Kapitel 0 – Einführung

- **SQL** ≡ Structured Query Language
- **Datenbank Schema**: Formale Definition der Struktur der Datenbankinhalten, bestimmt mögliche Datenbankzustände, wird zu Beginn definiert (umgspr. Tabellenkopf)
- **Datenbank Zustand**: eine Instanz des Schemas, enthält aktuelle Daten, ändert sich ständig (umgspr. Tabelleninhalt)
- **Attribute**: Spalten; **Tupel**: Zeile
- **Daten Modell**: definiert eine formale Sprache (Syntax & Semantik) zum deklarieren eines Schemas, zum Abfragen von aktuellen Zuständen und zum Ändern des Zustandes.
- **DB** (Datenbank): besteht aus Schema und Zustand
- **DBS** (Datenbank System): besteht aus DBMS und der DB
- „database application system“: besteht aus einem DBS und einer Menge an Programmen
- „Persistent information“: Informationen die länger leben als der ausführende Prozess, überlebt Neustart und Stromausfall

DBMS (Datenbank Management System):

- Softwaresystem das das Daten Modell implementiert (z.B. ein relationales DBMS wie MySQL)
- Speichert Daten üblicherweise in Dateien ab; bietet „higher level operations“
- verschiedene Algorithmen die man sonst selbst schreiben müsste: Sortierung, Suche, Speicherplatzmanagement, Statistische Auswertung, Buffer Management
- „multi-user“ Support (locks, Transaktionen)
- Weitere Funktionen: Backup, Recovery, Security, Integrity, Metadata (data about data, user lsit, access rights),

Daten Unabhängigkeit

- anhand der Abfragen werden automatisch Datenbankindexe im Hintergrund angelegt
- die interne Struktur ist für den Anwender nicht bekannt und unrelevant

Deklarative Sprachen

- Queries beschreiben nur, welche Information gefordert ist
- Solche Sprachen erlauben, benötigen aber auch mächtige Optimierungen

Logische Datenunbabhängigkeit: Erlaubt nachträgliche Änderungen am Schema (Spalten hinzufügen)

Datenbank User:

- Datenbank Administrator: Überwacht Performance, vergibt Rechte, etc.
- Application Programmer: Schreibt Programme für den naiven User
- Sophisticated User: kennt SQL, kennt nicht die Hintergründe der Auswertung
- Naiver User: verwendet DB nur über Programme

Kapitel 1/2 - Einführung in das relationale Modell

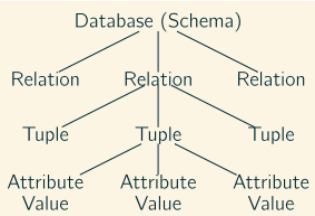
Ein relationales Datenbankschema definiert:

1. Namen der Tabellen der Datenbank
2. die Spalten jeder Tabelle (z.B. der Spaltenname und die Datentypen jedes Eintrags)
3. Integritäts Bedingungen

Ein relationaler Datenbankzustand definiert für jede Tabelle eine Menge an Zeilen, diese sind nicht geordnet! Ein **relationaler Schlüssel** ist immer einzigartig und identifiziert eine Zeile in einer Tabelle. Ein **Fremdschlüssel** ist ein logischer Pointer auf eine Zeile einer anderen Tabelle.

Theoretische Definition: Tabelle ≡ Relation; Zeile ≡ Tupel; Spalte ≡ Attribut; Eine Tabelle ist eine endliche Teilmenge des Kartesischen Produkts der Definitionsbereichs jeder Spalte.

