InfoDesign

IM4-Archiver

Datenarchivierung mit JSON

Referent:
Niklas Hackelberg, InfoDesign GmbH

Agenda


Einstieg
Export
Serialisierung
Datenintegrität
Import
Ausblick

Einstieg

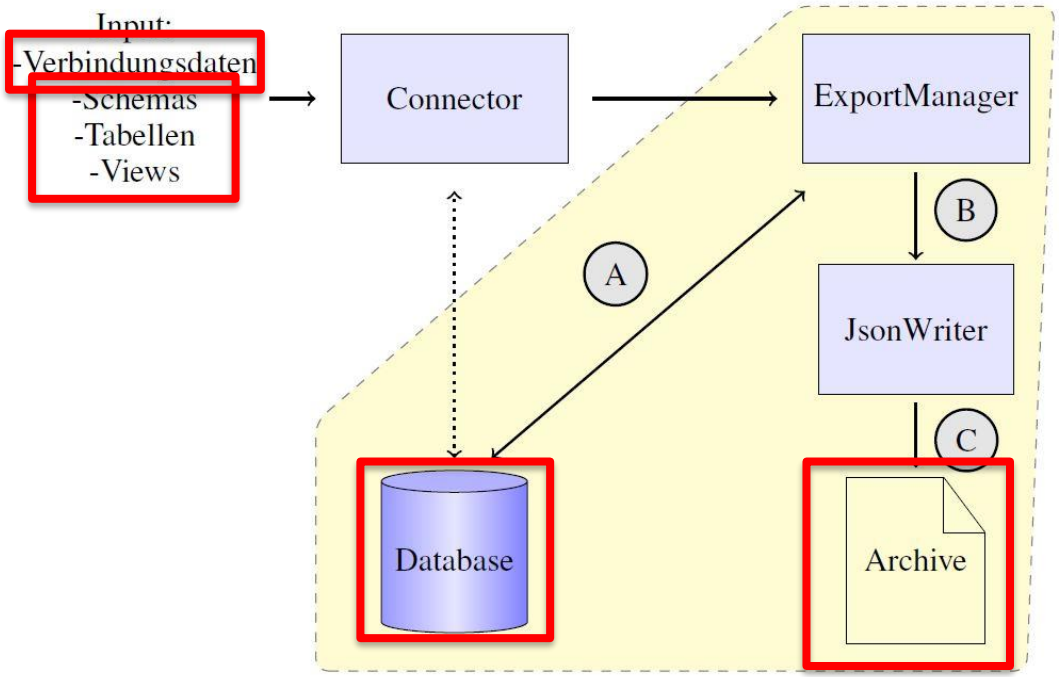
- Auslagerung von Daten…
- Aber langfristig gespeichert werden müssen
- Konflikt: Abhängigkeit von Produkten
- JSON:
  - Reines Textformat zum Datenaustausch
  - Les- und schreibbar in fast allen Programmiersprachen
  - Objekte mit Name/Wert Paare: {„Test“: 123}
  - Werte:
    - Objekte, Arrays/Listen
    - Beliebig lange Zeichenketten
    - Beliebig lange Ziffernketten
    - True, false & null
  - Keine binären Daten
- Java Programm: Alle OS mit JRE
- JDBC: Alle DB mit JDBC Treiber
  - Abbildung auf Java Klassen
  - Generische SQL Typen

