Entwicklungsumgebung für die Übung

VU Datenbanksysteme

Wolfgang Fischl

Arbeitsbereich Datenbanken und Artificial Intelligence
Institut für Informationssysteme
Technische Universität Wien

Wintersemester 2015/16

Gliederung

Übung

- Adresse: `bordo.dbai.tuwien.ac.at`
- Zugriff:
  - Von „überall“ her möglich (z.B. Übungsräume, andere Rechner im TU-Netz, zu Hause, in der Firma, etc.)
  - ausschließlich mittels **Secure Shell** (egal von wo aus)
- Betriebssystem auf bordo: Debian
- Linux Kennung:
  - Alle Übungsteilnehmer erhalten nach der Anmeldung eine Linux Kennung mit Usernamen `u<matrikelnummer>`.
  - Das Passwort wurde bereits an die in TISS hinterlegte E-Mail zugesandt.
  - Passwort ändern: mit Linux-Befehl **passwd**. Geändertes Passwort nicht vergessen!

Secure Shell

- Von Windows-PC aus:
  - SSH-Client (PuTTY) und SCP-Client (PSCP): `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`
  - Bequemer Datentransfer: WinSCP `http://winscp.net/`
- Von Linux-PC aus:
  - ssh und scp sind ohnehin vorhanden
  - Bequemer Datentransfer: sftp mit Konquerer oder Nautilus

Beispiel

- ssh username@bordo.dbai.tuwien.ac.at
- scp bsp1.sql username@bordo.dbai.tuwien.ac.at:abgabe1.sql

PostgreSQL Datenbank auf Bordo

DB-Server auf Bordo:

- PostgreSQL 9.4

DB-Benutzung:

- Alle LVA-Teilnehmer erhalten eine eigene DB-Kennung in der Datenbank
- DB-Username: gleich wie Linux-Username
- Starten einer interaktiven Session: mit dem Kommando psql (im ssh-Fenster). Authentifikation über SSH Account, d.h. kein Passwort notwendig

Editoren auf Bordo

- **emacs**: Hilfe im Web, z.B. `http://www.gnu.org/software/emacs/tour/`
- **vi**: Hilfe im Web, z.B. `http://www.library.yale.edu/wsg/docs/vi_hands_on/`
- **pico**: Hilfe im Web, z.B. `http://www.uic.edu/depts/accc/software/pine/pico.html`
- Alternativen: Entwicklung von SQL und PL/SQL auf dem lokalen PC und Upload nach Bordo mittels scp (oder sftp).

Gliederung

Datei Befehle

- Wichtige Befehle für die Laborübung

`\o datei`	Output wird in `datei` geschrieben
`\o`	Ende der Ausgabe in `datei`
`\i datei`	Führt die Befehle aus `datei` aus

Beschreibungsbefehle

- `\d+` Tabellen auflisten
- `\df+` Funktionen auflisten
- `\d+ table` Tabelle beschreiben
- `\df+ functions` Funktion beschreiben

