



TP-BDONN

Auteur : Otmane EL ALOI

Option : Informatique

Table des matières

0.1	Introduction	2
0.2	Présentation & Analyse du travail réalisé.	3
0.2.1	Modèle conceptuel de la base de données	3
0.2.2	Modèle physique de la base de données	4
0.2.3	Requêtes de création de la base de données	4
0.2.4	Requêtes d'insertion des données	5
0.2.5	Réponses aux messages	6
0.2.6	PL/SQL	13
0.2.7	JAVA	16
0.2.8	NoSQL avec MongoDB	17
0.3	Les difficultés rencontrées.	19
0.4	Analyse critique	20
0.5	Conclusion	20

Table des figures

1	Diagramme conceptuel de la base de données pour la gestion des commandes	3
2	Modèle physique correspondant au modèle conceptuel de la figure 1 . .	4
3	Requêtes pour la création des tables : camion et chauffeur	5
4	Requêtes d'insertion des données	5
5	Requête	6
6	Résultat de l'exécution de la requête de la figure 5	6
7	Ajout de la colonne <i>camion.date_disponibilite</i>	7
8	Requête mise à jour de la disponibilité d'un camion	7
9	Modification des données de disponibilité camion AC-543-AG	7
10	Requête pour avoir les camions disponibles et non attribué	8
11	Liste des camions non attribués et disponibles	8
12	Requête pour avoir les commandes en cours pour Novembre/Décembre avec leurs livraisons programmées	8
13	les commandes en cours pour Novembre/Décembre avec leurs livraisons programmées	9
14	Requête pour avoir les livraisons prévues en décembre pour le chauffeur Henry	9
15	Requête pour transformer la commande du 17 novembre en ordre de mission	9
16	Liste des ordre des missions	10
17	Vérification de la création de l'ordre de mission de Jacques WEBER . .	10
18	Ajout de la colonne type de camion	10
19	Ajout du nouveau chauffeur dans la table chauffeur	10
20	Résultat de l'ajout du nouveau chauffeur	11
21	Requête pour avoir le bilan d'activité de Henry le ROC'H	11
22	Requêtes pour la modification de la donnée certification	12
23	Résultat de la prise en considération des certifications de chauffeurs . .	12
24	Requêtes pour la création d'un trigger d'insertion	13
25	Requêtes pour la création d'un trigger de gestion des mises à jour des quantités produit dans une livraison.	14
26	Quantité initiale du produit 5 dans le quai de chargement 1	14
27	Requête de mise à jour	15
28	Quantité du produit 5 dans le quai de chargement après la mise jour de la commande.	15

29	Requêtes de création du trigger de suppression	16
30	Code pour récupérer les données du bordereau de livraison	17
31	Bordereau de livraison généré	17
32	POC python	18
33	Résultat de l'exécution du script de la figure 32	19

0.1 Introduction

Ce projet a pour but la mise en place de quelques bases de données pour une société fictive : la société BOFURI, c'est une entreprise de transport routier spécialisée dans la livraison de fruits et légumes.

La société BOFURI dispose d'une flotte de camions qui sillonnent les routes françaises en fonction des livraisons à effectuer. En premier lieu, elle veut mettre en place une base de données pour gérer les camions, les commandes et leurs livraisons correspondantes. Ensuite une autre base de données qui gère le fichiers des clients pour les commerciaux.

Ce rapport détaillera les importantes hypothèses considérées ainsi que les différentes démarches suivies pour monter les deux bases de données.

les codes et les fichiers utilisées sont dans mon GitHub dans le repo sauivant : https://github.com/otmane-el-aloi/BDONN_BOFURI

0.2 Présentation & Analyse du travail réalisé.

Dans cette partie, je détaillerai les différentes requêtes et modifications apportés au travail au fur et au mesure des messages qui arrivaient.

0.2.1 Modèle conceptuel de la base de données

Ci-dessous le modèle conceptuel de données réalisé.