Agenda


Einstieg
Export
Serialisierung
Datenintegrität
Import
Ausblick

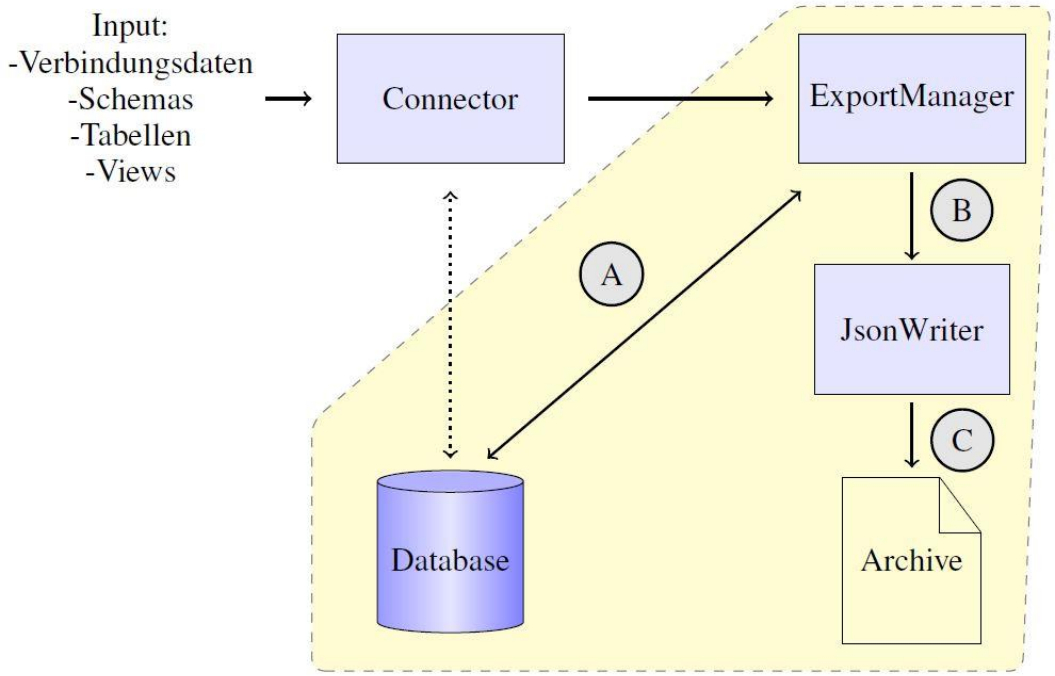
Export

- Was?
  - Schema
  - Tabellen
  - Views (ohne Def.)
  - Inhalt & Metadaten
- Wohin?
  - Archivdatei(en)
  - Reportdatei
  - IM4Web (Verwaltung)
- Von wo?
  - DB mit JDBC Treiber
  - Db2, PostgreSQL, MSSQL und Oracle
  - …



Export

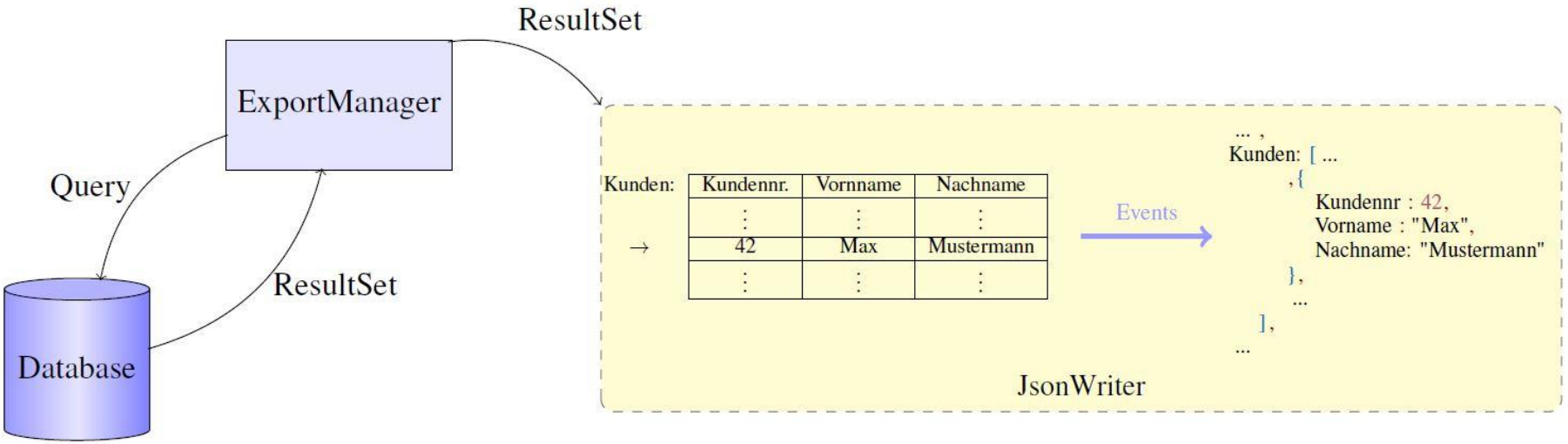
- A:
  - Ermitteln der Objekte
  - Abfrage der Metadaten
  - Abfrage des Inhalts
- B:
  - Serialisierung
  - Java Objekt -> JSON Objekt
- C:
  - Schreiben der JSON Datei
  - Prüfsummen
  - Speicherung im Dateisystem