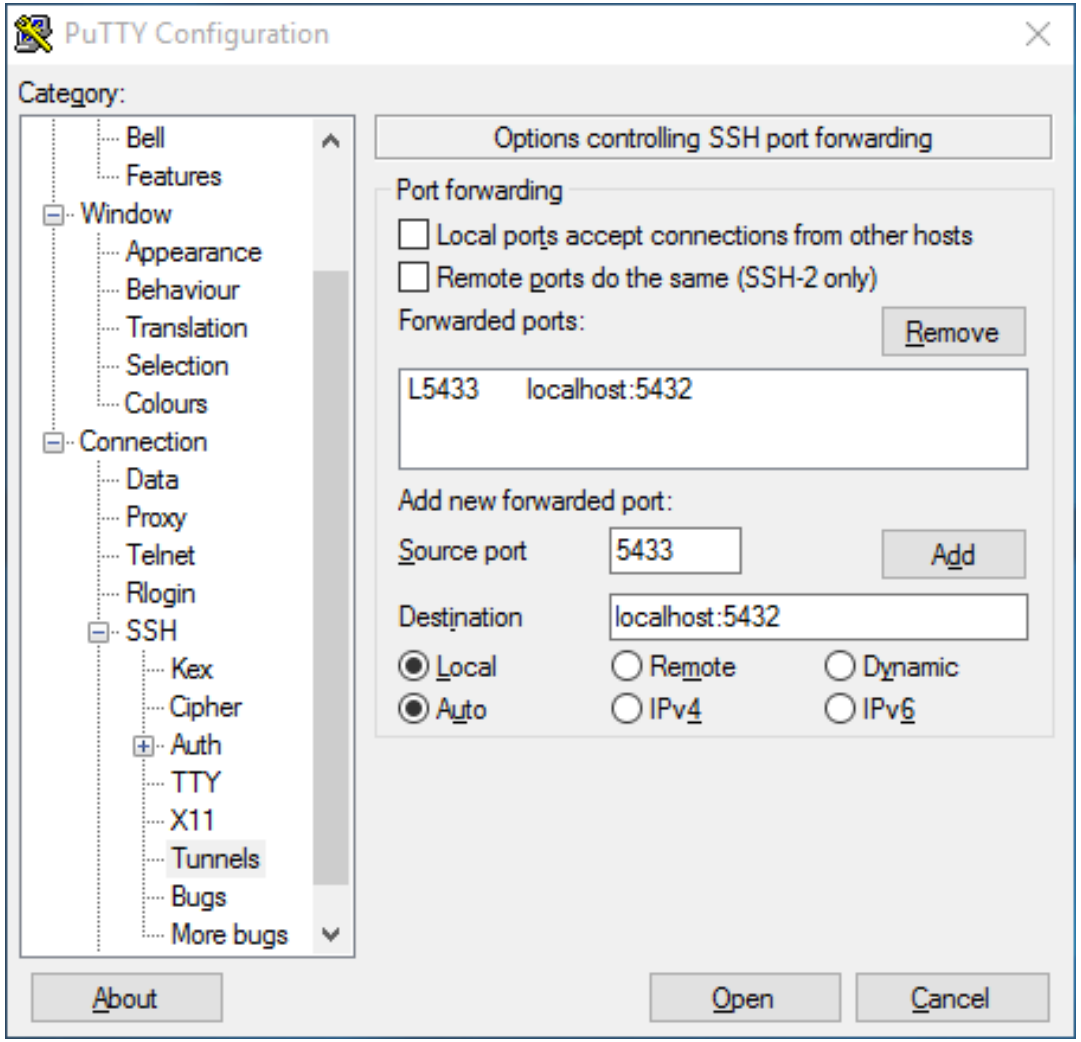
Beispiel

```
SELECT column_name FROM
  INFORMATION_SCHEMA.COLUMNS
  WHERE table_name = 'table';
```

pgAdmin

- Grafisches Tool zur Datenbankadministration
- SQL Editor, Anzeigen von Tabellen, Datenbanken
- Kann mit dem PostgreSQL Server auf Bordo verwendet werden
- `http://www.pgadmin.org`

