Projet de SQL: Game of Trones

Jeremy Wagemans Philippe Dragomir

7 décembre 2015

Table des matières

Table des matières **1**

1 Introduction **2**

2 Présentation de la solution **3**
2.1 Clarification de l'interprétation de l'énoncé 3
2.2 Structure de la base de données 4

3 Base de données **6**
3.1 Script d'installation 6
3.2 Script d'insertion de données valides 20
3.3 Script d'insertion de données invalides 21

4 Application java **23**
4.1 App.java 23
4.2 ClientsApp.java 24
4.3 HousesApp.java 35
4.4 Utils.java 46
4.5 PasswordHash.java 50

5 Conclusion **53**

Chapitre 1

Introduction

Afin d'appliquer les méthodologies et les notions enseignées au cours I2040 - DB : Langage de Requêtes et de Programmation, nous avions pour objectif de réaliser, par groupe de deux, une application de gestion de devis.

En effet, l'objectif du projet était d'informatiser le processus de soumission et d'acceptation des devis pour les maisons de WCrhoo et de leurs clients. Ils nous ont donc demandés de mettre en place une plateforme permettant de regrouper les demandes de devis des clients et permettant aux différentes maisons de pouvoir soumettre des devis à ces demandes.

La solution qui vous est présentée ci-après est celle du groupe composé de Dragomir Philippe et de Jeremy Wagemans.

Au terme du projet, nous avons donc dû délivrer une solution en parfaite adéquation avec les demandes des maisons et des clients répondant à des critères de qualité stricts. Ce rapport permet donc d'exposer de manière précise son fonctionnement. Il est structuré comme suit :

Dans un premier temps, nous développerons l'interprétation que nous avons faite des demandes des maisons ainsi que de celles des clients.

Ensuite, nous vous exposerons la structure de la base de données crée pour répondre aux demandes énoncés.

Enfin, nous proposerons le code source commenté des deux applications développées.

Chapitre 2

Présentation de la solution

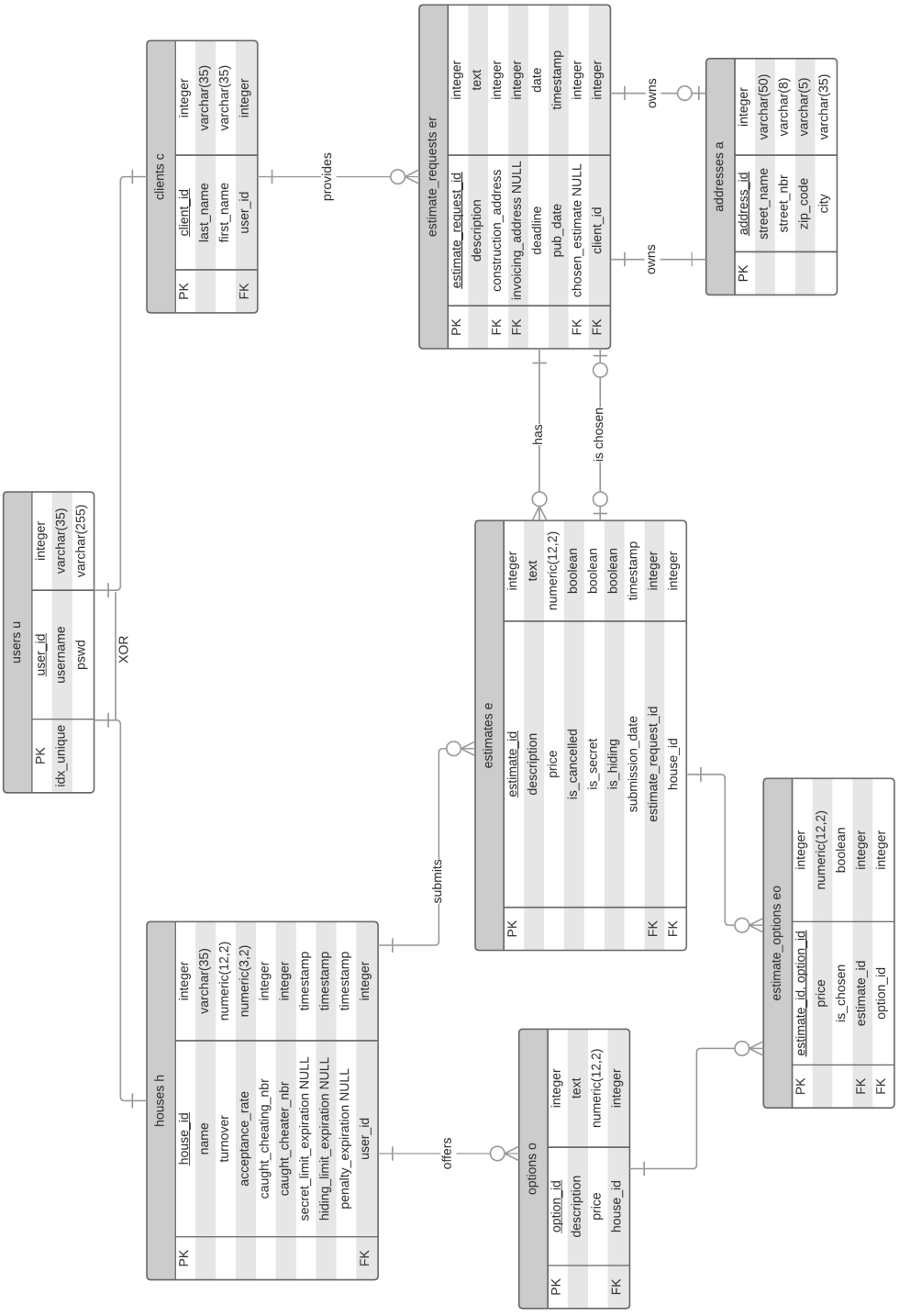
2.1 Clarification de l'interprétation de l'énoncé

Certains points du cahier des charges ont été approfondis lors la réalisation du projet, il est donc nécessaire de clarifier les détails suivants :

- Les options sont réutilisables. Une maison peut à tout moment créer et modifier des options. Elle peut joindre celles-ci à n'importe quelle offre.
- Toutes les statistiques demandées sont précalculées mis à part le nombre de devis en cours. En effet, vu que le système de gestion de bases de données ne sait pas gérer la notion d'expiration temporelle de manière systématique, il est nécessaire de recalculer ce nombre à chaque fois qu'on en a besoin.
- Lorsqu'une maison est dénoncée, le devis dénonciateur est transformé en un devis normal, c'est-à-dire non masquant et publique.

2.2 Structure de la base de données

Les données de l'application sont sauvegardées dans une base de données, structurée comme suit :



Il est essentiel de mentionner certaines spécificités du schéma :

- Un utilisateur représente un client ou une maison autorisé à utiliser l'application et est authentifié grâce à un nom d'utilisateur unique et à un mot de passe.
- Le fait qu'un devis soit caché et/ou masquant est déterminé sur base des champs booléens is_secret (vrai lorsqu'un devis est caché) et is_hiding (vrai lorsqu'un devis est masquant) de la table estimates.
- Un devis est annulé lorsque celui-ci a été soumis dans un délais de 24 heures avant que la maison soumissionnaire ne soit démasquée pour tricherie. Dans ce cas, le champs booléen is_cancelled de ces devis dans la table estimates vaut vrai.
- Le choix d'un devis par le client s'exprime au sein de la table estimate_requests au travers du champs chosen_estimate. Celui-ci référencie donc le devis choisi. Si aucun devis n'a encore été accepté, ce champs est vide. Par conséquent, il n'est pas nécessaire de parcourir l'entièreté des devis pour savoir si un devis a été accepté pour une demande et lequel a été accepté.
- Le champs pub_date d'une demande de devis exprime sa date de publication. Une demande est donc expirée lorsque cette date est antérieure à 15 jours.
- Les options sont réutilisables. Etant donné que leur prix peut être modifié, le prix d'une option est lié à un devis (table estimate_options).
- Les statistiques pré-calculées sont enregistrées au sein de la table maison.

