

InfMan Zusammenfassung 2022

Inhalt

Grundlagen	3
Daten	3
Informationen	3
Wissen	3
Informationskreislauf	4
Strukturierungsgrad	4
Informationssystem.....	4
ER-Modellierung.....	4
Graphische Notierung (Chen).....	5
Funktionalitäten/ Kardinalitäten	5
(min, max)-Notation	5
Schwache (existenzabhängige) Entitäten.....	6
Generalisierung (Vererbung).....	6
Aggregation (Teil-Ganzes Beziehung).....	6
Relationales Modell.....	6
Superschlüssel	6
Schlüsselkandidat	6
Primärschlüssel.....	6
Schlüsselbeziehungen	6
Abbildung eines ER-Diagramms auf ein Relationales Modell	7
Relationale Abfragesprachen	7
Relationale Algebra	7
RA-Ausdrücke	8
Relationale Entwurfstheorie	8
Funktionale Abhängigkeit.....	8
Normalformen	9
Synthesealgorithmus.....	9
Anfragesprache SQL	10
Data Query Language (DQL)	10
Data Definition Language (DDL)	11
Data Manipulation Language (DML)	12
Anfrageverarbeitung	12
Anfrage-Optimierung	12

Regel-basierte Optimierung	12
Äquivalenzen der Relationalen Algebra	12
Ablauf der Regel-basierten Optimierung	12
Kostenbasierte Optimierung	13
Schätzung der Zwischenergebnisse	13
Transaktionen	14
ACID-Garantien	14
Anomalien im Mehrnutzerbetrieb	14
Concurrency Control	14
Konfliktgraph	14
Lock-based Concurrency Control	15
Multi-Version Concurrency Control	15
Optimistic Concurrency Control	15
Natural Language Text	16
Character Encoding	16
Unicode	16
Linguistic Preprocessing	17
Tokenization	17
Morphologie	17
Wortbildung	18
Normalisierung	18
Syntax	18
Part of Speech Tagging (POS Tagging)	18
Parsing	18
Kontextfreie Grammatik	19
Syntaktische Ambiguity (Mehrdeutigkeit)	19
Semantik	19
Pragmatik	19
Linguistic Analysis Levels	19
Textkorpus	19
Annotationen	20
Korpus-Erstell-Workflow	20
IOB Annotations Schema	20
Lexikalische Ressourcen	20
Information Retrieval	21
Herausforderungen von IR	21

Bool'sche Suche	21
Vector Space Model (VSM)	22
Evaluation von IR-Systemen	22
Informationsextraktion	23
List Lookup Approach	23
Rule-based Approach (Pattern Matching).....	23
Machine Learning Approach	23
Knowledge Bases.....	23
Automatische Klassifikation	24
General (Supervised) ML Workflow	24

Grundlagen

Daten

Eine Zeichenkette, die nach bestimmten Regeln (Grammatik) gebildet wird, nennt man Nachricht. Daten sind ausgetauschte, aber noch nicht interpretierte Nachrichten.

Eine Folge von Symbolen heißt Zeichenkette oder Zeichenfolge. Ein Element aus einem Alphabet heißt Symbol oder Zeichen.

Informationen

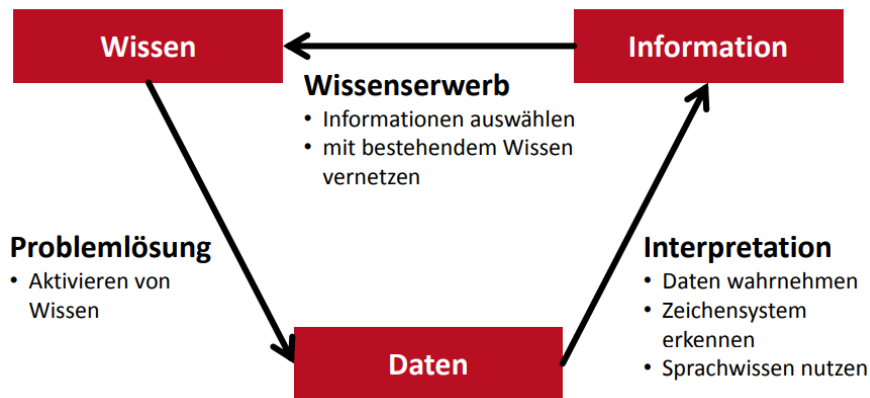
Information sind Nachrichten bzw. Daten mit einem Interpretationsbezug.

Wissen

Damit Informationen für einen Empfänger nützlich sind muss dieser die neuen Informationen mit aktuellen oder früheren Informationen verknüpfen. Wissen ist die Gesamtheit der Kenntnisse, Fähigkeiten, Fertigkeiten, die Personen zur Lösung von Problemen einsetzen.

Zeichenkette + Syntax = Daten + Interpretation = Information + Vernetzung = Wissen
--

Informationskreislauf



Strukturierungsgrad

Strukturierte Daten: Daten welche einem Datenmodell, einer gleichartigen Datenstruktur folgen. (Tabelle, Diagramme, Listen etc.)

Semistrukturierte Daten: Daten ohne festes Datenmodell, die Strukturen implizieren oder erweitern ein Datenmodell. (XML etc.)

Unstrukturierte Daten: Daten ohne formalisierte Struktur (Text, Bilder, Sprache etc.)

Aber: strukturierte Daten lassen sich häufig aus unstrukturierten ableiten.

Informationssystem

Anforderungen:

- **Informationserhalt:** Benötigte Daten lassen sich vollständig aus gespeicherten Daten ableiten
- **Effizienz:** Wiedergewinnung der Daten soll effizient sein
- **Konsistenzhaltung:** Nur „vernünftige“ Daten werden gespeichert
- **Redundanzfreiheit:** Daten werden nicht doppelt gespeichert, um Speicherplatz zu sparen und Anomalien zu vermeiden

Typischer Entwurfsprozess:

1. Anforderungsanalyse: Fachwissen, Terminologie, Kosten-/Nutzen-Analyse, informelle Dokumentation
2. Konzeptioneller Entwurf: Abstrakte Modellierung der Domäne mittels ER-Diagramms
3. Verteilungsentwurf: Fragmentierung, Synchronisation und Replikation
4. Logischer Entwurf: Abbildung in logisches, relationales Datenmodell
5. Datendefinition: Programmierung des Datenmodells im Informationssystem
6. Physischer Entwurf: Definition von Zugriffs- und Speicherstrukturen
7. Implementierung und Wartung

ER-Modellierung

Einfacher Formalismus zur konzeptionellen Modellierung von Daten auf Eigenschaften, Fokus auf statischen Eigenschaften der Daten.

Graphische Notierung (Chen)

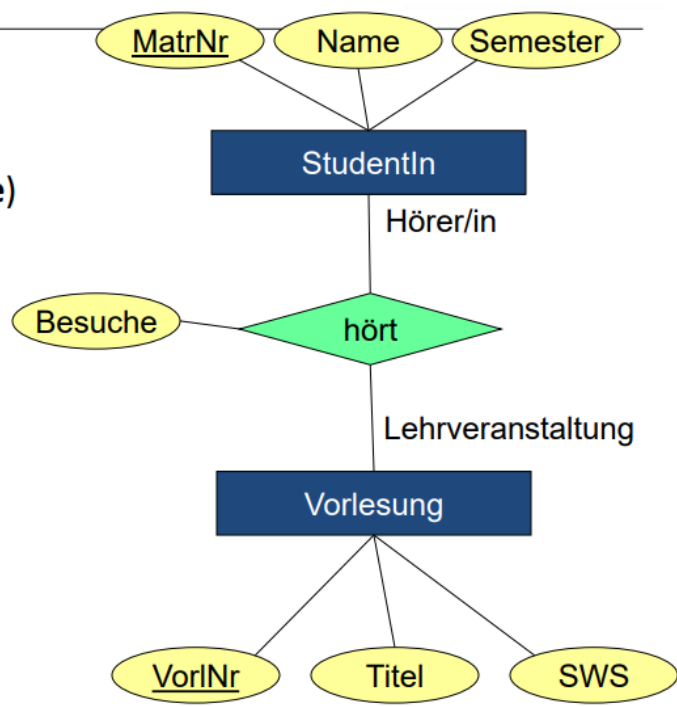
Entitätstyp (Entity Type)

Beziehungstyp (Relationship Type)

Merkmale / Attribute

Schlüsselattribute (Keys)

Rollen (Roles) -> werden meist nicht modelliert



Entitätstyp: Textuell: $E(A_1, A_2, \dots)$, der Entitätstyp bildet dabei den gemeinsamen Typen von Instanzen, eine Entität ist ein individuell identifizierbare Instanz eines Entitätstypen

Beziehungstyp: Textuell: $R(E_1, E_2, \dots; A_1, A_2, \dots)$, Beziehung zwischen Entitätstypen, eine konkrete Beziehung ist zwischen zwei oder mehr konkreten Entitäten