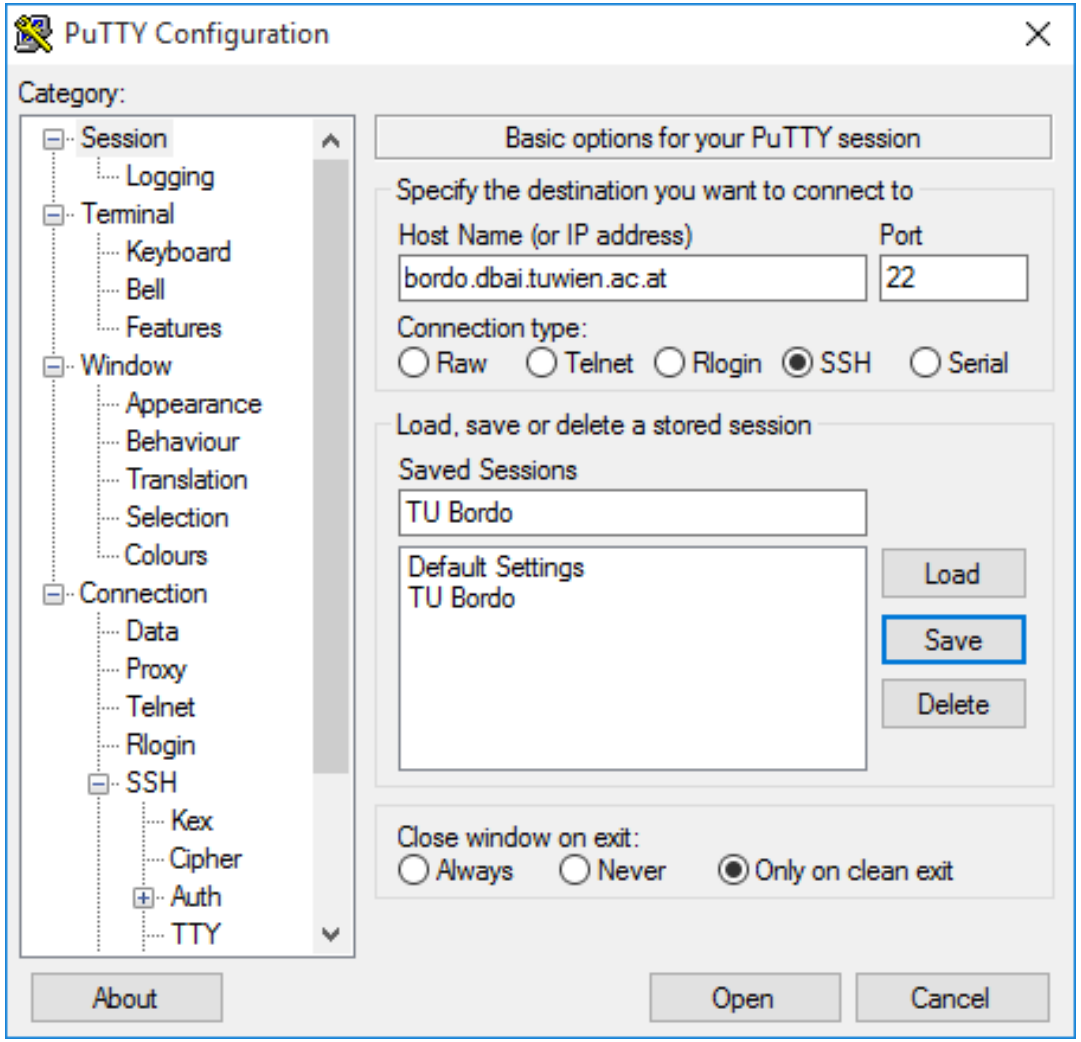
Einrichtung von pgAdmin - Port Forwarding



Linux

ssh -L 5433:127.0.0.1:5432 u<Matrikelnummer>@bordo.dbai.tuwien.ac.at

Einrichtung von pgAdmin - Port Forwarding



Session speichern und verbinden

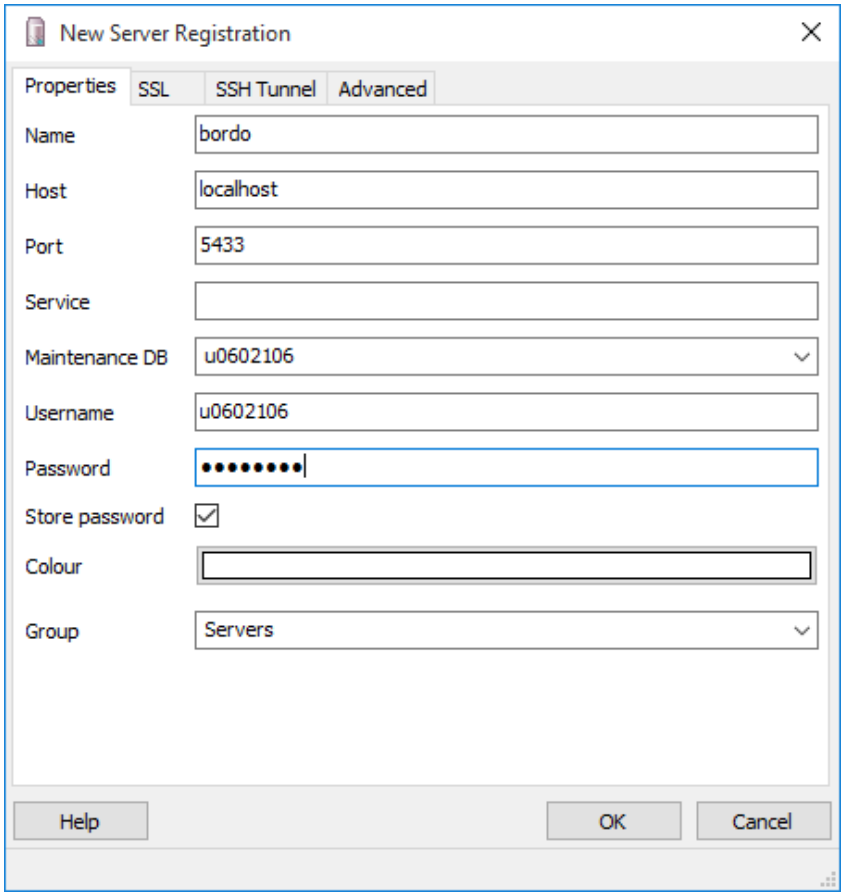
Einrichtung von pgAdmin - Port Forwarding