Agenda


Einstieg
Export
Serialisierung
Datenintegrität
Import
Ausblick

Serialisierung



- ResultSet:
  - Antwort auf Queries
  - Cursor-Prinzip
  - getX (String, Integer, …)
  - getObject

- Pro Zeile:
  {
  Spalte 1: Wert 1,
  …,
  Spalte N: Wert N
  }

Serialisierung

- Metadaten:
  - Abfrage per JDBC Methoden
  - Abbildung auf generische SQL Typen
  - Spalten (Datentyp, Größe, …) & Abschätzung
  - Primary & Foreign Key, Indices
- Daten:
  - Abfrage per SELECT: Aufbereitung der Datenbank & JDBC
  - Unterstützte Datentypen:
    - Numerisch, ausg. ±Infinity, NaN
    - Binäre, Base64 codiert
    - Charakter
    - Dokumente (XML, JSON) & LOBs, Base64 codiert
    - Zeit, Datum und Zeitstempel, ISO 8601 Format
  - Abfrage der Werte (getX):
    i. Bestimmte Java-Klassen
    ii. JDBC internes Mapping
    iii. String

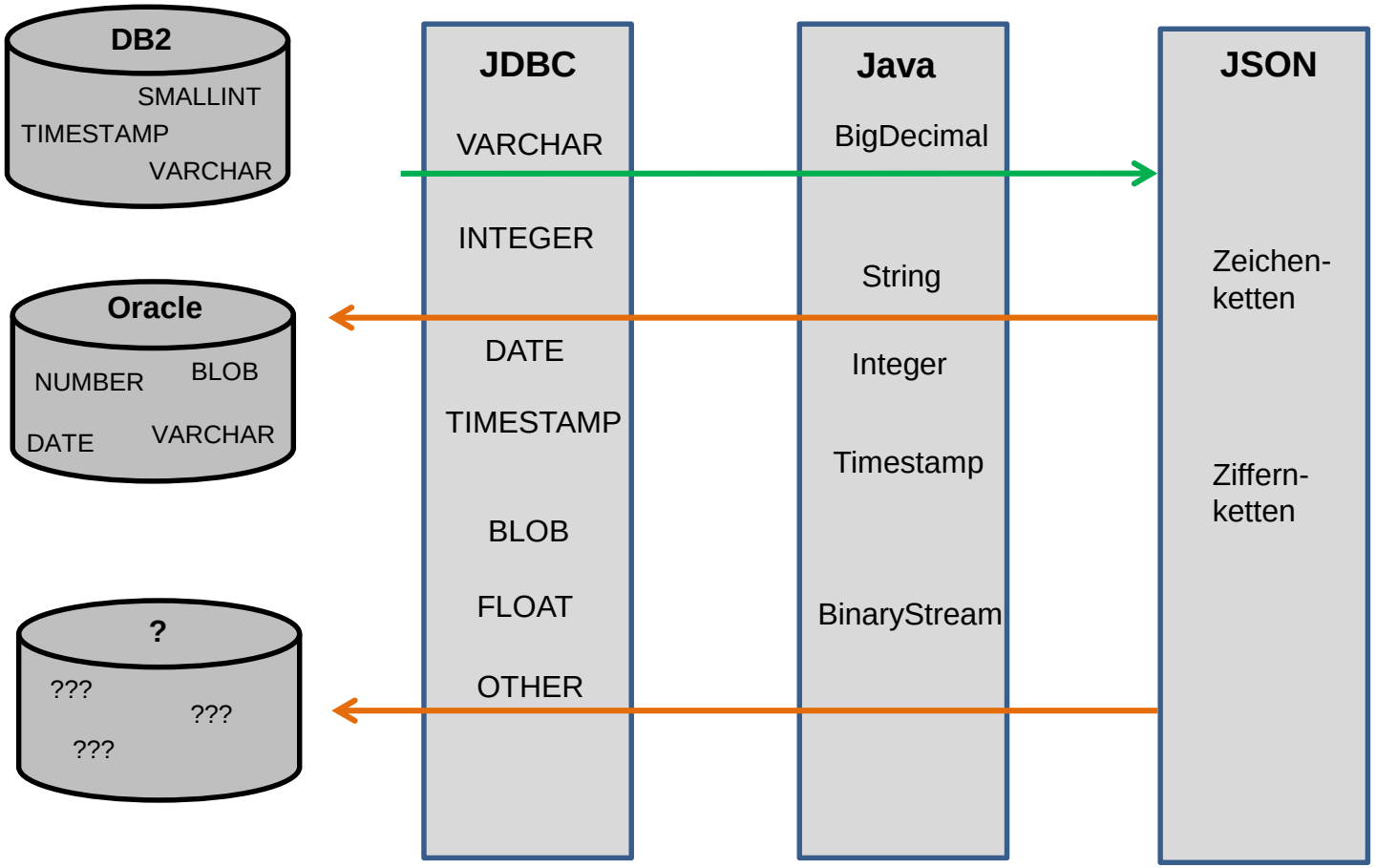
Serialisierung


```
{
 "DB_PROD" : "PostgreSQL",
 "DB_VER" : "10.5",
 "METADATA" : [ {
  "SCHEMA_NAME" : "awt",
  "TABLES" : [ {
   "TABLE_INFO" : {
    "TABLE_CAT" : null,
    "TABLE_SCHEM" : "awt",
    "TABLE_NAME" : "example",
    "TABLE_TYPE" : "TABLE"
   },
   "COLUMN_INFO" : [ {
    "COLUMN_NAME" : "kundennummer",
