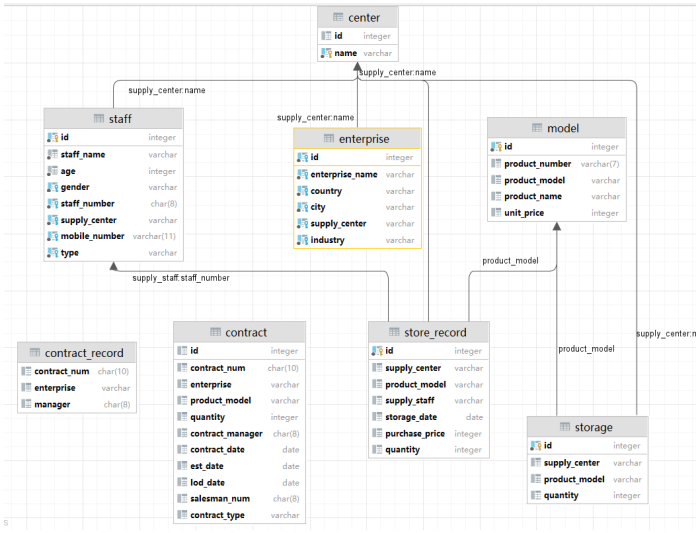
# CS307 Project 2 Report

安钧文 12012109 张海涵 12012222 lab session：7-8 Tue.
contributions: 50% 50%
**Task distribution:**
安钧文: Basic: 1), 5), 10)~13), Advanced: flexible query, Springboot, Vue.js, GUI design, index, trigger
张海涵: Basic: 2)~9), Advanced: Bill module, null arguments query, contract_type changed with time, Springboot, Vue.js, connection pool, user privilege, view

## Database structure

The fundamental table is `center`, `model`, `staff`, `enterprise`, `contract` and `store_record`. Considering that we need to retain contract_number when there is no order in contract. If only use table `contract`, the operation is complex. So we design table `contract_record` to store the common information of `contract`: `contract_num`, `enterprise`, `manager`.The items in stockIn can purchase one `model` many times, we need to know the total quantity of each `model`. So we design table `storage` to store the quantity of model in different `supply_center`.
Show the visialization:




## Basic Requirements

### 1) APIs for manipulating the original data:

`center`

```
Center selectCenter(String str); //str can be either id or center name, returns search result
void addCenter(int id,String name); //insert a row
void deleteCenter(String str); //delete a row by center name
```

`enterprise`

```
Enterprise selectEnterprise(String str); //str can be either id or enterprise
name, returns search result
void addEnterprise(int id, String name,String country,String city,String
supp,String industry); //insert a row
void deleteEnterprise(String str); //delete by enterprise name, str is name
void updateEnterprise(String industry,String name); //update enterprise's
industry by its name
```

`model`

```
Model selectModel(String str); //str can be id, product_model, product_number,
returns search result
void addModel(int id,String pro_num,String pro_model,String pro_name,int
unit_price); //insert a row
void deleteModel(String str); //delete by product_model
void updateModel(int u_price,String model); //update unit price by product_model
```

`staff`

```
Staff selectStaff(String str); //str is staff's number, returns search result
void addStaff(int id,String s_name,int age,String gender,String staff_num,String
center,String mobile,String type); //insert a row
void deleteStaff(String str); //delete by staff number
void updateStaff(String type,String staff_num); //update staff type by his/her
staff number
```

## 2) stockIn:

Reads a file and stock in inventory. `store_record` records each time a product is imported, `storage` records the total amount of each product in each supply center. Also filters illegal data.

```
void stockIn(String file) //The file should be in .csv format
```

## 3) placeOrder:

Reads a file and place orders. `contract_record` records each contract's contract number, enterprise and manager, `contract` records each contract and it's orders. Also filters illegal data.

```
void placeOrder(String file) //The file should be in .tsv format
```

## 4) updateOrder:

Reads a file and updates orders according to the contract number, product model and salesman number. Filters illegal data.

```
void updateOrder(String file) //The file should be in .tsv format
```

## 5) deleteOrder:

Reads a file and delete orders, if a contract has no order after deletion, the contract will not be removed. Filters illegal data.

```
void deleteOrder(String filename) //The file should be in .tsv format
```