Chapitre 3

Base de données

3.1 Script d'installation

```
-- Supprimer toutes les données existantes
DROP SCHEMA IF EXISTS marche_halibaba CASCADE;

-- Schema
CREATE SCHEMA marche_halibaba;

-- Users
CREATE TABLE marche_halibaba.users (
  user_id SERIAL PRIMARY KEY,
  username VARCHAR(35) NOT NULL CHECK (username <> '') UNIQUE,
  pswd VARCHAR(255) NOT NULL CHECK (pswd <> '')
);

-- Clients
CREATE TABLE marche_halibaba.clients (
  client_id SERIAL PRIMARY KEY,
  last_name VARCHAR(35) NOT NULL CHECK (last_name <> ''),
  first_name VARCHAR(35) NOT NULL CHECK (first_name <> ''),
  user_id INTEGER NOT NULL
    REFERENCES marche_halibaba.users(user_id)
);

-- Addresses
CREATE TABLE marche_halibaba.addresses (
  address_id SERIAL PRIMARY KEY,
  street_name VARCHAR(50) NOT NULL CHECK (street_name <> ''),
  street_nbr VARCHAR(8) NOT NULL CHECK (street_nbr <> ''),
  zip_code VARCHAR(5) NOT NULL CHECK (zip_code ~ '^[0-9]+$'),
  city VARCHAR(35) NOT NULL CHECK (city <> '')
);

-- Estimate requests
CREATE TABLE marche_halibaba.estimate_requests (
  estimate_request_id SERIAL PRIMARY KEY,
  description TEXT NOT NULL CHECK (description <> ''),
  construction_address INTEGER NOT NULL
    REFERENCES marche_halibaba.addresses(address_id),
  invoicing_address INTEGER
    REFERENCES marche_halibaba.addresses(address_id),
  pub_date TIMESTAMP NOT NULL DEFAULT NOW(),
```

```
  deadline DATE NOT NULL CHECK (deadline > NOW()),
  chosen_estimate INTEGER,
  client_id INTEGER NOT NULL
    REFERENCES marche_halibaba.clients(client_id)
);

-- Houses
CREATE TABLE marche_halibaba.houses (
  house_id SERIAL PRIMARY KEY,
  name VARCHAR(35) NOT NULL CHECK (name <> ''),
  turnover NUMERIC(12,2) NOT NULL DEFAULT 0,
  acceptance_rate NUMERIC(3,2) NOT NULL DEFAULT 0,
  caught_cheating_nbr INTEGER NOT NULL DEFAULT 0,
  caught_cheater_nbr INTEGER NOT NULL DEFAULT 0,
  secret_limit_expiration TIMESTAMP NULL,
  hiding_limit_expiration TIMESTAMP NULL,
  penalty_expiration TIMESTAMP NULL,
  user_id INTEGER NOT NULL
    REFERENCES marche_halibaba.users(user_id)
);

-- Estimates
CREATE TABLE marche_halibaba.estimates (
  estimate_id SERIAL PRIMARY KEY,
  description TEXT NOT NULL CHECK (description <> ''),
  price NUMERIC(12,2) NOT NULL CHECK (price > 0),
  is_cancelled BOOLEAN NOT NULL DEFAULT FALSE,
  is_secret BOOLEAN NOT NULL DEFAULT FALSE,
  is_hiding BOOLEAN NOT NULL DEFAULT FALSE,
  submission_date TIMESTAMP NOT NULL DEFAULT NOW(),
  estimate_request_id INTEGER NOT NULL
    REFERENCES marche_halibaba.estimate_requests(estimate_request_id),
  house_id INTEGER NOT NULL
    REFERENCES marche_halibaba.houses(house_id)
);

ALTER TABLE marche_halibaba.estimate_requests
ADD CONSTRAINT chosen_estimate_fk FOREIGN KEY (chosen_estimate)
REFERENCES marche_halibaba.estimates(estimate_id)
ON DELETE CASCADE;

-- Options
CREATE TABLE marche_halibaba.options (
  option_id SERIAL PRIMARY KEY,
  description TEXT NOT NULL CHECK (description <> ''),
  price NUMERIC(12,2) NOT NULL CHECK (price > 0),
  house_id INTEGER NOT NULL
    REFERENCES marche_halibaba.houses(house_id)
);

-- Estimate options
CREATE TABLE marche_halibaba.estimate_options (
  price NUMERIC(12,2) NOT NULL CHECK (price > 0),
  is_chosen BOOLEAN NOT NULL DEFAULT FALSE,
  estimate_id INTEGER NOT NULL
    REFERENCES marche_halibaba.estimates(estimate_id),
  option_id INTEGER NOT NULL
    REFERENCES marche_halibaba.options(option_id),
  PRIMARY KEY(estimate_id, option_id)
);
```

```
-- Afficher les utilisateurs

DROP VIEW IF EXISTS marche_halibaba.signin_users;

CREATE VIEW marche_halibaba.signin_users AS
  SELECT u.username as "u_username", u.pswd as "u_pswd", c.client_id as "c_id",
    c.first_name as "c_first_name", c.last_name as "c_last_name",
      h.house_id as "h_id", h.name as "h_name"
  FROM marche_halibaba.users u
    LEFT OUTER JOIN marche_halibaba.clients c
      ON u.user_id = c.user_id
    LEFT OUTER JOIN marche_halibaba.houses h
      ON u.user_id = h.user_id;


-- Afficher les détails d'un devis

DROP VIEW IF EXISTS marche_halibaba.estimate_details;

CREATE VIEW marche_halibaba.estimate_details AS
  SELECT e.estimate_id as "e_id", e.description as "e_description",
    e.price as "e_price", e.is_cancelled as "e_is_cancelled",
    e.submission_date as "e_submission_date",
    h.house_id as "e_house_id", h.name as "e_house_name",
    o.option_id as "e_option_id", o.description as "e_option_description",
    eo.price as "e_option_price"
  FROM marche_halibaba.estimates e
    LEFT OUTER JOIN marche_halibaba.estimate_options eo
      ON e.estimate_id = eo.estimate_id
    LEFT OUTER JOIN marche_halibaba.options o
      ON eo.option_id = o.option_id,
    marche_halibaba.houses h
  WHERE e.house_id = h.house_id;


-- Afficher les demandes de devis
DROP VIEW IF EXISTS marche_halibaba.list_estimate_requests;

CREATE VIEW marche_halibaba.list_estimate_requests AS
  SELECT er.estimate_request_id AS "er_id",
    er.description AS "er_description",
    er.deadline AS "er_deadline",
    er.pub_date AS "er_pub_date",
    er.chosen_estimate AS "er_chosen_estimate",
    a.street_name AS "er_construction_id",
    a.zip_code AS "er_construction_zip",
    a.city AS "er_construction_city",
    a2.street_name AS "er_invoicing_street",
    a2.zip_code AS "er_invoicing_zip",
    a2.city AS "er_invoicing_city",
    c.client_id AS "c_id",
    c.last_name AS "c_last_name",
    c.first_name AS "c_first_name",
    AGE(er.pub_date + INTERVAL '15' day, NOW()) AS "remaining_days"
  FROM marche_halibaba.clients c, marche_halibaba.addresses a, marche_halibaba.
    estimate_requests er
    LEFT OUTER JOIN marche_halibaba.addresses a2 ON er.invoicing_address = a2.
      address_id
```

```
  WHERE a.address_id = er.construction_address
    AND c.client_id = er.client_id
  ORDER BY er.pub_date DESC;


-- Enregistrer un client

CREATE OR REPLACE FUNCTION marche_halibaba.signup_client(VARCHAR(35), VARCHAR(50)
    , VARCHAR(35), VARCHAR(35))
  RETURNS INTEGER AS $$
DECLARE
  arg_username ALIAS FOR $1;
  arg_pswd ALIAS FOR $2;
  arg_first_name ALIAS FOR $3;
  arg_last_name ALIAS FOR $4;
  new_user_id INTEGER;
  new_client_id INTEGER;
BEGIN
  INSERT INTO marche_halibaba.users(username, pswd)
    VALUES (arg_username, arg_pswd)
    RETURNING user_id INTO new_user_id;

  INSERT INTO marche_halibaba.clients(first_name, last_name, user_id)
    VALUES (arg_first_name, arg_last_name, new_user_id)
    RETURNING client_id INTO new_client_id;
  RETURN new_client_id;
END;
$$ LANGUAGE 'plpgsql';


-- Afficher les devis visibles par un client

DROP VIEW IF EXISTS marche_halibaba.clients_list_estimates;

CREATE VIEW marche_halibaba.clients_list_estimates AS
  SELECT view.estimate_id as "e_id", view.description as "e_description",
    view.price as "e_price",
    view.submission_date as "e_submission_date",
    view.estimate_request_id as "e_estimate_request_id",
    view.house_id as "e_house_id",
    view.name as "e_house_name"
  FROM (
    (
    SELECT e.estimate_id, e.description, e.price,
      e.submission_date, e.estimate_request_id, e.house_id, h.name
    FROM marche_halibaba.estimates e, marche_halibaba.estimate_requests er,
      marche_halibaba.houses h
    WHERE e.estimate_request_id = er.estimate_request_id AND
      e.house_id = h.house_id AND
      er.chosen_estimate IS NULL AND
      e.is_cancelled = FALSE AND
      NOT EXISTS(
        SELECT *
        FROM marche_halibaba.estimates e2
        WHERE e2.estimate_request_id = e.estimate_request_id AND
          e2.is_hiding = TRUE AND
          e2.is_cancelled = FALSE
      )
    )
    UNION
```

```
    (
      SELECT e.estimate_id, e.description, e.price,
        e.submission_date, e.estimate_request_id, e.house_id, h.name
      FROM marche_halibaba.estimates e, marche_halibaba.estimate_requests er,
        marche_halibaba.houses h
      WHERE e.estimate_request_id = er.estimate_request_id AND
        e.house_id = h.house_id AND
        er.chosen_estimate IS NULL AND
        e.is_cancelled = FALSE AND
        e.is_hiding = TRUE
    )
    UNION
    (
      SELECT e.estimate_id, e.description, e.price,
        e.submission_date, e.estimate_request_id, e.house_id, h.name
      FROM marche_halibaba.estimates e, marche_halibaba.estimate_requests er,
        marche_halibaba.houses h
      WHERE e.estimate_id = er.chosen_estimate AND
        e.house_id = h.house_id
    )) view
  ORDER BY view.submission_date DESC;


-- Soumettre une demande de devis

CREATE OR REPLACE FUNCTION marche_halibaba.submit_estimate_request(TEXT, DATE,
    INTEGER, VARCHAR(50), VARCHAR(8), VARCHAR(5), VARCHAR(35), VARCHAR(50),
    VARCHAR(8), VARCHAR(5), VARCHAR(35))
  RETURNS INTEGER AS $$
DECLARE
  arg_description ALIAS FOR $1;
  arg_deadline ALIAS FOR $2;
  arg_client ALIAS FOR $3;
  arg_cons_street_name ALIAS FOR $4;
  arg_cons_street_nbr ALIAS FOR $5;
  arg_cons_zip_code ALIAS FOR $6;
  arg_cons_city ALIAS FOR $7;
  arg_inv_street_name ALIAS FOR $8;
  arg_inv_street_nbr ALIAS FOR $9;
  arg_inv_zip_code ALIAS FOR $10;
  arg_inv_city ALIAS FOR $11;
  new_construction_address_id INTEGER;
  new_invoicing_address_id INTEGER;
  new_estimate_request_id INTEGER;
BEGIN
  INSERT INTO marche_halibaba.addresses(street_name, street_nbr, zip_code, city)
    VALUES (arg_cons_street_name, arg_cons_street_nbr, arg_cons_zip_code,
        arg_cons_city)
    RETURNING address_id INTO new_construction_address_id;

  new_invoicing_address_id := NULL;

  IF arg_inv_street_name IS NOT NULL AND
    arg_inv_street_nbr IS NOT NULL AND
    arg_inv_zip_code IS NOT NULL AND
    arg_inv_city IS NOT NULL THEN

    INSERT INTO marche_halibaba.addresses(street_name, street_nbr, zip_code, city
        )
```

```
      VALUES (arg_inv_street_name, arg_inv_street_nbr, arg_inv_zip_code,
          arg_inv_city)
      RETURNING address_id INTO new_invoicing_address_id;

  END IF;

  INSERT INTO marche_halibaba.estimate_requests(description, construction_address
      , invoicing_address, deadline, client_id)
    VALUES (arg_description, new_construction_address_id,
        new_invoicing_address_id, arg_deadline, arg_client)
    RETURNING estimate_request_id INTO new_estimate_request_id;

  RETURN new_estimate_request_id;
END;
$$ LANGUAGE 'plpgsql';


-- Accepter une demande de devis

CREATE OR REPLACE FUNCTION marche_halibaba.approve_estimate(INTEGER, INTEGER[],
    INTEGER)
  RETURNS INTEGER AS $$

DECLARE
  arg_estimate_id ALIAS FOR $1;
  arg_chosen_options ALIAS FOR $2;
  arg_client_id ALIAS FOR $3;
  var_er_id INTEGER;
  var_er_client_id INTEGER;
  var_option INTEGER;
BEGIN
  SELECT e.estimate_request_id, er.client_id
  INTO var_er_id, var_er_client_id
  FROM marche_halibaba.estimate_requests er, marche_halibaba.estimates e
  WHERE e.estimate_request_id = er.estimate_request_id AND
    e.estimate_id = arg_estimate_id;

  IF var_er_client_id <> arg_client_id THEN
    RAISE EXCEPTION 'Vous␣n␣êtes␣pas␣autorisé␣à␣accepter␣ce␣devis';
  END IF;

  UPDATE marche_halibaba.estimate_requests er
  SET chosen_estimate = arg_estimate_id
  WHERE estimate_request_id = var_er_id;

  IF arg_chosen_options IS NOT NULL THEN
    FOREACH var_option IN ARRAY arg_chosen_options
    LOOP
      UPDATE marche_halibaba.estimate_options
      SET is_chosen = TRUE
      WHERE option_id = var_option AND
        estimate_id = arg_estimate_id;
    END LOOP;
  END IF;

  RETURN 0;
END;
$$ LANGUAGE 'plpgsql';
```

```
-- Enregistrer une maison

CREATE OR REPLACE FUNCTION marche_halibaba.signup_house(VARCHAR(35), VARCHAR(50),
    VARCHAR(35))
  RETURNS INTEGER AS $$
DECLARE
  arg_username ALIAS FOR $1;
  arg_pswd ALIAS FOR $2;
  arg_name ALIAS FOR $3;
  new_user_id INTEGER;
  new_house_id INTEGER;
BEGIN
  INSERT INTO marche_halibaba.users(username, pswd)
    VALUES (arg_username, arg_pswd) RETURNING user_id INTO new_user_id;

  INSERT INTO marche_halibaba.houses(name, user_id)
    VALUES (arg_name, new_user_id) RETURNING house_id INTO new_house_id;
  RETURN new_house_id;
END;
$$ LANGUAGE 'plpgsql';


-- Modifier une option

CREATE OR REPLACE FUNCTION marche_halibaba.modify_option(TEXT, NUMERIC(12,2),
    INTEGER, INTEGER)
  RETURNS INTEGER AS $$

DECLARE
  arg_description ALIAS FOR $1;
  arg_price ALIAS FOR $2;
  arg_option_id ALIAS FOR $3;
  arg_house_id ALIAS FOR $4;
BEGIN
  UPDATE marche_halibaba.options
  SET description= arg_description, price= arg_price
  WHERE arg_option_id= option_id
    AND arg_house_id= house_id;
RETURN arg_option_id;
END;
$$ LANGUAGE 'plpgsql';


-- Ajouter une option

CREATE OR REPLACE FUNCTION marche_halibaba.add_option(TEXT, NUMERIC(12,2),
    INTEGER)