bordo.dbai.tuwien.ac.at - PuTTY

```
login as: u0602106
u0602106@bordo.dbai.tuwien.ac.at's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Oct  7 17:24:59 2015 from t195-235.demo.tuwien.ac.at
u0602106@bordo:~$ psql
psql (9.4.4)
Type "help" for help.

u0602106=> \password
Enter new password:
Enter it again:
u0602106=>
```

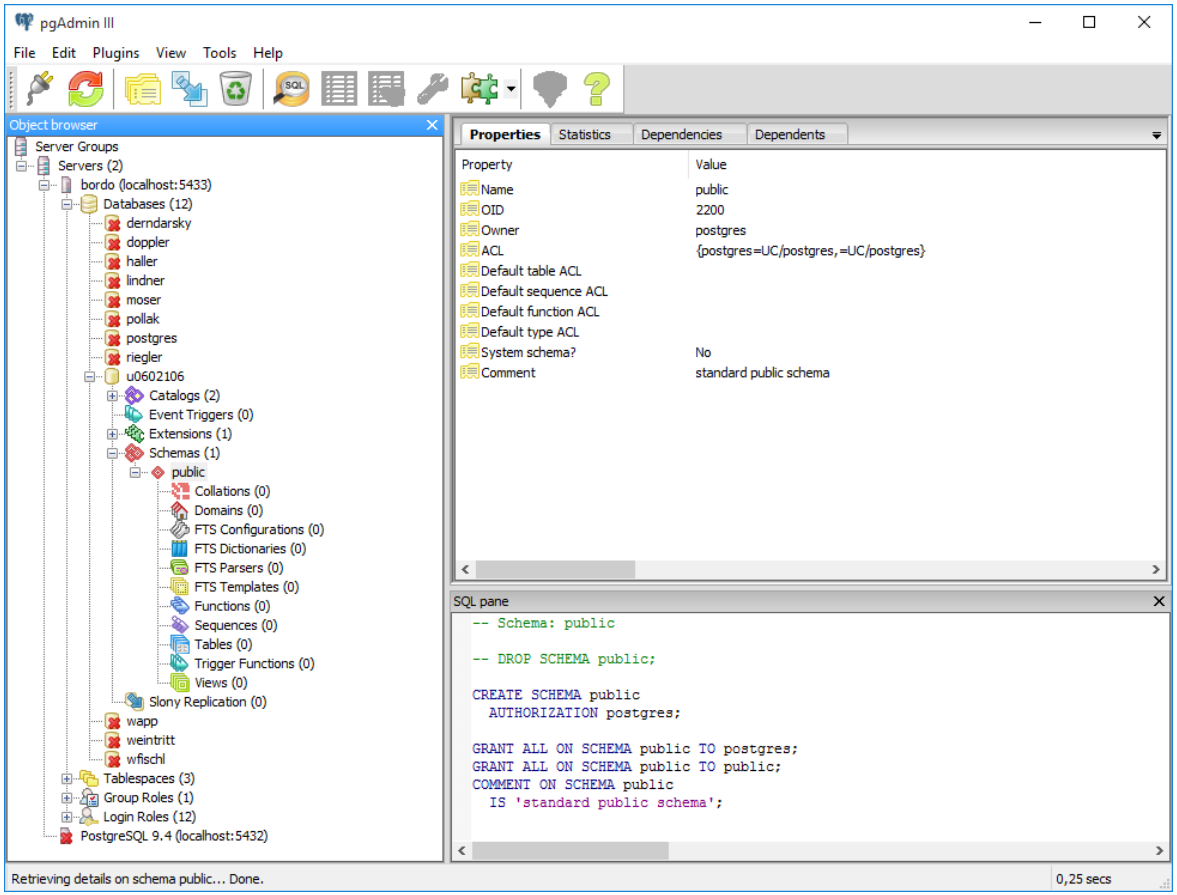
- Mittels `\password` in psql ein Passwort setzen
- Fenster geöffnet lassen