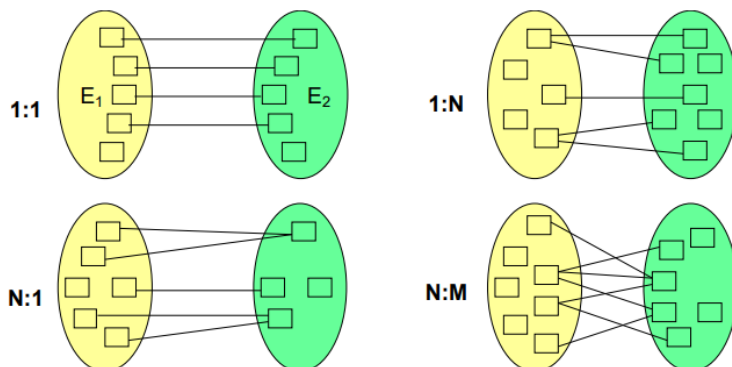
Attribute/Merkmale: Textuell: A

Schlüsselattribute: Teilmenge der Attribute eines Entitätstyps, welche eindeutig identifizierbar ist

Rekursiver Beziehungstyp: Setzt eine Entität mit einer anderen des selben Entitätstyps in Beziehung, Rollennamen sind hier hilfreich

Funktionalitäten/ Kardinalitäten

Kardinalitäten legen für einen Beziehungstyp fest, an wie vielen konkreten Beziehungen eine Entität maximal beteiligt sein kann. Diese muss für jeden beteiligten Entitätstypen bestimmt werden.



(min, max)-Notation

Präziser als Funktionalitäten, da Unter- und Obergrenze angegeben wird.

Schwache (existenzabhängige) Entitäten

Schwache Entitäten existieren nur, wenn starke Entität existiert. Der schwache Entitätstyp und die zugehörige Beziehung werden laut Chen-Notation mit doppelten Linien umrahmt. Der Primärschlüssel des schwachen Entitätstypen wird mit einer gestrichelten Linie markiert und ist nur eindeutig mit dem PK des starken Entitätstypen eindeutig.

Generalisierung (Vererbung)

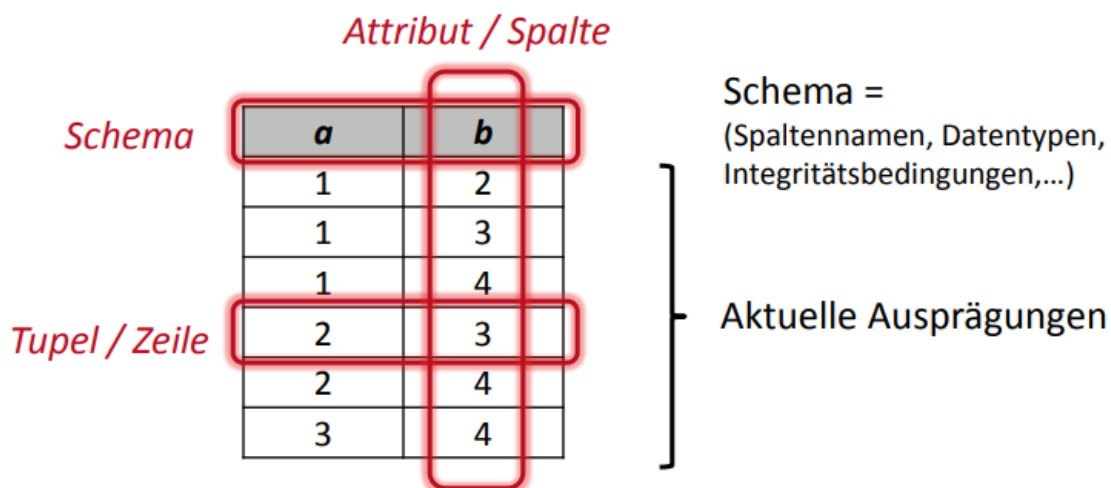
Visualisiert durch Sechseck (is-a-Beziehung), die Attribute des Obertyps werden an den Untertyp vererbt.

Aggregation (Teil-Ganzes Beziehung)

Visualisiert durch eine Raute (part-of-Beziehung).

Relationales Modell

Darstellung der zu speichernden Daten mit einem einzigen Konstrukt: Relation R (informell Tabelle). Hierbei ist die Relation R eine Menge von Tupeln. Das Schema der Relation ist die Attribut-Definition, welche aus dem Spaltennamen und dem Datentyp und weiteren Attributen besteht. Ein Tupel ist der Wert einer Spalte, ein Teiltupel sind die Werte mehrerer Spalten.



Superschlüssel

Eine Attributmenge K heißt Superschlüssel einer Relation, wenn jedes K-Teiltupel genau ein Tupel eindeutig identifiziert. Ein Superschlüssel K für eine Relation R heißt:

- Einfach: falls K nur aus einem Attribut besteht, $|K|=1$
- Zusammengesetzt: falls $|K|>1$
- Trivial: falls K alle Attribute von R enthält, $K = \text{Schema}(R)$
- Minimal, wenn es keine Teilmenge K' von K gibt, der auch Superschlüssel ist

Schlüsselkandidat

K ist Schlüsselkandidat, wenn K minimaler Superschlüssel ist.

Primärschlüssel

Ein Schlüsselkandidat für die Relation R wird als PK gewählt.

Schlüsselbeziehungen

Fremdschlüssel: Eine Attributmenge K' einer Relation S heißt Fremdschlüssel bezüglich der Relation R, falls Referenz $p(K')$ der PK von R ist und die Bedingung der referenziellen Integrität erfüllt ist.

Referenzielle Integrität: Werte für K' in S nehmen nur Werte an, für die in R eine Entsprechung existiert.

Abbildung eines ER-Diagramms auf ein Relationales Modell

1. Abbildung der Entitätstypen auf Relationen: Für jeden Entitätstypen wird eine eigene Relation definiert. Schlüssel aus Entitätstyp wird als PK in Relation übernommen, falls dieser minimal ist. Ansonsten neuer minimaler Schlüssel als PK. Hierbei wird der PK unterstrichen und bei allen Attributen der Datentyp angegeben.
2. Abbildung der Beziehungstypen auf Relationen: Für jede Beziehung wird eine eigene Relation erzeugt. FK beziehen sich auf Primärschlüssel der Relation, die aus Entitätstypen entstanden sind. Der PK einer solchen Relation setzt sich wie folgt zusammen:
 - Bei 1:1: PK der Relation ist PK einer der beiden Beteiligten Entitätstypen
 - Bei 1:N: PK der Relation ist PK der N-Seite
 - Bei N:M: PK ist zusammengesetzt aus den PKs der beteiligten Relationen
 - Bei n-stelligen Beziehungen ist der zusammengesetzte PK der Relation nur aus den PKs der Entitätstypen mit den Funktionalitäten N,M,...
3. Zusammenfassung von Relationen mit gleichen PKs
 - 1:1 Beziehungstypen können immer mit einer der beteiligten Relationen zusammengefasst werden.
 - 1:N Beziehungstypen können immer mit der Relation auf N-Seite zusammengefasst werden
 - N:M Beziehungstypen können nicht zusammengefasst werden
4. Behandlung von schwachen Entitätstypen

Relationale Abfragesprachen









Um Anfragen an eine Datenbank zu formulieren sind Relationale Abfragesprachen nötig, eine Abfrage definiert, „was“ zu extrahieren.

Relationale Algebra

Codd's Algebra beinhaltet Selektion, Projektion, Kreuzprodukt, Mengendifferenz, Vereinigung und Mengendurchschnitt, diese Operationen reichen aus, um alles zu extrahieren, zur Vereinfachung gibt es aber noch weitere Operatoren: Join, Outer Join, Left Outer Join, Right Outer Join, Left Semi-Join, Right Semi-Join, Relationale Division und Umbenennung.