  RETURNS INTEGER AS $$

DECLARE
  arg_description ALIAS FOR $1;
  arg_price ALIAS FOR $2;
  arg_house_id ALIAS FOR $3;
  new_option_id INTEGER;
BEGIN
  INSERT INTO marche_halibaba.options(description, price, house_id)
  VALUES (arg_description, arg_price, arg_house_id) RETURNING option_id INTO
      new_option_id;
  RETURN new_option_id;
END;
```

```
$$ LANGUAGE 'plpgsql';


-- Afficher le nombre de soumissions en cours par maison

DROP VIEW IF EXISTS marche_halibaba.valid_estimates_nbr;

CREATE VIEW marche_halibaba.valid_estimates_nbr AS
  SELECT h.house_id as "h_id", h.name as "h_name",
    count(e_id) as "h_valid_estimates_nbr"
  FROM marche_halibaba.houses h
    LEFT OUTER JOIN (
        SELECT e.estimate_id as "e_id", e.house_id as "e_house_id"
        FROM marche_halibaba.estimates e,
          marche_halibaba.estimate_requests er
        WHERE e.estimate_request_id = er.estimate_request_id AND
          e.is_cancelled = FALSE AND
          er.pub_date + INTERVAL '15' day >= NOW() AND
          er.chosen_estimate IS NULL) e
      ON h.house_id = e.e_house_id
  GROUP BY h.house_id, h.name;


-- Soumettre un devis

CREATE OR REPLACE FUNCTION marche_halibaba.submit_estimate(TEXT, NUMERIC(12,2),
    BOOLEAN, BOOLEAN, INTEGER, INTEGER, INTEGER[])
  RETURNS INTEGER AS $$

DECLARE
  arg_description ALIAS FOR $1;
  arg_price ALIAS FOR $2;
  arg_is_secret ALIAS FOR $3;
  arg_is_hiding ALIAS FOR $4;
  arg_estimate_request_id ALIAS FOR $5;
  arg_house_id ALIAS FOR $6;
  arg_chosen_options ALIAS FOR $7;
  new_estimate_id INTEGER;
  option INTEGER;
  option_price NUMERIC(12,2);
BEGIN
  INSERT INTO marche_halibaba.estimates(description, price, is_secret, is_hiding,
      submission_date, estimate_request_id, house_id)
  VALUES (arg_description, arg_price, arg_is_secret, arg_is_hiding, NOW(),
      arg_estimate_request_id, arg_house_id)
    RETURNING estimate_id INTO new_estimate_id;

  IF arg_chosen_options IS NOT NULL THEN
    FOREACH option IN ARRAY arg_chosen_options
    LOOP
      SELECT o.price INTO option_price
      FROM marche_halibaba.options o
      WHERE o.option_id = option AND
        o.house_id = arg_house_id;

      IF option_price IS NULL THEN
        RAISE EXCEPTION 'Cette␣option␣n␣appartient␣pas␣à␣la␣maison␣
            soumissionnaire.';
      END IF;
```

```
      INSERT INTO marche_halibaba.estimate_options(price, is_chosen, estimate_id,
          option_id)
      VALUES (option_price, FALSE, new_estimate_id, option);
    END LOOP;
  END IF;

  RETURN new_estimate_id;
END;
$$ LANGUAGE 'plpgsql';


-- Afficher les devis en cours de soumission par les maisons
-- Exemple d'exécution:
-- SELECT *
-- FROM marche_halibaba.valid_estimates_list
-- WHERE er_estimate_request_id = ? AND
--  (e_is_secret= FALSE OR (e_is_secret = TRUE AND e_house_id= ?));

DROP VIEW IF EXISTS marche_halibaba.valid_estimates_list;

CREATE VIEW marche_halibaba.valid_estimates_list AS
  SELECT e.estimate_id AS "e_estimate_id",
         e.description AS "e_description",
         e.price AS "e_price",
         e.house_id AS "e_house_id",
         e.submission_date AS "e_submission_date",
         e.is_secret AS "e_is_secret",
         er.estimate_request_id AS "er_estimate_request_id",
         er.deadline AS "er_deadline",
         er.description AS "er_description",
         h.name AS "h_name"
  FROM marche_halibaba.estimates e,
    marche_halibaba.estimate_requests er,
    marche_halibaba.houses h
  WHERE e.estimate_request_id= er.estimate_request_id
    AND e.house_id = h.house_id
    AND er.pub_date + INTERVAL '15' day > NOW()
    AND e.is_cancelled = FALSE
    AND er.chosen_estimate IS NULL
  ORDER BY e.pub_date DESC;


-- Trigger sur l'insertion de devis

CREATE OR REPLACE FUNCTION marche_halibaba.trigger_estimate_insert()
  RETURNS TRIGGER AS $$

DECLARE
  new_estimate_request_id INTEGER;
  caught_cheating_house_id INTEGER;
  house_times_record RECORD;

BEGIN

  SELECT h.penalty_expiration AS penalty_expiration,
    h.secret_limit_expiration AS secret_limit_expiration,
    h.hiding_limit_expiration AS hiding_limit_expiration
  INTO house_times_record
  FROM marche_halibaba.houses h
  WHERE h.house_id= NEW.house_id;
```

```
SELECT h.house_id
  INTO caught_cheating_house_id
FROM marche_halibaba.estimates e, marche_halibaba.houses h
WHERE e.estimate_request_id= NEW.estimate_request_id
  AND e.house_id= h.house_id
  AND e.is_hiding= TRUE AND e.is_cancelled= FALSE;

IF house_times_record.penalty_expiration > NOW()
THEN
  RAISE EXCEPTION 'Vous êtes interdit de devis pour encore % heures.', age(
      house_times_record.penalty_expiration, NOW());
END IF;

IF EXISTS( --If the estimate_request is expired, we raise a exception;
SELECT *
FROM marche_halibaba.estimate_requests er
WHERE er.estimate_request_id = NEW.estimate_request_id AND
  (er.pub_date + INTERVAL '15' day < NOW() OR er.chosen_estimate IS NOT NULL)
) THEN
  RAISE EXCEPTION 'Cette demande de devis est expirée/un devis a déjà été
      accepté pour cette demande.';
END IF;

IF NEW.is_hiding= TRUE
THEN
  IF house_times_record.hiding_limit_expiration > NOW() --On vérifie que l'on
      peut soumettre un devis hiding actuellement
  THEN
    RAISE EXCEPTION 'Vous ne pouvez pas poster de devis masquant pour encore %.
        ',age( house_times_record.hiding_limit_expiration, NOW()) ;
  ELSEIF caught_cheating_house_id IS NOT NULL --S'il y a déjà un devis masquant
      pour cette estimate_request
  THEN
    UPDATE marche_halibaba.houses
    SET penalty_expiration = NOW() + INTERVAL '1' day,
      caught_cheating_nbr = caught_cheating_nbr+1
    WHERE  house_id = caught_cheating_house_id;

    UPDATE marche_halibaba.houses
    SET caught_cheater_nbr = caught_cheater_nbr+1
    WHERE house_id= NEW.house_id;

    UPDATE marche_halibaba.estimates
    SET is_cancelled= TRUE
    WHERE house_id= caught_cheating_house_id
      AND estimate_request_id= NEW.estimate_request_id
      AND is_hiding= TRUE;

    UPDATE marche_halibaba.estimates
    SET is_cancelled= TRUE
    WHERE house_id= caught_cheating_house_id
      AND submission_date >= NOW() - INTERVAL '1' day;

    NEW.is_hiding:=FALSE;
    NEW.is_secret:=FALSE;

  ELSE
    UPDATE marche_halibaba.houses
    SET hiding_limit_expiration= NOW()+ INTERVAL '7' day
```

```
      WHERE house_id= NEW.house_id;
    END IF;
  END IF;

  IF NEW.is_secret= TRUE
  THEN
    IF house_times_record.secret_limit_expiration > NOW()
    THEN
      RAISE EXCEPTION 'Vous ne pouvez pas poster de devis secret pour encore %
          heures.',age( house_times_record.secret_limit_expiration, NOW()) ;
    ELSE
      UPDATE marche_halibaba.houses
      SET secret_limit_expiration= NOW()+ INTERVAL '1' day
      WHERE house_id= NEW.house_id;
    END IF;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_estimate_insert
BEFORE INSERT ON marche_halibaba.estimates
FOR EACH ROW
EXECUTE PROCEDURE marche_halibaba.trigger_estimate_insert();


-- Trigger sur l'insertion de demande de devis

CREATE OR REPLACE FUNCTION marche_halibaba.trigger_estimate_requests_update()
  RETURNS TRIGGER AS $$

DECLARE
  var_estimate_details RECORD;
  var_acceptance_rate NUMERIC(3,2);
BEGIN
  SELECT e.estimate_request_id as "estimate_request_id",
    e.is_cancelled as "is_cancelled", e.price as "price",
    e.house_id as "house_id"
  INTO var_estimate_details
  FROM marche_halibaba.estimates e
  WHERE e.estimate_id = NEW.chosen_estimate;

  -- An exception is raised if a estimate has already been approved for this
      estimate request
  IF OLD.chosen_estimate IS NOT NULL THEN
    RAISE EXCEPTION 'Un devis a déjà été approuvé pour cette demande.';
  END IF;

  -- An exception is raised because the estimate has been cancelled
  IF var_estimate_details.is_cancelled THEN
    RAISE EXCEPTION 'Ce devis n est plus valide. Il a été annulé.';
  END IF;

  -- An exception is raised because the estimate request has expired
  IF (OLD.pub_date + INTERVAL '15' day) < NOW() THEN
    RAISE EXCEPTION 'Cette demande de devis est expirée.';
  END IF;

  -- Updates house statistics
```

```
  SELECT ((
    SELECT count(estimate_id)
    FROM marche_halibaba.estimates e, marche_halibaba.estimate_requests er
    WHERE e.estimate_id = er.chosen_estimate AND
      e.house_id = var_estimate_details.house_id)::numeric(16,2)/(
    SELECT count(estimate_id)
    FROM marche_halibaba.estimates e
    WHERE e.house_id = var_estimate_details.house_id)::numeric(16,2))::numeric
      (3,2)
  INTO var_acceptance_rate;

  UPDATE marche_halibaba.houses
  SET turnover = turnover + var_estimate_details.price,
    acceptance_rate = var_acceptance_rate
  WHERE house_id = var_estimate_details.house_id;

  RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_estimate_requests_update
AFTER UPDATE OF chosen_estimate ON marche_halibaba.estimate_requests
FOR EACH ROW
EXECUTE PROCEDURE marche_halibaba.trigger_estimate_requests_update();


-- Trigger sur l'acceptation d'une option

CREATE OR REPLACE FUNCTION marche_halibaba.trigger_estimate_options_update()
  RETURNS TRIGGER AS $$

DECLARE
  house_to_update INTEGER;
  old_turnover NUMERIC(12,2);

BEGIN
  SELECT h.house_id, h.turnover
  INTO house_to_update, old_turnover
  FROM marche_halibaba.estimate_options eo, marche_halibaba.options o,
      marche_halibaba.houses h
  WHERE eo.option_id = o.option_id AND
    o.house_id = h.house_id AND
    eo.estimate_id = OLD.estimate_id AND
    eo.option_id = OLD.option_id;

  UPDATE marche_halibaba.houses
  SET turnover = old_turnover + OLD.price
  WHERE house_id = house_to_update;

  RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_estimate_options_update
AFTER UPDATE on marche_halibaba.estimate_options
FOR EACH ROW
WHEN (OLD.is_chosen IS DISTINCT FROM NEW.is_chosen)
EXECUTE PROCEDURE marche_halibaba.trigger_estimate_options_update();
```

```
-- Utilisateurs

-- Création de l'utilisateur pour l'interface client
DROP USER IF EXISTS app_clients;

CREATE USER app_clients
ENCRYPTED PASSWORD '2S5jn12JndG68hT';

GRANT CONNECT
ON DATABASE projet
TO app_clients;

GRANT USAGE
ON SCHEMA marche_halibaba
TO app_clients;

GRANT SELECT
ON marche_halibaba.clients_list_estimates,
  marche_halibaba.estimate_details,
  marche_halibaba.list_estimate_requests,
  marche_halibaba.signin_users,
  marche_halibaba.houses,
  marche_halibaba.estimates,
  marche_halibaba.options
TO app_clients;

GRANT SELECT, INSERT
ON marche_halibaba.users,
  marche_halibaba.clients,
  marche_halibaba.estimate_requests,
  marche_halibaba.addresses
TO app_clients;

GRANT SELECT, UPDATE, TRIGGER
ON marche_halibaba.estimate_requests,
  marche_halibaba.estimate_options,
  marche_halibaba.houses,
TO app_clients;

GRANT EXECUTE
ON FUNCTION marche_halibaba.approve_estimate(INTEGER, INTEGER[], INTEGER),
  marche_halibaba.signup_client(VARCHAR(35), VARCHAR(50), VARCHAR(35), VARCHAR
      (35)),
  marche_halibaba.submit_estimate_request(TEXT, DATE, INTEGER, VARCHAR(50),
    VARCHAR(8), VARCHAR(5), VARCHAR(35), VARCHAR(50), VARCHAR(8), VARCHAR(5),
        VARCHAR(35)),
  marche_halibaba.trigger_estimate_requests_update(),
  marche_halibaba.trigger_estimate_options_update()
TO app_clients;

GRANT ALL PRIVILEGES
ON ALL SEQUENCES IN SCHEMA marche_halibaba
TO app_clients;

-- Création de l'utilisateur pour l'interface maison
DROP USER IF EXISTS app_houses;

CREATE USER app_houses
ENCRYPTED PASSWORD '2S5jn12JndG68hT';
```

```
GRANT CONNECT
ON DATABASE projet
TO app_houses;

GRANT USAGE
ON SCHEMA marche_halibaba
TO app_houses;

GRANT SELECT
ON marche_halibaba.signin_users,
   marche_halibaba.list_estimate_requests,
   marche_halibaba.valid_estimates_list,
   marche_halibaba.houses,
   marche_halibaba.options,
   marche_halibaba.valid_estimates_nbr
TO app_houses;

GRANT SELECT, INSERT
ON marche_halibaba.users,
   marche_halibaba.houses,
   marche_halibaba.estimate_options,
   marche_halibaba.estimates,
   marche_halibaba.options
TO app_houses;

GRANT SELECT, UPDATE, TRIGGER
ON marche_halibaba.estimates,
   marche_halibaba.estimate_options,
   marche_halibaba.estimate_requests,
   marche_halibaba.houses,
   marche_halibaba.options
TO app_houses;

GRANT EXECUTE
ON FUNCTION
marche_halibaba.signup_house(VARCHAR(35), VARCHAR(50), VARCHAR(35)),
marche_halibaba.submit_estimate(TEXT, NUMERIC(12,2), BOOLEAN, BOOLEAN, INTEGER,
    INTEGER, INTEGER[]),
marche_halibaba.add_option(TEXT, NUMERIC(12,2), INTEGER),
marche_halibaba.modify_option(TEXT, NUMERIC(12,2), INTEGER, INTEGER),
marche_halibaba.trigger_estimate_insert()
TO app_houses;

GRANT ALL PRIVILEGES
ON ALL SEQUENCES IN SCHEMA marche_halibaba
TO app_houses;
```

