Zindel Marius

Begriffe

DB: Datenbasis, strukturierte Daten HD
DBMS: Datenbankmanagementsystem
DBS:(Datenbanksystem) DBMS + n* DB
Datenkonsistenz: korrekte Daten
Datensicherheit: vor phy. Verlust
Datenschutz: vor unberecht. Zugriff
Funktionen DBMS: Transaktionen, Mehrbenutzerbetrieb, Sicherheit, Backup & Recovery, Generische Datenstrukturen
Vorteile DBMS: Skalierbar, Sicherheit, Integrität, Live-Abfragen, Kapslung

DB-Modelle

Hierarchisches Datenbankmodell

Die zu speichernden Informationen werden in Hierarchie organisiert. Auch die beziehungen zwischen den Daten werden als Hierarchie gespeichert.
Nachteile: Es ist sehr schwer die Welt "Hierarchisch"darzustellen. Änderungen sind sehr schwer vorzunehmen. Trennung von Daten ist sehr schwer

Netzwerkmodell

Die zu speichernden Informationen werden in vernetzten Hierarchien organisiert. Beziehungen innerhalb der Hierarchie werden zusammen mit den Daten gespeichert, was eine Mehrfachbeziehung von Elementen möglich macht.
Nachteil: Effiziente Anwendungen nur möglich in Kenntnis der komplexen Netzwerk-Datenstruktur. Dadurch besteht eine starke Abhängigkeit zwischen Daten und Anwendung.

Relationenmodell

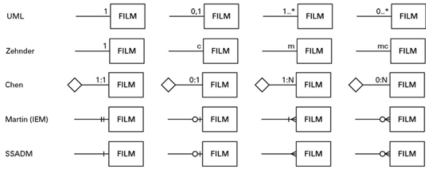
Alle Informationen werden als unstrukturierte Daten in Form von Tabellen gespeichert. Dadurch ist alles sehr flexibel und all mögliche abfragen lassen sich realisieren. Ebenfalls ist eine klare Trennung zwischen den Daten und Anwendungen möglich. Nachteile: Unterschiede zwischen den Typensystemen der DB und den Anwendungen ist schwer zu konvertierten.

Objektrelationales Modell

Ist eine weiterentwickeln des rel. Datenmodells. Es können aber auch zusätzlich Methoden gespeichert werden. Tabellen können verschachtelt werden (1. Normalform muss nicht erfüllt sein). Benutzerdefinierte Typen sowie Tabellen-Vererbung können erstellt werden.

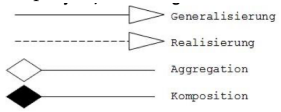
ER-Modell

Das ER-Modell modelliert den interessierenden Ausschnitt der 'Welt' grafisch als Entitätsmengen (Typen) mit den Beziehungen zwischen diesen Mengen (Assoziationen). Es ist vergleichbar mit dem Domain Model.



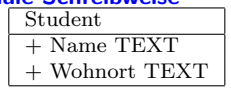
UML

UML erweitert die ER-Technik um dynamische Aspekte.

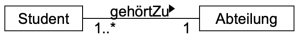


Aggregation: Kann in mehreren vorkommen und wird nicht gelöscht.
Komposition: Wird Vater gelöscht, dann auch Kind. Kann nur 0-1 haben

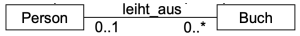
Relationale Schreibweise



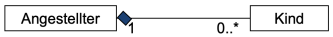

```
Student (StudID INT,
Name TEXT NOT NULL,
Wohnort TEXT NOT NULL)
```



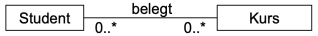

```
Abteilung (abtId, Name)
Student (StudId INT,
AbtId NOT NULL REFERENCES Abteilung)
```



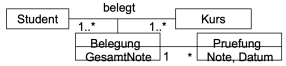

```
Person (PId, Name NOT NULL)
Buch (BuchId INT,
Ausleiher NULL REFERENCES Person)
```