- Selektion/Restriktion: Die Selektion $\sigma_p(R)$ ist die Auswahl aller Tupel aus R, welche die Bedingung p erfüllen. p besteht dabei aus $=, >, <, \leq, \geq, \wedge, \vee, \neg$
- Projektion: Die Projektion $\pi_\alpha(R)$ ist die Reduktion aller Tupel in R auf die Spalten α ; $\alpha \subseteq \text{Schema}(R)$ eine Teilmenge des Schemas von R, das Ergebnis der Projektion ist eine Menge, wobei Duplikate eliminiert werden
- Kartesisches Produkt: Das kartesische Produkt $R \times S$ ist die Menge aller möglichen Kombinationen der Tupel aus R und S
- Natürlicher Join: $L \bowtie R = \pi_{L.A, L.B, L.C, R.D, R.E}(\sigma_{L.C=R.C}(L \times R))$
- Allgemeiner Join (Theta Join): $L \bowtie_p R = \sigma_p(L \times R)$
- Left Outer Join:
- Right Outer Join:
- Full Outer Join:
- Left Semi Join: $L \ltimes R = \pi_{att(L)}(L \bowtie R)$
- Right Outer Join: $L \ltimes R = R \ltimes L$

- Mengendifferenz:
- Vereinigung
- Mengendurchschnitt
- Relationale Division
- Umbenennen von Relationen: =
- Umbenennen von Attributen: <-

LEFT JOIN  Everything on the left + anything on the right that matches	<pre>SELECT * FROM TABLE_1 LEFT JOIN TABLE_2 ON TABLE_1.KEY = TABLE_2.KEY</pre>	ANTI LEFT JOIN  Everything on the left that is NOT on the right	<pre>SELECT * FROM TABLE_1 LEFT JOIN TABLE_2 ON TABLE_1.KEY = TABLE_2.KEY WHERE TABLE_2.KEY IS NULL</pre>
RIGHT JOIN  Everything on the right + anything on the left that matches	<pre>SELECT * FROM TABLE_1 RIGHT JOIN TABLE_2 ON TABLE_1.KEY = TABLE_2.KEY</pre>	ANTI RIGHT JOIN  Everything on the right that is NOT on the left	<pre>SELECT * FROM TABLE_1 RIGHT JOIN TABLE_2 ON TABLE_1.KEY = TABLE_2.KEY WHERE TABLE_1.KEY IS NULL</pre>
OUTER JOIN  Everything on the right + Everything on the left	<pre>SELECT * FROM TABLE_1 OUTER JOIN TABLE_2 ON TABLE_1.KEY = TABLE_2.KEY</pre>	ANTI OUTER JOIN  Everything on the left and right that is unique to each side	<pre>SELECT * FROM TABLE_1 OUTER JOIN TABLE_2 ON TABLE_1.KEY = TABLE_2.KEY WHERE TABLE_1.KEY IS NULL OR TABLE_2.KEY IS NULL</pre>
INNER JOIN  Only the things that match on the left AND the right	<pre>SELECT * FROM TABLE_1 INNER JOIN TABLE_2 ON TABLE_1.KEY = TABLE_2.KEY</pre>	CROSS JOIN  All combination of rows from the right and the left (cartesian product)	<pre>SELECT * FROM TABLE_1 CROSS JOIN TABLE_2</pre>

RA-Ausdrücke

RA-Ausdrücke können aus mehreren Operatoren und Relationen zusammengesetzt werden

Relationale Entwurfstheorie

Aus redundanten Daten entsteht Speicherplatzverschwendung und Anomalien in Daten.

Update-Anomalie: Bei einer Änderung von Daten müssen alle zugehörigen Tupel aktualisiert werden.

Einfüge-Anomalie: Neue Daten können nur gespeichert werden, wenn übergeordnete Daten vorhanden sind.

Lösch-Anomalie: Beim Löschen von Daten gehen andere abhängige Daten verloren.

Funktionale Abhängigkeit

Die Menge β ist von α funktional abhängig genau dann, wenn für jeden Wert aus α genau ein Wert für β existiert, wobei α und β Attribut-Teilmengen eines Relationenschema sind. Man schreibt $\alpha \rightarrow \beta$.

Schlüsselabhängigkeit: ist eine spezielle FD, für alle Schlüsselkandidaten K von R gilt: $K \rightarrow \text{Schema}(R)$.

Voll funktionale Abhängigkeit: β heißt voll funktional abhängig von α genau, dann wenn $\alpha \rightarrow \beta$ gilt und es kein $\alpha' \subset \alpha$ gibt mit $\alpha' \rightarrow \beta$

Partielle funktionale Abhängigkeit: β heißt partiell funktional abhängig von α genau dann, wenn β nicht voll funktional abhängig von α ist

Attributhülle: Eine Attributhülle α^+ für eine Teilmenge α ist die Menge aller Attribute, die von α funktional abhängig sind

TODO ATTRIBUT ALGO

Schlüsselkandidaten aus FDs bestimmen:

1. Attribute, die nicht funktional abhängig sind, müssen in jedem Schlüsselkandidaten enthalten sein
2. Superschlüssel aus Attributhüllen der „rechten“ Seite einer Abhängigkeit bestimmen
3. Prüfen auf Schlüsselkandidaten

Normalisierung: Zerlegung von großen Relationen in kleinere Relationen, um Relationen redundanzfrei zu machen und Anomalien zu vermeiden. Hierfür gibt es zwei Kriterien:

1. Verlustlosigkeit: Die Daten in der ursprünglichen Relation R müssen verlustfrei aus den zerteilten Relationen rekonstruierbar sein
2. Abhängigkeitserhaltung: Die für R geltenden FDs müssen alle in den zerteilten Relationen enthalten sein

Normalformen

- 1NF: Eine Relation ist in 1NF, wenn alle Attributwerte atomar sind. Das bedeutet, dass ein Attribut nicht mehrwertig ist, also nicht mehr als ein Eintrag erhält.
- 2NF: Eine Relation ist in 2NF, wenn sie in 1NF ist und alle Nichtprimattribute vollständig von jedem Schlüsselkandidaten funktional abhängen. Nichtprimattribute sind an keinem Schlüsselkandidaten beteiligt.
- 3NF: Eine Relation ist in 3NF, wenn sie in 2NF ist und alle Nichtprimattribute nicht transitiv von einem Schlüsselkandidaten abhängen. Eine Attributmenge β heißt transitiv abhängig von α genau dann, wenn $\alpha \rightarrow \gamma$ und $\gamma \rightarrow \beta$ voll funktional abhängig sind, aber $\gamma \rightarrow \alpha$ keine voll funktionale Abhängigkeit ist.

Synthesealgorithmus

Der Synthesealgorithmus dient dazu eine Relation zu Zerlegen und diese in 3NF zu bringen.

1. Bestimme die kanonische Überdeckung von den Funktionalen Abhängigkeiten
 - Linksreduktion: Überprüfe für $\alpha \rightarrow \beta$ ob $\beta \subseteq (\alpha - A) +$ wobei $A \in \alpha$.
Umgangssprachlich: Wenn die linke Seite einer FD aus mehr als einem Attribut besteht, überprüfen, ob eins der Attribute weggelassen kann, hierzu muss das Attribut auf der rechten Seite in der Attributhülle des zu behaltenden Attributs auftauchen
 - Rechtsreduktion: für alle FDs $\alpha \rightarrow \beta$:
 - Überprüfe für alle $B \in \beta$ ob $B \in \alpha^+$ mit $F - (\alpha \rightarrow \beta) \cup (\alpha \rightarrow (\beta - B))$
 - Falls dies der Fall ist, dann ersetze $\alpha \rightarrow \beta$ durch $\alpha \rightarrow (\beta - B)$
 - Umgangssprachlich: Wenn A von B abhängig und C von A abhängig ist, C aber auch in der Attributhülle von B ist, kann C entfernt werden.
 - Entfernung von FDs der Form $\alpha \rightarrow \emptyset$
 - Zusammenfassung gleicher linker Seiten
2. Erzeuge neue Relationen
 - Für jede funktionale Abhängigkeit $\alpha \rightarrow \beta \in F$ aus Schritt 1 erzeuge eine Relation $R := \{(\alpha, \beta)\}$ per FD
 - Weise alle F aus Schritt 1 den einzelnen Relationen zu
3. Überprüfe Schlüsselkandidat
 - Falls eines der in Schritt 2 erzeugten Schemata Attribute eines Schlüsselkandidaten von R vollständig enthält, sind wir fertig
 - Sonst wähle einen Schlüsselkandidaten k aus und definiere folgende neue Relation $R_k := \{(k)\}$
4. Fasse Relationen zusammen

- Entferne diejenigen Relationen R' die in einer anderen Relation R komplett enthalten sind

Anfragesprache SQL

Deklarative Anfragesprache für relationale Datenbanksysteme, standardisiert durch ANSI und ISO

Data Query Language (DQL)