    "DATA_TYPE" : 4,
    "TYPE_NAME" : "serial",
    "COLUMN_SIZE" : 10,
    "DECIMAL_DIGITS" : 0,
    "IS_NULLABLE" : "NO",
    "DATA_EST" : 4
   }, …
```


java.sql.Types
Integer


```
"CONTENT" : [ {
  "SCHEMA_NAME" : "awt",
  "TABLES" : [ {
   "TABLE_INFO" : {
    "TABLE_CAT" : null,
    "TABLE_SCHEM" : "awt",
    "TABLE_NAME" : "example",
    "TABLE_TYPE" : "TABLE"
   },
   "RECORDS" : [ {
    "kundennummer" : 1,
    "vorname" : "Max",
    "nachname" : "Mustermann"
   }, {
    "kundennummer" : 2,
    "vorname" : "Erika",
    "nachname" : "Mustermann"
   } ]
  } ]
 } ]
}
```

Serialisierung



Serialisierung

- PostgreSQL:
  - JDCB getObject() nutzt internes Mapping
  - Bit(n): „010101“ -> getObject() als boolean
  - Money: „200.20€“ -> getObject() als double
- Generischer Datentyp OTHER & „eigene Typ Konstanten“:
  - Db2: Timestamp with Time Zone, DECFLOAT
  - PostgreSQL: Varbit, JSON, Geometry (Line, Point, …), …
  - MSSQL: -155 DatetimeOffset (Timestamp with Time Zone)
  - Oracle: 100 (Real), 101 (Double), -101 (Timestamp with Time Zone), …
- Large Objects:
  - MSSQL: Varchar(n), Varchar(MAX) -> VARCHAR
  - Postgres: Varchar(n), Varchar, Text -> VARCHAR, TOAST Werte, …

