Data Modelling & SQL - Day 2

Architecture, infrastructure and networking

In this intermezzo we'll try to clarify some things about architecture, infrastructure and deployment of our software.

How do we run our code?

First, lets create a web server in Java

```
public class Main {
    public static void main(String[] args) {
        // Listen on port 8080
    }
}
```

Compile it and run it on our laptop

```
$ javac Main.java
$ java Main
```

Access it on the laptop

Go to your browser and access http://localhost:8080

Access it from your network

- Make sure your firewall allows it
    - Traffic to port 8080
    - On the Java process
- Bind your app to the IP address of your laptop (ipconfig/iftables)
    - Or bind to 0.0.0.0

Access it from outside your network

- Not so easy…
- Which is a good thing. Attackers shouldn’t be able to access your network.
- Having a fixed IP is handy
- Setup port forwarding on your gateway (Router)

And your webapp is accessible, at a cost

- This costs you download and upload bandwidth
- It requires a stable connection
- And now you cannot restart your laptop anymore
- Opens up your internal network to the outside world
    - Which is a risk and requires work (updates, hardening…)

What is a server?

- "Just a computer", without peripherals, like screens
    - Only network connectivity
- Used to offer services over the network

Typical server requirements

- Stability
- Good network connectivity
    - So also secure
- Low energy usage
- "Good enough" performance

Home server


LÖWIK
Synology

Datacenter

Hosting

- Reserve a server from a hosting provider
- They provision a server (VM) for you
- And give you access information

- Sometimes even a fancy control panel (like Plesk)

Installing your app on the server

- Package your app (in Java a Jar)
- Write an init daemon script which starts the app
    - Or register a service

DNS

- Domain Name System
- Phone book of the internet
- Maps logical names to IP addresses
- You need to set up DNS records for www, email, etc

Et voila!

- Now you can access your webapp
- On http://mydomain.com:8080/

Why the port?

- The default HTTP port is 80 and HTTPS 443
- All other ports need to be specified explicitly in the browser

But I want port 80

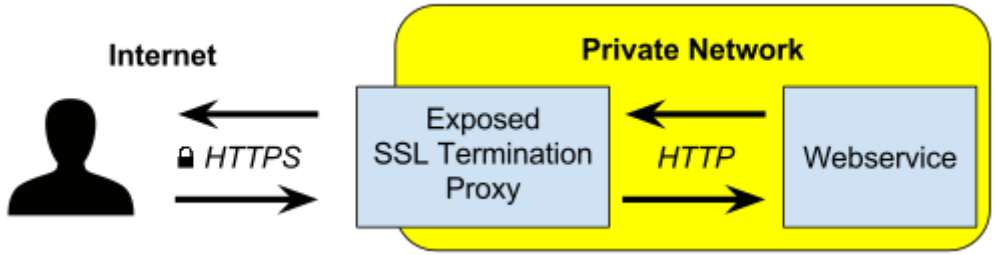
- Then listen on that port in your Java code
- Listening on port 80 (below 1024) requires root privileges
- Please don’t do this, don’t expose your Java app to the internet
    - Especially running as root

Web proxies

- Software which sits between the internet and your app
- Listens to i.e. port 80 and proxies to your app (on localhost:8080)
- I.e. envoyproxy, traefik, Apache Httpd or Nginx
- This proxy now needs to run as root!

SSL/TLS/HTTPS

- Still, the site is not using HTTPS
- Requires special request handling, which is not in our simple Java app
- The previously mentioned proxies do this
- "SSL offloading/termination"