- Select: Auswahl einzelner Attribute oder „*“ aller Attribute

```
select distinct p.PersNr as Nummer, p.Name, p.Rang
from ProfessorIN as p
where Name != 'Müller' and Nummer > 3;
order by Rang desc;
```
- From: Ursprungstabelle aus denen die Daten entnommen werden sollen
- Where: Einschränkung der auszuwählenden Tupel, hier können AND, OR, NOT und die Vergleichsoperatoren verwendet werden
- Order by: Sortierung nach einem oder mehrerer Attribute (asc or desc)
- Umbenennung: Attribute können für die Ergebnistabelle umbenannt werden, dies geht mit dem Stichwort „as“
- Select distinct: Tupelduplikate werden eliminiert
- Joins:
 - Cross join: Kreuzprodukt

```
select s.Name, v.Titel
from StudentIN s
join hoeren h on s.MatrNr = h.MatrNr
join Vorlesungen v on h.VorlNr = v.VorlNr
```
 - Join/inner join
 - Natural join
 - Left, right, full outer join
- Mengenoperationen:
 - Union (ohne Duplikate), union all (mit Duplikaten)
 - Intersect (all)
 - Minus/except (all)

```
select gelesenVon, Name, sum (SWS)
from Vorlesungen, ProfessorIN
where gelesenVon = PersNr and Rang = 'C4'
group by gelesenVon, Name
having avg (SWS) >= 3;
```
- Aggregation und Gruppierung
 - Group by gruppiert alle Zeilen im Ergebnis nach der angegebenen Spalte. Pro Gruppe wird eine Aggregationsfunktion angewandt
 - Aggregationsfunktionen: count, sum, avg, max, min
- Unteranfragen / Verschachtelung sind möglich bei Select, From, Where, Group-by und Having
 - Operationen in der where-Klausel, die auf eine Tabelle T angewandt und einen booleschen Wert ergeben:

```
select a.*
from AssistentIn a
where EXISTS
( select p.* from ProfessorIN p
  where a.Boss = p.PersNr );
```

 - EXISTS T : ergibt True, wenn T nicht leer
 - S IN T : ergibt True, wenn s in T
 - $S > ALL T$: ergibt True, wenn s größer als alle Werte in T
 - $S > ANY T$: ergibt True, wenn s größer als mindestens ein Wert in T ist
 - Jede dieser Bedingungen kann auch mit NOT negiert werden und anstelle von $>$ können auch andere Vergleichsoperatoren verwendet werden, T kann auch wieder eine Unteranfrage sein

- Relationale Division: nativ hat SQL keine relationale Division, aber sie lässt sich mithilfe einer count-Aggregation ausdrücken

```
select h.MatrNr, count(*)
from hoeren h, Vorlesungen v
where v.VorlNr = h.VorlNr and v.sws = 4
group by h.MatrNr
having count(*) =
( select count(*) from Vorlesungen where sws = 4 );
```

- NULL-Werte
 - Kommt null in arithmetischen Ausdrücken vor, wird dieser immer zu null ausgewertet
 - Kommt null in Where oder Having vor (bei True/False Bestimmung) wird der Ausdruck folgendermaßen ausgewertet:

AND	True	False	Unknown	OR	True	False	Unknown	NOT	
True	True	False	Unknown	True	True	True	True	True	False
False	False	False	False	False	True	False	Unknown	False	True
Unknown	Unknown	False	Unknown	Unknown	True	Unknown	Unknown	Unknown	Unknown

- Spezielle Sprachkonstrukte
 - Between: zwischen zwei Werten
 - In: Menge an Werten
 - Like: Pattern Matching, _ steht für beliebiges Zeichen % steht für eine beliebige Zeichenkette
 - Case when ... then ... else ... end: Um zum Beispiel Noten als Text darzustellen, when note < 1.5 then ‚sehr gut‘
 - Rekursion auch möglich, hier unterscheidet sich der genaue Terminus je nach SQL-Standard, siehe VL 07b Folie 40ff.
 - Sichten: Sichten werden in SQL häufig für Benutzer und Rechte, Vereinfachung von Anfragen und Abbildung von Vererbung genutzt, Views sind Read-only

Data Definition Language (DDL)

DDL Befehle dienen dazu Tabellen anzulegen, zu ändern oder zu entfernen, sie dienen nicht dazu um neue Daten einzufügen bzw. Daten zu ändern/entfernen.

- Tabelle Anlegen: creat table, hier kann überprüft werden, ob die Tabelle bereits existiert und ggf. kann der Primärschlüssel

angegeben werden. Es besteht auch die Möglichkeit automatisch inkrementierende PKs hinzuzufügen. Mit den Keywords start with X increment by X kann die Sequenz auch angepasst werden. Aber auch hier unterscheiden sich die unterschiedlichen SQL-Standards

- Entfernen einer Tabelle: drop table, hier können auch alle abhängigen Objekte mit entfernt werden (Keyword cascade)

```
create table if not exists ProfessorIN (
    PersNr int,
    Name   varchar(30),
    Rang   char(2),
    primary key (PersNr, Rang)
);

create sequence PersNrSeq;

insert into ProfessorIN
values (nextval('PersNrSeq'), 'XX', 'XX');

drop table if exists ProfessorIN cascade;

alter table ProfessorIN add column Raum integer;
```

- Ändern einer Tabelle, um z.B. Tabellename/Attributname zu ändern, Attribute hinzuzufügen oder zu löschen oder den Datentyp eines Attributs zu ändern

Data Manipulation Language (DML)

DML Befehle dienen dazu um Daten in einer Tabelle zu ändern, hinzuzufügen oder zu löschen

```
insert into ProfessorIN(PersNr, Name, Rang)
values (2125, 'Sokrates', 'C4');
```

```
delete from StudentIN
where Semester > 13;
```

```
update StudentIN set Semester = Semester + 1
where MatrNr=29120
```

Anfrageverarbeitung

Die SQL-Anfrage wird vom Anfrage-Compiler in einen Logischen Plan kompiliert (relationale Algebra), dieser Plan wird dann vom Anfrage-Optimierer optimiert und in einen Ausführbaren Algebra-Plan übersetzt, dieser wird dann ausgeführt.

Anfrage-Optimierung

Die Anfrage-Optimierung besteht aus zwei Phasen, der Regel-basierten Optimierung und der Kosten-basierten Optimierung. Das Ziel der Optimierung ist ein effizienter Ausführungsplan der Anfrage, meistens im Bezug auf Minimierung der Ausführungszeit.

Regel-basierte Optimierung

Regeln schreiben Plan auf algebraischer Ebene um, dass logischer Ausführungsplan verbessert wird. Das Basis Regelwerk folgt aus den Äquivalenzregeln der relationalen Algebra. Die Regeln sind unabhängig von den Daten in der Datenbank.

Äquivalenzen der Relationalen Algebra

1. Aufbrechen von Konjunktion im Selektionsprädikat: $\sigma_{c_1 \wedge c_2 \wedge \dots \wedge c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\dots(\sigma_{c_n}(R)) \dots))$
2. σ ist kommutativ: $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$
3. π -Kaskaden: Falls $L_1 \subseteq L_2 \subseteq \dots \subseteq L_n$, dann gilt $\pi_{L_1}(\pi_{L_2}(\dots(\pi_{L_n}(R)) \dots)) \equiv \pi_{L_1}(R)$
4. Vertauschen von σ und π : Falls die Selektion sich nur auf die Attribute A_1, \dots, A_n der Projektionsliste bezieht: $\pi_{A_1, \dots, A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, \dots, A_n}(R))$
5. \times, \cup, \cap und JOINS sind kommutativ
6. Vertauschen von σ mit JOIN: Falls das Selektionsprädikat c nur auf Attribute der Relation R zugreift, kann man die beiden Operationen vertauscht: $\sigma_{c_1}(R \bowtie S) \equiv \sigma_{c_1}(R) \bowtie S$. Falls das Selektionsprädikat c eine Konjunktion der Form $c_1 \wedge c_2$ ist und c_1 sich nur auf Attribute aus R und c_2 sich nur auf Attribute aus S bezieht, gilt folgende Äquivalenz: $\sigma_{c_1 \wedge c_2}(R \bowtie S) \equiv \sigma_{c_1}(R) \bowtie (\sigma_{c_2}(S))$
12. Join Ersetzung: Sei c eine Bedingung der Form $A \text{ op } B$, mit A ein Attribut von R und B ein Attribut aus S und op ein Vergleichsoperator: $\sigma_c(R \times S) \equiv R \bowtie_c S$

Ablauf der Regel-basierten Optimierung

1. Selektions-Aufspaltung: Mittels Regel 1 werden konjunktive Selektionsprädikate in Kaskaden von σ -Operationen zerlegt
2. Selektions-Pushdown: Mittels Regeln 2,4,6 werden Selektionsoperationen soweit nach unten propagiert wie möglich

3. Join-Ersetzung: Mittels Regel 12 wird eine X-Operation, die von einer σ -Operation gefolgt wird, in eine Join-Operation umgewandelt
4. Projektions-Pushdown: Mittels Regeln 3,4,7 und 10 werden Projektionen soweit wie möglich nach unten propagiert
5. Zusammenfassung: Versuche Operationsfolgen gleicher Operatoren wieder zusammenzufassen (Regeln 1 und 3)

Kostenbasierte Optimierung

Kostenbasierte Optimierung wählt einen Plan anhand von geringen geschätzten Ausführungskosten (Laufzeit) aus.

1. Auswahl einer optimalen Join-Reihenfolge: Join-Reihenfolge hat großen Einfluss auf Laufzeit eines Plans
2. Auswahl von „physischen Operatoren“ für jeden logischen Operator

Hier wird die Datenbankengröße häufig als Messwert genommen, diese Information wird im Datenbanken-katalog gespeichert.