Einrichtung von pgAdmin - New Server Registration



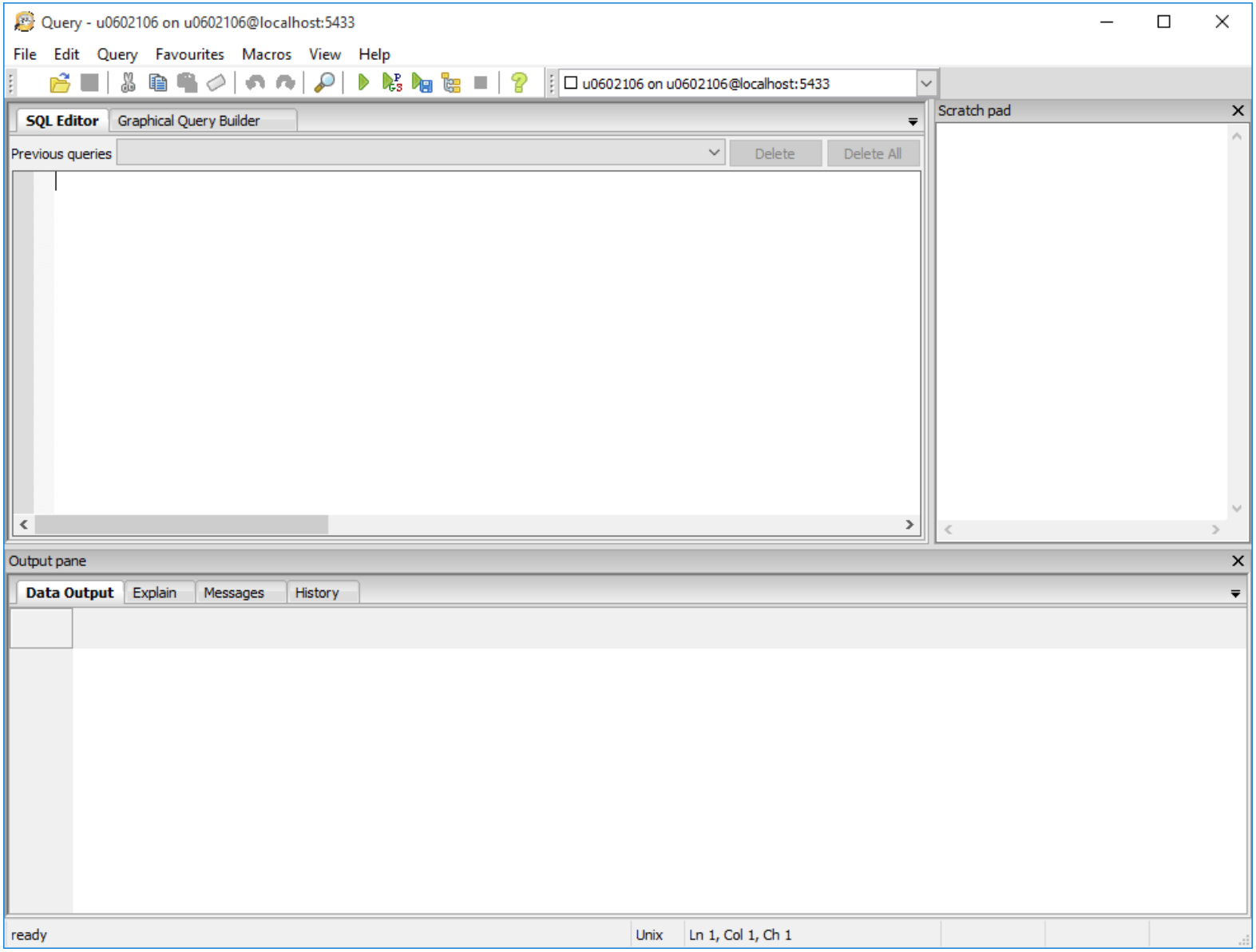
- Maintenance DB und Username sind u<Matrikelnummer>
- Password wie vorhin gesetzt

