

# Uni DB1 Notes

Notes for the DB1 (databases) course at HdM Stuttgart

---

Felicitas Pojtinger

2022-02-01

*“The true courage is to admit that the light at the end of the tunnel is probably the headlight of another train approaching” - Slavoj Žižek, The Courage of Hopelessness*

Mehr Details unter <https://github.com/pojntfx/uni-db1-notes>.

## Aufbau eines DBMS

---

- **Persistent:** Über mehrere Programmläufe verfügbar
- **Temporär:** Nur während der Laufzeit verfügbar

Applikation → DBMS → Datenbank

- Präsentationsschicht (temporäre Daten)
- Logik-Schicht (temporäre Daten)
- Daten-Zugriffs-Schicht (temporäre Daten)
- API zwischen App und DBMS
- DBMS (persistente Daten)
- Datenbank (persistente Daten)

- Sammelt Daten
- Verwaltet Daten
- Definiert Struktur (Modell)
- Definition, Manipulation & Abfrage von Daten
- Services

Entwickler:

- Effiziente Modellierung
- Einfache Sprache
- Gutes Tooling

Admins:

- Effiziente Ressourcennutzung
- Einbindung in Systemverwaltung
- Monitoring
- Anpassung
- Zugriffssteuerung



## SMARASTD:

- Service to ensure Data Integrity
- Multi-User Support
- Authorization Service
- Recovery Service
- Access via Network
- Storage, Query & Manipulation of Data
- Transactional Support
- Data Dictionary & System Catalog

- **Data Dictionary:** Metadaten der DB-Objekte
- **System Catalog:** Status und Konfiguration

- $n$  Datendateien ( $n = 1 \dots MAXDATAFILES$ )
- Control-Files
- Logfiles

- **DBWR** (Database Write-Prozess): Lesen & Schreiben auf Daten-Dateien
- **LGWR** (Log Write-Prozess): Logging aller Veränderungen
- **PMON** (Process-Monitor): Garbage Collector; führt in konsistenten Zustand nach Abbruch von z.B. einer Transaktion
- **SMON** (System-Monitor): Consistency Check; führt in konsistenten Zustand nach Crash von DBMS, OS oder Hardware
- **ARCH** (Archiv-Prozess): Archivierung von Daten

Notwendig für ...

- Konsistenz
- Wiederherstellbarkeit

Log-Dateien sind ...

- Groß: Ineffizienter Zugriff
- Wichtig: Verlust muss vermieden werden

→ Round-Robin-Prozess mit Archiv-Prozess

- Datenbank: Objekte zusammen von DBMS **verwaltet**
- Schema: Objekte zusammen von DBMS **betrachtet**

- Trennung der Anwendungsschicht vom Unternehmensmodell (menhir)
- Sicherheit
- Zugriffsstrukturen
- Vereinfachte Schemaevolutionen
- Einfügen und Löschen einschränken

→ Am besten immer nur via Views auf Daten zugreifen

## Keys

---



Ein Key ist eine Menge von Spalten.

- **Eindeutigkeit:** Es gibt keine zwei Zeilen mit demselben Candidate Key
- **Irreduzibilität:** Nimmt man eine oder mehrere Spalten aus dem Key, so ist dieser nichtmehr eindeutig.

Ein gewählter Candidate Key (oft der mit der kleinsten Anzahl von Spalten).

Es werden zwei Tabellen A und B betrachtet.

Der Foreign Key, welcher B aus A referenziert, ist ein Candidate Key von B (meist der Primary Key).

## Skripte

---

1. Löschen Constraints
2. Löschen Objekte
3. Anlegen Objekte
4. Anlegen Constraints

Bei einer Erweiterung des Modells dürfen bestehende Daten nicht ungültig werden.

- alter table: Neue Spalte einfügen
- update: Default-Werte für alte Zeilen einfügen (falls not null)
- insert: Fehlende Zeilen anlegen (falls not null)
- alter table: Foreign Key-Constraint hinzufügen
- alter table: not null-Constraint hinzufügen

# Mengenoperationen

---

## Typen von Multi-Tabellen-Abfragen:

- **Additive** Mengenoperationen: Mehrere Teilabfragen (in etc.)
- **Multiplikative** Mengenoperationen: Kartesisches Produkt (join etc.)



Wenn Abfragen über mehrere Tabellen gemacht werden, so müssen alle Abfragen fertig sein, damit verglichen werden kann. Deshalb union all verwenden (Vorsicht: Duplikate werden nicht entfernt!)

- **Inner Join:** Zeilen in linker Tabelle, für welche in der rechten Tabelle keine entsprechenden Zeilen existieren, werden nicht dargestellt.
- **Outer Join (+):** Zeilen in Tabelle A, für welche in Tabelle B keine entsprechende Zeilen existieren, werden mit null gefüllt.
  - **Left Outer Join:** Rechts kann null-Werte haben
  - **Right Outer Join:** Links kann null-Werte haben
  - **Full Outer Join:** Beide könnten null-Werte haben

- Bulk Join (Kartesisches Produkt)
- Restricted Join (mit zwei Where-Bedingungen)
- Natural Join (min. ein Attribut gleich)
- Semi Join (nur Attribute einer Tabelle im select-Statement)
- Multiple Join (z.B. join aus drei Tabellen)
- Auto Join (Tabelle mit sich selbst joinen; z.B. Stückliste)

# Modellierung

---

1. **Input:** Reale Welt
2. Anforderungen analysieren
3. Entwurf (konzeptionell) erstellen
4. Entwurf (logischen) erstellen
5. Implementieren
6. **Output:** System

Dabei wird nebenläufig kontinuierlich getestet.