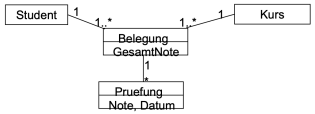

```
Angestellter (AngId, Name NOT NULL)
Kind (AngId REFERENCES
Angestellter NOT NULL, Name)
```




```
Student (StudId , Name NOT NULL)
Kurs (KursId,Bezeichnung NOT NULL)
Belegung (StudId REFERENCES
Student, KursId REFERENCES Kurs)
```




```
Student (StrudId, Name)
Kurs (KursId, Bez.)
Belegung (StudId, KursId,
GesamtNote)
Prüfung (StudId REFERENCES
Belegung, KursId REFERENCES
Belegung, Datum, Note NOT NULL)
```




```
Student (StrudId, Name)
Kurs (KursId, Bez.)
Belegung (BelId, StudId, KursId,
GesamtNote)
Prüfung (BelId, Datum Note NOT NULL)
```

Vererbung

Die Superklasse und Subklasse wird auf eine Tabelle abgebildet,
Primärschlüssel der Subklassen-Tabellen = Primärschlüssel der Superklasse.
Der Primärschlüssel der Subklassen ist zugleich Fremdschlüssel.
Vorteil: Flexibelste Lösung, Redundanzfrei, geeignet für überlappende Vererbung
Nachteil: Viele Tabellen, Komplexe Zugriffe, Zusätzlicher Typ-Attribut `Fahrzeug (FzgId INT, Marke STRING, Gewicht DECIMAL, FzgTyp INT NOT NULL)`
`PKW (FzgID REFERENCES Fahrzeug, AnzPlaetze INT NOT NULL)`

Eine Tabelle pro Klasse
Vorteil: Einfache Zugriffe auf die Tabellen
Nachteil: Semantikverlust (nicht mehr klar was gemeinsam ist), Schlüssel-Eindeutigkeit separat kontrollieren, keine Überlappende Vererbung

Nur eine "Super"-Tabelle
Vorteil: Einfache Zugriffe auf die Tabellen
Nachteil: Viele NULL-Werte pro Tupel, 3. oder höhere Normalform verletzt

Normalform

Die Redundanzfreiheit wird geprüft:
1.NF: Wertebereiche der Attribute atomar (nur ein). Strukturierte Werte wie Mengen, Gruppen, etc. sind also nicht zugelassen.
2.NF: jedes Nichtschlüsselattribut von jedem Schlüsselkandidaten voll funktional abhängig ist. Ein Attribut B ist funktional abhängig vom Attribut A, falls zu jedem Wert von A genau ein Wert von B existiert: jede ISBN gibt es nur ein Buch.
3.NF: kein Nichtschlüsselattribut von irgendeinem Schlüssel über "Umwege" abhängig.

Window Function

```
SELECT row_number() over(),
name, salaer, salaer -
lead(salaer,1,salaer) OVER
(ORDER BY salaer) FROM
angestellter ORDER BY 2 DESC;
```

Common Table Expressions

Wie temp. Tabellen, sind Hilfqueries

```
WITH angestprojekten AS (
SELECT a.persnr, a.name, a.chef,
a.abtnr, proj.bezeichnung FROM
angestellter a JOIN
projektzuteilung pz ON a.persnr
= pz.persnr
SELECT * FROM angestprojekten;
```

Recursive Query

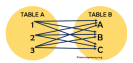
Dazu muss `WITH RECURSIVE` zusätzlich geschrieben werden.

```
WITH RECURSIVE query_name
     (column_name) AS (
-- Nicht rekursiver Teil.....
UNION ALL
-- Recursiver Teil...)
SELECT c1,c2 FROM query_name;
```

View

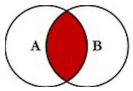
```
CREATE VIEW AngPublic (Persnr,
Name, Tel, Wohnort) AS SELECT
Persnr, Name, Tel, Wohnort FROM
Angestellter; -- Create View

SELECT * FROM AngPublic; -- Use
```

