





- Georgiy Shevoroshkin
- uml diagramm (konzeptionell)
    - assoziationen, bedingungen
  - normalisierung (+beispiele)
    - transitiv,(voll) funktional abhängig
  - transaktionen
    - isolation levels erklären, welche fehler sie beheben
    - fuzzy read, deadlock, dirty read, write skew, phantom read, serializable snapshot isolation, cascading rollbacks
    - schedule analysieren + Serialisierbarkeitsgraph
  - begriffe (physisches schema, DBMS)
  - b-baum indexe einfügen

### Unified Modeling Language

 Assoziation   
  Komposition  
 Aggregation   
  Vererbung

### Normalisierung

1NF: Atomare Attributwerte  
 2NF: Nichtschlüsselattr. voll vom Schlüssel abhängig  
 3NF: Keine transitiven Abhängigkeiten  
 BCNF: Nur abhängigkeiten vom Schlüssel

**Vererbung (vor & nachteile) (einzige tabelle für superklasse, tabelle pro subklasse, tabelle pro sub- und superklasse)** **Data Definition Language**

```
CREATE SCHEMA s ();
CREATE TABLE t (
  id SERIAL PRIMARY KEY,
  name TEXT UNIQUE,
  grade DECIMAL(2,1) NOT NULL,
  added TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,
  u VARCHAR(9) DEFAULT CURRENT_USER,
  fk INT FOREIGN KEY REFERENCES
t2.id ON DELETE CASCADE,
  CHECK (grade between 1 and 6)
);
ALTER TABLE t2 ADD CONSTRAINT c
PRIMARY KEY (a, b);
```

### Usermanagement

```
CREATE ROLE r WITH LOGIN PASSWORD ' '
GRANT INSERT ON TABLE t TO r;
REVOKE CREATE ON SCHEMA s FROM r;
ALTER ROLE r CREATEROLE, CREATEDB,
INHERIT;
GRANT r TO user_name;
-- read all future created tables
ALTER DEFAULT PRIVILEGES IN SCHEMA s
```

```
GRANT SELECT ON TABLES TO
readonlyuser;
CREATE POLICY p ON t FOR ALL TO
PUBLIC USING (u = current_user);
ALTER TABLE t ENABLE ROW LEVEL
SECURITY;
```

### Data Manipulation Language

```
INSERT INTO t (id, grade) VALUES (1,
1) RETURNING id;
```

### Views

```
CREATE VIEW v (id, grade, u) AS
SELECT id, grade, u FROM t;
```

### Common Table Expressions

```
WITH RECURSIVE q AS (SELECT * FROM t
WHERE grade>1 UNION ALL SELECT *
FROM t INNER JOIN q ON q.u = t.name)
SELECT id as "ID" FROM q;
```

### Window Functions

```
SELECT id, RANK() OVER (ORDER BY
grade DESC) as r FROM t;
```

### Subqueries

```
SELECT * FROM t WHERE grade > ANY/
IN/EXISTS (SELECT g FROM t2);
```

### JOIN

```
SELECT y.*, x.* FROM t AS y, JOIN
LATERAL (SELECT * FROM t2 WHERE
t2.id = y.id) AS x;
```

### GROUP BY

```
SELECT id, COUNT(*) FROM t GROUP BY
grade, id HAVING COUNT(*) > 2;
```

### WHERE

```
BETWEEN 1 AND 5; LIKE '___%'
IN (1, 5) ; LIKE '%asd'
```

### INDEX

```
CREATE INDEX i ON t /*USING BTREE*/
(grade, UPPER(u)) INCLUDE added;
DROP INDEX i;
```

### Transaktionen

```
BEGIN; SAVEPOINT s;
COMMIT; ROLLBACK /*TO SAVEPOINT s*/;
```

### Isolation

```
READ UNCOMMITTED; READ COMMITTED
REPEATABLE READ ; SERIALIZABLE
```

### Relationale Algebra

```
 $\pi_{R1,R4}(R)$  SELECT R1,R4 FROM R;
 $\sigma_{R1>30}(R)$  SELECT * FROM R WHERE R1 > 30;
 $\rho_a \leftarrow R$  SELECT * FROM R AS a;
 $R \times S$  SELECT * FROM R,S;
 $R \bowtie_{A=B} S$  SELECT * FROM R JOIN S ON R.A=S.B;
```

### Transaktionen

### Serialisierbarkeit