## 6) APIs for select:(Q6 to Q13)

```
List<Staff> getAllStaffCount();  //Q6
List<Contract> getContractCount(); //Q7
List<Contract> getOrderCount(); //Q8
List<Storage> getNeverSoldProductCount(); //Q9
List<Model> getFavoriteProductModel(); //Q10
List<Center> getAvgCenter(); //Q11
List<Generic> getProductByNumber(String product_number); //Q12 Parameters can
either be entered one by one (on backend server or in script) or enter multiple
parameters separated by comma (only in script)
List<Generic> getContractInfoByNym(String contract_number); //Q13 arameters can
either be entered one by one (on backend server or in script) or enter multiple
parameters separated by comma (only in script)
```

select statement:

```
select type, count(*) as cnt from staff group by type; --Q6
select count(*) as cnt from contract_record; --Q7
select count(*) from contract;--Q8
select count(distinct product_model) as cnt from storage where product_model not
in (select distinct product_model from contract);--Q9
with q as (select product_model, sum(quantity) as cnt from contract group by
product_model) select * from q where cnt = (select max(cnt) from q);--Q10
select supply_center, round(((100.0 * sum) / (100.0 * cnt)), 1) as avg
from (select distinct supply_center,sum(quantity) over (partition by
supply_center) as sum, count(*) over (partition by supply_center) as cnt
from storage) s order by supply_center;--Q11
```

# Advanced

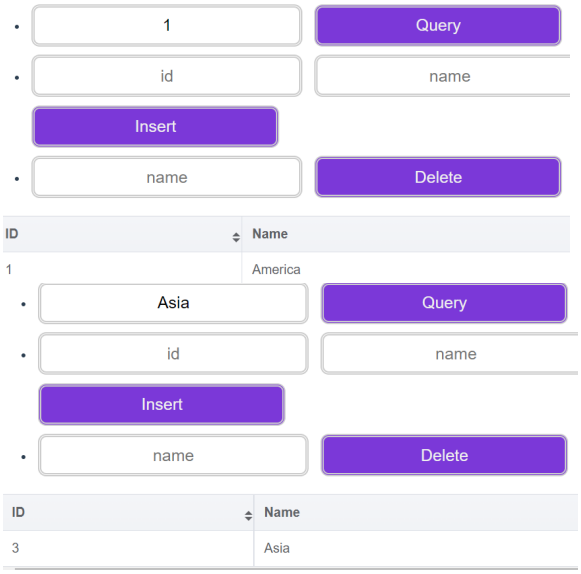
## 1) More APIs

### Query with flexiable arguments:

- We designed an select order API that receive three arguments which can be null or not. We only use the not null arguments as filters. If three arguments are all null, the query result is whole table. The interface is:

  ```
  public List<Contract> queryOrder(String contract_num, String model, String
  enterprise);
  ```

- We designed single-argument query APIs that can receive multiple types of argument. And then distinguish the actual type as filter. For example, in table `center`, if input is a digit number, it will be considered as `id` filter, if the input is a string, it will be treated as `name` filter. Show the sample result:




### Bill module:

Noticed that both table `contract` and table `store_record` have column `price` and `quantity`. So we design 4 **views** with different standard, and use the sql format `select * from view_name` to query the bill.

```
public List<Price> getOrderCostGroupByEnterprise();//the product_model cost in
placeOrder grouped by enterprise
public List<Price> getOrderCostGroupByModel();//the product_model cost in
placeOrder grouped by model
public List<Price> getStockCostByCenter();//the product_model cost in stockIn
grouped by supply_center
public List<Price> getStockCostByModel();//the product_model cost in stockIn
grouped by model
```

The details of view shows in 5) User privilege, Trigger, Index, View.

### Mechanism to change order status according to time:

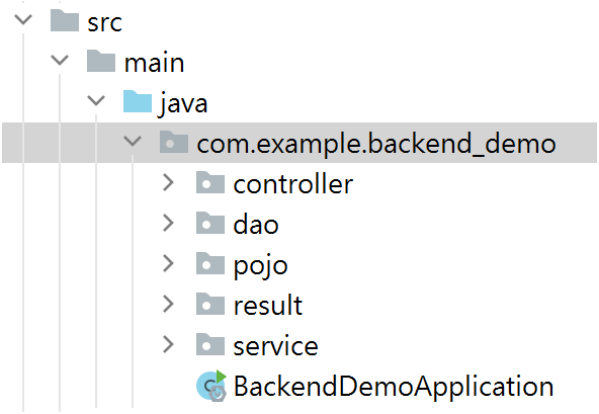
We design a new contract_type "Unfinished" in table `contract`, definited as the `lod_date` is null.When we insert "Unfinished" items, the `storage` won't update.
When we update this order and give `lod_date` a value, the "Unfinished" changed to "Finished" and the `storage` updated.
We also define an API to check the invalid items with `lod_date` not null and `contract_type` "Finished", which caused by directly change the value in datebase. The interface is:

```
public void updateUnfinished(String file);//update through file
public void updateType();//check table and update
```

## 2) Backend server

We used **Springboot** as backend framework, **Vue.js** as frontend framework.