Praktischer Ansatz: Tabelle ≡ Datei; Spalte ≡ Feld; Zeile ≡ record/struct



Allgemeines

- Jeder Eintrag einer Spalte ist atomar! Sie besitzen nicht mehrere Werte
- Es gibt keine doppelten Tupel.
- **Update Operationen**:
  - Einfügen: $\mathrm{I}_{\mathrm{new}}(\mathrm{R}) := \mathrm{I}_{\mathrm{old}}(\mathrm{R}) \cup \{(d_1, \ldots, d_n)\}$
  - Löschen: $\mathrm{I}_{\mathrm{new}}(\mathrm{R}) := \mathrm{I}_{\mathrm{old}}(\mathrm{R}) - \{(d_1, \ldots, d_n)\}$
  - Ändern: $\mathrm{I}_{\mathrm{new}}(\mathrm{R}) := \mathrm{I}_{\mathrm{old}}(\mathrm{R}) - \{(d_1, \ldots, d_n)\} \cup \{(d'_1, \ldots, d'_n)\}$
- "**null**" ist unterschiedlich zu allen Werten aller Typen.
- Vorteile von "null": Ohne müssten Relationen in viele Subklassen aufgeteilt werden.
- **Primärschlüssel**: Kann über mehrere Spalten gehen und ist für jede Zeile eindeutig (nie null).
- **Fremdschlüssel**: Ermöglicht eine "one-to-many" Beziehung und gleichzeitig eine Existenzgarantie, schützen die referentielle Integrität der Datenbank. Kann immer nur auf einen Primär- oder Sekundärschlüssel verweisen. Können eventuell null sein.

Kapitel 3 - Das Entity-Relationship Model

Datenbankdesign

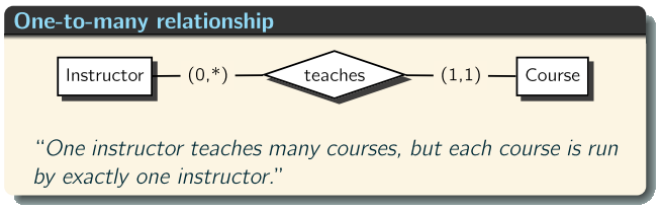
Datenbankdesign ist der Prozess zur Entwicklung eines Datenbankschemas für eine gegebene Anwendung. Es ist nicht leicht aufgrund Fachwissen, Flexibilität (alle Sonderfälle bedacht) und der Größe. Es gibt drei übliche Schritte zum Datenbankdesign:

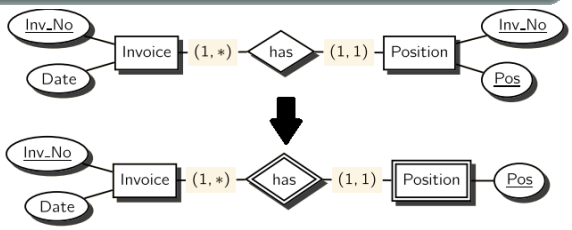
1. Konzeptuelles Datenbankdesign: (z.B. ER-Modell)
2. Logisches Datenbankdesign: Übersetzung des konzeptuellen Schemas in ein Datenmodell, das von DBMS unterstützt wird.
3. Physikalisches Datenbankdesign: Performance maximieren (z.B. Index erstellen, Buffergröße)

Das ER Modell (Entity-Relationship Model)

- **Entitäten** (Entities): Hält endliche Informationen über ein Objekt der Mini-Welt. Es muss möglich sein zwischen Entitäten zu unterscheiden. (z.B. Personen, …)
- **Attribute**: Der Wert eines Attributs ist ein Element mit primitivem Datentyp.

- **Beziehung** (Relationship): (z.B. Prof *(entity)* hält *(relationship)* einen Kurs *(entity)*), sie können auch Attribute enthalten, hat immer 2 Beziehungen, (min, max) Notation
- **Beziehungsarten**:
  - *many-to-many*
  - *one-to-many*
  - *many-to-one*
  - *one-to-one*
- Eine Entität muss keinen **Schlüssel** besitzen.
- **Weak Entities**: Beschreiben ein Entität, die nicht ohne eine andere Entität existieren kann.





Transformation in das Relationale Modell

1. Erstelle eine Tabelle für jede Entität
2. Die Spalten sind die Attribute der Entitätstypen
3. Der Schlüssel wird übernommen, falls keiner vorhanden, eine ID-Spalte hinzufügen.
4. Beziehungen auflösen: One-To-Many, wird zu einem Fremdschlüssel aufgelöst, Many-To-Many → eigene Tabelle mit beiden Primärschlüsseln der jeweiligen Tabellen
5. Kardinalitäten ungleich 0, 1 oder * sind nicht möglich und werden angepasst.