Einrichtung von pgAdmin



- Zu PostgreSQL verbunden
- Schaltfläche SQL öffnet Query Fenster

Einrichtung von pgAdmin - SQL

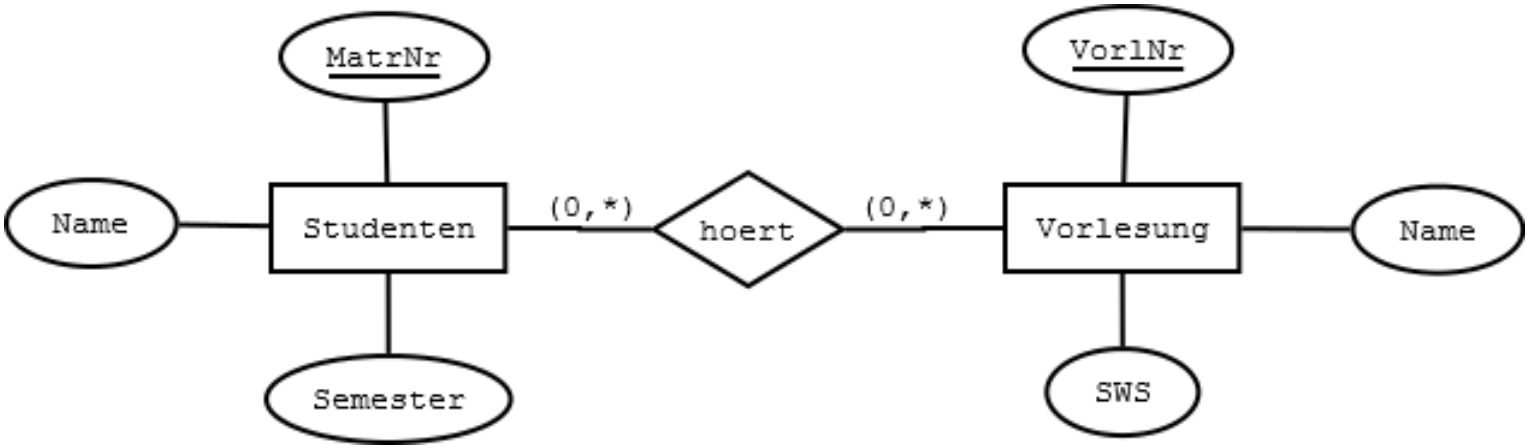


Einrichtung von pgAdmin - Zusammenfassung

- Port-Forwarding einrichten
- DB Passwort in `psql` setzen
- Mittels pgAdmin zur Datenbank auf Bordo verbinden

Gliederung

CREATE Tables



CREATE Tables

- Datei 01_create.sql:

```
CREATE TABLE Vorlesungen (
    VorlNr INTEGER PRIMARY KEY,
    SWS INTEGER,
    name VARCHAR(50)
);

CREATE TABLE Studenten (
    MatrNr INTEGER PRIMARY KEY,
    semester INTEGER,
    name VARCHAR(50)
);

CREATE TABLE hoeren (
    VorlNr INTEGER REFERENCES Vorlesungen,
    MatrNr INTEGER REFERENCES Studenten,

    PRIMARY KEY(VorlNr,MatrNr)
);
```

- Einlesen und Kompilieren: \i 01_create.sql

CREATE Tables

- Datei 02_insert.sql:

```
INSERT INTO Vorlesungen VALUES (1,4,'DBS');
INSERT INTO Vorlesungen VALUES (2,4,'DM');
INSERT INTO Vorlesungen VALUES (3,2,'SSD');

INSERT INTO Studenten VALUES (1,1,'Franz');
INSERT INTO Studenten VALUES (2,2,'Anna');
INSERT INTO Studenten VALUES (3,2,'Maria');
INSERT INTO Studenten VALUES (4,4,'Karl');

INSERT INTO hoeren VALUES (1,2);
INSERT INTO hoeren VALUES (2,2);
INSERT INTO hoeren VALUES (1,3);
INSERT INTO hoeren VALUES (2,3);
INSERT INTO hoeren VALUES (3,3);
INSERT INTO hoeren VALUES (3,1);
INSERT INTO hoeren VALUES (2,4);
INSERT INTO hoeren VALUES (3,4);
```

- Einlesen und Kompilieren: \i 02_insert.sql

SQL Statement

Beispiel

Suche die Namen jener Studenten, die alle 4-stündigen Vorlesungen gehört haben.

