

Neugestaltung der Datenbank des Chemnitzer Studentennetzes

Verteidigung Bachelorarbeit

Morris Jobke
Prüfer: Dr. Frank Seifert
Betreuer: Dipl.-Inf. Johannes Fliege

**Neugestaltung der
Datenbank des
Chemnitzer Studentennetzes**

Neugestaltung der
Datenbank des
Chemnitzer Studentennetzes

Chemnitzer Studentennetz

- studentische Initiative
- Ziel:
 - Netzwerk innerhalb der Wohnheime
 - Anbindung an das Uni-Netz
- ehrenamtliche Mitarbeiter
- Gründung: 1994

Neugestaltung der
Datenbank des
Chemnitzer Studentennetzes

Datenbank

- persistente Speicherung
- strukturierte Daten
- Beziehungen zwischen Daten

Neugestaltung der
**Datenbank des
Chemnitzer Studentennetzes**

Aufgabe der Datenbank des CSN

- Daten für den Betrieb des CSN
 - Nutzerinformationen
 - Netztechnik
 - Finanzdaten
 - Inventar
 - Nachrichten
 - ...

Neugestaltung der
Datenbank des
Chemnitzer Studentennetzes

Bisheriger Stand der Technik

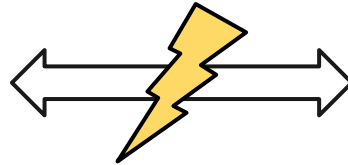
"Never touch a running system."

Funktionierendes System

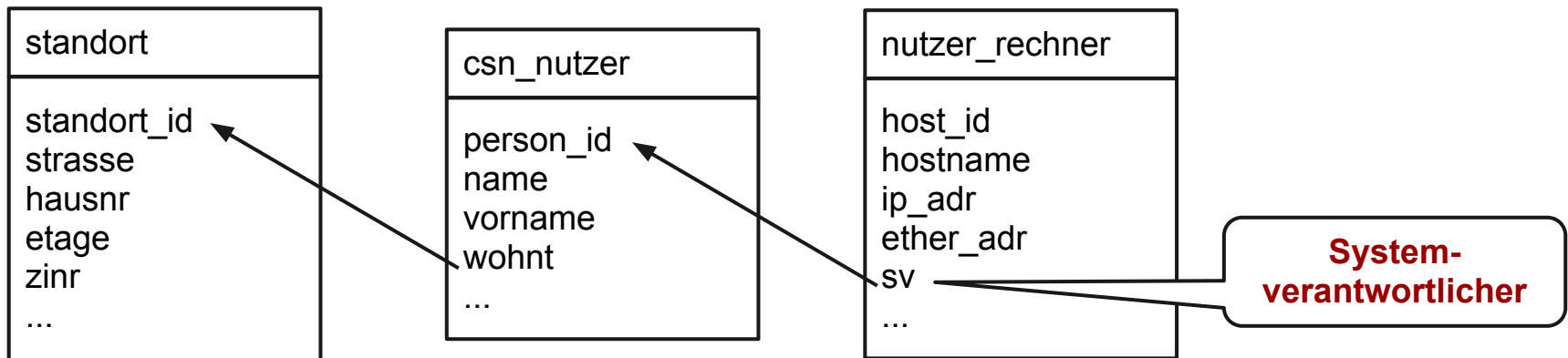
ABER

minimale Anpassungsmöglichkeiten/
extremer Zeitaufwand

traffic_current
traffic_overlimit
hub_config_last



finanz_jahr
nutzerrechner
standort



Benennung

SELECT * FROM kategorie **ORDER BY** kategorie_id;

kategorie_id	katname	katchar
1	Nutzungsgebuehr	N
2	Material	M
3	Sonstiges	S

[...]

(4 rows)

SELECT * FROM finanz_titel **ORDER BY** titel_id;

titel_id	titel	titelchar
1	Nutzungsgebühr	N
2	Material	M
3	Sonstiges	S

[...]