Kapitel 4 – Relational Algebra

Die Relationale Algebra ist eine Query-Sprache für das relationale Modell mit einer theoretischen Grundlage. Nahezu jedes RDBMS verwendet RA für die interne Repräsentation von Querys.

SELECT - Operatoren

1. σ_p Selection: Wählt die Teilmenge aus, die das Prädikat P erfüllt (Filter)
2. π_L Projection: Wählt die Spalten aus. (z.B. L = A, B ← C||D, Strings werden mit || verknüpft, map)
3. x Cartesian Product: Erstellt das kartesische Produkt aus zwei Spalten.
4. $\bowtie_P$ (Natural) Join: Verbindet zwei Tabellen abhängig vom Prädikat P (auch $\ltimes$, $\rtimes$, $⟕$, $⟖$, $⟗$) (redundant)

SET – Operatoren

1. R ∪ S (Vereinigung)
2. R ∩ S (Schnitt) (redundant)
3. R – S (Differenz)

Allgemeines

- RA behandelt Relationen als Mengen (keine Duplikate)
- Null-Werte werden in der Definition in der Regel ausgeschlossen
- Mit ← können auch Ergebnistabellen einen Namen zugewiesen werden (z.B. A ← π_C(B))

- Es gibt Algebraische Gesetze die der Query Optimierer verwendet. z.B. selection push-down ($\sigma_P(R \bowtie S) \equiv R \bowtie \sigma_P(S)$, falls sich P nur auf S bezieht)
- Lediglich ∩ und – sind nicht monotone Operatoren (**Monoton**: Wenn sich die Relation vergrößert gibt es immer noch gleich viel oder mehr Ergebnisse)
- *Syntaxzucker* (anti-join): $R \bar{\ltimes} S \equiv R \bowtie (\pi_B(R) - \pi_B(S))$ (wobei R(A,B), S(B,C))
- Jedes Problem das mit der relationalen Algebra formuliert werden kann, ist in P berechenbar.

Äquivalenzen

- Q_1 und Q_2 sind äquivalent falls $I(Q_1) = I(Q_2)$ für alle Datenbankzustände I.
- $\sigma_{p1}(\sigma_{p2}(Q)) = \sigma_{p2}(\sigma_{p1}(Q))$
- $(Q_1 \times Q_2) \times Q_3 = Q_1 \times (Q_2 \times Q_3)$

Kapitel 1/2/5 - Einführung in SQL

Allgemeines

Datentypen (Data Values): nur primitive möglich (z.B. NUMERIC(n), VARCHAR(n), DATE, ...)

Definitionsbereiche (Domains): Man kann mit (CREATE DOMAIN *name* AS *type* [CHECK(*bedingung*)]) eigene Typen benennen, um konsistent Daten die Typen zu verwalten.

Integritäts Bedingungen (Constraints): Bedingungen die bei jedem Datenbankzustand überprüft werden. (z.B. NOT NULL, (Fremd-)Schlüssel, CHECK())

SELECT-Statements

SELECT [DISTINCT] *spalte1* [AS otherName],*konstante* [AS otherName] ...
FROM *tabelle1, tabelle2* [AS otherName], ...
[LEFT,RIGHT,NATURAL,UNION] JOIN *tabelle* ON join*bedingung*
WHERE *bedingung*
GROUP BY *spalteX*
HAVING group*bedingung*
ORDER BY *spalteX* [ASC,DESC], *spalteY*
LIMIT *anzahl*
OFFSET *start*

Verschiedene Arten von Bedingungen:

- **Pattern**: zum Beispiel % für beliebige Sequenz, _ für beliebiges einzelnes Zeichen
- **Verknüpfungen**: AND, OR, NOT
- **Sonstige**: *spalte* [NOT] IN (*values)* oder auch (*s1*, *s2*) [NOT] IN (SELECT *s3*, *s4* …))
  [NOT] EXISTS (SELECT * FROM …)
  spalte1 < [ALL,ANY/SOME] (SELECT *spalte2* FROM …)
  spalte IS NULL

Aggregatsfunktionen

- **Beispiele**: COUNT([DISTINCT] *spalte*), SUM(*spalte*), AVG(*spalte*), MIN(*spalte*), MAX(*spalte*)
- Mit GROUP BY kann eine Aggregatsfunktion auf einer „Gruppe“ ausgeführt werden und mit HAVING können davon einzelne Ergebnisse ausgeschlossen werden.
- Wenn das Resultat leer ist liefern alle Funktionen „null“ außer COUNT.