- Ersetzung von „for all“ durch „exists“:
  Suche die Namen jener Studenten, für die *nicht* gilt: Es gibt eine 4-stündige VO, für die *nicht* gilt: der Student hat diese VO gehört.
- Geschachteltes SQL-Statement erstellen: „schrittweise“ von innen nach außen.

SQL Statement

Innerstes **SELECT**:

- Wähle MatrNr eines beliebigen Studenten sowie Vorlesungsnummer einer beliebigen Vorlesung:

```
SELECT * FROM hoeren ORDER BY VorlNr
SELECT MatrNr FROM studenten
SELECT DISTINCT MatrNr FROM hoeren
```

  z.B., wähle MatrNr 1 und VorlNr 3

- **SELECT** Statement (in Datei test1.sql editieren):

```
SELECT * FROM hoeren h
WHERE h.VorlNr = 3 AND h.MatrNr = 1
```

- Datei einlesen und ausführen in psql: \i test1.sql

SQL Statement

Mittleres **SELECT**:

- Gesucht: Informationen über alle 4-stündigen Vorlesungen, die von einem bestimmten Studenten (z.B. wieder der Student mit MatrNr 1) nicht gehört wurden.
- **SELECT** Statement (in Datei test2.sql editieren):

```
SELECT * FROM Vorlesungen v
WHERE v.SWS = 4 AND NOT EXISTS
   (SELECT * FROM hoeren h
    WHERE h.VorlNr = v.VorlNr
    AND h.MatrNr = 1)
```

- Datei einlesen und ausführen in psql: \i test2.sql

SQL Statement

Äußeres **SELECT**:

- Gesucht: Informationen über Studenten, für die es keine 4-stündigen Vorlesungen gibt, die von diesen Studenten nicht gehört wurden.
- **SELECT** Statement (in Datei abgabe_bsp1.sql editieren):

```
SELECT * FROM Studenten s
WHERE NOT EXISTS
   (SELECT * FROM Vorlesungen v
    WHERE v.SWS = 4 AND NOT EXISTS
       (SELECT * FROM hoeren h
        WHERE h.VorlNr = v.VorlNr
        AND h.MatrNr = s.MatrNr))
```

- Datei einlesen und ausführen in SQL*Plus:
  \i abgabe_bsp1.sql

PL/SQL Prozedur

- Gesucht: Prozedur, die den Namen und die Semesteranzahl eines Studenten ausgibt
- Mögliche Vorgangsweise:
  - Entwicklung mittels pgAdmin
  - Am Anfang: nur Prozedur-Kopf und „NULL“-Body
  - „Schrittweise“ den Body hinzufügen
  - Nach jedem Schritt kompilieren.
  - „Test-Treiber“ erstellen
  - Bildschirmausgaben mittels RAISE NOTICE / RAISE DEBUG zwecks Fehlersuche

PL/SQL Prozedur

```
CREATE OR REPLACE FUNCTION suche(matrnr integer)
  RETURNS void AS $$
DECLARE
BEGIN
  NULL;
END;
$$ LANGUAGE plpgsql;

SELECT suche(1);
```

PL/SQL Prozedur

```
CREATE OR REPLACE FUNCTION suche(mnr integer)
  RETURNS void AS $$
DECLARE
name     varchar(30);
semester numeric(2);
BEGIN
  SELECT s.name, s.semester INTO name, semester
    FROM studenten s WHERE s.matrnr = mnr;
    RAISE NOTICE 'Name: %, Semester: %', name, semester;
END;
$$ LANGUAGE plpgsql;

SELECT suche(1);
```

- Aber was wenn kein Ergebnis gefunden wurde?