3.2 Script d'insertion de données valides

```
-- Crée un utilisateur client
SELECT marche_halibaba.signup_client('dgrolaux', 'nb_iterations:salt:hash', '
    Donatien', 'Grolaux');

-- Crée un utilisateur maison
SELECT marche_halibaba.signup_house('debouchetout', 'nb_iterations:salt:hash', '
    Debouchetout Inc.');
SELECT marche_halibaba.signup_house('specialisteswc', 'nb_iterations:salt:hash',
    'Les specialistes du WC');

-- Insère des demandes de devis
SELECT marche_halibaba.submit_estimate_request('Installation de sanitaires VIP
    pour Mr. Grolaux', '2016-04-18', 1, 'Rue chapelle aux champs', '43', '1200', '
    Bruxelles', null, null, null, null);
SELECT marche_halibaba.submit_estimate_request('Nettoyage des toilettes des é
    tudiants', '2016-05-31', 1, 'Rue chapelle aux champs', '43', '1200', '
    Bruxelles', 'Alma', '2', '1200', 'Bruxelles');

-- Insère des options
SELECT marche_halibaba.add_option('Toilettes en or massif', 6000, 1);
SELECT marche_halibaba.add_option('Toualèt vere pom', 1000, 1);
SELECT marche_halibaba.add_option('Toilettes en bronze', 2000, 2);

-- On modifie une option
SELECT marche_halibaba.modify_option('Toilettes vertes pomme', 1000, 2); -- pas
    très fort en orthographe ce nouveau stagiaire ;)

-- Insère des devis

-- Devis sans option
SELECT marche_halibaba.submit_estimate('Toilettes VIP', 2000, FALSE, FALSE, 1, 1,
     '{}');

-- Devis avec options
SELECT marche_halibaba.submit_estimate('Toilettes confortables', 1600, FALSE,
    FALSE, 1, 1, '{1,2}');

-- Devis masquant
SELECT marche_halibaba.submit_estimate('Nettoyage au Karcher', 400, FALSE, TRUE,
    2, 2, '{}');

-- Devis caché
SELECT marche_halibaba.submit_estimate('Nettoyage avec Cillit Bang', 600, TRUE,
    FALSE, 2, 2, '{}');

-- Devis masquant et caché
SELECT marche_halibaba.submit_estimate('Toilettes révolutionnaires', 800, TRUE,
    TRUE, 1, 1, '{}');

-- Accepter un devis sans option
SELECT marche_halibaba.approve_estimate(4, '{}', 1);

-- Accepter un devis avec option
SELECT marche_halibaba.approve_estimate(2, '{1}', 1);
```

3.3 Script d'insertion de données invalides

```
-- Création d'un utilisateur client
-- Un utilisateur possède déjà un compte avec ce nom d'utilisateur
-- Aucun champs ne peut être vide
SELECT marche_halibaba.signup_client('dgrolaux', 'nb_iterations:salt:hash', '
    Donatien', 'Grolaux');
SELECT marche_halibaba.signup_client('dgrolaux', 'nb_iterations:salt:hash', '
    Petitrigolo', '123');
SELECT marche_halibaba.signup_client('Petitrigolo', 'nb_iterations:salt:hash', ''
    , '');

-- Crée un utilisateur maison
SELECT marche_halibaba.signup_house('debouchetout', 'nb_iterations:salt:hash', '
    Debouchetout Inc.');
SELECT marche_halibaba.signup_house('specialisteswc', 'nb_iterations:salt:hash',
    'Les specialistes du WC');

-- Insertion d'une demandes de devis
-- La date souhaitée pour l'accomplissement des travaux doit être ultérieure à
    aujourd'hui
-- Aucun champs (à part l'adresse de facturation) ne peut être vide.
-- Le code postal doit être numérique
-- Une exception est levée.
SELECT marche_halibaba.submit_estimate_request('Installation de sanitaires VIP
    pour Mr. Grolaux', '2014-04-18', 1, 'Rue chapelle aux champs', '', 'ad', '
    Bruxelles', null, null, null, null);

-- Insertion et modification des options
-- Aucun champs ne peut être vide
-- Le montant de l'option ne peut être négatif. Une exception est levée.
SELECT marche_halibaba.add_option('', 200, 1);
SELECT marche_halibaba.modify_option('Toualèt vere pom', -23.3, 1);

-- Insertion de devis
-- La description d'un devis ne peut être vide
-- Le montant d'un devis ne peut-être négatif. Une exception est levée.
SELECT marche_halibaba.submit_estimate('', 2000, FALSE, FALSE, 1, 1, '{}');
SELECT marche_halibaba.submit_estimate('', -1000, FALSE, FALSE, 1, 1, '{}');

-- Insertion d'un devis pour une demande de devis expirée
-- Pré-condition: la demande de devis est expirée. Une exception est levée.
SELECT marche_halibaba.submit_estimate('Toilettes VIP', 2000, FALSE, FALSE, 1, 1,
     '{}');

-- Insertion d'un devis pour une demande de devis pour laquelle un devis a déjà é
    té accepté
-- Pré-condition: la demande de devis est expirée. Une exception est lancée.
SELECT marche_halibaba.submit_estimate('Toilettes VIP', 2000, FALSE, FALSE, 1, 1,
     '{}');

-- Insertion d'un devis avec option
-- Pré-condition: la maison soumissionnaire n'a pas d'option disponible
-- L'option en argument n'existe pas/la maison soumissionnaire ne possède pas
    cette option. Une exception est levée.
SELECT marche_halibaba.submit_estimate('Toilettes VIP', 2000, FALSE, FALSE, 1, 1,
     '{1}');

-- Insertion d'un devis caché
```

```
-- Pré-condition: la maison soumissionnaire a soumis un devis caché il y a moins
    de 24 heures
SELECT marche_halibaba.submit_estimate('Premier devis caché', 1600, TRUE, FALSE,
    1, 1, '{}');
-- La maison ne peut plus poster de devis caché pendant 24h. Une exception est
    levée.
SELECT marche_halibaba.submit_estimate('Deuxième devis caché', 1600, TRUE, FALSE,
     1, 1, '{}');

-- Insertion d'un devis masquant
-- Pré-condition: la maison soumissionnaire a soumis un devis masquant il y a
    moins de 7 jours
SELECT marche_halibaba.submit_estimate('Premier devis masquant', 1600, FALSE,
    TRUE, 1, 1, '{}');
-- La maison ne peut plus poster de devis masquant pendant 7 jours. Une exception
     est levée.
SELECT marche_halibaba.submit_estimate('Deuxième devis masquant', 1600, FALSE,
    TRUE, 1, 1, '{}');

-- Insertion d'un devis par une maison dénoncée
-- Pré-condition: une maison a soumis un devis masquant pour une demande possé
    dant déjà un devis masquant
SELECT marche_halibaba.submit_estimate('Devis dénoncé.', 1600, FALSE, TRUE, 1, 1,
     '{}');
SELECT marche_halibaba.submit_estimate('Devis dénonceur.', 1600, FALSE, TRUE, 1,
    2, '{}');
-- La maison dénoncée ne peut plus soumettre de devis pendant 24 heures. Une
    exception est levée.
SELECT marche_halibaba.submit_estimate('Nouveau devis', 600, FALSE, FALSE, 1, 1,
    '{}');

-- Accepter un devis pour une demande de devis expirée
-- Pré-condition: la demande de devis est expirée
-- Le devis ne peut être accepté. Une exception est levée.
SELECT marche_halibaba.approve_estimate(1, '{}', 1);

-- Accepter un devis lié à une demande pour laquelle un devis a déjà été accepté
-- Pré-condition: un devis pour la demande a déjà été accepté
-- Le devis ne peut être accepté. Une exception est levée.
SELECT marche_halibaba.approve_estimate(1, '{}', 1);

-- Accepter un devis annulé à cause d'une maison dénoncée
-- Pré-condition: le devis accepté
-- Le devis ne peut être accepté. Une exception est levée.
SELECT marche_halibaba.approve_estimate(1, '{}', 1);

-- Accepter un devis avec une option inexistante
-- Pré-condition: le devis n'offre aucune option
-- Le devis est accepté. L'option demandée est ignorée.
SELECT marche_halibaba.approve_estimate(1, '{1}', 1);
```