Serialisierung

- Oracle:
  - Number(p,s), Number(p), Number(*,s), Number(*), Number
  - Number(p,s), Number(p) -> p Ziffern, davon s/0 Nachkomma
  - Number(*), Number(*,s), Number -> 38 SIGNIFIKANTE Ziffern
  - 1.111e+125 hat 4 Signifikante Ziffern
  - JDBC: Number(38) = Number(*), Number = Number(0,-127)

Agenda


Einstieg
Export
Serialisierung
Datenintegrität
Import
Ausblick

Datenintegrität: Übersicht

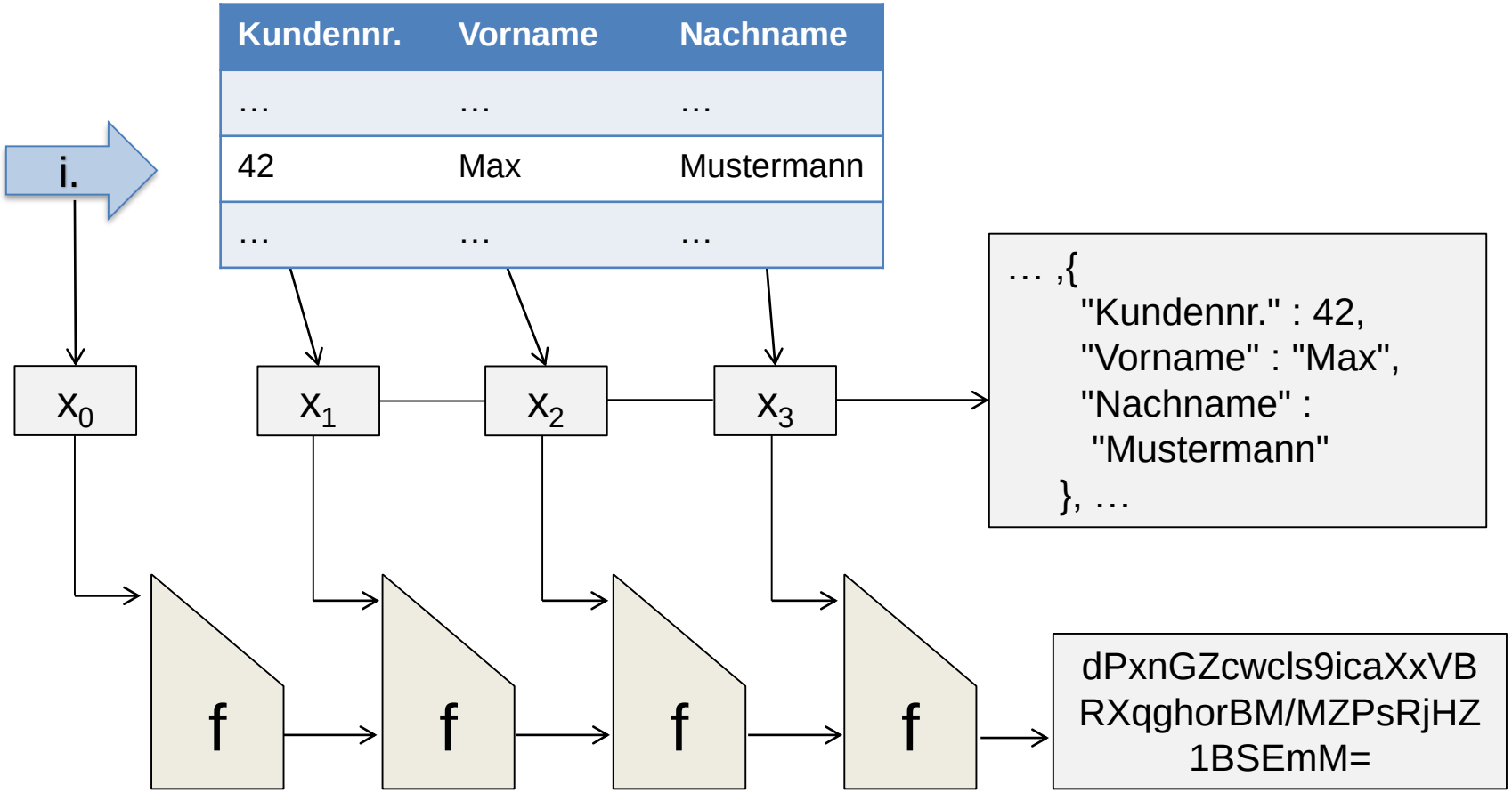
- Kompression & Prüfsummen:
  - GZIP
  - CRC32
- Reportdatei:
  - Übersicht über:
    - Dateien: Inhalt & Prüfsummen
    - Tabellen: Datei & Anzahl
  - Produkt, Version, DB-Name
  - Bemerkung (String, Datei, Dateityp)
- Hash auf Zeilen Ebene:
  - SHA256 Hash
  - Bildet beliebigen Input auf 256 Bit
  - Vergleich der Werte des Exports:
    - Mit den Werten im Archiv
    - Mit den Werten beim Import