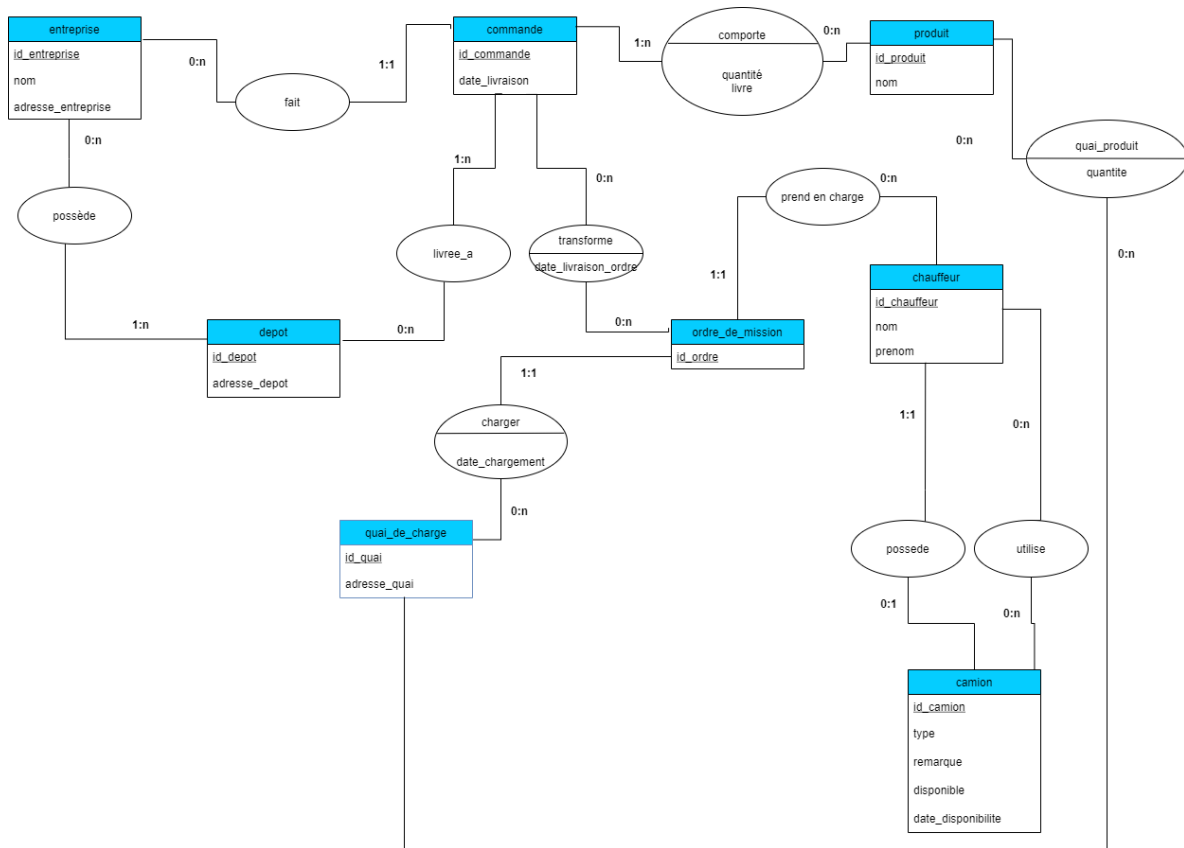


FIGURE 1 – Diagramme conceptuel de la base de données pour la gestion des commandes

La réalisation de ce modèle conceptuel était basé sur plusieurs hypothèses. Je citerai ci-dessous les plus pertinentes d'entre elles.

- Le chargement des produits dans un ordre de mission se fait dans un seul endroit ; chargement depuis un seul quai de chargement.
- La colonne disponible est ajouté pour faciliter la recherche des camions disponibles. Dans le cas de non disponibilité j'ai ajouté une colonne date disponibilité (qui peut avoir éventuellement des valeurs nulles vu qu'en général on ne connaît pas au moment de l'empanne la date de disponibilité (durée de réparation inconnue au début)).

- Une commande peut être associée à plusieurs ordre de mission. En effet, dans le cas où une commande n'a pas pu être livrée on peut l'associer à un autre ordre sans la supprimer du premier ordre déjà livré et ceci pour garder la trace de la première livraison non aboutie.

0.2.2 Modèle physique de la base de données

Le modèle physique ci-dessous a été réalisé dans **SQL Power Architect**.

