

Grundlagen

DB: Datenbasis, strukturierte Daten auf HDD

DBMS: Datenbankmanagement System

Redundanzfreiheit: Element nur 1x pro DB, **Datenintegrität:** Datenkonsistenz:

korrekte Daten **Datensicherheit:** vor phy. Verlust, **Datenschutz:** vor unberecht. Zugriff
DBS = DBMS + (n *) DB

Funktionen DBMS Transaktionen, Mehrbenutzerbetrieb, Sicherheit, Backup & Recovery, Generische Datenstrukturen

Vorteile DBMS Skalierbar, Sicherheit, Integrität, Live-Abfragen, Kapslung

Datenbankmodelle Hierarchisch., relational, Netzwerk, Objektrelationale, -orientiert, NOSQL

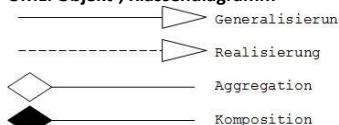
Datenbankentwurf

1. Anforderungsanalyse -> Anforderungsspezifikation
2. Konzeption. Entwurf -> konzeptuelles Schema(UML)
 - Unabhängig von eingesetztem DBMS
3. Logischer Entwurf -> logisches Schema
 - Rel. Schreibweise, angepasst an DBMS
4. Physischer Entwurf -> Physisches Schema
 - SQL-Code (+ Hardwareplanung)

1 Tier: DBMS gleicher Prozess wie Client

2 Tier gleiche Maschine

UML: Objekt-, Klassendiagramm



Disjoint: 1 Obj. = 1 Subcl.

Overlapping: 1 Obj. = n Sub.

Complete: All Subclas. def.

Incomplete: More sub. poss.

Abbildungsregel/strategie

1 Tabelle pro Sub- und Superklasse

- + Redundanzfreiheit(kleine Lösung)[auch bei overlapping]
- Komplexe Zugriffe(«wegen vielen Tabellen»)

2. Je eine Tabelle pro Subklasse. PK = FK ref. Superclass

- + Einfache Zugriffe auf die Elemente der Subklasse
- Komplexe Abfragen über alle Elemente der Superklasse

3. Eine Superklasse mit allen möglichen Attributen(NULL)

- + Einfache Zugriffe & effiziente Zugriffe
- 3. NF verletzt(→Abhängigkeiten innerhalb Tabelle)

ANSI 3: externe, konzeptionelle, interne

Schlüsselkandidaten: Attribut oder Attributkombination, welche Datensatz eindeutig identifizierbar macht

Eindeutig, laufend zuteilbar, knapp, invariant

Surrogatschlüssel: künstlicher Schlüssel

Referentielle Integrität: FK NN→Datenkonsis.

Einfügeanomalie, Löschanomalie, Änderungsanomalie

1. NF: Attribute sind atomar(z.B nicht Adresse)

2. NF: Jedes Nichtschlüsselattribut voll funktional abhängig von Schlüsselkandidat

3. NF: Jedes NSA transitiv abhängig(A←C, A←B←C)

Boyce-Codd-NF, 4. & 5. NF

Relationales Modell / Rel. Schreibweise

```

practice (practiceID INT PK, name TEXT UNIQUE, day DATE NOT NULL, teamid INT FK REFERENCES team on delete restrict);
  
```

Structured Query Language(SQL)

Data Definition Language(DDL): CREATE ALTER DROP

ON DELETE

- CASCADE: löscht Elemente, die darauf referenzieren
- RESTRICT(DEFAULT): Bei Referenzen, Fehler/Rollback
- SET NULL
- SET DEFAULT

CREATE DB somedb owner someuser;

```

CREATE TABLE angestellter(persnr INTEGER PRIMARY KEY, name VARCHAR(80) NOT NULL UNIQUE, abteilung INTEGER REFERENCES abteilung);
  
```

PRIMARY KEY(att1, att2);

```

ALTER TABLE angestellter ADD CONSTRAINT abteilung FOREIGN KEY (abteilung) REFERENCES abteilung [ON DELETE CASCADE];
  
```

```

ALTER TABLE angestellter ADD CONSTRAINT check_PLZ CHECK(plz BETWEEN 1000 AND 9999);
  
```

DROP CONSTRAINT check_PLZ;

```

ALTER TABLE ang ADD COLUMN street varchar(30);
  
```

```

ALTER TABLE ang DROP COLUMN street [RESTRICT];
  
```

```

ALTER TABLE ang ALTER COLUMN street TYPE varchar(80);
  
```

```

ALTER TABLE ang RENAME COLUMN street to streetname;
  
```

```

DROP TABLE [IF EXISTS] ang [CASCADE | RESTRICT]
  
```

Data Manipulation Language(DML)

INSERT INTO, UPDATE SET, DELETE /SELECT FROM

Bedingungen:

- | | |
|---------------------|--------------------|
| - BETWEEN 10 AND 20 | • IN('Zug 'Chur',) |
| • NOT | • IS NULL |
| • (I)LIKE | • IS NOT NULL |

% → beliebige und beliebig viele Character

_ → genau ein beliebiger Character

Aggregat Funktionen

```

SELECT pid as Personalnummer FROM angestellter WHERE name LIKE '%a' OR name ILIKE 'e%' order by 1 desc limit 2;
  
```

```

SELECT DISTINCT wohnort FROM angestellter;
  
```

```

SELECT sum(salaer) as Abteilungslohn FROM angestellter GROUP BY abtnr ORDER BY 1 desc ;
  
```

```

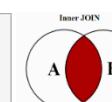
SELECT abtnr, avg(salaer) as Durchschnittslohn FROM angestellter group by abtnr HAVING avg(salaer) >= 5000 order by avg(salaer) desc;
  
```

SUM(), AVG(), MIN(), MAX()

Andere wichtige Funktionen
COUNT(), ROUND(), MOD() [Modulo], NOW() [aktuelles Datum],
JOINS

```

SELECT a.name as Angestellter, ab.name as Abteilung FROM angestellter a INNER JOIN abteilung ab on a.abtnr = ab. abtnr;
  
```



```

SELECT p.bezeichnung, a.name FROM projekt p LEFT [OUTER] JOIN angestellter a on a.persnr = p.ProjLeiter;
  
```

```

SELECT zeitanteil, name FROM projektzuteilung pz RIGHT [OUTER] JOIN angestellter a on pz.persnr = a.persnr;
  
```

```

SELECT a.name, ab.name FROM angestellter a FULL [OUTER] JOIN abteilung ab on a.abtnr = ab.abtnr;
  
```

```

SELECT a.name, ab.name FROM angestellter a CROSS JOIN abteilung ab order by a.name;
  
```

```

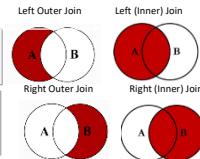
SELECT a.name, a.salaer, b.name, b.salaer FROM angestellter a INNER JOIN angestellter b on a.chef = b.persnr where a.salaer > b.salaer;
  
```

Natural Join: Join wenn gleichnamige Attribute

Theta Join: Relationale Algebra(<, >, =, ≥, ≤)

Equi Join: Join wenn etwas gleich(«equals-Join»)

Lateral Join: Auf äusser/innere Attr. zugreifen



EMP_ID	EMP_NAME	DT_OF_JOIN	EMP_SUPV
20001	Umesh Gehlot	15-APR-09	20001
20002	Ashish Gehlot	25-OCT-09	20001
20003	Avantika Gehlot	02-DECEMBER-09	20001
20005	Vineet Gehlot	27-NOV-08	20001
20007	Mukesh Singh	26-JAN-11	20007

Unterabfragen

Korriert: Unterabfrage greift auf Attribute der äusseren Abfrage zu → funktioniert nur mit äusseren Abfrage

Unkorriert: Unterabfrage greift nicht auf Attribute der äusseren Abfrage → funktioniert ohne äussere Abfrage

Lateral Join(zugreifen auf äussere/innere Elemente)

```

SELECT ab.name,eintritt.name, eintritt.eintrittsdatum FROM abteilung ab cross join
lateral( Select * from angestellter a where a.abtnr = ab.abtnr order by eintrittsdatum asc limit 2) eintritt;
  
```

```

SELECT name FROM angestellter where persnr IN( select AbtChef from abteilung);
  
```

IN : Alle Abteilungsleiter ausgeben

EXISTS: Alle die an einem Projekt arbeiten(keine Duplikate)

ANY: Angestelle, welche mehr verdienen als irgendein Chef

```

SELECT name FROM angestellter a where EXISTS( select * FROM projektzuteilung pz
where a.persnr = pz.persnr);
  
```

```

SELECT name FROM angestellter a where salaer > ANY( select salaer from
angestellter where chef is null);
  
```

```

SELECT name FROM angestellter where abtnr = 2 and salaer > ALL(select salaer from
angestellter where abtnr = 3);
  
```

ALL: MA Abt 3, welche Lohn > Lohn von allen MA von Abt 3

Mengenoperationen(Gleiche Anzahl Spalten!)

UNION: Verbindet zwei Mengen-> Essensgeld der Abteilungen

```

SELECT a.name, '9' as Essensgeld FROM angestellter a where abtnr = 1 UNION [ALL]
select a.name, '0' as Essensgeld from angestellter a where abtnr = 2;
  
```

EXCEPT: Linke Menge ohne rechte Menge -> MA ohne Projekt

```

SELECT persnr, name FROM angestellter EXCEPT SELECT persnr, projnr::varchar
FROM projektzuteilung;
  
```