Chapitre 4

Application java

4.1 App.java

```
package marche_halibaba;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Map;

public abstract class App {

  Connection dbConnection;
  Map<String, PreparedStatement> preparedStmts;

  public App(String dbUser, String dbPswd) {

    try {
      Class.forName("org.postgresql.Driver");
    } catch (ClassNotFoundException e) {
      System.out.println("Driver PostgreSQL manquant !");
      System.exit(1);
    }

    // Dev
    String url = "jdbc:postgresql://localhost:5432/projet?user=" + dbUser + "&
        password=" + dbPswd;

    // Prod
    //String url = "jdbc:postgresql://localhost:5432/projet?user=app&password=2
        S5jn12JndG68hT";

    try {
      this.dbConnection = DriverManager.getConnection(url);
    } catch (SQLException e) {
      System.out.println("Impossible de joindre le server !");
      System.exit(1);
    }

  }

}
```

4.2 ClientsApp.java

```
package marche_halibaba;

import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.sql.Array;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

public class ClientsApp extends App {

  private int clientId;

  public static void main(String[] args) {

    try {
      ClientsApp session = new ClientsApp("app_clients", "2S5jn12JndG68hT");

      boolean isUsing = true;
      while(isUsing) {
        System.out.println("\n**********************************************");
        System.out.println("* Bienvenue sur le Marché d'Halibaba - Clients *");
        System.out.println("**********************************************");
        System.out.println("1 - Se connecter");
        System.out.println("2 - Créer un compte");
        System.out.println("3 - Quitter");

        System.out.println("\nQuel est votre choix? (1-3)");
        int userChoice = Utils.readAnIntegerBetween(1, 3);

        switch(userChoice) {
        case 1:

          if(session.signin()) {
            session.menu();
          }

          session.clientId = 0;
          break;
        case 2:

          if(session.signup()) {
            session.menu();
          }

          session.clientId = 0;
          break;
        case 3:
          isUsing = false;
          break;
        }

      }

      System.out.println("\nMerci de votre visite.   bientôt!");
```

```
      session.dbConnection.close();

    } catch(SQLException e) {
      e.printStackTrace();
      System.exit(1);
    }

  }

  public ClientsApp(String dbUser, String dbPswd) throws SQLException {
    super(dbUser, dbPswd);

    this.preparedStmts = new HashMap<String, PreparedStatement>();

    preparedStmts.put("signup", dbConnection.prepareStatement(
        "SELECT marche_halibaba.signup_client(?, ?, ?, ?)"));

    preparedStmts.put("signin", dbConnection.prepareStatement(
        "SELECT c_id, u_pswd " +
        "FROM marche_halibaba.signin_users " +
        "WHERE u_username = ?"));

    preparedStmts.put("estimateRequests", dbConnection.prepareStatement(
        "SELECT er_id, er_description, remaining_days " +
        "FROM marche_halibaba.list_estimate_requests " +
        "WHERE er_pub_date + INTERVAL '15' day >= NOW() AND " +
        "er_chosen_estimate IS NULL AND " +
        "c_id = ?"));

    preparedStmts.put("approvedEstimateRequests", dbConnection.prepareStatement(
        "SELECT er_id, er_description, er_pub_date, remaining_days " +
        "FROM marche_halibaba.list_estimate_requests " +
        "WHERE er_chosen_estimate IS NOT NULL AND " +
        "c_id = ?"));

    preparedStmts.put("submitEstimateRequests",
        dbConnection.prepareStatement("SELECT marche_halibaba.
            submit_estimate_request(?,?,?,?,?,?,?,?,?,?,?)"));

    preparedStmts.put("estimates", dbConnection.prepareStatement(
        "SELECT e_id, e_description, e_price, " +
            "e_house_name " +
        "FROM marche_halibaba.clients_list_estimates " +
        "WHERE e_estimate_request_id = ?"));

    preparedStmts.put("estimate", dbConnection.prepareStatement(
        "SELECT e_description, e_price, e_house_name, " +
            "e_option_id, e_option_description, e_option_price " +
        "FROM marche_halibaba.estimate_details " +
        "WHERE e_id = ?"));

    preparedStmts.put("approveEstimateRequests", dbConnection.prepareStatement("
       SELECT marche_halibaba.approve_estimate(?, ?, ?)"));

    preparedStmts.put("statistics", dbConnection.prepareStatement(
        "SELECT h.name, h.turnover, h.acceptance_rate, " +
            "h.caught_cheating_nbr, h.caught_cheater_nbr " +
        "FROM marche_halibaba.houses h "));

  }
```

```
private boolean signin() throws SQLException {
  System.out.println("\nSe connecter");
  System.out.println("**********************************************\n");

  boolean isUsing = true;
  while(isUsing) {
    System.out.print("Votre nom d'utilisateur : ");
    String username = Utils.scanner.nextLine();
    System.out.print("Votre mot de passe : ");
    String pswd = Utils.scanner.nextLine();

    try {
      PreparedStatement ps = preparedStmts.get("signin");
      ps.setString(1, username);
      ResultSet rs = ps.executeQuery();

      if(rs.next() &&
          rs.getInt(1) > 0 &&
          PasswordHash.validatePassword(pswd, rs.getString(2))) {
        clientId = rs.getInt(1);
        isUsing = false;
      } else {
        System.out.println("\nVotre nom d'utilisateur et/ou mot de passe est
            erroné.");
        System.out.println("Voulez-vous réessayer? Oui (O) - Non (N)");

        if(!Utils.readOorN()) {
          isUsing = false;
        }
      }

      rs.close();
    } catch (NoSuchAlgorithmException e) {
      e.printStackTrace();
    } catch (InvalidKeySpecException e) {
      e.printStackTrace();
    }

  }

  return clientId > 0;
}

private boolean signup() throws SQLException {
  System.out.println("\nInscription");
  System.out.println("**********************************************\n");

  boolean isUsing = true;
  while (isUsing) {
    System.out.print("Votre nom: ");
    String lastName = Utils.scanner.nextLine();
    System.out.print("Votre prenom: ");
    String firstName = Utils.scanner.nextLine();
    System.out.print("Votre nom d'utilisateur: ");
    String username = Utils.scanner.nextLine();
    System.out.print("Votre mot de passe: ");
    String pswd = Utils.scanner.nextLine();

    try {
```

```
        pswd = PasswordHash.createHash(pswd);
      } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        System.exit(1);
      } catch (InvalidKeySpecException e) {
        e.printStackTrace();
        System.exit(1);
      }

      PreparedStatement ps = preparedStmts.get("signup");
      ps.setString(1, username);
      ps.setString(2, pswd);
      ps.setString(3, firstName);
      ps.setString(4, lastName);
      ResultSet rs = null;

      try {
        rs = ps.executeQuery();
        rs.next();

        System.out.println("\nVotre compte a bien été crée.");
        System.out.println("Vous allez maintenant être redirigé vers la page d'
            accueil de l'application.");
        Utils.blockProgress();

        clientId = rs.getInt(1);
        isUsing = false;
      } catch (SQLException e) {

        e.printStackTrace();

        if(e.getSQLState().equals("23505")) {
          System.out.println("\nCe nom d'utilisateur est déjà utilisé.");
        } else {
          System.out.println("\nLes données saisies sont incorrectes.");
        }

        System.out.println("Voulez-vous réessayer? Oui (O) - Non (N)");

        if(!Utils.readOorN()) {
          isUsing = false;
        }

      } finally {

        if(rs != null) {
          rs.close();
        }

      }

    }

    return clientId > 0;
  }

  private void menu() throws SQLException {

    boolean isUsing = true;
    while(isUsing) {
```

```
      System.out.println("\nMenu");
      System.out.println("**********************************************\n");

      System.out.println("1. Consulter mes demandes de devis en cours");
      System.out.println("2. Consulter mes demandes de devis acceptées");
      System.out.println("3. Soumettre une demande de devis");
      System.out.println("4. Afficher les statistiques des maisons");
      System.out.println("5. Se déconnecter");

      System.out.println("\nQue desirez-vous faire ? (1 - 5)");
      int choice = Utils.readAnIntegerBetween(1, 5);

      switch(choice) {
      case 1:
        displayEstimateRequests();
        break;
      case 2:
        displayApprovedEstimateRequests();
        break;
      case 3:
        submitEstimateRequest();
        break;
      case 4:
        displayStatistics();
        break;
      case 5:
        isUsing = false;
        break;
      }

    }

  }

  private void displayEstimateRequests() throws SQLException {

    boolean isUsing = true;
    while(isUsing) {
      System.out.println("\nListe des demandes de devis en cours : ");
      System.out.println("**********************************************\n");

      HashMap<Integer, Integer> estimateRequests = new HashMap<Integer, Integer
          >();
      String estimateRequestsStr = "";

      PreparedStatement ps = preparedStmts.get("estimateRequests");
      ps.setInt(1, clientId);
      ResultSet rs = ps.executeQuery();

      int i = 1;
      while(rs.next()) {
        estimateRequests.put(i, rs.getInt(1));
        estimateRequestsStr += i + ". " + rs.getString(2) + " - " +
            Utils.SQLIntervalToString(rs.getString(3)) + "\n";
        i++;
      }

      rs.close();

      if(estimateRequests.size() > 0) {
```

```
        System.out.println(estimateRequestsStr);

        System.out.println("Que voulez-vous faire ?");
        System.out.println("1. Consulter les devis soumis pour une demande");
        System.out.println("2. Retour");

        if(Utils.readAnIntegerBetween(1, 2) == 1) {
          System.out.println("\n" + estimateRequestsStr);
          System.out.println("Pour quelle demande voulez-vous voir les devis
              soumis?");
          int userChoice = Utils.readAnIntegerBetween(1, estimateRequests.size())
              ;
          displayEstimates(estimateRequests.get(userChoice));
        } else {
          isUsing = false;
        }

      } else {
        System.out.println("Il n'y a aucune demande de devis en cours");
        Utils.blockProgress();
        isUsing = false;
      }

    }

  }

  private void displayApprovedEstimateRequests() throws SQLException {
    System.out.println("\nListe des demandes de devis acceptées :");
    System.out.println("*****************************************\n");

    HashMap<Integer, Integer> approvedEstimateRequests = new HashMap<Integer,
        Integer>();
    String estimateRequestsStr = "";

    PreparedStatement ps = preparedStmts.get("approvedEstimateRequests");
    ps.setInt(1, clientId);
    ResultSet rs = ps.executeQuery();

    int i = 1;
    while(rs.next()) {
      approvedEstimateRequests.put(i, rs.getInt(1));
      estimateRequestsStr += i + ". " + rs.getString(2) + "\n";
      i++;
    }

    rs.close();

    if(approvedEstimateRequests.size() > 0) {
      System.out.println(estimateRequestsStr);
      Utils.blockProgress();
    } else {
      System.out.println("Il n'y a aucune demande de devis acceptées.");
      Utils.blockProgress();
    }

  }

  private void displayEstimates(int id) throws SQLException {
```

```
    boolean isUsing = true;
    while(isUsing) {
      HashMap<Integer, Integer> estimates = new HashMap<Integer, Integer>();
      String estimatesStr = "";

      PreparedStatement ps = preparedStmts.get("estimates");
      ps.setInt(1, id);
      ResultSet rs = ps.executeQuery();

      int i = 1;
      while(rs.next()) {
        estimates.put(i, rs.getInt(1));
        estimatesStr += i + ". " + rs.getString(2) + " - Prix: " + rs.getDouble
            (3) + " euros - Maison: " + rs.getString(4) + "\n";
        i++;
      }

      rs.close();

      System.out.println("\nListe des devis soumis : ");
      System.out.println("*********************************************\n");

      if(estimates.size() > 0) {
        System.out.println(estimatesStr);
        System.out.println("Que voulez-vous faire ?");
        System.out.println("1. Afficher les détails d'un devis");
        System.out.println("2. Retour");

        if(Utils.readAnIntegerBetween(1, 2) == 1) {
          System.out.println(estimatesStr);
          System.out.println("Quel devis voulez-vous consulter ?");
          int userChoice = Utils.readAnIntegerBetween(1, estimates.size());
          isUsing = !displayEstimate(estimates.get(userChoice));
        } else {
          isUsing = false;
        }

      } else {
        System.out.println("Il n'y a aucun devis soumis pour cette demande.");
        Utils.blockProgress();
        isUsing = false;
      }

    }

}

private boolean displayEstimate(int estimateId) throws SQLException {
  String optionsStr = "";
  Map<Integer, Integer> options = new HashMap<Integer, Integer>();

  PreparedStatement ps = preparedStmts.get("estimate");
  ps.setInt(1, estimateId);
  ResultSet rs = ps.executeQuery();

  if(rs.next()) {
    System.out.println("\nDevis : " + rs.getString(1));
    System.out.println("*********************************************\n");
    System.out.println("Prix : " + rs.getDouble(2) + " euros");
    System.out.println("Maison : " + rs.getString(3));
```

```
    int i = 1;
    do {

      if(rs.getInt(4) != 0) {
        optionsStr += i + ". " + rs.getString(5) + " - prix : " + rs.getDouble
            (6) + " euros\n";
        options.put(i, rs.getInt(4));
        i++;
      }

    } while(rs.next());

    if(options.size() > 0) {
      System.out.println("\nListes des options disponibles : ");
      System.out.println(optionsStr);
    }

  }

  rs.close();

  System.out.println("\nQue voulez-vous faire ?");
  System.out.println("1. Accepter ce devis");
  System.out.println("2. Retour");

  if(Utils.readAnIntegerBetween(1, 2) == 1) {
    return approveEstimate(estimateId, optionsStr, options);
  }

  return false;
}

private boolean approveEstimate(int estimateId, String optionsStr, Map<Integer,
    Integer> options) throws SQLException {
  System.out.println("\nEtes-vous sur de vouloir accepter ce devis ? Oui (O) - Non (N)");

  if(Utils.readOorN()) {
    boolean status = false;
    Array chosenOptions = null;

    if(options.size() > 0) {
      System.out.println("Voulez-vous choisir des options ? Oui (O) - Non (N)");

      if(Utils.readOorN()) {
        System.out.println("Quels options voulez-vous choisir ? (exemple: 1, 2, 3)");
        int[] integers = Utils.readIntegersBetween(1, options.size());
        Object[] userChoices = new Object[integers.length];

        for(int i = 0; i < integers.length; i++) {
          userChoices[i] = (Object) options.get(integers[i]);
        }

        chosenOptions = dbConnection.createArrayOf("integer", userChoices);
      }

    }
```

```
    PreparedStatement ps = preparedStmts.get("approveEstimateRequests");
    ps.setInt(1, estimateId);
    ps.setArray(2, chosenOptions);
    ps.setInt(3, clientId);
    ResultSet rs = null;

    try {
      rs = ps.executeQuery();
      rs.next();
      System.out.println("\nLe devis a bien été accepté!");
      Utils.blockProgress();
      status = true;
    } catch (SQLException e) {
      System.out.println("Malheureusement, ce devis ne peut-etre accepté.\n");
    } finally {

      if(rs != null) {
        rs.close();
      }

    }

    return status;
  }

  return false;
}

private void submitEstimateRequest() throws SQLException {
  System.out.println("\nSoumettre une demande de devis");
  System.out.println("**********************************************\n");

  boolean isUsing = true;
  while(isUsing) {
    System.out.print("Description : ");
    String description = Utils.scanner.nextLine();
    System.out.print("Date souhaitée de fin des travaux (jj/mm/aaaa) : ");
    Date deadline = Utils.readDate();
    Map<String, String> constructionAddress = enterAddress();

    System.out.println("L'adresse de facturation est-elle differente de l'
        adresse des travaux ? O (oui) - N (non)");
    Map<String, String> invoicingAddress = null;

    if(Utils.readOorN()) {
      invoicingAddress = enterAddress();
    }

    PreparedStatement ps = preparedStmts.get("submitEstimateRequests");
    ps.setString(1, description);
    ps.setDate(2, new java.sql.Date(deadline.getTime()));
    ps.setInt(3, clientId);
    ps.setString(4, constructionAddress.get("streetName"));
    ps.setString(5, constructionAddress.get("streetNbr"));
    ps.setString(6, constructionAddress.get("zipCode"));
    ps.setString(7, constructionAddress.get("city"));

    if(invoicingAddress == null) {
      ps.setString(8, null);
```

```
        ps.setString(9, null);
        ps.setString(10, null);
        ps.setString(11, null);
      } else {
        ps.setString(8, invoicingAddress.get("streetName"));
        ps.setString(9, invoicingAddress.get("streetNbr"));
        ps.setString(10, invoicingAddress.get("zipCode"));
        ps.setString(11, invoicingAddress.get("city"));
      }

      ResultSet rs = null;

      try {
        rs = ps.executeQuery();
        System.out.println("\nFelicitations! Votre demande de devis a bien été
            publiée.");
        Utils.blockProgress();
        isUsing = false;
      } catch (SQLException e) {
        System.out.println("Les donnees entrées sont erronnées. Veuillez
            recommencer.\n");
      } finally {

        if(rs != null) {
          rs.close();
        }

      }

    }

  }

  private void displayStatistics() throws SQLException {
    System.out.println("\nStatistiques des maisons");
    System.out.println("*****************************************");

    PreparedStatement ps = preparedStmts.get("statistics");
    ResultSet rs = ps.executeQuery();

    while(rs.next()) {
      System.out.println("\n" + rs.getString(1));
      System.out.println("\tChiffre d'affaire: " + rs.getDouble(2) + "    ");
      System.out.println("\tTaux d'acceptation: " + (rs.getDouble(3)*100) + " %")
          ;
      System.out.println("\tNombre de fois que la maison s'est fait attraper en
          train de tricher: " + rs.getInt(4) + " fois");
      System.out.println("\tNombre de fois que la maison a attrapé un tricheur :
          " + rs.getInt(5) + " fois");
    }

    rs.close();
    Utils.blockProgress();
  }

  private Map<String, String> enterAddress() {
    Map<String, String> address = new HashMap<String, String>();

    System.out.print("Nom de la rue: ");
    address.put("streetName", Utils.scanner.nextLine());
```

```
    System.out.print("Numero: ");
    address.put("streetNbr", Utils.scanner.nextLine());

    System.out.print("Code postal: ");
    address.put("zipCode", Utils.scanner.nextLine());

    System.out.print("Ville: ");
    address.put("city", Utils.scanner.nextLine());

    return address;
  }

}
```