(12 rows)

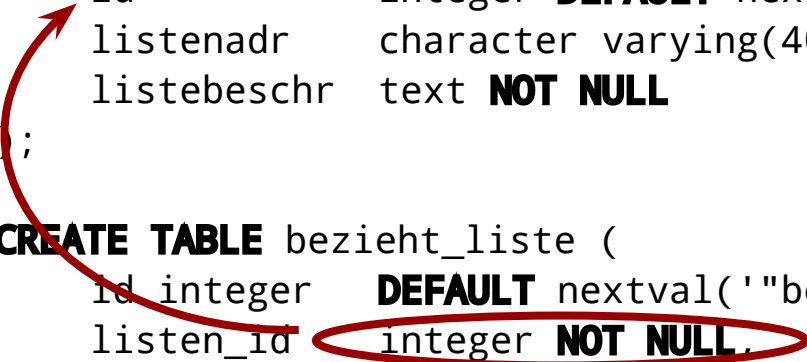
Doppelte Daten

```
CREATE TABLE mailingliste (  
    id            integer DEFAULT nextval('"mailingliste_id_seq"'::text) NOT NULL,  
    listenadr     character varying(40) NOT NULL,  
    listebeschr   text NOT NULL  
);
```

```
CREATE TABLE bezieht_liste (  
    id integer DEFAULT nextval('"bezieht_liste_id_seq"'::text) NOT NULL,  
    listen_id   integer NOT NULL,  
    aufg_id     integer NOT NULL  
);
```

Beziehungen

```
CREATE TABLE mailingliste (  
    id            integer DEFAULT nextval('"mailingliste_id_seq"'::text) NOT NULL,  
    listenadr     character varying(40) NOT NULL,  
    listebeschr   text NOT NULL  
);  
  
CREATE TABLE bezieht_liste (  
    id integer DEFAULT nextval('"bezieht_liste_id_seq"'::text) NOT NULL,  
    listen_id integer NOT NULL,  
    aufg_id      integer NOT NULL  
);
```



Beziehungen

```
CREATE FUNCTION abstime_as_german_date(abstime) RETURNS text
AS '
SELECT CASE WHEN isfinite($1)
    THEN date_part(''day'', $1) || ''.'' ||
        date_part(''month'', $1) || ''.'' ||
        date_part(''year'', $1)
    ELSE ''infinity''::text END;
'
LANGUAGE sql;
```

Neugestaltung

Nachteile

- Benennung
- Daten-
Doppelungen
- Obsolete Daten
- Logik innerhalb der
Datenbank

Vorteile

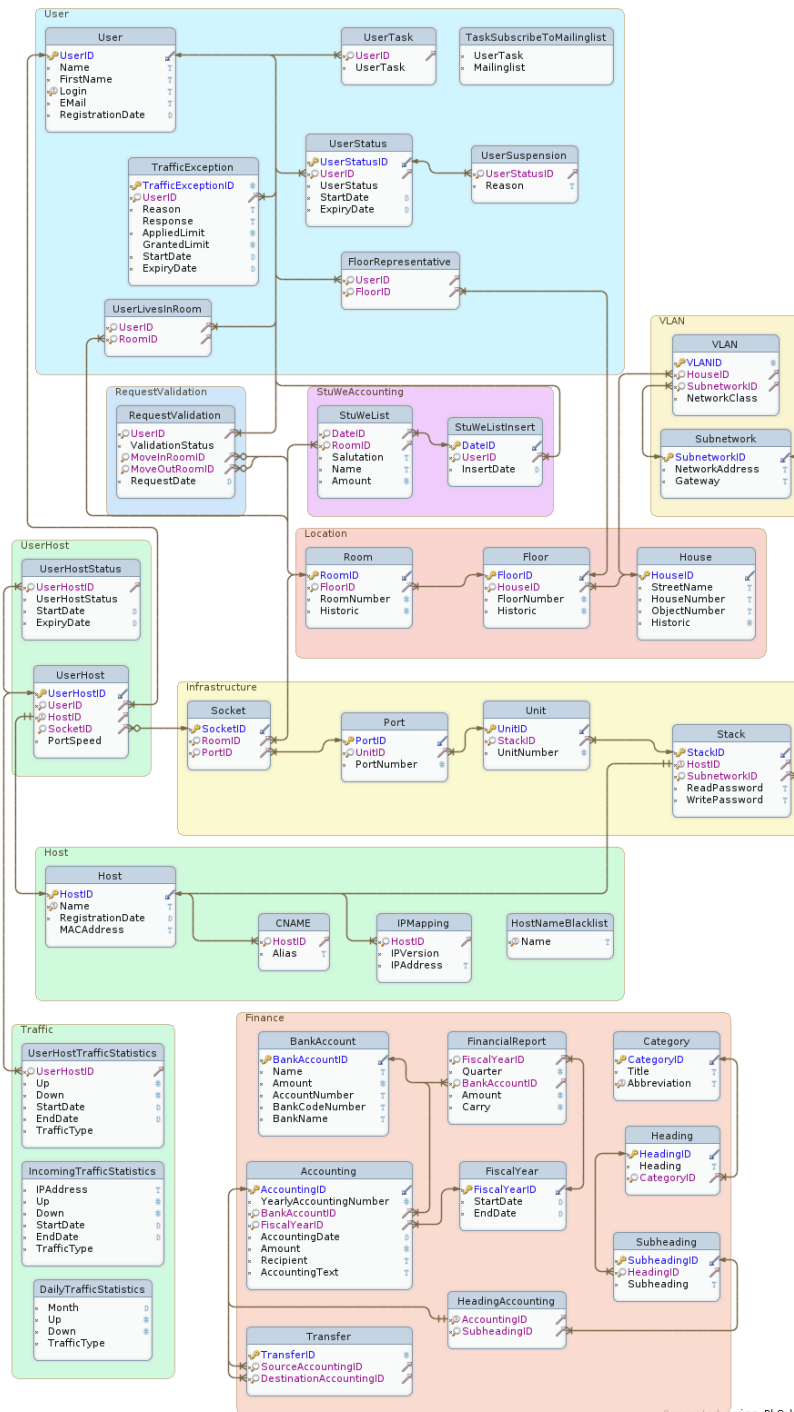
- Funktionierendes
Gesamtkonzept
- Daten