Join



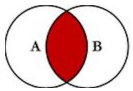
Cross Join
(Kartesisches Produkt)

```
SELECT * FROM ang,
abt;
```



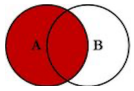
Inner Join

```
SELECT * FROM tab1
JOIN tab2 ON
tab1.spalte =
tab2.spalte;
```



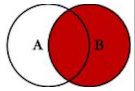
Natural Join
automatisch Spalten, mit gleichen Namen.

```
SELECT * FROM
tabelle1 NATURAL
JOIN tabelle2;
```



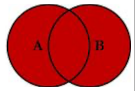
Left Join
wie Inner, nur + alle Elemente links.

```
SELECT * FROM
tabelle1 LEFT
JOIN tabelle2 ON
tabelle1.spalte =
tabelle2.spalte;
```



Right Join
wie Inner, nur + alle Elemente rechts.

```
SELECT * FROM
tabelle1 RIGHT
JOIN tabelle2 ON
tabelle1.spalte =
tabelle2.spalte;
```



Full Outer Join
Verknüpfung von Left und Right Join.

```
SELECT * FROM tab1
FULL JOIN tab2 ON
tab1.spalte =
tab2.spalte;
```

Self Join
Tabelle mit sich selbst.

```
SELECT * FROM tab1
a, tab1 b WHERE
a.chefid = b.id;
```

ANSI 3-Ebenen

Logische Ebene	Logische Strukur der Daten
Interne Ebene	Speicherstrukturen
Externe Ebene	Sicht einer Benutzerklasse auf Teilmenge der Daten
Mapping	Zw. Ebenen Abbildung nötig

Zindel Marius

SQL-Abfragen

Mit DISTINCT werden entstehende Duplikate ausgeschlossen.

Mit LIKE wird nach einem String-Muster gesucht, wobei % (0 bis n beliebige Zeichen) oder _ (genau ein beliebiges Zeichen) eingesetzt werden können.

Aggregatfunktionen liefern als Resultat nur eine Zeile. NULL-Werte werden übersprungen.

```
SELECT AVG( salaer ) FROM ang;
```

HAVING-Klausel kann nur nach einer GROUP-BY-Klausel stehen.

```
GROUP BY abtnr HAVING AVG(salaer)
 >= 7000 ORDER BY AVG(salaer);
```

IN Liste der Angestellten (Name), die in keinem Projekt mitarbeiten:

```
SELECT name FROM ang WHERE
persnr NOT IN (SELECT DISTINCT ..
```

EXISTS Liste der Angestellten, die in mindestens einem Projekt mitarbeiten:

```
SELECT name FROM ang a WHERE
EXISTS ( SELECT * FROM projzut
WHERE persnr = a.persnr);
```

ANY Gesucht sind die 'Marketing'-Angestellten, die weniger als irgend einer der Entwickler verdienen: ANY liefert für jedes solche Tupel 'true'.

```
SELECT ang.name, ang.salaer FROM
angestellter ang INNER JOIN
Abteilung Abt ON ang.abtnr =
abt.abtnr WHERE abt.name =
'Marketing' AND ang.salaer < ANY
( SELECT salaer FROM angestellter
ang1 INNER JOIN Abteilung abt1
ON ang1.abtnr = abt1.abtnr
WHERE abt1.name='Entwicklung');
```

ALL Gesucht sind die 'Marketing'-Angestellten, die mehr als jeder der Entwickler verdienen: ALL liefert für jedes solche Tupel 'true'

```
SELECT ang.salaer, ang.salaer
FROM angestellter ang INNER JOIN
abteilung abt ON ang.abtnr =
abt.abtnr WHERE abt.name =
'Marketing' AND ang.salaer > ALL
(SELECT salaer FROM angestellter
ang1 INNER JOIN Abteilung abt1
ON ang1.abtnr=abt1.abtnr
WHERE abt1.name='Entwicklung');
```

count (*)>0 → True oder False

Security

- Identifizierung und Authentisierung von Benutzern
- Überprüfung der Benutzerprivilegien für Ausführung von Systemoperationen.
- Kontrolle der Benutzung von CPU, Disk.