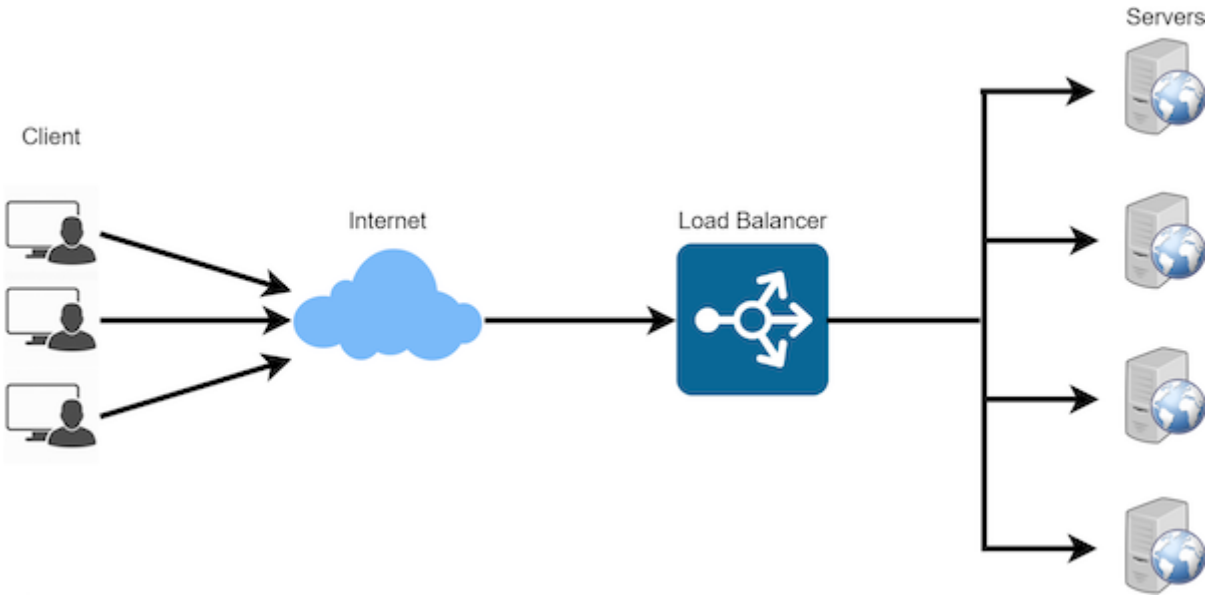


Adding more servers

- Our app becomes popular, we need more capacity
- Either a bigger server (scale up)
- Or more servers (scale out)
- More servers require load balancing

Load balancing

- Hardware or software in front of your app
- Which routes requests to all our backend servers
- Opaque for clients



More on load balancing

- SSL termination at load balancer
- Advanced balancing strategies

Adding persistent data

- We don’t yet store data
- But we want to
- So let’s add a database

Database hosting choices

- Managed
- Unmanaged

Unmanaged database hosting

- You rent a server or VM
- You install the database software (like Postgres)
- You configure it, secure it and run it
- You take care of updates, etc
- Gives you more control and much more work

Managed database hosting

- Often cloud offerings
- Just press a button and voila!
- No access to the underlying OS
- Often provides additional tools, i.e. for monitoring
- (Auto)scaling features

Connecting to a database in Java

```
String url = "jdbc:postgresql://dbhost:4321/dbname";
Properties props = new Properties();
props.setProperty("user", "username");
props.setProperty("password", "password");
Connection conn = DriverManager.getConnection(url, props);
```

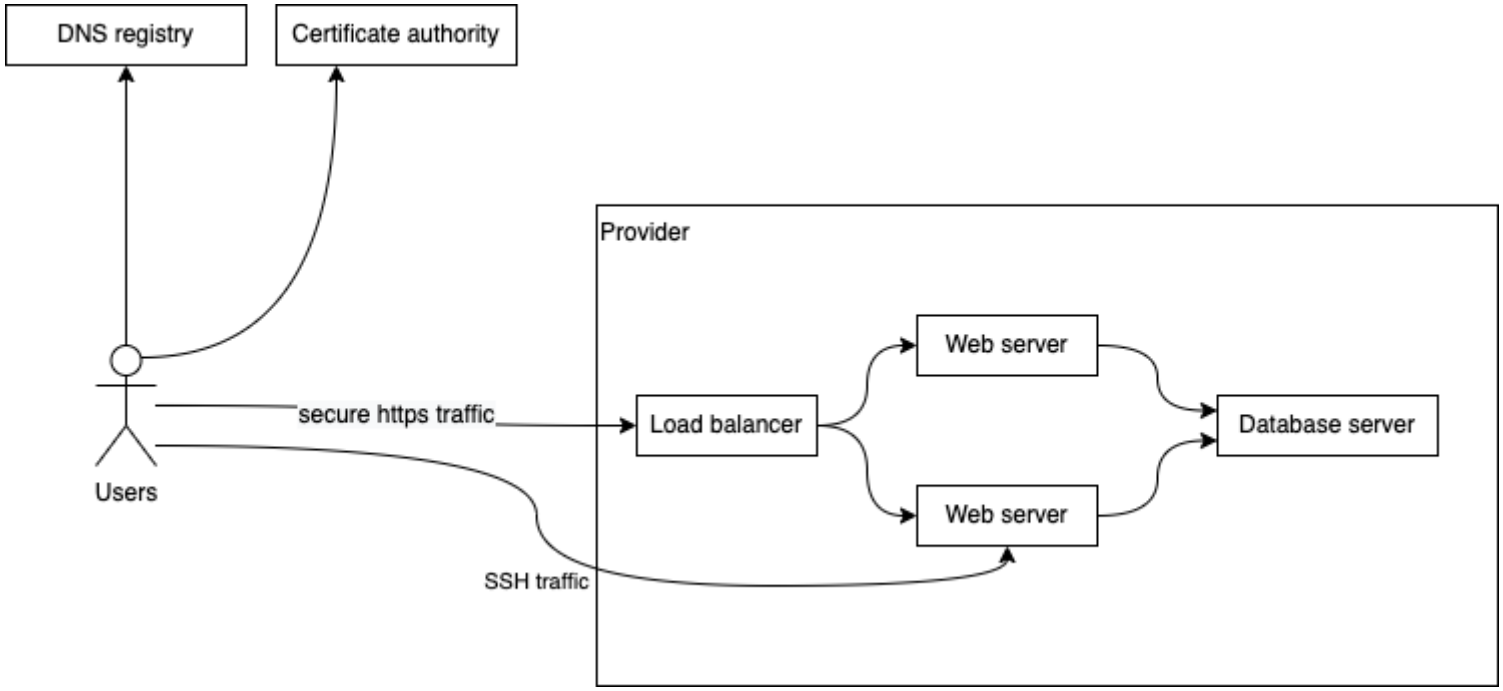
Running SQL queries in Java

```
Statement statement = connection.createStatement();
String sql = "UPDATE people SET first_name='John' WHERE id=123";
int rowsAffected = statement.executeUpdate(sql);
statement.close();
```