**Backend structure:** (`result` was used as login verification package, but is not implemented in the final design)




- `pojo` contains objects that map to tables and columnns in the database.
- `dao` contains implementations of database manipulation intefaces, such as select and insert etc.
- `service` is used to encapsulate methods in `dao`, and send encapuslated methods to `controller`.
- `controller` is used to communicate with frontend using post and get requests.
- The server runs on localhost:8443

We used Springboot's embed Tomcat to act as the backend server.
Most APIs are implemented into backend, however, a few APIs are only available as scripts, they will also be in the attachment.

**Frontend structure**

- We use Axios to send post and get requests to backend.
- An 3rd-party template `Vue-good-table` is used to display the data.
- The application starts on localhost:8080.

## 3) Connection pool

We rewrite the connection pool demo, modify the arguments and write the test query.

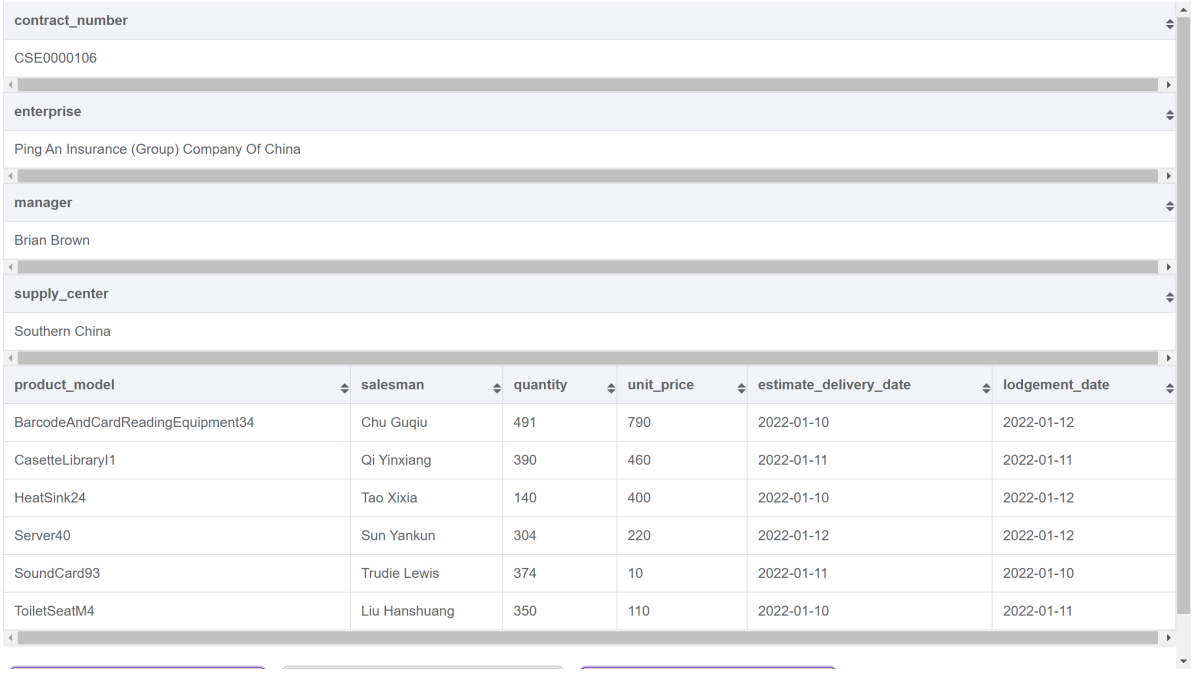
```
------Thread 13 visiting DB!------
------Thread 12 visiting DB!------
--- Active:2    Available:1     Max:15 ---
--- Active:2    Available:1     Max:15 ---
------Thread 18 visiting DB!------
------Thread 19 visiting DB!------
------Thread 20 visiting DB!------
------Thread 21 visiting DB!------
```

We also try Druid, but failed after configuration because can not login to monitor page.