**Neugestaltung der
Datenbank des
Chemnitzer Studentennetzes**

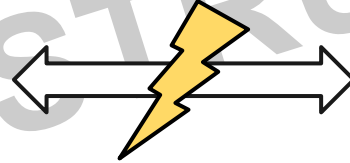
Anforderungen an das neue System

- Abbildung sämtlicher Daten
- Workarounds ausbessern
- zukünftige Probleme
- Benennungskonventionen
- Abbildung sämtliche Strukturen im Modell
- Vermeidung von Datendoppelung, DB-Prozeduren und überflüssigen Strukturen

Modell



traffic_current
traffic_overlimit
hub_config_last



finanz_jahr
nutzerrechner
standort

DailyTraffic

Socket

Port

Unit

House

UserStatus

UserTask

FloorRepresentative

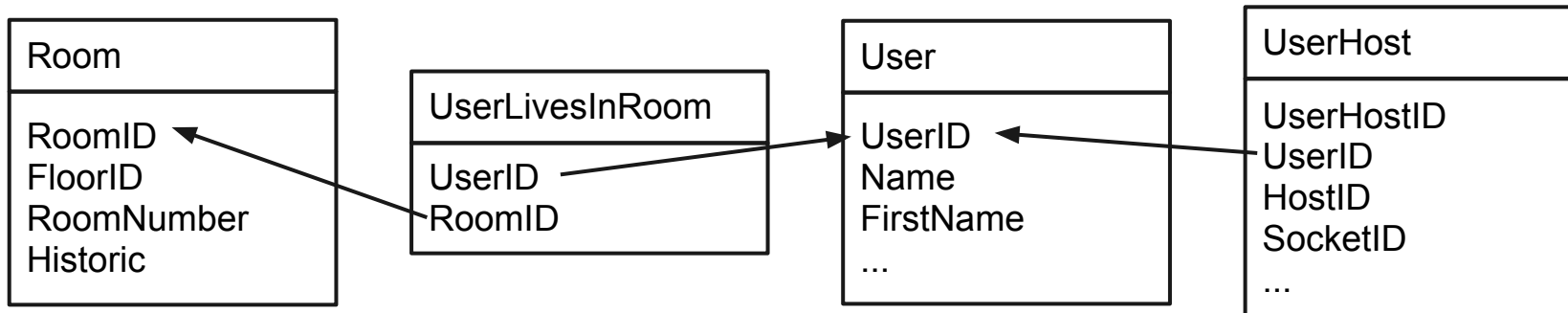
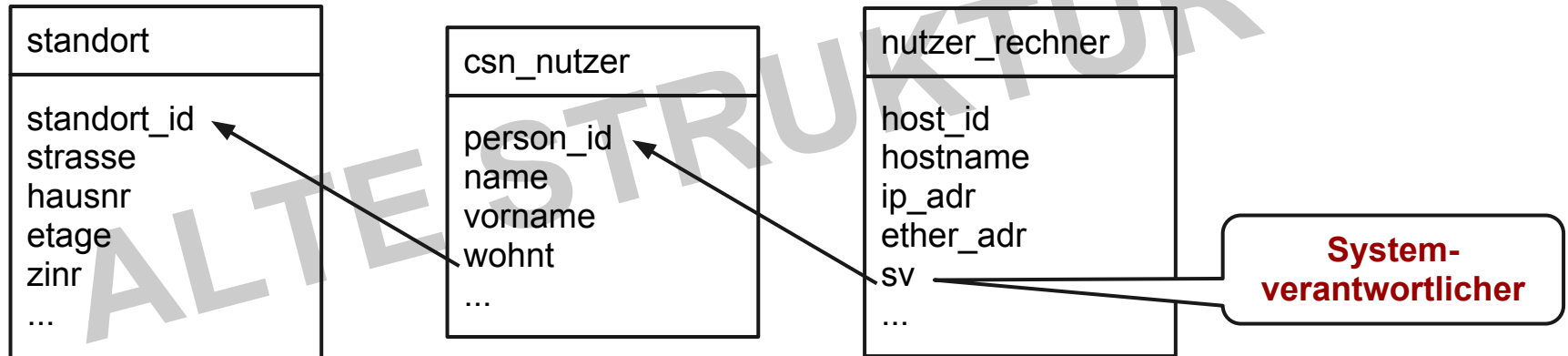
FiscalYear

UserHost

Room

Floor

Benennung



Benennung

SELECT * FROM kategorie **ORDER BY** kategorie_id;

kategorie_id	katname	katchar
1	Nutzungsgebuehr	N
2	Material	M
3	Sonstiges	S

[...]

(4 rows)

SELECT * FROM finanz_titel **ORDER BY** titel_id;

titel_id	titel	titelchar
1	Nutzungsgebühr	N
2	Material	M
3	Sonstiges	S

[...]

(12 rows)

Doppelte Daten

```
CREATE TABLE mailingliste (  
    id            integer DEFAULT nextval('"mailingliste_id_seq"'::text) NOT  
NULL,  
    listenadr     character varying(40) NOT NULL,  