Datenintegrität: Reportdatei


```
{
  "DIR_NAME" : "archiv",
  "ARCHIVE_NAME" :
"AWT_EXAMPLE",
  "DB_NAME" : "PG01",
  "DB_PROD" : "PostgreSQL",
  "DB_VER" : "10.5",
  "UID" : "postgres",
  "START_TIME" : 1542190637366,
  "END_TIME" : 1542190637426,
  "REMARK" : "Hier könnte ein
Vermerk stehen",
  "REMFILE" : null,
  "REMTYPE" : null,
…
```

```
…,
"FILES" : [ {
   "FILE_NAME" : "AWT_EXAMPLE.json",
   "CHECKSUM" : 486048327,
   "TABLES" : [ {
     "SCHEMA_NAME" : "awt",
     "TABLE_NAME" : "example",
     "RECORD_COUNT" : 2
   } ]
 } ],
 "TOTAL_RECORD_COUNTS" : [ {
   "SCHEMA_NAME" : "awt",
   "TABLE_NAME" : "example",
   "TOTAL" : 2
 } ]
}
```

Datenintegrität: Hashsummen

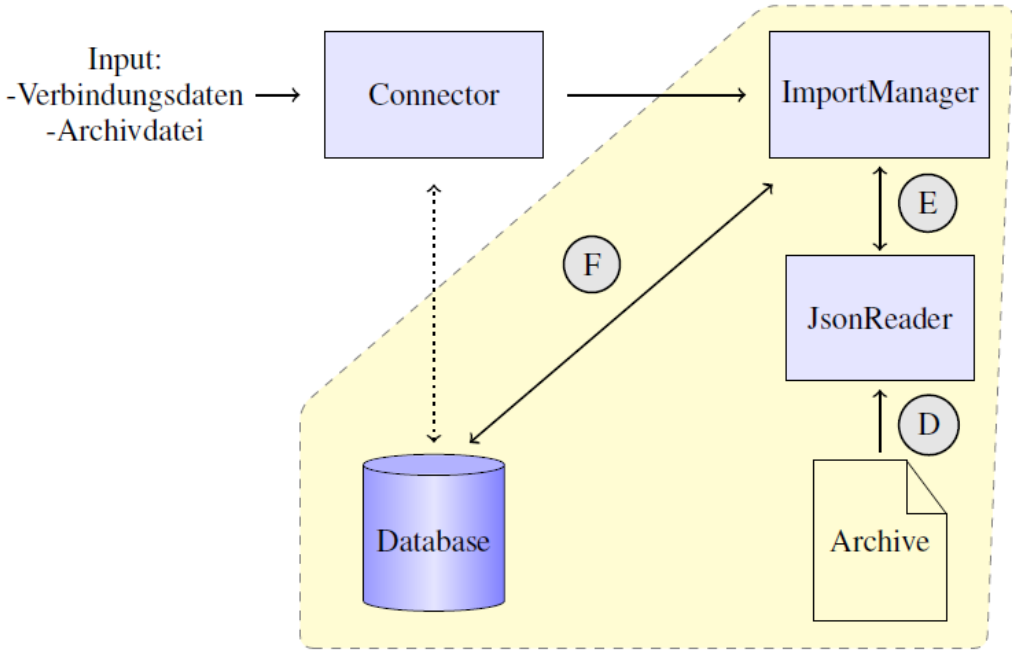


Agenda


Einstieg