Benutzern sind Privilegien zugeordnet für Datenbankoperationen.

- Rollen gelten über alle alle Datenbanken
- Schemas fassen Datenbank-Objekte zusammen in einer bestimmten Datenbank
- Eine Datenbank kann n * Schemas haben
- Eine Rolle kann n * Schemas besitzen
- Ohne Angabe eines Schema-Namens werden alle Objekte im Default Schema "public" erzeugt

```
CREATE ROLE angproj WITH LOGIN
PASSWORD 'angproj';
GRANT INSERT ON TABLE
"Angestellter" TO "angproj";
```

Systemprivilegien legen fest, welche Operationen ein Benutzer ausführen darf.

- REATEDB: Erlaubt dem Benutzer Datenbanken auf Server zu erstellen.
- CREATEROLE: Erlaubt der Rolle / Benutzer neue Rollen/Benutzer zu erstellen.
- NOREADDB und NOCREATEROLE sind Negationen davon.

Mit REVOKE kann der Benutzer, das Recht wieder entziehen.

```
REVOKE [GRANT OPTION FOR]{
privilege [, privilege...]|ALL
[PRIVILEGES]} ON object FROM
{user[, user...]|GROUP
group|PUBLIC} [CASCADE|RESTRICT];
```

Gruppen sind Sammlungen von logisch zusammengehörenden Benutzern. Gruppen können Objektprivilegien zugeordnet werden. Benutzer können Gruppen zugeordnet werden, die Benutzer erhalten dann die in den Gruppen definierten Privilegien.

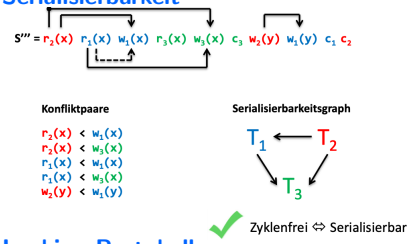
```
CREATE ROLE angmanager;
GRANT SELECT, UPDATE
ON angproj.angestellter TO
angmanager;
GRANT angmanager TO Blake;
```

Transaction

A	Atomicity	entweder vollständig oder gar nicht
C	Consistency	führt Daten von konsistenten Zustand in einen anderen
I	Isolation	Soll ausgeführt werden, als sei sie isoliert
D	Durability	Änderungen einer Transaktion gehen nicht durch Fehler verloren

Mit ROLLBACK oder ABORT kann Transaktion abgebrochen werden. Mit SAVEPOINT SavepointName kann aktuelle Punkt gespeichert werden. Mit ROLLBACK TO SavepointName kann zurückgesprungen werden. Nach COMMIT oder ROLLBACK gibt DBMS sämtliche Ressourcen frei.

Serialisierbarkeit



Locking Protokolle

- Exclusive Lock (xlock)
  Schreib- und Lesezugriffe sind gesperrt
- Shared Lock (slock)
  Nur Lesezugriffe sind gesperrt (mehrere können S-Lock auf Objekt halten)

Iso-Level

- READ UNCOMMITTED (schwächste)
  Lesezugriffe nicht synchronisiert (keine Read-Locks)
- READ COMMIETT
  Lesezugriffe nur kurz /temporär synchronisiert
- REPEATABLE READ
  Einzeln zugegriffene Rows sind synchronisiert
- SERIALIZABLE (stärkste)
  Vollständige Isolation

Isolation Level	Dirty Read	Fuzzy Read	Phantom
READ UNCOMMITED	mögl.	mögl.	mögl.
READ COMMITTED	nicht mögl.	mögl.	mögl.
REPEATABLE READ	nicht mögl.	nicht mögl.	mögl.
SERIALIZABLE	nicht mögl.	nicht mögl.	nicht mögl.

Dirty Read Lese Daten von anderer nicht committed Transaktion.