4.3 HousesApp.java

```
package marche_halibaba;

import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Array;
import java.util.ArrayList;
import java.math.BigDecimal;
import java.util.HashMap;

public class HousesApp extends App{

  private int houseId;

  public static void main(String[] args) {

    try{

      HousesApp session = new HousesApp("app_houses", "2S5jn12JndG68hT");

      boolean isUsing = true;
      while(isUsing) {
        System.out.println("\n***********************************************");
        System.out.println("* Bienvenue sur le Marche d'Halibaba - Maisons *");
        System.out.println("***********************************************");
        System.out.println("1 - Se connecter");
        System.out.println("2 - Créer un compte");
        System.out.println("3 - Quitter");

        System.out.println("\nQuel est votre choix? (1-3)");
        int userChoice = Utils.readAnIntegerBetween(1, 3);

        switch(userChoice) {
        case 1:

          if(session.signin()) {
            session.menu();
          }

          session.houseId = 0;
          break;
        case 2:

          if(session.signup()) {
            session.menu();
          }

          session.houseId = 0;
          break;
        case 3:
          isUsing = false;
          break;
        }

      }
```

```
      System.out.println("\nMerci␣de␣votre␣visite.  ␣bientôt!");
      session.dbConnection.close();

    } catch(SQLException e) {
      e.printStackTrace();
      System.exit(1);
    }


}

public HousesApp(String dbUser, String dbPswd) throws SQLException{
  super(dbUser, dbPswd);

  this.preparedStmts = new HashMap<String, PreparedStatement>();

  preparedStmts.put("signup", dbConnection.prepareStatement(
      "SELECT␣marche_halibaba.signup_house(?,␣?,␣?)"));

  preparedStmts.put("signin", dbConnection.prepareStatement(
      "SELECT␣h_id,␣u_pswd␣" +
      "FROM␣marche_halibaba.signin_users␣" +
      "WHERE␣u_username␣=␣?"));

  preparedStmts.put("estimates", dbConnection.prepareStatement(
      "SELECT␣e_description,␣e_price,␣e_submission_date,␣"+
          "er_estimate_request_id,␣␣h_name␣"+
      "FROM␣marche_halibaba.valid_estimates_list␣"+
      "WHERE␣er_estimate_request_id␣=␣?␣AND␣"+
      "(e_is_secret=␣FALSE␣OR␣(e_is_secret␣=␣TRUE␣AND␣e_house_id=␣?))"));

  preparedStmts.put("estimateRequests", dbConnection.prepareStatement(
      "SELECT␣er_id,␣er_description,␣remaining_days␣" +
      "FROM␣marche_halibaba.list_estimate_requests␣" +
      "WHERE␣er_pub_date␣+␣INTERVAL␣'15'␣day␣>=␣NOW()␣AND␣" +
      "er_chosen_estimate␣IS␣NULL␣"));

  preparedStmts.put("submit_estimate", dbConnection.prepareStatement(
      "SELECT␣marche_halibaba.submit_estimate(?,␣?,␣?,␣?,␣?,␣?,␣?)"
      ));

  preparedStmts.put("add_option", dbConnection.prepareStatement(
      "SELECT␣marche_halibaba.add_option(?,␣?,␣?)"));

  preparedStmts.put("list_options", dbConnection.prepareStatement(
      "SELECT␣o.option_id,␣o.description,␣o.price␣"+
      "FROM␣marche_halibaba.options␣o␣"+
      "WHERE␣house_id=␣?"));

  preparedStmts.put("statistics", dbConnection.prepareStatement(
      "SELECT␣h.name,␣h.turnover,␣h.acceptance_rate,␣" +
      "h.caught_cheating_nbr,␣h.caught_cheater_nbr␣" +
      "FROM␣marche_halibaba.houses␣h␣"));

  preparedStmts.put("valid_estimates", dbConnection.prepareStatement(
      "SELECT␣h_valid_estimates_nbr␣"+
      "FROM␣marche_halibaba.valid_estimates_nbr␣"+
      "WHERE␣h_id=␣?"));

  preparedStmts.put("modify_option", dbConnection.prepareStatement(
```

```
        "SELECT␣marche_halibaba.modify_option(?,?,?,?)␣"));

  }

  private boolean signin() throws SQLException {
    System.out.println("\nSe␣connecter");
    System.out.println("**********************************************\n");

    boolean isUsing = true;
    while(isUsing) {
      System.out.print("Votre␣nom␣d'utilisateur␣:␣");
      String username = Utils.scanner.nextLine();
      System.out.print("Votre␣mot␣de␣passe␣:␣");
      String pswd = Utils.scanner.nextLine();

      try {
        PreparedStatement ps = preparedStmts.get("signin");
        ps.setString(1, username);
        ResultSet rs = ps.executeQuery();

        if(rs.next() &&
            rs.getInt(1) > 0 &&
            PasswordHash.validatePassword(pswd, rs.getString(2))) {
          houseId= rs.getInt(1);
          isUsing= false;
        } else {
          System.out.println("\nVotre␣nom␣d'utilisateur␣et/ou␣mot␣de␣passe␣est␣
              errone.");
          System.out.println("Voulez-vous␣reessayer?␣Oui␣(O)␣-␣Non␣(N)");

          if(!Utils.readOorN()) {
            isUsing = false;
          }
        }

        rs.close();
      } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
      } catch (InvalidKeySpecException e) {
        e.printStackTrace();
      }
    }

    return houseId > 0;
  }

  private boolean signup() throws SQLException{
    System.out.println("\nInscription");
    System.out.println("**********************************************\n");

    boolean isUsing = true;
    while (isUsing) {
      System.out.print("Nom␣de␣votre␣maison␣:␣");
      String name = Utils.scanner.nextLine();
      System.out.print("Votre␣nom␣d'utilisateur␣:␣");
      String username = Utils.scanner.nextLine();
      System.out.print("Votre␣mot␣de␣passe␣:␣");
      String pswd = Utils.scanner.nextLine();

      try {
```

```
        pswd = PasswordHash.createHash(pswd);
      } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        System.exit(1);
      } catch (InvalidKeySpecException e) {
        e.printStackTrace();
        System.exit(1);
      }

      PreparedStatement ps = preparedStmts.get("signup");
      ps.setString(1, username);
      ps.setString(2, pswd);
      ps.setString(3, name);
      ResultSet rs = null;

      try {
        rs = ps.executeQuery();
        rs.next();

        System.out.println("\nVotre compte a bien été créé.");
        System.out.println("Vous allez maintenant être redirigé vers la page d'
            accueil de l'application.");
        Utils.blockProgress();

        houseId = rs.getInt(1);
        isUsing = false;
      } catch (SQLException e){

        e.printStackTrace();

        if(e.getSQLState().equals("23505")){
          System.out.println("\nCe nom d'utilisateur est déjà utilise.");
        } else {
          System.out.println("\nLes données saisies sont incorrectes.");
        }

        System.out.println("Voulez-vous reessayer? Oui (O) - Non (N)");

        if(!Utils.readOorN()) {
          isUsing = false;
        }

      } finally {

        if(rs != null){
          rs.close();
        }

      }

    }

    return houseId > 0;
  }

  private void menu() throws SQLException{

    boolean isUsing = true;
    while(isUsing) {
      System.out.println("\nMenu");
```

```
      System.out.println("**********************************************\n");

      System.out.println("1. Lister les demandes de devis en cours");
      System.out.println("2. Ajouter des options au catalogue d'options");
      System.out.println("3. Modifier des options du catalogue d'options");
      System.out.println("4. Statistiques");
      System.out.println("5. Se déconnecter");

      System.out.println("\nQue désirez-vous faire ? (1 - 5)");
      int choice = Utils.readAnIntegerBetween(1, 5);

      switch(choice) {
      case 1:
        displayEstimateRequests();
        break;
      case 2:
        addOption();
        break;
      case 3:
        modifyOption();
        break;
      case 4:
        displayStatistics();
        break;
      case 5:
        isUsing = false;
        break;
      }

    }

  }


private void displayEstimatesForRequest(int requestId) throws SQLException {

  HashMap<Integer, Integer> estimates = new HashMap<Integer, Integer>();
  String estimateStr = "";


  PreparedStatement ps = preparedStmts.get("estimates");
  ps.setInt(1, requestId);
  ps.setInt(2, houseId);
  ResultSet rs = ps.executeQuery();

  int j= 1;
  while(rs.next()) {
    if(rs.getInt(4)== requestId){
      estimates.put(j, rs.getInt(4));
      estimateStr += j+ ". "+ rs.getString(1) +
          "\n\tPrix: " + rs.getDouble(2) +
        "\n\tSoumis le : "+rs.getDate(3)+
        "\n\tDevis soumis par la maison: "+rs.getString(5)+"\n\n";
      j++;
    }

  }
  rs.close();
```

```
  if(estimates.size() > 0) {
    System.out.println(estimateStr);
    System.out.println("Que voulez-vous faire ? (1 - 2)");
    System.out.println("1. Soumettre un nouveau devis pour cette demande");
    System.out.println("2. Retour");
    int userChoice= Utils.readAnIntegerBetween(1, 2);

    if(userChoice== 1)
      submitEstimate(requestId);

  } else {
    System.out.println("Il n'y a aucun devis pour la demande en question\n");
    System.out.println("Que voulez-vous faire ?(1 - 2)\n");
    System.out.println("1. Soumettre un nouveau devis pour cette demande");
    System.out.println("2. Retour au menu");
    int userChoice= Utils.readAnIntegerBetween(1, 2);

    if(userChoice== 1)
      submitEstimate(requestId);

  }

}


private void displayEstimateRequests() throws SQLException{

  HashMap<Integer, Integer> estimateRequests = new HashMap<Integer, Integer>();
  String estimateRequestsStr = "";

  PreparedStatement ps = preparedStmts.get("estimateRequests");
  ResultSet rs = ps.executeQuery();

  int i = 1;
  while(rs.next()) {
    estimateRequests.put(i, rs.getInt(1));
    estimateRequestsStr += i + ". " + rs.getString(2) +
    "\n\tTemps restant: " + Utils.SQLIntervalToString(rs.getString(3)) + "\n\n";
    i++;
  }
  rs.close();

  if(estimateRequests.size() > 0) {
    System.out.println(estimateRequestsStr);

    System.out.println("Que voulez-vous faire ? (1 - 2)");
    System.out.println("1. Consulter les devis soumis pour une demande");
    System.out.println("2. Retour");

    if(Utils.readAnIntegerBetween(1, 2) == 1) {
      System.out.println(estimateRequestsStr);
      System.out.println("Pour quelle demande voulez-vous voir les devis soumis?"
          );
      int userChoice = Utils.readAnIntegerBetween(1, estimateRequests.size());
      displayEstimatesForRequest(estimateRequests.get(userChoice));

    }

  } else {
```

```
    System.out.println("Il n'y a aucune demande de devis en cours\n");
    Utils.blockProgress();
  }

}


private void submitEstimate(int estimateRequest) throws SQLException{

  System.out.println("Soumettre un devis");
  System.out.println("*****************");

  System.out.println("Description:");
  String description = Utils.scanner.nextLine();

  System.out.println("Prix du devis:");
  double price= Utils.readADoubleBetween(0, 1000000000.0);

  System.out.println("Voulez-vous que le devis soit caché? Oui (O) - Non (N)");
  boolean secret= Utils.readOorN();

  System.out.println("Voulez-vous que le devis soit masquant? Oui (O) - Non (N)")
      ;
  boolean hiding= Utils.readOorN();

  System.out.println("Voulez-vous rajouter des options ? Oui (O) - Non (N)");
  boolean options= Utils.readOorN();

  ArrayList<Integer> choosedOptions= new ArrayList<>();

  while(options){

    System.out.println("Menu des options");
    System.out.println("****************");
    System.out.println("1.Sélectionner une option existante dans le catalogue d'
        options");
    System.out.println("2.Ajouter une nouvelle option au calatogue et l'utiliser
        pour le devis");
    System.out.println("3.Soumettre le devis avec les options sélectionnées");

    System.out.println("\nQue désirez-vous faire ? (1 - 3)");
    int userChoice= Utils.readAnIntegerBetween(1, 3);
    switch(userChoice){
      case 1:

        int selectedOption= selectOption();
        if(choosedOptions.contains(selectedOption)){
          System.out.println("Veuillez choisir des options pas encore choisies\n"
              );

        }else if(selectedOption == -1){

        }else{
          choosedOptions.add(selectedOption);
          System.out.println("L'option a correctement été ajoutée pour le devis\n
              ");
        }

        break;
```

```
      case 2:

        int optionId = addOption();
        if(optionId != -1)
          choosedOptions.add(optionId);
        else
          System.out.println("L'ajout d'option a échoué");
        break;

      case 3:
        options= false;
        break;
    }

  }

  try{

    PreparedStatement ps = preparedStmts.get("submit_estimate");
    ps.setString(1, description);
    ps.setBigDecimal(2, new BigDecimal(price));
    ps.setBoolean(3, secret);
    ps.setBoolean(4, hiding);
    ps.setInt(5, estimateRequest);
    ps.setInt(6, houseId);

    Object[] userChoices= new Object[choosedOptions.size()];
    for (int i = 0; i < choosedOptions.size(); i++) {
      userChoices[i]= choosedOptions.get(i);
    }

    Array chosenOptions= dbConnection.createArrayOf("integer", userChoices);

    ps.setArray(7, chosenOptions);

    ps.executeQuery();

    System.out.println("Le devis a été soumis correctement !\n");
    Utils.blockProgress();


  }catch(SQLException e){
    String message= e.getMessage();
    System.out.println(message.split("\n")[0]);

  }

}

private int selectOption() throws SQLException{

  HashMap<Integer, Integer> options = new HashMap<Integer, Integer>();
  String optionsStr = "";

  PreparedStatement ps = preparedStmts.get("list_options");
  ps.setInt(1, houseId);
  ResultSet rs = ps.executeQuery();

  int i = 1;
  while(rs.next()) {
```

```
    options.put(i, rs.getInt(1));
    optionsStr += i + ". " + rs.getString(2) +
      "\n\tPrix de l'option: " + rs.getDouble(3) + "\n\n";
    i++;
  }
  rs.close();

  if(options.size() > 0){
    System.out.println(optionsStr);

    System.out.println("Voulez-vous ajouter une de ces options? Oui (O) - Non (N)
        ");
    if(Utils.readOorN()){

      System.out.println(optionsStr);
      System.out.println("Quelle option voulez-vous ajouter? 1 - "+ options.size
          ());
      return options.get(Utils.readAnIntegerBetween(1, options.size()));

    }

  }else{
    System.out.println("Vous n'avez pas encore d'options dans votre catalogue");
  }

  return -1;

}

private int addOption() throws SQLException{

  System.out.println("Ajout d'une nouvelle option");
  System.out.println("**************************");
  System.out.println("Veuillez entrer une description pour l'option");
  String description = Utils.scanner.nextLine();
  System.out.println("Veuillez entrer un prix pour cette option");
  double price= Utils.readADoubleBetween(0, 1000000000.0);

  PreparedStatement ps = preparedStmts.get("add_option");

  ps.setString(1,description);

  ps.setBigDecimal(2, new BigDecimal(price));

  ps.setInt(3, houseId);

  ResultSet result= ps.executeQuery();
  result.next();

  int optionId= result.getInt(1);
  result.close();

  System.out.println("Votre option a bien été ajoutée!\n\n");

  return optionId;

}

private void displayStatistics() throws SQLException {
```

```
  System.out.println("\nStatistiques des maisons");
  System.out.println("*****************************************");

  PreparedStatement ps = preparedStmts.get("statistics");
  ResultSet rs = ps.executeQuery();

  while(rs.next()) {
    System.out.println("\n" + rs.getString(1));
    System.out.println("\tChiffre d'affaire: " + rs.getDouble(2) + "    ");
    System.out.println("\tTaux d'acceptation: " + (rs.getDouble(3)*100) + " %");
    System.out.println("\tNombre de fois que la maison s'est fait attraper en
        train de tricher : " + rs.getInt(4) + " fois");
    System.out.println("\tNombre de fois que la maison a attrapé un tricheur : "
       + rs.getInt(5) + " fois");
  }
  rs.close();

  ps = preparedStmts.get("valid_estimates");
  ps.setInt(1, houseId);
  rs = ps.executeQuery();
  rs.next();

  System.out.println("\nVous avez actuellement "+
  rs.getBigDecimal(1)+" devis en cours de soumission.");
  rs.close();

  Utils.blockProgress();
}

private void modifyOption() throws SQLException{

  System.out.println("Modification d'une option");
  System.out.println("************************");

  HashMap<Integer, Integer> options = new HashMap<Integer, Integer>();
  String optionsStr = "";

  PreparedStatement ps = preparedStmts.get("list_options");
  ps.setInt(1, houseId);
  ResultSet rs = ps.executeQuery();

  int i = 1;
  while(rs.next()) {
    options.put(i, rs.getInt(1));
    optionsStr += i + ". " + rs.getString(2) +
      "\n\tPrix de l'option: " + rs.getDouble(3) + "\n\n";
    i++;
  }
  rs.close();

  if(options.size() > 0){
    System.out.println(optionsStr);

    System.out.println("Quelle option voulez-vous modifier? 1 - "+ options.size()
        );
    int userChoice= options.get(Utils.readAnIntegerBetween(1, options.size()));

    System.out.println("\nModification de l'option");
    System.out.println("************************");
    System.out.println("Veuillez entrer une description pour l'option");
```

```
    String description = Utils.scanner.nextLine();

    System.out.println("Veuillez entrer un prix pour cette option");
    double price= Utils.readADoubleBetween(0, 1000000000.0);

    ps = preparedStmts.get("modify_option");

    ps.setString(1,description);
    ps.setBigDecimal(2, new BigDecimal(price));
    ps.setInt(3, userChoice);
    ps.setInt(4, houseId);

    ps.executeQuery();
    System.out.println("L'option a été modifiée avec succès !\n");

  }else{

    System.out.println("Vous n'avez pas encore d'options dans votre catalogue");
    Utils.blockProgress();
    System.out.println("\n\n");

  }

}


}
```

