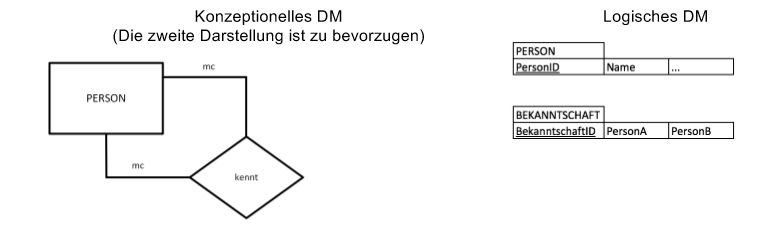
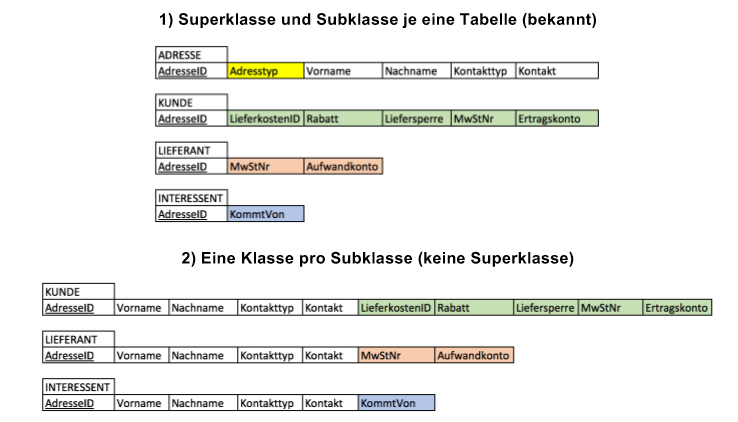
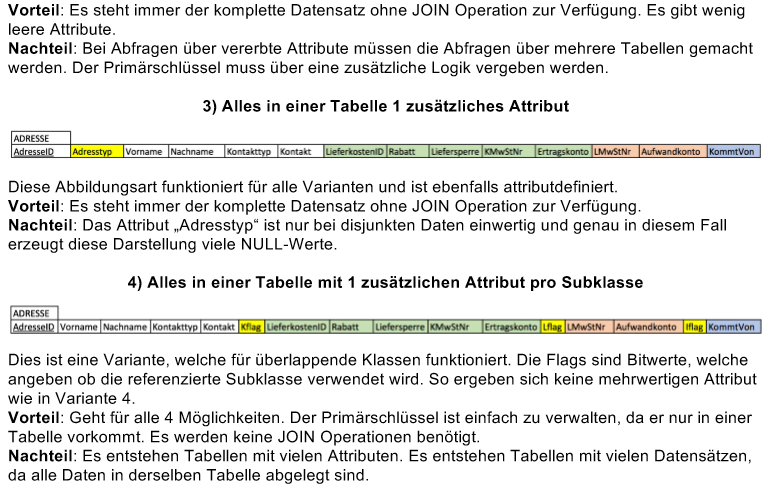
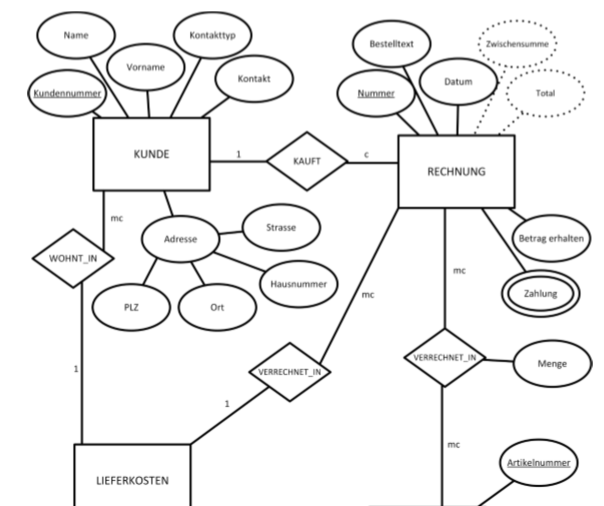
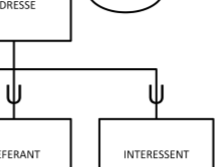
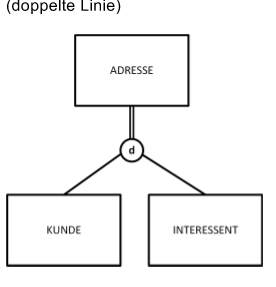
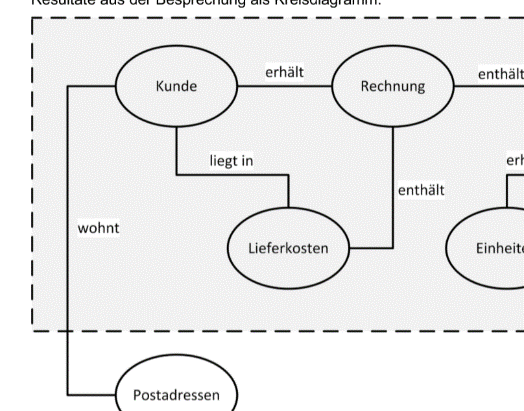
Im Beispiel rechts ist die Klasse Privatkunde eine Spezialisierung von Person. Sie deklariert das [Attribut](https://de.wikipedia.org/wiki/Attribut_(UML)) kundennummer und verfügt zusätzlich implizit über die Attribute name und vorname aus der Klasse Person