Schätzung der Zwischenergebnisse

Die Schätzung der Zwischenergebnisgrößen erfolgt über Schätzung der sogenannten „Selektivität“ eines Operators. Die Selektivität setzt Eingabe- und Ausgabe eines Operators in Verhältnis ($sel = 0 \dots 1$)

- Selektivität des Selektionsoperators σ_p : $sel(\sigma_p(R)) = |\sigma_p(R)|/|R|$
 - $sel(\sigma_{A=Konstante}(R)) = 1/|A|$
 - $sel(\sigma_{A=B}(R)) = 1/\max(|A|, |B|)$
 - $sel(\sigma_{p1 \text{ AND } p2}(R)) = sel(\sigma_{p1}(R)) * sel(\sigma_{p2}(R))$
 - $sel(\sigma_{p1 \text{ OR } p2}(R)) = sel(\sigma_{p1}(R)) + sel(\sigma_{p2}(R)) - sel(\sigma_{p1}(R)) * sel(\sigma_{p2}(R))$
- Selektivität des Join-Operators \bowtie : $sel(R \bowtie S) = |R \bowtie S|/|R \times S|$
 - $sel(R \bowtie_{R.A=S.B} S) = 1/|R|$
 - $sel(R \bowtie_{A=B} S) = 1/\max(|A|, |B|)$

Weitere (bessere) Verfahren:

- Stichprobenverfahren: Berechne Selektivität für eine Stichprobe der Eingabe, dann Verallgemeinerung auf gesamte Eingabe
- Histogramme: Vereinfachte Häufigkeitsverteilung meist auf einem Attribut, berechne die relative Häufigkeit dieser Teilbereiche
- Parametrisierte Verteilung (Synopsen): Annahmen über die Verteilung, nutze geschlossene Formeln für Verteilungen um Selektivität zu berechnen

Join-Reihenfolge: Anzahl der unterschiedlichen Binärbäume mit $n+1$ Blättern (Relationen) ist durch die Catalan-Zahl C_n gegeben, wobei $C_n = (2n)!/((n+1)! * n!)$. Die Anzahl der Binärbäume wird durch alle $(n+1)!$ Permutationen der Relationen vervielfacht zu $(2n)!/n!$. Alle Pläne können bei einer großen Anzahl Eingabe-Relationen nicht analysiert werden.

Dynamische Programmierung: Ein klassischer Ansatz mit einem großen Suchraum umzugehen, ist die dynamische Programmierung. Die dynamische Programmierung wird daher auch in Datenbanken zur Suche nach einer Join-Reihenfolge eingesetzt. Die Idee dahinter ist eine Bottom-Up Konstruktion des Anfrageplans aus optimalen Teilplänen. Für n Eingabe-Relationen wird in n Phasen der bestmögliche Plan gesucht. In Phase k werden nur Pläne mit k Relationen analysiert und schlechte Pläne verworfen (Pruning). Jede Phase k greift dabei auf alle Ergebnisse der vorhergehenden Phase zurück.

Transaktionen

Eine Transaktion ist eine Folge von logisch zusammengehörigen SQL-Anfragen und SQL-Änderungsoperationen, dabei werden die ACID-Garantien beachtet.

ACID-Garantien

- Atomicity: Transaktionen werden vollständig oder nicht ausgeführt (Logging und Recovery)
- Consistency: Transaktionen produzieren nur konsistente Zustände (Constraint Checking)
- Isolation: Keine gegenseitige beeinflussung, Änderung erst nach Commit (Concurrency Control)
- Durability: Veränderungen sind permanent (Logging und Recovery)

Anomalien im Mehrnutzerbetrieb

- Inkonsistentes lesen: Leseoperation liefert unterschiedliche Ergebnisse
- Verlorene Updates: Zeitliche Überschneidung von zwei TX, Änderung der ersten geht verloren
- Schreib-Lese-Konflikt: TX2 benutzt falschen Wert, da TX1 fehlgeschlagen ist
- Phantomproblem: Wie inkonsistentes Lesen nur mit veränderter Anzahl an Elementen

Concurrency Control

Der Transaktions-Scheduler bestimmt die Reihenfolge, in der die Schritte von nebenläufigen Transaktionen ausgeführt werden.

Eine Transaktion ist eine geordnete Folge von Schritten $T_i = \langle s_i^1, \dots, s_i^n \rangle$ für Schritte 1...n in Transaktion i wobei Schritt-Indizes häufig weggelassen werden. Mögliche Schritte:

- Lesen $r_i(o_k)$
- Schreiben $w_i(o_k)$
- Beginnen b_i
- Commiten c_i
- Abort a_i

Ein Schedule S für eine Menge von Transaktionen $T = \{T_1, \dots, T_n\}$ ist eine (partielle) Ordnung der Schritte aller Transaktionen. Bei einem Ein-Prozessor-System ist S eine totale Ordnung. Die Reihenfolge der Schritte innerhalb einer Transaktion bleiben in S erhalten.

Ein serieller Schedule ist ein spezieller Schedule bei dem alle Schritte einer Transaktion konsekutiv aufeinander folgen. Serielle Schedules erzeugen keine Anomalien aber der Durchsatz ist in der Regel sehr gering.

Ein Schedule S ist serialisierbar, wenn S äquivalent zu irgendeinem seriellen Schedule S' ist. Alles Leseaktionen lesen den gleichen Wert und alle geänderten Werte sind nach S und S' gleich.

Konfliktgraph

Die Serialisierbarkeit eines Schedules S kann mit Hilfe eines Konfliktgraphen G überprüft werden. Der Graph G enthält einen Knoten pro Transaktion und eine Kante pro Konflikt zwischen zwei Transaktionen. Die gerichtete Kante im Graph G stellt einen Konflikt zwischen Transaktion T_x nach T_y , dar, wenn $s_x(o_k)$ vor $s_y(o_k)$ auf dem gleichen Objekt o_k im Schedule liegt und s_x und/ oder s_y eine Schreiboperation auf o_k ist. Ist der Konfliktgraph G zyklensfrei, dann ist der Schedule S serialisierbar.

Ein Scheduler kann auf unterschiedliche Arten implementiert werden. Pessimistische Ansätze vermeiden von vorne herein Konflikte (durch Locks), optimistische Ansätze erkennen Konflikte im

Nachhinein. Dadurch haben optimistische Ansätze häufig einen höheren Durchsatz, wenn es weniger Konflikte gibt.

Lock-based Concurrency Control

Transaktionen holen sich zum Lesen oder Ändern ein Lock auf das Objekt, nebenläufige Transaktionen müssen auf Locks warten und Transaktionen geben das Lock nach erfolgreicher Änderung wieder frei.

Bei Two-Phase Locking findet Locking in zwei Phasen statt (Grow- und Release-Phase) -> In Release, dann keine neuen Locks. 2PL garantiert serialisierbare Schedules aber hat zwei Probleme: Deadlocks und Cascading Aborts.

Deadlocks: Transaktionen können sich gegenseitig blockieren, dadurch müssen alle Transaktionen unendlich lange warten. Deadlocks können durch Timeouts oder Warte-Graphen aufgelöst werden bzw. durch Preclaiming vermieden werden. Beim Preclaiming werden alle Locks atomar vor Beginn der Transaktion angefordert, dabei sind aber bei Beginn einer Transaktion nicht alle Locks bekannt.

Cascading Aborts: spätere Transaktionen müssen zurückgerollt werden, wenn eine frühere Transaktion abgebrochen wird. Die Commit Reihenfolge muss korrekt ausgeführt werden. Um das zu vermeiden, nutzt man die Strict 2PL, dabei werden alle Locks bis zum Ende der Transaktion gehalten und dann atomar freigegeben.

Isolationsstufen: Niedrigere Isolationsstufen führen i.d.R. zu einem höheren Durchsatz von Transaktionen aber mehr Anomalien: (niedrig->hoch):

- Read uncommitted: Transaktionen fordern keine Sperren an. Diese Isolationsstufe ist nur für Lese-Transaktionen
- Read committed: Transaktionen fordern Write-Locks mit Strict 2PL an. Lese-Locks werden direkt freigegeben
- Read stability: Schreib- und Lese-Locks werden mit Strict 2PL angefordert
- Serializable: Schreib- und Lese-Locks werden mit Strict 2PL angefordert mit Lösung für Phantom-Problem

Multi-Version Concurrency Control

Jedes Datenobjekt liegt in mehreren Versionen vor. Jeder Schreiber erzeugt eine neue Version, alte Versionen bleiben erhalten, Garbage Collection notwendig. Damit besitzen Leser die zusätzliche Freiheit, eine passende Version zu lesen. Der Vorteil ist, dass Leser und Schreiber sich nicht gegenseitig blockieren können.