4.4 Utils.java

```
package marche_halibaba;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.InputMismatchException;
import java.util.Iterator;
import java.util.Locale;
import java.util.Scanner;

public class Utils {
  public static Scanner scanner = new Scanner(System.in);

  public static void blockProgress() {
    System.out.println("\n[Appuyez sur ENTER pour continuer]");

    try {
          scanner.nextLine();
        } catch(Exception e) {}

  }

  public static int readAnIntegerBetween(int number1, int number2){
    int number = 0;

    boolean isLegal = false;
        while(!isLegal) {

          try {
            number = scanner.nextInt();

            if(number>=number1 && number<=number2) {
              isLegal = true;
            } else {
              System.out.println("Le nombre doit être compris entre " + number1 +
                  " et " + number2 + ". Veuillez recommencer.");
            }


          } catch(InputMismatchException e) {
            System.out.println("Vous ne pouvez entrer que des chiffres. Veuillez
                recommencer.");
          } finally {
            scanner.nextLine();
      }

        }

        return number;
  }

  public static double readADoubleBetween(double number1, double number2){
    double number = 0;

    boolean isLegal = false;
    while(!isLegal){
```

```
    try{
      number= scanner.nextDouble();

      if(number >= number1 && number<= number2){
        isLegal = true;
      }else{
            System.out.println("Le nombre doit être compris entre " + number1 +
                "et " + number2 + ". Veuillez recommencer.");
      }


    }catch(InputMismatchException e){
          System.out.println("Vous ne pouvez entrer que des chiffres. Veuillez
              recommencer.");
    } finally{
      scanner.nextLine();
    }

  }

  return number;
}

public static Date readDate() {
  Date date = null;

  boolean isLegal = false;
  while(!isLegal) {
    String str = scanner.nextLine();

    DateFormat format = new SimpleDateFormat("dd/MM/yyyy", Locale.ENGLISH);

    try {
      date = format.parse(str);
      isLegal = true;
    } catch (ParseException e) {
      System.out.println("Veuillez entrer une date au format correct (jj/mm/
          aaaa).");
    }

  }

  return date;
}

public static int[] readIntegersBetween(int number1, int number2) {
  int[] integers = null;

  boolean isLegal = false;
  while(!isLegal) {
    String str = scanner.nextLine();
    str = str.replaceAll("[^-?0-9]+", "-");
      String[] strs = str.split("-");
      integers = new int[strs.length];

      if(strs.length == 0) {
        System.out.println("Veuillez entrer des nombres compris entre " +
            number1 + "et " + number2 + ".");
      } else {
```

```
        isLegal = true;
      }

      for(int i=0; i<strs.length; i++) {
        int j = Integer.parseInt(strs[i]);

        if(j < number1 || j > number2) {
          System.out.println("Les nombres doivent être compris entre " +
              number1 + " et " + number2 + ". Veuillez recommencer.");
          isLegal = false;
          break;
        }

        integers[i] = j;
      }

  }

    return integers;
}

public static String SQLIntervalToString(String interval) {
  String str = "";

  String days = interval.substring(0, 2).replaceAll(" ", "");
  String hours = interval.replaceAll("[0-9]{1,2} days ", "").replaceAll("[0-9] 
      day ", "").substring(0, 2).replaceAll(":", "");

  str = days + " jour(s) " + hours + " heure(s) restant(s)";

  return str;
}


public static boolean readOorN(){
  String str = scanner.nextLine();
  char response = 'z';

  if(str.length() == 1) {
    response = str.charAt(0);
  }

  while (response != 'O' && response != 'o' &&
      response != 'N' && response != 'n'){
    System.out.println("Veuillez répondre O (oui) ou N (non).");
    str = scanner.nextLine();

    if(str.length() == 1) {
      response = str.charAt(0);
    }

  }

  return response == 'O' || response == 'o';
}

public static int[] convertIntegers(ArrayList<Integer> integers) {
    int[] ret = new int[integers.size()];
    Iterator<Integer> iterator = integers.iterator();
    for (int i = 0; i < ret.length; i++)
```

```
        {
            ret[i] = iterator.next().intValue();
        }
        return ret;
    }
}
```