Export
Serialisierung
Datenintegrität
Import
Ausblick

Import

- Auswahl:
  - Gesamtes Archiv, best. Datei
  - Gesamtes Schema, best. Tabellen, …
  - Umbenennen von Tabellen und Schema
- Einfügen:
  - INSERT Statements
- Erstellen der Strukturen:
  - Grundlegend, nur um Daten zu halten
  - Ziel == Quelle: Datentypname des JDBC Treibers
  - Ziel != Quelle: Datentyp der Schätzung

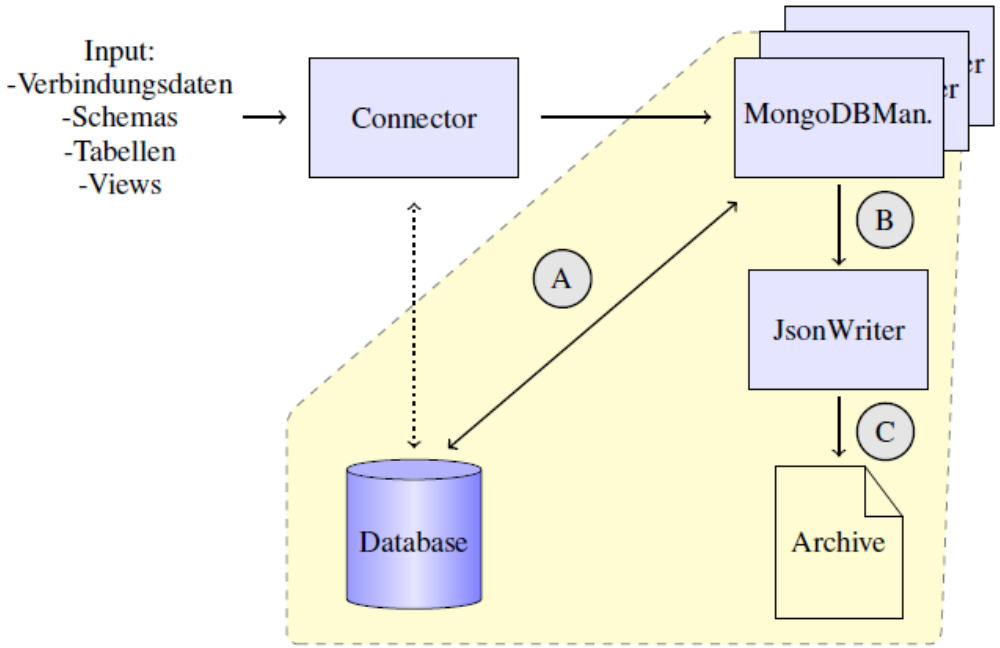


Agenda


Einstieg
Export
Serialisierung
Datenintegrität
Import
Ausblick

Ausblick

- Neue Datenbanken:
  - Relationale
  - Nicht-Relationale/kein JDBC
- Datenintegrität:
  - Verschlüsselung



Vielen Dank für Ihre Aufmerksamkeit!

Fragen?

Niklas Hackelberg
InfoDesign GmbH

✆ 04101 693154
✉Hackelberg@infodesign.de