Optimistic Concurrency Control

Konflikte werden erst am Ende einer Transaktion erkannt, die Transaktionen werden in 3 Phasen ausgeführt:

1. Execute Phase: Transaktionen lesen nur Daten und Schreiben Änderungen in einen privaten Bereich (ohne Prüfung)
2. Validation Phase: Transaktionen prüfen erst vor dem Commit ob es Konflikte gab, falls ja wird abgebrochen
3. Installation Phase: Transaktion schreibt ihre Änderungen vom privaten Bereich in die Datenbank

Phase 2 und 3 müssen atomar ausgeführt werden.

Natural Language Text

Text sind geschriebene, kohärente Daten einer Sprache, Kohärenz gibt einem Text eine semantische Bedeutung. Daten sind nicht interpretierte, ausgetauschte Nachrichten. Eine Nachricht sind strings, die einer Grammatik folgen. Strings sind eine Sequenz von Symbolen oder Zeichen aus einem Alphabet (Script). Das Script definiert das Alphabet (symbol set). Es gibt unterschiedliche Arten von Scripts: Alphabetic, Logographic/syllabic, Consonant-based oder Segment-based. Diese unterscheiden sich auch durch ihre Schreib-Richtung. Sprache definiert das Vokabular und die Grammatik.

Eigenschaften von Text:

- Texttyp / Genre: Bucht, Artikel, News, Social Media Post, ...
- Register: Sprachvariante, Dialekt, ...
- Style: Schreibstil
- Domain/topic
- Time of writing

Textstruktur: Chapter/Section -> Paragraph -> Sentence -> Phrase -> Word -> Buchstabe /Zeichen

Um Text in einer relationalen Datenbank zu speichern, nutzt man entweder ein Binary large object (BLOB), für Dateien, Bilder, Audio oder Videos oder man nutzt Character large objects (CLOB) für Text.

Character Encoding

Eine encode-Funktion um Zeichen auf eine Bitfolge abzubilden und eine decode-Funktion um Bitfolgen auf Zeichen abzubilden. Es gibt verschieden Arten von Encoding:

- Single byte: Jedes Zeichen wird zu einem einzigen Byte enkodiert
- Multi byte: Jedes Zeichen wird zu einer festgelegten Anzahl an Bytes enkodiert
- Variable length: Die Bit-Länge ändert sich entsprechend einem Zeichen und vorgeschriebenen Regeln
- Escape-code-based: Wechseln zwischen verschiedenen Zeichensätzen mithilfe von Escape-Codes

Unicode

Zeichensatz, der alle Zeichen beinhaltet, jedes Zeichen hat 4 Byte mit 21 significant bits. Unicode ist in 17 planes unterteilt:

- Plane 0: Basic Multilingual Plane (BMP)
- Plane 1: Supplementary Multilingual Plane (SMP)
- Plane 2: Supplementary Ideographic Plane (SIP)
- Planes 3-13: noch leer
- Plane 14: Supplementary Special-purpose Plane (SSP)
- Planes 15-16: Supplementary Private Use Area (PUA-A/B)

UTF-32/UCS-4 Encoding: Jedes Zeichen mit genau 4 Bytes enkodieren, direkte Korrespondenz zum Unicode Zeichensatz, Nachteile: längere Verarbeitungszeit und Verschwendung von Speicherplatz

UCS-2 Encoding: Jedes Zeichen wird mit genau 2 Bytes enkodiert, direkte Korrespondenz zum Unicode Zeichensatz

UTF-16 Encoding: Jedes Zeichen wird mit 2 bis 4 Bytes enkodiert, direkte Korrespondenz zu Unicodes
BMP, Teilen der Zeichen der Planes in High und Low Surrogates

U+0	=	0000	0000	0000	0000	
U+00DF	=	0000	0000	1101	1111	
U+FFFF	=	1111	1111	1111	1111	
U+10FFFF	=	1101	1011	1111	1111	1101 1111 1111 1111
		<i>high surrogate</i>				<i>low surrogate</i>

UTF-8 Encoding: Jedes Zeichen wird mit 1 bis 4 Bytes enkodiert, direkte Korrespondenz zu ASCII, nicht-ASCII Zeichen brauchen extra Bytes

U+0 = 0000 0000

U+0068 = 0110 1000

U+00DF = 1100 0011 1001 1111

U+FFFF = 1110 1111 1011 1111 1011 1111

U+10FFFF = 1111 0100 1000 1111 1011 1111 1011 1111

single bytes

continuation bytes

leading bytes

trailing bytes

Encoding bringt in Software und DB einige Probleme mit sich, welche Länge hat ein enkodiertes Zeichen, Suchen und Sortieren.

Collation Framework: Das Framework hilft beim Definieren einer Such-Reihenfolge:

1. Anfangsbuchstaben normalisieren
2. Diakritische Zeichen behandeln (ä,ö,ü,ä, ...)
3. Den Wortfall untersuchen (resume, résumé)
4. Sonderzeichen betrachten

Linguistic Preprocessing

Tokenization

Tokenization ist der Prozess des Aufteilens eines input streams in eine geordnete Sequenz von Tokens. Normalerweise entsprechen Tokens einem Wort oder einer Wortform. Ein häufiges Problem ist, dass manchmal Zeichen zu einem Token dazu gehören und nicht geteilt werden (Beispiel: 1,543 oder 1, und 543 oder ganz anders).

Morphologie

Die Studie von Wortformen und Wortformationen, Wörter bestehen aus Morphemen, Morpheme sind die kleinste lautliche Einheit, welche in unterschiedlichen sprachlichen Zusammenhängen die gleiche Bedeutung trägt.

Ein freies Morphem oder Stamm ist ein Wort, welches nicht weiter zerlegt werden kann (z.B. Katze).

Gebundene Morpheme oder Affixe sind Anhänge an Wortstämme (z.B. -en bei Katzen).

Affixtypen:

- Suffix: nach einem Wortstamm
- Prefix: vor einem Wortstamm
- Infix: in einem Wortstamm

- Circumfix: um einen Wortstamm

Wortbildung

Derivation: Bilden neuer Wörter durch freie Morpheme und Affixe

Konversion: Sprachänderung eines Wortstammes ohne ein Affix

Komposition: Bilden neuer Wörter durch Verbinden mehrerer freier Morpheme

Dekomposition: Aufteilen eines komponierten Wortes in seine Ursprungsteile

Normalisierung

Stemming: Algorithmischer Ansatz, um Wortendungen zu entfernen (aus indices wird indic)

- Under-Stemming: Stemmer transformiert verwandte Wörter nicht zum selben Stamm
- Over-Stemming: Stemmer kann unterschiedliche (gleichgeschriebene) Wortstämme nicht unterscheiden
- Ambiguity: Gleichbedeutung gleich geschriebener Wörter (Teekesselchen)

Lemmatization: echten Wortstamm bestimmen (aus indices wird index)

Syntax

Syntax beschreibt die Art, in welcher Wörter angeordnet sind.

Part of Speech Tagging (POS Tagging)

Der Prozess des Zuweisens bestimmter Marker zu jedem Wort in einem Text. Zum Beispiel Nomen, Verben, Adjektive, Adverbien, Präposition, Pronom, Artikel, ... Part-of-speech tagged Daten bieten wichtige Informationen über die Wort-Formation, mögliche Wortnachbarn, korrekte Aussprache, Wortsinn, und Bedeutung eines Satzes.

Rule-based Tagging: Taggen via Regeln, welche vorher festgelegt sind.

Probabilistisches Tagging: Die Wahrscheinlichkeit berechnen, dass ein Wort einen bestimmten POS tag bekommt, das Wort bekommt dann den wahrscheinlichen POS tag.

Parsing

Parsing beschreibt die grammatische Struktur eines Satzes mithilfe eines Baumes.

Phrasenstrukturgrammatik: Satz aufteilen in seine Konstituente.

Konstituente: Gruppe von Wörtern, die wie eine einzelne Einheit agieren.

Phrases: Kontinuierliche Sequenz von Wörtern die zu einer einzigen syntaktischen Einheit gehören:

- Noun phrase (NP) mit Nomen als Kopf
- Prepositional phrase (PP) mit einer Präposition als Kopf
- Verb phrase (VP) mit einem Verb als Kopf
- Adjectival phrase (AP) mit einem Adjektiv als Kopf
- Adverbial phrase (AdvP) mit einem Adverb als Kopf

Der Phrasehead bestimmt den syntaktischen Typ einer Phrase, das „key“ Wort.

Grammatische Modifier, optionales Element einer Phrase, welches die Bedeutung der Phrase verändert. Premodifier (vor dem Wortkopf) und Postmodifier (nach dem Wortkopf).

Beim parsing wird für einen gegebenen Satz überprüft, ob dieser Teil einer Sprache ist, kann s durch eine Grammatik G generiert werden? Falls ja ist s ein grammatischer Satz, falls nicht, ist s ein ungrammatischer Satz.