Rekursive Querys

WITH *acc*(*spalte1, spalte2*) AS

(<SELECT als Basis>
UNION ALL
<SELECT das auf bisheriger Ergebnismenge weitere findet (Rekursion)>)

<SELECT zur Auswertung>

- Wird solange ausgeführt bis der rekursive SELECT leer ist.
- Endlosschleifen sind damit möglich.

Sonstige Befehle

- UPDATE *tablelle* SET *spalte* = *neuerWert* WHERE *bedingung*
- INSERT INTO *tabelle* [(*spalte1*, …)] VALUES (*wert1*, …), …
- [CREATE,ALTER,DROP] [VIEW,TABLE,TRIGGER,DATABASE]
- UNION, INTERSECT, EXCEPT
- CASE WHEN *X* IS NOT NULL THEN *X* ELSE *Y* END ≡ COALESCE(*X*,*Y*)

Kapitel 6 – Relationale Normalformen

Functional Dependencies (FDs, Funktionale Abhängigkeiten)

- FDs sind eine Generalisierung von Schlüsseln und gleichzeitig ein „Constraint“
- Die Theorie definiert in welcher Normalform eine Relation ist. Wenn eine Normalform verletzt wird, dann ist Information redundant gespeichert (→ schlechter Stil).
- In der Praxis werden Normalisations-Algorithmen als Überprüfung verwendet, ob die aus dem ER-Modell entstandene Relation in BCNF liegt. Normalerweise ist das automatisch der Fall, meist sogar in 4NF.
- Bei FDs bestimmt die linke Seite zusammen die rechte Seite eindeutig! Üblicherweise liegt der Focus auf einer Relation wenn man über eine einzelne FD redet.
- Eine Determinante ist stärker als eine FD. Eine Determinante ist ein partieller Schlüssel, der allein Attribute bestimmt, aber nicht im Allgemeinen.
- **Armstrong Axiome**
  - **Reflexivity**: Wenn $\alpha \subseteq \beta$, dann $\alpha \rightarrow \beta$.
  - **Augmentation**: Wenn $\alpha \rightarrow \beta$, dann $\alpha \cup \gamma \rightarrow \beta \cup \gamma$.
  - **Transitivity**: Wenn $\alpha \rightarrow \beta$ und $\beta \rightarrow \gamma$, dann $\alpha \rightarrow \gamma$.
- **Cover**: Berechnet wie welche Attribute von α eindeutig bestimmt werden. Anhand einer Menge von FDs F.
  $\alpha^+ := \{B | F\ impliziert\ \alpha \rightarrow B\}$
- Wenn $\alpha^+ = A$ und A die Menge mit allen Attributen ist, dann ist α Schlüssel der Relation. Und α ist minimaler Schlüssel, falls keine Teilmenge von α Schlüssel ist.

Key Computation (call with $x = \varnothing$)

```
procedure key(x, A, F)
  foreach α → B ∈ F do
    if α ⊆ x and B ∈ x and B ∉ α then
      return;        /* x not minimal */
    fi
  od
  if x+ = A then
    print x;    /* found a minimal key x */
  else
    X ← any element of A − x+;
    key(x ∪ {X}, A, F);
    foreach α → X ∈ F do
      key(x ∪ α, A, F);
    od
  fi
```

- Eine Menge A ist **Determinante** für die Attribute B wenn folgendes gilt:
  - Linke und Rechte Seite sind verschieden (distinct): $\{A_1,…,A_n\} \neq \{B_1,…,B_m\}$
  - Die FD $A_1,…,A_n \rightarrow B_1,…,B_m$ gilt.
  - Die linke Seite ist minimal.
- Es gibt auch andere Arten von Beschränkungen zum Beispiel MVDs und JDs

Normalformen

First Normal Form (1NF)

- Es wird gefordert, dass alle Tabelleneinträge atomar sind, also eine Spalte nicht mehrere Werte (Listen, Records, etc.) trägt.