Summary

- We’ve created a simple server app
- Running on our laptop
- Exposed it to the internet
- Migrated it to the cloud
- Gave it a DNS entry
- Secured it, added loadbalancing
- And added a database

Final solution



How to read the command syntax

See https://www.postgresql.org/docs/13/sql-createtable.html

Alternative notation, see https://docs.oracle.com/cd/B19188_01/doc/B15917/sqcmd.htm#BABFBEFF (graphical notation, also BNF - Backus-Naur form)

DDL

Stands for **D**ata **D**efinition **L**anguage.

Recap: 4 groups of SQL commands.[1]:

- DDL (Data definition language)
- DML (Data modification language)
- DCL (Data control language)
- TCL (Transaction control language)

DDL provides syntax for creating, modifying and deleting database objects such as tables, columns, indices and users.[2]== Demo: basic DDL

First steps:

- CREATE, ALTER, DROP TABLE
- ADD, ALTER COLUMN

```
---
--- demo1: basic ddl
---

--- create table
create table demo1 (
    key serial primary key,
    value varchar(20)
);

--- alter table, add column
```

```
alter table demo1 add column comment text;

--- insert value too long -> fail
insert into demo1(value) values ('abcdefghijklmnopqrstuvwxyz');

--- insert value -> ok
insert into demo1(value) values ('abcdefghijklmnopqrst');

--- alter table, change column, reduce length -> fail due to already inserted data
alter table demo1 alter column value type varchar(10);

--- alter table, change column, reduce length -> ok
truncate table demo1;
insert into demo1(value) values ('a');
alter table demo1 alter column value type varchar(10);

--- delete table
drop table demo1;
----== Further DDL features

=== Recap: Demo1

Known till now:

* CREATE, ALTER, DROP TABLE
* ADD/ALTER COLUMN
** changing column type
```

```
create table if not exists people (
                        id INT,
                        first_name VARCHAR(50),
                        last_name VARCHAR(50),
                        full_name VARCHAR(50),
                        email VARCHAR(50),
```

```
                        gender VARCHAR(50),
                        ip_address VARCHAR(20),
                        ssn VARCHAR(50)
);

ALTER TABLE people ADD CONSTRAINT people_pk PRIMARY KEY (id);

CREATE TABLE orders(
    order_id serial primary key,
    product_name varchar(50),
    person_id integer
);


ALTER TABLE orders ADD CONSTRAINT orders_person_id_fk
FOREIGN KEY (person_id) REFERENCES people (id);
```

Demo2

Recap:

- primary keys
- foreign keys
- joins

Next steps:

- Inheritance/composition
- NULL/ NOT NULL values

- DEFAULT values
- GENERATED values
- PRIMARY KEYs
- FOREIGN KEYs
    - referential integrity.[3].[4]
- CONSTRAINTS

```
---
--- demo2: further ddl
---

--- create tables

--- show documentation on `inherits` vs. `like` (inheritance vs. composition ?), see
https://www.postgresql.org/docs/13/sql-createtable.html
create table identifiable (
    id serial primary key not null
);

create table users (
    name varchar(100),
    like identifiable including all
);

create table posts (
    author_id int not null references users(id),
    content text not null,
    like identifiable including all
);
```