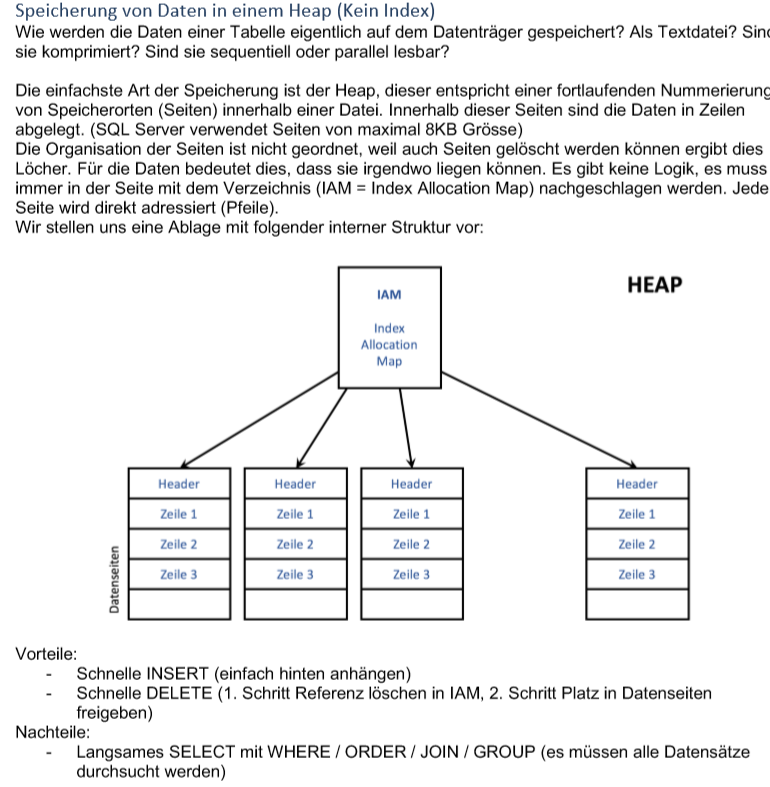




Disjoint – overlap

totale spezialisierung

Speicherung von Daten in einem clustered Index (Primärspeicher) Ein clustered Index ist nichts anderes als ein Index, bei welchem alle Attribute einer Tabelle in den Blättern abgespeichert werden. (wie ein include \*). Da der clustered Index alle Daten im Original vorliegen – es gibt kein Heap mehr daneben – nennt sich dieser auch Primärspeicher.

Vorteile: - Daten werden schnell gefunden wie bei einem „normalen“ Index - Keine weiteren Sprünge auf Heap notwendig - Keine Redundanz Nachteile: - Nur für ein Attribut (oder zusammengesetzter Schlüssel) möglich

Speicherung von Daten in einem clustered Index plus Zusatzindex

Dabei zeigt die Adressierung des Zusatzindex’ auf den Primärschlüssel des clustered Index und nicht wie beim Heap direkt auf die Speicherseite. Dies kann dazu führen, dass der Zugriff langsamer ist als beim normalen nonclustered Index. Vorteile: - Mehrere Indizes möglich Nachteile: - Zugriff über nonclustered Index minimal langsamer als über Heap, da beide Bäume durchlaufen werden müssen

Speicherung von Daten in einem Heap (mit Index) Um die Nachteile des Heap (langsame SELECT) auszugleichen, kann zusätzlich zu den Datenseiten ein Index angelegt werden. Wie funktioniert dieser Index? Ein Index ist in einer Baumstruktur abgelegt. Der SQL Server verwendet einen „Balanced Tree“ (BTree), welcher an allen Verästelungen links und rechts gleich viele Blätter trägt. Ein Index auf Zeichenkette ist unten abgebildet. Im obersten (root) Element wird nicht fix das Alphabet halbiert, sondern es kommt auf die Anzahl der Einträge pro jeweiligem Anfangsbuchstaben an! Es gibt B-Trees, welche in den Blättern mehrere Elemente zulassen (siehe Bild unten A-D) und es gibt B-Trees, welche auch in der untersten Hierarchie (Blätter) nur maximal 2 Elemente zulassen. Die letzteren sind einfacher zu berechnen, haben aber die grössere Anzahl Hierarchiestufen.

Ersparnis: Je nach Aufbau des B-Trees lässt sich die Suchzeit enorm verkürzen. Stellen wir uns einen B-Tree vor mit Zahlen zwischen 1-1024, welche keine Duplikate enthält (wie ein Primärschlüssel). Das bedeutet, dass in maximal 10 Schritten jedes Element aufgefunden werden kann! Wären die Daten in einem Heap (ohne Index) gespeichert worden, wären folgende Suchen notwendig: - Wenn Duplikate vorkommen ist eine komplette Suche durch alle 1024 Elemente („Full-RowScan“) nötig - Wenn keine Duplikate vorkommen wird im Durchschnitt nach der Hälfte der Elemente das gesuchte gefunden. Es werden 512 Elemente durchsucht.