Second Normal Form (2NF)

- Es wird gefordert, dass jedes Attribut, dass nicht Teil des Schlüssels, ist abhängig vom kompletten Schlüssel ist. (z.B. results(stud_id, ex_no, points, max_points) ist nicht in 2NF, da max_points bereits durch ex_no (Teil des Schlüssels) eindeutig bestimmt ist)

Third Normal Form (3NF)

- Es wird gefordert, dass für alle FDs ($A_1, \dots, A_n \rightarrow B$) eines der folgenden Bedingungen gilt:
  - FD ist trivial
  - FD ist bereits Schlüssel
  - B ist ein Schlüsselattribut
- 3NF bedeutet, dass jede Determinante eines Attributs, dass nicht Teil eines Schlüssels ist, ein Schlüssel ist.

Boyce-Codd Normal Form (BCNF)

- Es wird gefordert, dass alle FDs bereits als Schlüssel repräsentiert sind oder selbst trivial (z.B. $\{B_1,\dots,B_m\} \subseteq \{A_1,\dots,A_n\}$ sind.

4NF

BCNF split algorithm

Input: relation $R(\mathcal{A})$ with schema (attribute set) $\mathcal{A}$, set of FDs $\mathcal{F}$

Output: splitted relational schema (relations $R_{1,2,\dots}$)

```
repeat
  if non-trivial FD α → B ∈ F and α does not contain a key then
    Decompose R into
    (1) R1(α ∪ (A − α+)) and
    (2) R2(α+);
  fi
until no further decomposition of R1, R2 possible
```

Splitting/Decomposition

Wenn eine FD $A_1,\dots,A_n \rightarrow B_1,\dots,B_m$ die BCNF verletzt, dann erstelle eine neue Relation $R_2(A_1,\dots,A_n,B_1,\dots,B_m)$ und lösche $B_1,\dots,B_m$ aus der originalen Relation.

Die Dekomposition ist garantiert verlustfrei, wenn der Schnitt der Attribute der neuen Tabellen ein Key von einer der Tabellen ist.

3NF Synthesis Algorithm

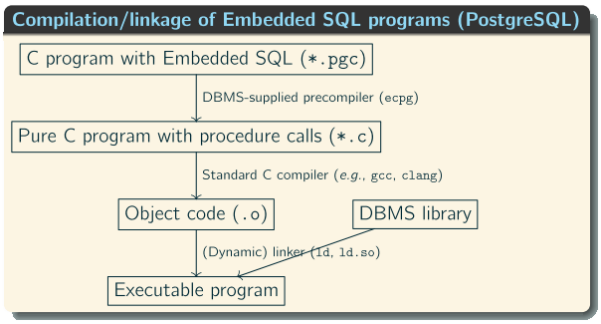
1. Ersetze alle FDs $\alpha \rightarrow B_1, \dots, B_m$ zu $\alpha \rightarrow B_i$. Das Ergebnis ist dann F
2. Minimiere alle FDs in F: für jede FD $A_1,\dots,A_n \rightarrow B$ und jedes i = 1, ..., n berechne $\{A_1,\dots,A_{i-1},A_{i+1},\dots,A_n\}_F^+$, wenn B dann lösche A_i aus der FD und wiederhole.
3. Lösche implizite FDs: für jede FD $\alpha \rightarrow \beta$, berechne den α_F^+, wobei $F' = F - \{\alpha \rightarrow \beta\}$. Wenn B im Cover enthalten ist, dann mach mit F' weiter.
4. Für jedes α einer FD in F erstelle eine Relation mit den Attributen $A = \alpha \cup \{B | \alpha \rightarrow \beta \in F\}$.
5. Wenn keine der Relationen in 4. ein Schlüssel der originalen Relation besitzt dann füge eine Relation die hinzu die den Schlüssel der originalen Relation besitzt hinzu.
6. Wenn eine Relation in einer anderen enthalten ist, so lösche diese.