Fuzzy Read Gelesene Daten ändern sich plötzlich durch andere nebenläufige Transaktionen

Phantom Read INSERT oder DELETE von nebenläufigen Transaktionen. Tritt in Postgres nicht auf → Snapshot Isolation!

Recovery

Änderungen der Transaktionen in Logfiles geschrieben.

- Implementation des globalen Undo's und des globalen Redo's nach einem Fehlerfall.
- Checkpoint: DBMS schreibt alle modifzierten Pages auf Disk und macht Checkpoint Eintrag ins Log
- globales REDO: Die Wirkung sämtliche Transaktionen, welche mit einem Commit abgeschlossen wurden, darf nicht verlorengehen gehen
- globales UNDO: Die Wirkung sämtliche Transaktionen, welche noch nicht mit einem Commit abgeschlossen wurden, müssen rückgängig gemacht werden.

- Logischer Backup (mit pg_dump)
  Blockiert keine schreibende und lesende Transaktionen
- Physikalisches Backup
  Datenbank muss heruntergefahren warden – ist aber schneller
- Cloud Backup
  Alle Cloud Provider bieten Backups an in ihren PostgreSQL-Angeboten

Indexe

Arten von Indexen:

- ISAM
- B-Bäume
- B+ -Bäume
- Hash

Primär-Index: Index mit Primärschlüssel

Sekundär-Indexe: Alle anderen Indexe

Zusammengesetzter Index über mehrere zusammengesetzte Attribute/Kolonnen:

```
CREATE INDEX angestellter_name
_idx ON angestellter (vorname,
nachname);
```

B-tree Index mit zusätzlicher INCLUDE-Klausel als ≪Zusatzfracht≫. INCLUDE gibt Liste von Attriuten an, die als Nicht-Schlüsselteil in den Index aufgenommen werden.

```
CREATE INDEX magic_idx
ON test (nr,id) INCLUDE (txt);
```

Partieller Index mit WHERE-Klausel.

```
CREATE INDEX mytable_col_part_idx
ON mytable (col)
WHERE archived IS NOT NULL;
```

Funktionaler Index mit Funktion

```
CREATE INDEX angestellter_lower_
name_idx ON angestellter
(lower(name));
```

B-Baum Eigenschaften: Geeignet für Hintergrundspeicher, viele Index-Bedürfnisse und Fast optimal für verschiedene Queries und Einfügen.

Ein B+-Baum ist ein B-Baum, bei dem nur die Blätter Daten enthalten und zudem verkettet sind (z.B. als Linked List). Die Verkettung erlaubt schnelle Iterationen.

Logische Optimierung: Abfrage so umformulieren, dass sie dasselbe Resultat liefert aber effizienter berechnet werden kann.

- WHERE"-Statements so früh wie möglich, um Zwischenergebnisse klein zu halten
- Basisoperationen sollten ohne Zwischenspeicherung von Zwischenrelationen realisiert werden
- Nur Berechnungen ausführen, die auch einen Beitrag zum Gesamtergebnis liefern
- Zusammenfassen gleicher Teilausdrücke

Der Planer

- Full Table Scan
  Bsp: `SELECT * FROM angestellter;`
- Index Scan
  Falls in der WHERE-Klausel ein Attribut ist, zu dem es einen Index gibt.
  Bsp: `SELECT * FROM angestellter WHERE abtnr = 2`
- Bitmap Index Scan
  Ein Bitmap Index Scan lädt alle Tupel-Zeiger auf einmal aus dem Index, benutzt eine Bitmap-Struktur, um sie im Hauptspeicher zu sortieren und lädt die Tabellen-Tupel entsprechend der physischen Speicherreihenfolge.
- Index Only Scan Falls ein Attribut in der Projektion vorkommt. Bsp: `SELECT abtnr FROM angestellter WHERE abtnr=2;`
- Nested loop join
  Die rechte tabelle wird für jede Zeile der linken Tabelle gescannt. Ist einfach, kann aber zeitaufwändig sein.