partielle Spezialisierung**, da es auch Adressen gibt, welche keiner Subklasse angehören. (nicht zutreffend in unserem Fallbeispiel)

(einfache Linie)



Konzeptuelles DM



Logisches DM

SCHULE

SchuleID	Name	...

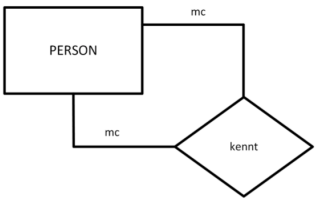
KLASSE

KlasseID	SchuleID	Name	...

LERNENDER

LernenderID	KlasseID	Vorname	...

Konzeptionelles DM
(Die zweite Darstellung ist zu bevorzugen)



Logisches DM

PERSON

PersonID	Name	...

BEKANNTSCHAFT

BekanntschaftID	PersonA	PersonB