ER Modell und Normalformen

- Wenn eine FD verletzt wird, dann muss die Entität geteilt werden
- FDs zwischen Attributen von Beziehungen, falsche Platzierung von Attributen, 3-fach Beziehungen verletzten die BCNF
- wenn unabhängige Entitäten verbunden werden, verletzt das 4NF

Denormalization

Der Prozess bei dem redundante Informationen hinzugefügt werden, damit die Performance besser wird, dadurch können Änderungsanomalien auftreten. Durch Trigger kann man das verhindern.



Kapitel 7 - Embedded SQL

- Das DBMS unterscheidet zwischen internen und externen Typen und bietet bei manchen auch Konversionen zwischen diesen an
- Für den Precompiler relevante Variablen müssen separat deklariert werden

```
EXEC SQL BEGIN DECLARE SECTION;
   int     sid;
   VARCHAR first[21];
   char    last[21];
EXEC SQL END DECLARE SECTION;
```

- VARCHAR first[21] wird übersetzt zu
  struct { int len; char arr[21]; } first;
- Fehler werden mit SQLCA bearbeitet.
  0: okay
  1403: no more tuples

- Statisch:
  EXEC SQL DECLARE c1 CURSOR FOR *sqlquery*
  EXEC SQL OPEN c1;
  EXEC SQL WHENEVER NOT FOUND DO <stmt>;
  EXEC SQL FETCH c1 INTO :var1, :var2 INDICATOR :null, …;
  EXEC SQL CLOSE c1;
- Dynamisch:
  EXEC SQL PREPARE q FROM :sql_cmd;
  EXEC SQL DECLARE c1 CURSOR FOR q;
  EXEC SQL OPEN c1 USING v1, v2, …;
  …

Kapitel 8 – Transaction Management

ACID Properties

- **A**tomicity: Entweder alle oder keine Änderungen werden an der Datenbank ausgeführt
- **C**onsistency: Nach einer Transaktion ist die Datenbank wieder im konsistenten zustand sein
- **I**solation: nebenläufige Transaktionen beeinflussen sich nicht gegenseitig
- **D**urability: Die Effekte eine erfolgreichen Transaktion bleiben Dauerhaft in der Datenbank

Begriffe

- **Scheduler** (S): Bestimmt die Zugriffsreihenfolge von Transaktionen, wenn sich die Datenbankzugriffe überschneiden.
- **Transaktion** (T): Ist eine Sequenz von Schritten. Jeder Schritt ist ein Paar einer Zugriffsoperation verknüpft mit einem Objekt. Die Länge ist seine Anzahl an Schritte.
- **Serielle Ausführung**: Ein Schedule S ist seriell, wenn für jede enthaltende Transaktion alle seine Schritte direkt am Stück ausgeführt wurden und nicht verteilt.
- Jede Serielle Ausführung ist korrekt. (also keine Anomalien treten auf)
- Zwei Operationen sind im Konflikt, wenn ihre Reihenfolge relevant ist für das Ergebnis.
- Wenn man den Konfliktgraph topologisch sortiert (er darf keine Kreise haben), dann ist er serialisierbar. Ansonsten sind locks nötig.

Conflicting operations

Two operations (T_i, a, e) and (T_j, a', e') are **in conflict** ($\leftrightarrow$) in S if
1. they belong to two **different transactions** ($T_i \neq T_j$), and
2. they access the **same database object**, *i.e.*, $e = e'$, and
3. at least one of them is a `write` operation.

- **Two-Phase Locking**: Eine Restriktion wie Transaktionen vollzogen werden sollen. (es gibt eine Lock und eine Release-Phase)
  Varianten: Preclaiming 2PL, Strict 2PL
- **Deadlock**: (l_1 (A), l_2(B), l_1(B), l_2(A))
- **Granularität der Locks**: Datenbanklevel → Tabellen → Seite → Zeile
  Zur Umsetzung verwenden Datenbanken „intention locks“ (IS/IE)

Alternative: Optimistic Concurrency Control

- 3 Phasen: Read Phase, Validation Phase, Write Phase
- Es muss aber garantiert werden, dass 2. und 3. ohne Unterbrechungen ausgeführt wird!