- **Realwelt**
  - Vielschichtig
  - Unikate
  - Umfangreiche Beziehungen
- **Semantisches Datenmodell**
  - Zusammenfassung zu Gruppen, abstrahiert
  - Integritätsbedingungen
  - Explizit modellierte Beziehungen
- **Relationales Datenbankmodell**
  - Einfach
  - Tabellen
  - Implizit modellierte Beziehungen

SSRWKV:

- Syntaktische & semantische Richtigkeit
- Systematischer Aufbau
- Relevanz
- Wirtschaftlichkeit
- Klarheit
- Vergleichbarkeit

Ein gutes Anforderungsdokument sollte die Eigenschaften haben ...

- Korrektheit
- Vollständigkeit
- Konsistenz
- Einfachheit
- Eindeutig

Ein gutes Anforderungselement sollte bestehen aus ...

- Informationsanforderungen
- Bearbeitungsanforderungen
- Funktionale Anforderungen
- Dynamische Anforderungen



- **Atribut:** Datenelement
- **Entität:** Gruppierungselement
- **Beziehung:** Verknüpfung (n:m-Beziehungen via schwacher Entität)
- **Kardinalität:** Maximale Anzahl an Elementen in Beziehung



Folgende Anomalien treten durch Redundanzen auf:

- Änderungsanomalie
- Löschanomalie
- Einfügeanomalie

- **Erste Normalform:** Spalten sind nicht weiter auftrennbar
- **Zweite Normalform:** Alle Attribute hängen vom Schlüssel ab (keine funktionalen Abhängigkeiten)
- **Dritte Normalform:** Beziehungen werden über Foreign Key-Constraints abgebildet (keine transitiven Abhängigkeiten)

1. Erheben von Infos
2. Identifikation der Attribute
3. Formalisierung von Infos
4. Gruppierung der Attribute

- Im temporären Speichern funktionieren Indizes nicht mehr
- Falsche Anwendung von Indizes kann sogar langsamer als keine Indizes sein. Ohne Indizes werden immer alle Zeilen einer Tabelle evaluiert; bei Indizes wird immer von einer Position ausgehend, bis die where-Clause eintritt, evaluiert. Letztere Strategie besitzt damit auch einen Overhead, welcher teurer als die Ersparnis durch das frühere Abbrechen nach dem Eintreten der where-Clause sein kann.

Bei der Erstellung eines Indexes sollte immer die Spalte mit der höchsten Selektivität ( $> 0,8$ ) zuerst angegeben werden, welche sich mit folgender Formel berechnen lässt:

$$\textit{Selektivitt} = 1 - \frac{n - \textit{distinct}(n)}{n}$$

$n$ : Anzahl von Elementen

$\textit{distinct}(n)$ : Anzahl von eindeutigen Elementen

## Weitere Services

---

Nutzt eine Allowlist.

- Beschränkung von Nomen (i.e. “Nutzer x darf auf Tabelle products zugreifen”): **Objektprivilegien**
- Beschränkung von Prädikaten (i.e. “Nutzer x darf updatenen”): **Systemprivilegien**



- Sichtbarkeit von Daten
- Änderbarkeit von Daten
- Trennung in Anwender und Admins
- Schonung von Ressourcen
- Einfache Verwaltung

Daten dürfen weder physisch noch semantisch fehlerhaft sein, weshalb folgende Dinge existieren müssen:

- Transaktionen
- Virtueller Single-User-Betrieb

Aktionen werden entweder vollständig oder gar nicht ausgeführt.

- **Atomicity:** Alles oder nichts
- **Consistency:** Zustand 1 → Zustand 2 (Unterbrechung: Zustand 2 = Zustand 1)
- **Isolation:** Virtueller Single-User-Betrieb
- **Durability:** Zustand 2 bleibt erhalten, egal was passiert

- begin: Start einer Transaktion (SQL: Nicht definiert)
- end: Ende einer Transaktionen (SQL: commit)
- undo: Verwerfen offener Transaktionen (SQL: rollback)
- redo: Wiederherstellung abgeschlossener Transaktionen (SQL: Nicht definiert)
- savepoint: Sub-Transaktionen (SQL only)

- **Constraints:** In Tabellen
- **Transaktionen:** In Ablaufebene
- **Trigger:** In Prozeduralen Erweiterungen

Verhindern von ...

- **Lost Update:** Verlorengegangenen Änderungen
- **Dirty Read/Write:** Zugriff auf “schmutzige” Daten

Umsetzung durch ...

- Lese-, Schreib- und Exklusiv-Sperren (**Funktionale Sperr-Ebene**)
- Table-, Page- und Row-Level-Sperren (**Physische Sperr-Ebene**)

→ Z.B. durch `select ... for update of ...`

- Low Code-Umgebungen (z.B. LibreOffice Base, IFTTT)
- Embedding
- APIs

- `too_many_rows`: Mehr als ein Datensatz
- `no_data_found`: Null Datensätze (nicht streng genommen ein Impedance Mismatch)



Ergebnis einer Abfrage wird in einer Tabelle abgelegt, von welcher dann  $n$ -mal gefetched werden kann.