    listebeschr   text NOT NULL  
);
```

```
CREATE TABLE bezieht_liste (  
    id integer DEFAULT nextval('"bezieht_liste_id_seq"'::text) NOT NULL,  
    listen_id   integer NOT NULL,  
    aufg_id    integer NOT NULL  
);
```

```
CREATE TYPE Mailinglist AS ENUM (  
    ...  
);
```

```
CREATE TABLE TaskSubscribeToMailinglist (  
    "UserTask"      UserTask NOT NULL,  
    "Mailinglist"   Mailinglist NOT NULL  
);
```

Beziehungen

```
CREATE FUNCTION abstime_as_german_date(abstime) RETURNS text
AS '
SELECT CASE WHEN isfinite($1)
    THEN date_part(''day'', $1) || ''.''' ||
        date_part(''month'', $1) || ''.''' ||
        date_part(''year'', $1)
    ELSE 'infinity'::text END;
'
LANGUAGE sql;
```

```
#!/usr/bin/env python
```

```
timeObject.strftime("%d.%m.%Y")
```


Umsetzung

- virtueller Server im CSN
- Ubuntu 12.04 LTS
- PostgreSQL 9.1

Praktische Probleme

- marginale Unterstützung von Vererbung in PostgreSQL

```
CREATE TABLE "Host" {  
    "HostID"      serial PRIMARY KEY,  
    "Name"        varchar  
};
```

```
CREATE TABLE "UserHost" {  
    "UserName"    varchar  
} INHERIT ("Host");
```

