* Danke, Ich werde nun mit SW-DB fortfahren
* Anfangen werde ich damit, was eine SW-DB eigentlich ist

**Was ist eine SW-DB (Eng. Key-Value-Store)**

* NoSQL 🡪 verfolgt nichtrelationalen Ansatz
* Besteht aus SW-paaren 🡪 Vorstellen wie eine Tabelle mit den Spalten Schlüssel/Wert
* Operieren meist im Hauptspeicher 🡪 + in Performance durch Hardware
* Vorteile
  + Schnelle Lese-/Schreiboperationen // durch einfache Struktur
  + Flexibel
    - Attribute lassen sich beliebig einfügen/entfernen
  + Alle Datenformate möglich //
    - Von True oder False repräsentiert durch 1 oder 0 bis BLOBs (Binary Large Objects)
* Use Cases
  + Sitzungsdetails in Webanwendungen
    - Informationen können einfach in ein Profil (Schlüssel) als Sitzungsdetails (Werte) gespeichert/abgerufen werden
  + Datencaching
    - Daten in-Memory zwischenspeichern, um langwierigen Lese-/Schreibvorgängen auf Festplatten zu entgehen
    - Analysedaten zwischenspeichern (Analyseergebnisse einer komplexen SQL-Abfrage zwischenspeichern, damit die Abfrage nicht immer neu ausgeführt werden muss)
* Hier gewählt Redis (Remote Dictionary Server)
  + Open-Source
  + Als verbreitetstes Schlüssel-Werte-Datenbank DBMS
    - Gut dokumentiert
    - Wird in vielen Programmiersprachen durch Bibliotheken unterstützt
      * Redis-py für Python 🡪 Alle für uns relevanten Operationen abbildbar

**Prinzip Schlüssel-Werte-Datenbank**

* Einfacher Schlüssel (Matrikelnummer) mit einfachem Wert (Name als String)
* Sprechender Schlüssel
  + Wert als Hash (Hash mappt Felder mit Werten 🡪 Objekte repräsentieren)
  + Wert als Liste
  + Wert boolean
* Man sieht
  + Datensätze bis auf Eindeutigkeit der Primärschlüssel unabhängig voneinander
  + Insbesondere unterschiedliche Attributanzahl und Attributformen
  + Unabhängigkeit führt auch dazu, dass Datensätze in keiner Beziehung zueinander stehen -> schwierig für Abfragen im Sinne von SQL

**Aktuelles Modell für unseren Anwendungsfall**

* Wichtig zu verstehen: Zur Veranschaulichung kategorisiere ich hier meine Datensätze, in Redis stehen diese wie vorhin beschrieben einfach hintereinander weg
* (Legende) Farbcodierung für jeden Datensatz Bildungsvorschrift Schlüssel (rot), Beispiel Schlüssel (grün) und Beispiel Wert (blau)
* Rohdaten 🡪 Daten wie sie uns zur Verfügung gestellt werden, abgebildet als Hash
  + Bildungsvorschrift „in“ (zur Kategorisierung), „:“ (Trennzeichen für die Informationen im Primärschlüssel), „\*inCounter\*“ (Zähler, um Eindeutigkeit des Schlüssels zu wahren)
  + Werte als Hash 🡪 Wir erinnern uns, quasi Objekt abgebildet (Eigenschaft SNR hat Ausprägung 325…)
* Verknüpfungen 🡪 Verknüpfen der Rohdaten
  + Bildungsvorschrift „InDat“ (Datum In), Trennzeichen, „inCounter“ als Information zur Herstellung der Verbindung zu den Eigenschaften, „OutDat“ (Datum des letzten Out), „SNR“ zur Kategorisierung der Informationen
  + Werte als Liste 🡪 „Verweise“ auf die Rohdaten
* Eigenschaften 🡪 Bilden Verbindungen der Ausprägungen der Eigenschaften mit den Verknüpfungen, welche diese Ausprägung besitzen
  + Bildungsvorschrift „Name der Eigenschaft“, Trennzeichen, Ausprägung
  + Werte als Bitmaps 🡪 0 heißt in der Verknüpfung ist diese Ausprägung nicht vorhanden, 1 heißt in der Verknüpfung vorhanden
  + Beispiel braun

**Eigenschaften der Analysen**

* Wichtige, große Datenmengen 🡪 Alle Verknüpfungen, gebrauchte Eigenschaften
* Weniger Abfragen an Datenbank 🡪 Viele Daten auf einmal holen
* Sehr gute Performance 🡪 Wenig Latenz (nicht immer gegen DB schießen), Zeit maßgeblich durch das genutzte Pythonscript beeinflusst
* Komplexe Struktur 🡪 Nicht leicht nachzuvollziehen wie SQL Statements, Abfragen letztendlich recht verschachtelt