4.5 PasswordHash.java

```
package marche_halibaba;

import java.security.SecureRandom;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.SecretKeyFactory;
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;

/*
 * PBKDF2 salted password hashing.
 * Author: havoc AT defuse.ca
 * www: http://crackstation.net/hashing-security.htm
 */
public class PasswordHash
{
    public static final String PBKDF2_ALGORITHM = "PBKDF2WithHmacSHA1";

    public static final int SALT_BYTES = 24;
    public static final int HASH_BYTES = 96;
    public static final int PBKDF2_ITERATIONS = 1000;

    public static final int ITERATION_INDEX = 0;
    public static final int SALT_INDEX = 1;
    public static final int PBKDF2_INDEX = 2;

    /**
     * Returns a salted PBKDF2 hash of the password.
     *
     * @param   password    the password to hash
     * @return              a salted PBKDF2 hash of the password
     */
    public static String createHash(String password)
        throws NoSuchAlgorithmException, InvalidKeySpecException
    {
        return createHash(password.toCharArray());
    }

    /**
     * Returns a salted PBKDF2 hash of the password.
     *
     * @param   password    the password to hash
     * @return              a salted PBKDF2 hash of the password
     */
    public static String createHash(char[] password)
        throws NoSuchAlgorithmException, InvalidKeySpecException
    {
        // Generate a random salt
        SecureRandom random = new SecureRandom();
        byte[] salt = new byte[SALT_BYTES];
        random.nextBytes(salt);

        // Hash the password
        byte[] hash = pbkdf2(password, salt, PBKDF2_ITERATIONS, HASH_BYTES);
        // format iterations:salt:hash
        return PBKDF2_ITERATIONS + ":" + toHex(salt) + ":" +  toHex(hash);
    }
```

```
/**
 * Validates a password using a hash.
 *
 * @param   password    the password to check
 * @param   goodHash    the hash of the valid password
 * @return              true if the password is correct, false if not
 */
public static boolean validatePassword(String password, String goodHash)
    throws NoSuchAlgorithmException, InvalidKeySpecException
{
    return validatePassword(password.toCharArray(), goodHash);
}

/**
 * Validates a password using a hash.
 *
 * @param   password    the password to check
 * @param   goodHash    the hash of the valid password
 * @return              true if the password is correct, false if not
 */
public static boolean validatePassword(char[] password, String goodHash)
    throws NoSuchAlgorithmException, InvalidKeySpecException
{
    // Decode the hash into its parameters
    String[] params = goodHash.split(":");
    int iterations = Integer.parseInt(params[ITERATION_INDEX]);
    byte[] salt = fromHex(params[SALT_INDEX]);
    byte[] hash = fromHex(params[PBKDF2_INDEX]);
    // Compute the hash of the provided password, using the same salt,
    // iteration count, and hash length
    byte[] testHash = pbkdf2(password, salt, iterations, hash.length);
    // Compare the hashes in constant time. The password is correct if
    // both hashes match.
    return slowEquals(hash, testHash);
}

/**
 * Compares two byte arrays in length-constant time. This comparison method
 * is used so that password hashes cannot be extracted from an on-line
 * system using a timing attack and then attacked off-line.
 *
 * @param   a       the first byte array
 * @param   b       the second byte array
 * @return          true if both byte arrays are the same, false if not
 */
private static boolean slowEquals(byte[] a, byte[] b)
{
    int diff = a.length ^ b.length;
    for(int i = 0; i < a.length && i < b.length; i++)
        diff |= a[i] ^ b[i];
    return diff == 0;
}

/**
 *  Computes the PBKDF2 hash of a password.
 *
 * @param   password    the password to hash.
 * @param   salt        the salt
 * @param   iterations  the iteration count (slowness factor)
 * @param   bytes       the length of the hash to compute in bytes
```

```
     * @return                the PBDKF2 hash of the password
     */
    private static byte[] pbkdf2(char[] password, byte[] salt, int iterations,
        int bytes)
        throws NoSuchAlgorithmException, InvalidKeySpecException
    {
        PBEKeySpec spec = new PBEKeySpec(password, salt, iterations, bytes * 8);
        SecretKeyFactory skf = SecretKeyFactory.getInstance(PBKDF2_ALGORITHM);
        return skf.generateSecret(spec).getEncoded();
    }

    /**
     * Converts a string of hexadecimal characters into a byte array.
     *
     * @param   hex           the hex string
     * @return                the hex string decoded into a byte array
     */
    private static byte[] fromHex(String hex)
    {
        byte[] binary = new byte[hex.length() / 2];
        for(int i = 0; i < binary.length; i++)
        {
            binary[i] = (byte)Integer.parseInt(hex.substring(2*i, 2*i+2), 16);
        }
        return binary;
    }

    /**
     * Converts a byte array into a hexadecimal string.
     *
     * @param   array         the byte array to convert
     * @return                a length*2 character string encoding the byte array
     */
    private static String toHex(byte[] array)
    {
        BigInteger bi = new BigInteger(1, array);
        String hex = bi.toString(16);
        int paddingLength = (array.length * 2) - hex.length();
        if(paddingLength > 0)
            return String.format("%0" + paddingLength + "d", 0) + hex;
        else
            return hex;
    }

}
```

Chapitre 5

Conclusion

A l'issue d'un mois de travail intensif, nous pouvons affirmer que ce projet s'est terminé sans encombre et dans les délais. Nous avons atteint les objectifs que nous nous sommes fixés initialement et avons réalisé une solution répondant parfaitement au cahier des charges.

Nous estimons la période de réalisation de l'entièreté de l'application à 50 heures réparties comme suit : 5h pour l'analyse, 30 heures pour la conception de la base de données et 15h pour le développement de l'application java.

Nous avons eu l'opportunité, grâce à ce projet, d'améliorer et d'approfondir nos connaissances en SQL ainsi qu'à nous familiariser aux bonnes pratiques de jdbc, le driver SQL pour java. Nous avons également pu appliquer l'ensemble des savoir-faire acquis en cours de conception de bases de données.

Du point de vue humain, il nous a permis d'apprendre à mieux nous connaître. Nous avons appris à travailler ensemble de manière efficace en répartissant la charge de travail selon nos forces et faiblesses.

C'est donc pleinement satisfaits que nous délivrons ce projet aujourd'hui.