- keine Vererbung von Constraints und keine tabellenübergreifenden Constraints

```
CREATE TABLE "Host" {  
    "HostID"          serial PRIMARY KEY,  
    "Name"            varchar  
};
```

```
CREATE TABLE "UserHost" {  
    "HostID"          integer REFERENCES "Host"  
    ("HostID"),  
    "UserName"        varchar  
};
```

```
CREATE TABLE "Stack" {  
    "HostID"          integer REFERENCES "Host"  
    ("HostID"),  
    "ReadPassword"    varchar  
};
```

Problemlösung - Vererbung

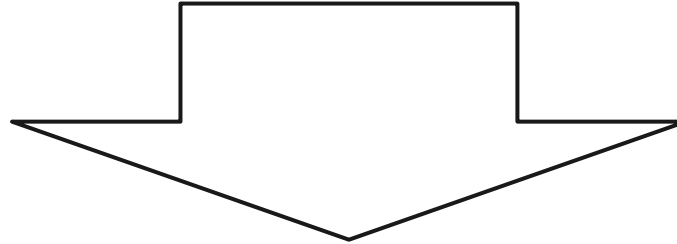
Daten-Migration

- eine Migrationsanweisung pro Tabelle:
 - Bezug der benötigten Daten
 - Anpassungen (Zeichensatz, ...)
 - Erstellung der Einfüge-Statements

```
SELECT strasse, hausnr, hkz, objekt_nr FROM haus ORDER BY haus_id;
```

strasse	hausnr	hkz	objekt_nr
Thueringer Weg	3	Tw3	8235
Reichenhainer Strasse	35	Rh35	8110
...			
Vettersstrasse	72	Ve72	8180

(10 rows)



```
INSERT INTO "House" ("StreetName", "HouseNumber", "Abbreviation",  
    "ObjectNumber") VALUES  
( 'Thüringer Weg',      '3',  'Tw3',      8235),  
( 'Reichenhainer Straße', '35', 'Rh35', 8110),  
...  
( 'Vettersstraße',      '72', 'Ve72', 8180);
```

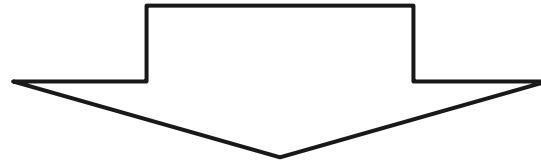
Migrationsbeispiel

Evaluation

- Erfüllung sämtliche Anforderungen
- Ausnahme: Vermeidung von Datendoppelung
 - praktisch sinnvolle Umsetzung
 - ansonsten komplexe und zeitintensive Abfrage (über drei weitere Tabellen)

Zusammenfassung

- einheitliches und übersichtliches Schema
- vollständige Migration



darauf aufbauende Neugestaltung der
Software-technischen Grundlage im CSN

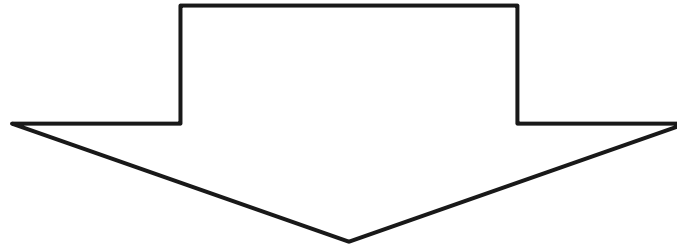
Dokumentation, Dokumentation,
Dokumentation

Fragen?


```
SELECT hkz, etage, zinr FROM standort JOIN haus USING (haus_id) ORDER BY
    hkz, etage, zinr;
```

hkz	etage	zinr
Rh35	0	11
Rh35	0	12
Rh35	0	13
...		

(2370 rows)



```
INSERT INTO "Room" ("FloorID", "RoomNumber") VALUES
((SELECT "FloorID" FROM "Floor"
    WHERE "FloorNumber" = 0 AND "HouseID" = 'Rh35'), 11),
((SELECT "FloorID" FROM "Floor"
    WHERE "FloorNumber" = 0 AND "HouseID" = 'Rh35'), 12),
((SELECT "FloorID" FROM "Floor"
    WHERE "FloorNumber" = 0 AND "HouseID" = 'Rh35'), 13),
...;
```

Migrationsbeispiel