Kontextfreie Grammatik

Eine kontextfreie Grammatik ist $G = (T, N, S, R)$ mit einer Menge Terminalen T, einer Menge Nicht-Terminalsymbole N, einem Startsymbol $S \in N$ und einer Menge an Produktionsregeln R der Form $X \rightarrow y$ wobei X ein Nichtterminalsymbol und y eine Sequenz von Nichtterminalen ist. Eine Grammatik G generiert eine Sprache $L(G)$.

Syntaktische Ambiguity (Mehrdeutigkeit)

Manchmal hat ein Satz mehr als nur eine korrekte syntaktische Bedeutung.

Attachment ambiguity: Ein Konstituent kann an unterschiedlichen Stellen im Parse Baum platziert werden.

Coordination ambiguity: Unterschiedliche Umfang einer Wortverbindung

Garden path sentences: Grammatische Sätze in denen eine Phrase offensichtlich nicht funktioniert.

Semantik

Semantik ist die Untersuchung über Bedeutung. Lexikalische Semantik ist die Untersuchung über die Bedeutung eines lexikalischen Elements (z.B. Wörter). Strukturelle Semantik ist die Untersuchung über die Beziehung zwischen lexikalischen Bedeutungen verschiedener Elemente in einem größeren Kontext.

Lexikalische Mehrdeutigkeit sind unterschiedliche Interpretationen eines Wortes.

Syntaktische Mehrdeutigkeit sind unterschiedliche Interpretationen eines Satzes im Bezug auf syntaktische Unterschiede (z.B. bekommt ein Satz eine andere Bedeutung durch die Position eines Kommas).

Lesartendisambiguierung ist der Prozess des korrekten Interpretierens eines Wortes mithilfe Hintergrundinformationen.

Pragmatik

Was ist die Intention in einer Äußerung?

Linguistic Analysis Levels

Phonetics and Phonology -> Segmentation -> Morphology -> Syntax -> Semantics -> Pragmatics

Textkorpus

Eine Sammlung von Texten, die zum Trainieren oder Evaluieren eines Systems genutzt werden.

Ein paralleler Korpus ist ein Dokument, welches in zwei oder mehr Sprachen übersetzt wird, wobei jeder Satz einzeln übersetzt sind.

Parameter eines Textkorpus: Sprache: Kommunikation, Größe, Annotationen, Statisch/Dynamisch, Genre/Texttyp, Domain/Thema, Zeit der Zusammenstellung

Annotationen

Erweitern eines Korpus mit unterschiedlichen Informationen, diese können auf unterschiedlichen Ebenen angebracht werden. Am Wort, an einer Phrase, an einem Satz oder an einem Diskurs. Der Annotierungsprozess ist meistens teuer und dauert lang.

Korpus-Erstell-Workflow

1. Abrufen und speichern der originalen Dokumente
2. In Plaintext konvertieren
3. Segmentieren
4. Manuell oder automatisch Annotationen erzeugen
5. Alle Korpus nutzbaren Formate speichern
6. Analysieren oder benutzen des Korpus

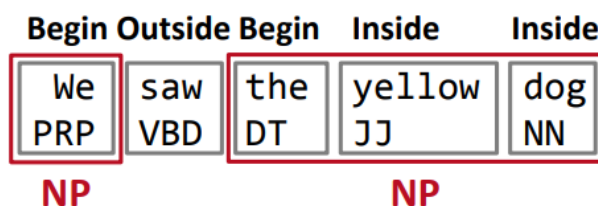
Inline Annotationen: Annotationen werden direkt im Text gespeichert, verändert das Originalformat.

Stand-off Annotationen: Annotationen werden in einem separaten Dokument gespeichert.

IOB Annotations Schema

Wird häufig für Annotationen genutzt, die mehrere Worte umspannen sollen.

- B = Beginn eines annotierten Bereichs
- I = in einem annotierten Bereich
- O = außerhalb eines annotierten Bereichs



We	PRP	B-NP
saw	VBD	O
the	DT	B-NP
yellow	JJ	I-NP
dog	NN	I-NP

Kollokationen: Sequenz von Wörtern, welche häufig gemeinsam auftreten

Distributional Hypothesis: Wörter, die im gleichen Kontext auftreten haben eine ähnliche Bedeutung.

Die Distributional Hypothesis wird häufig für Begriffserklärung und Word Space Models (Wort- oder Textähnlichkeit berechnen) genutzt. Anwendung finden diese Techniken häufig bei Dokument Klassifizierung/ Gruppierung, Informationsgewinnung, Fragen beantworten, Paraphrasierungshilfe, Worteinbettung und Deep Learning.

Lexikalische Ressourcen

Wörterbücher, Enzyklopädien, Wortnetze, Thesauri und weitere Typen.

Textkorpus	Lexikalische Ressource
Gesammelt von echtem Text und Sprache	Abgeleitet von Textkorpus
Beinhaltet mehrere Vorkommen eines Lemmas	Beinhaltet ein Lemma meistens nur einmal
Häufige Phänomene treten häufiger auf	Seltene und häufige Phänomene werden gleich behandelt
Zeigt wie Sprache genutzt wird	Beschreibt wie Sprache genutzt wird
Bietet typischen Kontext und Häufigkeiten	Bietet Meta Informationen

Lexikalische Ambiguität und Synonyme:

Lexikalische Ambiguität bedeute, dass ein Wort verschiedene Bedeutungen haben kann, Synonyme sind verschieden Worte die die gleiche Bedeutung haben.

Information Retrieval

Informationssuche (IR) ist das Finden von Material (meistens Dokumente) einer unstrukturierten Natur (meistens Text), welches einen Informationsbedarf aus einer größeren Sammlung stillt.

Informationsbedarf: Status (einer Person), welche Informationen benötigt, um ein Problem zu lösen (auch Intent genannt).

Suchanfrage: Systeminterpretierbare Darstellung des Informationsbedarfs

Relevanz: Eigenschaft eines Dokuments in Abhängigkeit zum Informationsbedarf, wie lässt sich Relevanz tatsächlich beurteilen?

IR Engine vs DBMS: DBMS benötigt strukturierte Daten, eine IR Engine arbeitet mit unstrukturierten Daten.

Herausforderungen von IR

- Ergebnisanzahl: Welche relevanten Dokumente auswählen?
- Relevanz: Manche Ergebnisse sind relevanter als andere
- Absicht: Die Suchanfrage erfüllt möglicherweise nicht den kompletten Informationsbedarf, was möchte der User tatsächlich?
- Lexikalische Lücke
- Ambiguität

Indexieren: Eine Dokumentationsrepräsentation, um eine Suche zu verbessern

Suchen: Interpretieren einer Suchanfrage und Dokumente zurückgeben, welche möglichst nah der Dokumentationsrepräsentation kommen, Relevanz festlegen, Ergebnisreihenfolge festlegen und Verbessern durch User Feedback.

Bool'sche Suche

Einfachste Informationssuche, Dokumente werden als Wortmengen behandelt es wird nach Dokumenten gesucht, die ein bestimmtes Wort enthalten, ein bestimmtes Wort nicht enthalten, Wort 1 und Wort 2 enthalten oder Wort 1 oder Wort 2 enthalten.

Inverted Index: TODO

Probleme mit bool'schen Suchen:

- Need for Syntax: Nutzer müssen bool'sche Ausdrücke verwenden zum Suchen, Lösung: Keyword-based Suchen erlauben
- Feast or famine: Bool'sche Suche liefern entweder zu wenige (ca 0) oder zu viele (>>1000) Ergebnisse, Lösung: Rangfolge

Ranked Retrieval: Sortieren der Ergebnisse nach Relevanz

Wortmenge: Ein Dokument enthält einen Begriff oder nicht (Duplikate werden ignoriert)

Wortbeutel: Ein Dokument enthält einen Begriff unterschiedlich oft (Duplikate werden nicht ignoriert)

Term Frequency (TF): Funktion welche angibt, wie häufig ein Wort in einem Wortbeutel auftritt.