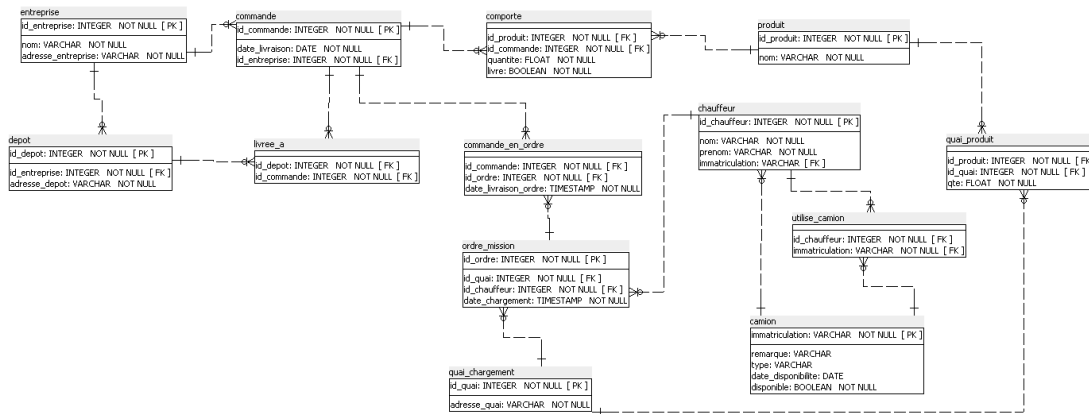


FIGURE 2 – Modèle physique correspondant au modèle conceptuel de la figure 1

0.2.3 Requêtes de création de la base de données

Les requêtes de création de la base de données sont générées automatiquement par **SQL power Architect**. La figure ci-dessous illustre quelques exemple de requête de création des tables.

```

CREATE TABLE public.camion (
    immatriculation VARCHAR NOT NULL,
    remarque VARCHAR,
    type VARCHAR,
    date_disponibilite DATE,
    disponible BOOLEAN NOT NULL,
    CONSTRAINT pk_camion PRIMARY KEY (immatriculation)
);

CREATE SEQUENCE public.chauffeur_id_chauffeur_seq;

CREATE TABLE public.chauffeur (
    id_chauffeur INTEGER NOT NULL DEFAULT nextval('public.chauffeur_id_chauffeur_seq'),
    nom VARCHAR NOT NULL,
    prenom VARCHAR NOT NULL,
    immatriculation VARCHAR,
    CONSTRAINT pk_chauffeur PRIMARY KEY (id_chauffeur)
);

ALTER TABLE public.chauffeur ADD CONSTRAINT camion_chauffeur_fk
FOREIGN KEY (immatriculation)
REFERENCES public.camion (immatriculation)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

```

FIGURE 3 – Requêtes pour la création des tables : camion et chauffeur

0.2.4 Requêtes d'insertion des données

La figure suivantes montre un exemple des requêtes exécutés dans **pgAdmin** pour remplir les tables de la base de données.

```

INSERT INTO camion VALUES
('AC-543-AG', NULL, NULL, NULL, TRUE),
('AD-671-KA', NULL, NULL, NULL, TRUE),
('AH-126-GG', NULL, NULL, NULL, TRUE),
('AM-654-TU', NULL, NULL, NULL, TRUE),
('BA-865-PF', NULL, NULL, NULL, TRUE),
('BA-921-AA', 'au garage', NULL, NULL, FALSE),
('CK-221-KW', NULL, NULL, NULL, TRUE),
('CL-128-TR', NULL, NULL, NULL, TRUE),
('CN-225-AB', NULL, NULL, NULL, TRUE);

INSERT INTO chauffeur VALUES
(1, 'DENT', 'Arthur', 'AC-543-AG'),
(2, 'LE ROC', 'H', 'Henry', 'BA-865-PF'),
(3, 'DUPUIS', 'Nathalie', 'AH-126-GG'),
(4, 'WEBER', 'Jacques', 'BA-921-AA'),
(5, 'EMOUCHET', 'David', 'AD-671-KA'),
(6, 'CE', 'NEDRA', 'Alexandra', 'CN-225-AB');

```

FIGURE 4 – Requêtes d'insertion des données

Remarque : Dans le cas des tables avec beaucoup de colonnes, il vaut mieux ajouter les noms de colonnes dans les requêtes d'insertion pour éviter de se tromper dans le remplissage. Dans mon cas les tables contenaient que peu de colonne ainsi je n'ai pas écrits les noms des colonnes dans les requêtes d'insertion.

0.2.5 Réponses aux messages

Dans cette section je présenterai les différentes requêtes exécutées sur la base de données afin de répondre aux messages correspondants.

Gestion des camions-chauffeurs correspondants (message 2) :

Dans la base de données, j'ai :

- Une table **camion** qui contient toutes les camions disponible et non disponible avec leurs types correspondants.
- Une table **chauffeur** qui contient les chauffeurs avec leurs camions associés.
- Et une table **utilise_camion** qui contient la liste des camions en utilisation par les chauffeurs. Ici, j'ai fais l'hypothèse qu'un système de pointage est déjà mis en place, et qui permet de donner l'information d'utilisation et de fin d'utilisation.