## 4) Beautiful GUI

### Main page


### Query page



| contract_number |
| --- |
| CSE0000106 |

| enterprise |
| --- |
| Ping An Insurance (Group) Company Of China |

| manager |
| --- |
| Brian Brown |

| supply_center |
| --- |
| Southern China |

| product_model | salesman | quantity | unit_price | estimate_delivery_date | lodgement_date |
| --- | --- | --- | --- | --- | --- |
| BarcodeAndCardReadingEquipment34 | Chu Guqiu | 491 | 790 | 2022-01-10 | 2022-01-12 |
| CasetteLibraryI1 | Qi Yinxiang | 390 | 460 | 2022-01-11 | 2022-01-11 |
| HeatSink24 | Tao Xixia | 140 | 400 | 2022-01-10 | 2022-01-12 |
| Server40 | Sun Yankun | 304 | 220 | 2022-01-12 | 2022-01-12 |
| SoundCard93 | Trudie Lewis | 374 | 10 | 2022-01-11 | 2022-01-10 |
| ToiletSeatM4 | Liu Hanshuang | 350 | 110 | 2022-01-10 | 2022-01-11 |

# 5) User privilege, Trigger, Index, View

### User privilege

We design 5 different users: `checker`, `visitor`, `people_manager`, `enterprise` and `model_manager`. The fundamental privilege is able to `select` on origin table: `center`, `enterprise`, `staff` and `model`.

User checker is the super user of database, it can change any table, manage other role. Password: 123456
User visitor only has the fundamental privilege. Password: 654321
User people_manager can also update, delete, insert on table `staff`, can select on table `contract_record` and `contract`. Password: 114514
User enterprise can also update, delete, insert on table `contract_record` and `contract`, select on on table `contract_record` and `contract`. Password: 1919

User model_manager can also update, delete, insert on table `model`, `storage` and `store_record`. Password: 810

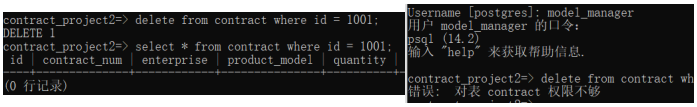
Some test cases:

```
select * from contract;--1
insert into contract (id, contract_num, enterprise, product_model, quantity,
contract_manager,contract_date,est_date,lod_date, salesman_num, contract_type)
values (1001,'CSE0000500','Alibaba','DatabaseSoftwareO6',1,'11711129','2022-05-
22','2022-05-26',null,'11310409','Unfinished'); --2
delete from contract where id = 1001;--3
```

run 1 on `visitor` and `people_manager`:


run 2 on `people_manager` and `enterprise`:


run 3 on `enterprise` and `people_manager`:




The test result meet expectation.

**Index**

A number of indexes are added to frequently-used columns to improve efficiency.

```
create index on contract(contract_num);
create index on contract(product_model);
create index on contract(salesman_num);
create index on store_record(product_model);
create index on contract_record(contract_num);
-- full code in attachment
```

**Trigger**

Triggers that detects whether illegal data will be inserted into original data tables `center, model, enterprise, staff` (Duplicate primary key/Duplicate unique column/Foriegn key constraint violation)

```
create or replace function model_check() returns trigger
as
$$
declare
    ex_id    int;
```

```
    cur_id   int;
    ex_name  varchar;
    cur_name varchar;
begin
    cur_name := new.product_model;
    cur_id := new.id;
    select product_model into ex_name from model where product_model = cur_name;
    select id into ex_id from model where id = cur_id;
    if cur_name = ex_name or cur_id = ex_id then
        return null;
    end if;
    return new;
end;
$$ language plpgsql;
-- full code in attachment
```

**View**

Use view in bill module can check the bill in current time,  encapsulate the detail information in table `contract` and `stroe_record`, simplify and clearify the query sql statement.

```
--bill1
create or replace view orderCostGroupByEnterprise as
select enterprise, sum(total_price)
from (select enterprise,quantity * model.unit_price as total_price
         from contract c
                  join model on model.product_model = c.product_model) sub
group by enterprise;
--bill2
create or replace view orderCostGroupByModel as
select product_model, sum(total_price)
from (select c.product_model, quantity * model.unit_price as total_price
         from contract c
                  join model on model.product_model = c.product_model) sub
group by product_model;
--bill3
create or replace view  storageCostGroupByCenter as
select supply_center, sum(purchase_price * quantity )as total_price
from store_record group by supply_center;
--bill4
create or replace view storageCostGroupByModel as
select product_model, sum( purchase_price * quantity )as total_price
from store_record group by product_model;
```