Creating database

Using postgresql as a database to learn how to make tables inside a database, how to open and close a session in a database, how to handle a transaction.

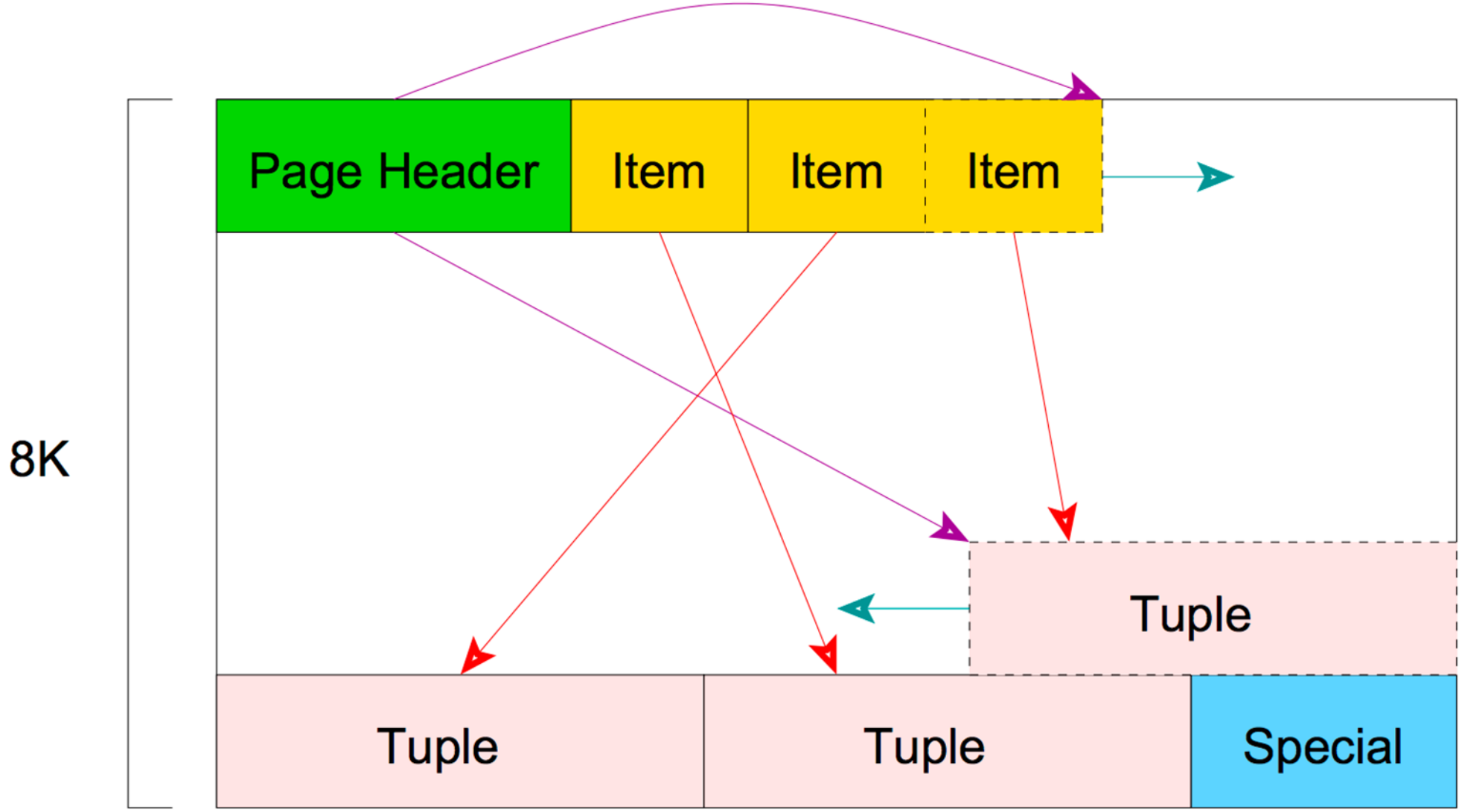
Also know how postgresql work, how pages stored in the hard disk, how to handle a complex relations (tables) and the low levels of how postgresql add, update, delete column in a table or an entire row.

Data-structure behind how rows are stored and fixed-size pages.

Low-level Postgresql

- How the rows are stored
- Every table stored as an array of pages of a fixed size (usually 8Kb). In a table, all the pages are logically equivalent, so a particular item (row) can be stored in any page.
- The structure used to store the table is a heap file. Heap files are lists of unordered records of variable size. The heap file is structured as a collection of pages (or block), each containing a collection of items. The term item refers to a row that is stored on a page.
- Further topics to discuss:
- limitations
- TOAST (The Oversized-Attribute Storage Technique)
- Free Space map (vacuum)
- Visibility map

- A page structure looks like the following:



Creating a web-service using Restful Api

- It stands for Representational State Transfer and it can mean a lot of things, but usually when you are talking about APIs and applications, you are talking about REST as a way to do web services or get programs to talk over the web.
- REST(architectural design) is basically a way of communicating between systems and does much of what SOAP RPC was designed to do, but while SOAP(protocol) generally makes a connection, authenticates and then does stuff over that connection, REST works pretty much the same way that that the web works. You have a URL and when you request that URL you get something back. This is where things start getting confusing because people describe the web as a the largest REST application and while this is technically correct it doesn't really help explain what it is.
- In a nutshell, REST allows you to get two applications talking over the Internet using tools that are similar to what a web browser uses. This is much simpler than SOAP and a lot of what REST does is says, "Hey, things don't have to be so complex."
- Worth reading:
- How I Explained REST to My Wife (now available here)
- Architectural Styles and the Design of Network-based Software Architectures
- shareeditflag

Using java to create a Restful web-service

- Creating Get request to fetch data from the database :
- @GET
- @Path("/getAllCustomer")
- @Produces(MediaType.APPLICATION_JSON)
- //   @GET
- //   @Produces("text/plain")
- public List<Customer> getAllCustomer() throws ClassNotFoundException, SQLException {
- Class.forName("org.postgresql.Driver");
- Connection c = DriverManager
- .getConnection("jdbc:postgresql://localhost:5432/mydb",
- "postgres", "123456");
- c.setAutoCommit(false);
- System.out.println("Opened database successfully");
- Statement stm;
- stm = c.createStatement();

```
String sql = "Select * From COMPANY";
        ResultSet rst;
        rst = stm.executeQuery(sql);
        ArrayList<Customer> customerList = new ArrayList<>();
        while (rst.next()) {
            Customer customer = new Customer(rst.getInt("id"), rst.getString("name"), rst.getString("address"), rst.getDouble("salary"));
            customerList.add(customer);
        }
        System.out.println("Opened database successfully");
        rst.close();
        stm.close();
        c.close();

        //GenericEntity<List<Customer>> list = new GenericEntity<List<Customer>>(customerList) {};

        return customerList;
    }
```

Description

- Define annotation for a restful api as : GET, POST, PRODUCE.
- Creating a list to get all the data needed from the database and customer interface.
- Connect to a database driver, then execute a sql query on it to get all the data needed from the database.
- Then assign all the data to the list we created before.
- Then close the session.

Restful web-service to post to DB

- Specify the annotations required to post to the DB:
- @POST
- @Path("/insertCustomer")
- @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
- @Produces(MediaType.APPLICATION_JSON)
- Then, using @FormParam annotation to get data directly from the our form when we insert the data inside the DB. Also, to map data that we get from the form to the right parameters.
- Then we insert sql query to post to the database.
- Using a preparedStatement to compile data directly to the DB as it keeps the transaction open until we finished then it update the DB.
- Finally we close the DB connection and close the preparedStatement(transaction).

CORS(cross origin resource sharing)

- Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin. A web application makes a **cross-origin HTTP request** when it requests a resource that has a different origin (domain, protocol, and port) than its own origin.

- We use CORS to specify some headers to the browser as for security resone it won’t let you to open connection to a database from the browser directly.

Angular

- Using Angular to create a frontend application to connect with our web-service.
- Angular is a SAP (single page application), so we have a single page inside our application (index.html). What angular does is inject pieces of code in that page when needed (will discuss further what needed means in angular using ‘routing’)to appear to the user as it have many pages.
- Angular application is working on node.js(is a backend server side dependent on google v8 engine for javascript (to make code working in the browser works in your machine)) which we will need to install also to use it’s package manager tool(npm) to handle all the directive and libraries we need to install.
- We also use TypeScript is a superset of JavaScript which primarily provides optional static typing, classes and interfaces. One of the big benefits is to enable IDEs to provide a richer environment for spotting common errors *as you type the code*.

- Angular consists of components , each component handle the view for a specific page that we need the user to see when it make a specific actions ($event). It also contain a class with it’s properties and methods and finally a meta-data which defines the class as an angular component and lays out the view managed by the component.
- We can bind data in angular using several methods as interpolation(one way binding from the class to the template(html page related to the component).).
- Directive: custom html element or attribute used to power up and extend our html. it has two custom and built-in.
- Custom as to view our template in an html page as <pm-product></pm-product> .
- Built-in as ngFor directive. To loop over all our list when we fetch data from the DB.

- Property binding as [] , event binding () so we use [()] to get property from our component class and reflect user action inside the component(event)
- Also we have pipes which we can use to make changes on the data before we display it. As to put a dollar sign in-front of money.
- Retrieving data using http: we must make an observable(is a behavioral design pattern as façade design pattern on jquery) to wait for any change happen on the service we call and then subscribe to that observable and listen for any change that happen to reflect it on the component.
- We can think of an observable as an array whose items arrives async over time.

- We must declare our component before we use it in our app module to it available to all the application.
- We use constructor to inject dependencies as httpclient dependency which provide interface some how that maps between our web-service and get and post methods in our observables.
- Then we need to import our httpclient from angular common http package. Then we add our injected service to our imports array in the appmodule.
- Routing is how we change views depending on a specific action from the user (when the action happen we specify which component will render and where it will render)