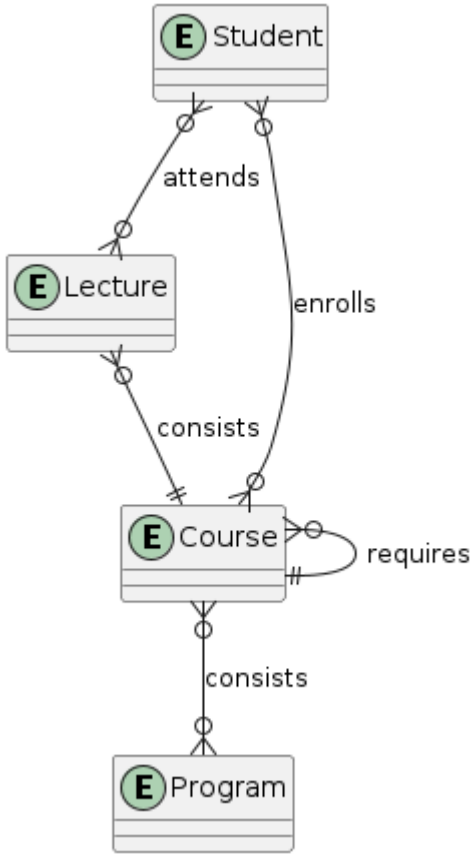
```
create table comments (
    author_id int not null references users(id),
    post_id int not null references posts(id),
    content text not null,
    upvotes int default 0,
    downvotes int default 0,
    rating int generated always as (upvotes - downvotes) stored,
    like identifiable including all
);

--- show er diagram using "right click on database in pgadmin -> generate erd"


--- try to populate the database demonstrating current constraints and defaults
insert into users(name) values ('jan');

insert into posts(author_id, content) values ((select id from users where name='jan'), 'Starting a new great
course!');

select * from posts;

insert into users(name) values ('alexei');

select * from users; -- note, that the same sequence is used for auto-generating the id in all tables!

insert into comments(author_id, post_id, content) values ((select id from users where name='jan'), 100, '');
-- error

insert into comments(author_id, post_id) values ((select id from users where name='jan'), 2); -- error,
content may not be null

insert into comments(author_id, post_id, content) values ((select id from users where name='jan'), 2, ''); --
ok
```

```
--- try to add constraints to enforce some basic logic on content and usernames

alter table users add constraint udx_users_name unique(name);

insert into users(name) values ('jan'); -- error

truncate table comments;

alter table comments add constraint c_comments_content_length check (length(content) > 10);

insert into comments(author_id, post_id, content) values ((select id from users where name='jan'), 2, ''); --
error

insert into comments(author_id, post_id, content) values ((select id from users where name='alexei'), 2, 'A
great course indeed, enjoyed it!'); -- ok

select * from comments;

update comments set upvotes = 47, downvotes = 11;

select * from comments;

--- drop tables, demo wrong order wrt. foreign keys
drop table comments;
drop table posts;
drop table users;
drop table identifiable;
```

Exercise time!

Write DDL for the ER diagram from "ER diagram" lecture.

Recap (conceptual):



Solution draft

```
---
--- exercise
---

create table identifiable (
    id serial primary key not null
);
```

```
create table programs (
    name text not null,
    syllabus text,
    like identifiable including all
);

create table courses (
    name text not null,
    syllabus text,
    like identifiable including all
);

create table lectures (
    name text not null,
    -- ordinal int not null,
    -- course_id int not null references courses(id),
    -- constraint uniq_lecture_course_ordinal unique(course_id, ordinal),
    like identifiable including all
);

create table students (
    name text not null,
    number text not null constraint uniq_student_number unique,
    date_of_birth date not null,
    constraint uniq_student_name_birth unique (name, date_of_birth),
    like identifiable including all
);

create table program_courses (
    program_id int not null references programs(id),
    course_id int not null references courses(id),
    constraint pk_program_courses primary key (program_id, course_id)
);
```

```
create table course_lectures (
    course_id int not null references courses(id),
    lecture_id int not null references lectures(id),
    ordinal int not null,
    constraint pk_course_lectures primary key (course_id, lecture_id),
    constraint uniq_lecture_course_ordinal unique(course_id, ordinal)
);

create table course_prerequisites (
    course_id int not null references courses(id),
    required_course_id int not null references courses(id),
    constraint pk_course_prerequisites primary key (course_id, required_course_id)
);

create table student_course_enrolls (
    student_id int not null references students(id),
    course_id int not null references courses(id),
    enrollment_date date not null,
    constraint pk_student_course_enrolls primary key (student_id, course_id)
);

create table student_lecture_attends (
    student_id int not null references students(id),
    lecture_id int not null references lectures(id),
    attendance_date date not null,
    constraint pk_student_lecture_attends primary key (student_id, lecture_id)
);
```

[1] https://en.wikipedia.org/wiki/Data_control_language

[2] User being a kind of grey area imo, since some databases treat users as entries in system/information_schema tables

[3] https://en.wikipedia.org/wiki/Referential_integrity

[4] https://en.wikipedia.org/wiki/Foreign_key