PL/SQL Prozedur

```
CREATE OR REPLACE FUNCTION suche(mnr integer)
   RETURNS void AS $$
DECLARE
name     varchar(30);
semester numeric(2);
BEGIN
  SELECT s.name, s.semester INTO name, semester
    FROM studenten s WHERE s.matrnr = mnr;
  IF (name IS NULL) THEN
    RAISE NOTICE 'Student mit der MatrNr % nicht gefunden', mnr;
  ELSE
    RAISE NOTICE 'Name: %, Semester: %', name, semester;
  END IF;
END;
 $$ LANGUAGE plpgsql;

SELECT suche(10);
```

- Datei einlesen und ausführen in SQL*Plus: \i plpgsql.sql

Gliederung

Beispiel 1- Aufgabenstellung

- `http://dbai.tuwien.ac.at/education/dbs/current/uebung/bsp1.html`

- Auftragsverwaltung einer Eventtechnikfirma
  - Beschreibung und ER-Diagramm gegeben
  - Anlegen der Tabellen in PostgreSQL
  - Anlegen von Testdaten
  - Datenauswertung mit SQL Queries
  - PL/pgSQL Trigger und Functions
  - Vollständiges Löschen der angelegten Objekte

Beispiel 1- Punktevergabe

- Beim Abgabegespräch wird das Verständnis der Konzepte überprüft!

- Punkteverteilung
  - DB anlegen/löschen: max. **4** Punkte
  - Queries: max. **2** Punkte
  - PL/pgSQL: max. **4** Punkte

Beispiel 1- Korrekte Abgabe

- **1** ZIP Datei mit folgenden Dateien:
  1. `create.sql`
  2. `insert.sql`
  3. `drop.sql`
  4. `queries.sql`
  5. `plpgsql-teil.sql`
  6. `test.sql`
  7. `listing.txt`

Beispiel 1- `listing.txt`

- Kopiere alle Dateien auf Bordo
- Auf Bordo:

```
u0602106@bordo:~$ ls
create.sql  drop.sql  insert.sql  plpgsql-teil.sql  test.sql
u0602106@bordo:~$ psql
psql (9.4.4)
Type "help" for help.

u0602106=> \o listing.txt
u0602106=> \i create.sql
u0602106=> \i plpgsql-teil.sql
u0602106=> \i insert.sql
u0602106=> \i test.sql
psql:test.sql:32: ERROR:  Der zu behandelnde Patient ist bereits mit einer anderen Krankheit i
psql:test.sql:35: ERROR:  Der Arzt kann nur Krankheiten behandeln auf die die Abteilung in der
psql:test.sql:38: ERROR:  Koordinator muss in der Abteilung arbeiten, die er koordiniert!
psql:test.sql:41: ERROR:  Leiter muss in einer der Abteilung dieses Krankenhauses arbeiten!
psql:test.sql:44: ERROR:  new row for relation "krankheit" violates check constraint "krankhei
DETAIL:  Failing row contains (10, 0.80, Geteerte Lungen).
psql:test.sql:47: ERROR:  new row for relation "mitarbeiter" violates check constraint "mitarb
DETAIL:  Failing row contains (7564280686, -1.00, 2015-10-13, 10, 1).
psql:test.sql:50: ERROR:  new row for relation "akteneintrag" violates check constraint "akten
DETAIL:  Failing row contains (5287081081, 1, 2014-10-28, 1970-01-01, 10, 5).
u0602106=> \i drop.sql
u0602106=> \q
```

Beispiel 1- `beispiel1.zip`

- zB.: auf Bordo:

```
u0602106@bordo:~$ zip beispiel1.zip *
  adding: create.sql (deflated 71%)
  adding: drop.sql (deflated 61%)
  adding: insert.sql (deflated 63%)
  adding: listing.txt (deflated 82%)
  adding: plpgsql-teil.sql (deflated 70%)
  adding: test.sql (deflated 56%)
```

Gliederung

Zusammenfassung

- Verbindung zu Übungsserver `bordo.dbai.tuwien.ac.at` herstellen
- Mit PostgreSQL DB verbinden

- Entwickeln einer einfachen Beispieldatenbank
  - Create Statements
  - SQL Statements
  - Stored Procedures
  - Drop Statements

- Durchbesprechung des ersten Übungsbeispiels