Vector Space Model (VSM)

Die Terme der Term-Dokument-Matrix spannen einen Vektor-Raum auf. Jedes Dokument ist ein Dokumentvektor in diesem Raum

Cosinus Ähnlichkeit: Suchvektor q und Dokumentvektor v : $\text{sim}(q, v) = \frac{q \cdot v}{|q| \cdot |v|}$

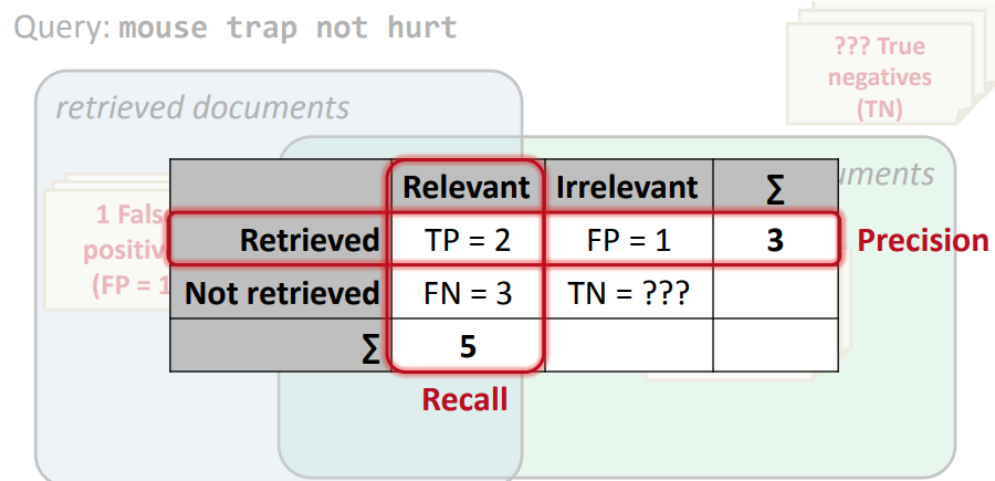
Vector Weights: bestimmen des Vektorinhalts eines Dokumentenvektors $v = \{w_{t1}, w_{t2}, \dots, w_{tn}\}$

- Binary: $w_t = \begin{cases} 1 & \text{if } t \in d \\ 0 & \text{otherwise} \end{cases}$
- Term Frequency (TF): $w_t = \begin{cases} tf_d(t) & \text{if } t \in d \\ 0 & \text{otherwise} \end{cases}$ mit $tf_d(t)$ die Anzahl der Vorkommen von t in d
- Normalized TF: $w_t = \begin{cases} \log(tf_d(t)) + 1 & \text{if } t \in d \\ 0 & \text{otherwise} \end{cases}$
- Document Frequency (DF): $w_t = \begin{cases} \text{some function of } \log(tf_d(t)) + 1 \text{ and } df(t) & \text{if } t \in d \\ 0 & \text{otherwise} \end{cases}$
mit $df(t)$ die Anzahl an Dokumenten, in denen t vorkommt
- Inverse DF (IDF): $w_t = \begin{cases} \text{some function of } \log(tf_d(t)) + 1 \text{ and } idf(t) & \text{if } t \in d \\ 0 & \text{otherwise} \end{cases}$ mit
 $idf(t) = \log\left(\frac{\#documents}{df(t)}\right) = \log\left(\frac{|D|}{|\{d \in D | t \in d\}|}\right)$
- TF.IDF: $w_t = \begin{cases} (\log(tf_d(t)) + 1) \cdot idf(t) & \text{if } t \in d \\ 0 & \text{otherwise} \end{cases}$
 - Unterschiedliche Normalisierung möglich: $w_t = \begin{cases} tf_d(t) \cdot idf(t) & \text{if } t \in d \\ 0 & \text{otherwise} \end{cases}$

Evaluation von IR-Systemen

Precision: Anteil der relevanten Dokumente der bezogenen Dokumente $P = \frac{\text{retrieved} \cap \text{relevant}}{\text{retrieved}}$

Recall: Anteil aller relevanten Dokumente bezogen auf die bezogenen Dokumente $R = \frac{\text{retrieved} \cap \text{relevant}}{\text{relevant}}$



$$P = \frac{\text{\#TP}}{\text{\#TP} + \text{\#FP}} = \frac{2}{3}$$

$$R = \frac{\text{\#TP}}{\text{\#TP} + \text{\#FN}} = \frac{2}{5}$$

F1-Score: $\frac{2PR}{P+R}$

Precision at Rank: $P@R = \frac{\text{\#relevant docs before } R}{R}$

Informationsextraktion

Information Retrieval	Information Extraction
Output: Liste relevanter Dokumente	Output: Menge an strukturierten Fakten aus einem Dokument
Einfacher	Schwerer (da zunächst IR stattfinden muss)
Domain-Unabhängig	Domain-Abhängig
Suchtyp nicht beschränkt	Vordefinierter Suchtyp
Schneller	Langsamer
Weniger effektiv	Effektiver, da der Nutzer direkt Antworten erhält

Entity Types: Organization, Person, Location, Date, Time, Money, Percent, Facility, ...

Herausforderungen an Entity Recognition:

- Entity vs non-entity
- Coverage issues
- Variation
- Ambiguity
- Time dependency
- Multi-word expressions (Mehrwortausdrücke)
- Metonymy (Metonymie): Wenn eine Entität nicht mit ihrem Namen genannt, sondern mit etwas, was man sofort mit ihr assoziiert genannt wird

List Lookup Approach

Entitäten werden nur anhand einer Entitäten-Liste erkannt. Dieser Ansatz ist einfach, schnell und einfach anpassbar aber die Listen müssen erstellt und maintained werden, mit Namensvariationen, Abkürzungen und Ambiguitäten kann nicht umgegangen werden.

Rule-based Approach (Pattern Matching)

Handgeschriebene Regeln, meistens durch eine Menge an regulären Ausdrücken. Dieser Ansatz ist sehr aufwändig, liefert aber einigermaßen gute Performanz, richtig gute Performanz zu erreichen ist schwer und das maintainen und anpassen auf neue Domains ist aufwändig.

Machine Learning Approach

Supervised Machine Learning: Erstellen eines hand-gelabelten Datensatzes, die Maschine lernt Regeln oder statistische Modelle um bisher ungesehen Daten zu labeln.

Corpus-based probabilistic Models: Schätzen der Wahrscheinlichkeit für bestimmte Sprachstrukturen anhand großer Textkorpusse.

Knowledge Bases

Knowledge Bases sind Knowledge Graphs, in ihnen werden die Daten einer IE gespeichert.

Konstruktions Modelle:

- Curated approaches: Triples werden manuell von einer geschlossenen Gruppe von Experten erstellt, (+) Hohe Qualität, (-) Niedriger Skalierbarkeit
- Collaborative approaches: Triples werden manuell von einer offenen Gruppe von Freiwilligen erstellt, (+) bessere Skalierbarkeit, (-) trotzdem limitiert

- Automated Semi-structured approaches: Triples werden automatisch aus semi-strukturiertem Text extrahiert, (+) große Knowledge Graphen, (-) es wird nur ein kleiner Teil des Internets abgebildet
- Automated unstructured approaches: Triples werden automatisch aus unstrukturiertem Text extrahiert, (+) riesen Knowledge Graphen

Automatische Klassifikation

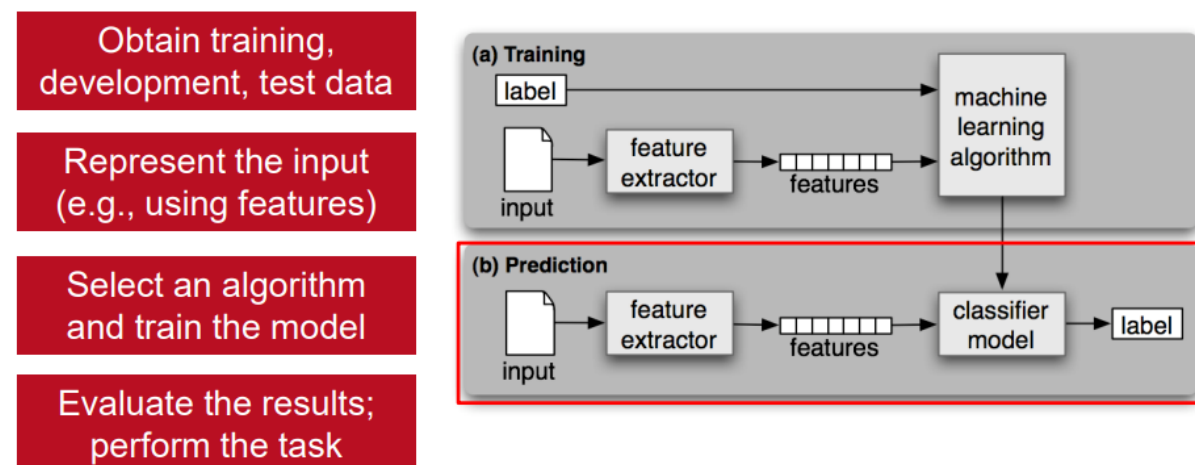
Klassifikation ist der Prozess, die richtige Klassenbezeichnung für einen gegebenen Input basierend auf seinen Eigenschaften zu wählen.

Varianten:

- Binär vs multi-class Klassifikation
- Single-label vs multi-label Klassifikation
- Sequenzklassifikation

Ein classifier wird supervised genannt, wenn er auf Trainings-Korpusen trainiert wurde, welche die korrekten Label für jeden Input beinhalten.

General (Supervised) ML Workflow



Wenn möglich verschiedene Datensätze für Entwicklung und Testen benutzen, sonst einen großen Datensatz in Training, Dev-Test und Test Set aufteilen.