Étant donnée qu'on manque d'information sur l'utilisation des camions dans l'énoncé. Je peux fournir que la liste des chauffeurs avec leurs camions associés au lieux de la lise des camions effectivement en utilisation.

Ci-dessous la requête exécuté pour avoir la liste des chauffeur avec leur camion associés.

```
SELECT CONCAT(nom, ' ', prenom) AS chauffeur, immatriculation AS camion FROM chauffeur;
```

FIGURE 5 – Requête

On obtient la liste des chauffeurs et camion associés :

chauffeur text	camion character varying
DENT Arthur	AC-543-AG
LE ROC'H Henry	BA-865-PF
DUPUIS Nathalie	AH-126-GG
WEBER Jacques	BA-921-AA
EMOUCHET David	AD-671-KA
CE'NEDRA Alexandra	CN-225-AB

FIGURE 6 – Résultat de l'exécution de la requête de la figure 5

Gestion des disponibilités des camions (message 3) :

Le message indique qu'un camion ne serai plus disponible, L'arrivée de ce message, m'a poussé à ajouter la colonne suivante : *camion.date_disponibilite*.

La requête exécuté pour ajouter la nouvelle colonne est comme suit :

```
ALTER TABLE camion
ADD date_disponibilite VARCHAR;
```

FIGURE 7 – Ajout de la colonne *camion.date_disponibilite*

Une fois la nouvelle colonne ajoutée, j'ai modifié la disponibilité du camion en panne.

```
UPDATE camion SET remarque = 'au garage', date_disponibilite = '2021-11-30', disponible= FALSE
WHERE LTRIM(RTRIM(immatriculation)) = 'AC-543-AG';
```

FIGURE 8 – Requête mise à jour de la disponibilité d'un camion

Ci-dessous les résultats de la modification apporté aux données du camion **AC-543-AG**

immatriculation [PK] character varying	remarque character varying	type character varying	date_disponibilite date	disponible boolean
AD-671-KA	[null]	[null]	[null]	true
AH-126-GG	[null]	[null]	[null]	true
AM-654-TU	[null]	[null]	[null]	true
BA-865-PF	[null]	[null]	[null]	true
CK-221-KW	[null]	[null]	[null]	true
CL-128-TR	[null]	[null]	[null]	true
CN-225-AB	[null]	[null]	[null]	true
BA-921-AA	au garage	[null]	[null]	false
AC-543-AG	au garage	[null]	2021-11-30	false

FIGURE 9 – Modification des données de disponibilité camion **AC-543-AG**

Camions disponibles non utilisé par quelqu'un (message 4) :

Vu que le tableau utilise_camion ou doivent figurer les camions en utilisation est vide(par manque de l'information). J'utiliserai à sa place la table chauffeur qui contient les camions attribués aux chauffeur qui sont bien différents des camions effectivement en utilisation.

Ci-dessous la requête exécutée :

La figure suivante montre le résultat de la requête :

```
SELECT immatriculation FROM camion
WHERE camion.disponible = TRUE
EXCEPT
SELECT immatriculation FROM chauffeur;
```

FIGURE 10 – Requête pour avoir les camions disponibles et non attribué


immatriculation	
character varying	
AM-654-TU	
CL-128-TR	
CK-221-KW	

FIGURE 11 – Liste des camions non attribués et disponibles

Commandes en cours pour Novembre/Décembre avec leurs livraisons programmées éventuelles (message 5) :

Ci-dessous la requête exécutée :

```
SELECT commande.id_commande, commande.date_livraison, entreprise.nom as entreprise,
CASE
    WHEN commande_en_ordre.id_ordre ISNULL THEN 'livraison non programmée'
    ELSE CAST (commande_en_ordre.id_ordre as text)
END as id_livraison

FROM commande INNER JOIN entreprise ON (commande.id_entreprise = entreprise.id_entreprise)
LEFT JOIN commande_en_ordre ON (commande.id_commande = commande_en_ordre.id_commande)
WHERE date_part('month',commande.date_livraison) = '11' OR date_part('month', commande.date_livraison) = '12' ;
```

FIGURE 12 – Requête pour avoir les commandes en cours pour Novembre/Décembre avec leurs livraisons programmées

La figure suivant montre le résultat de l'exécution : **Remarque :** Je viens de remarquer qu'il faut aussi inclure l'année dans la condition de la requête. Parce qu'on demande que les commandes en cours.

Livraison prévues pour Henryy le ROC'h (message 6) :

Ci-dessous la requête exécutée :

id_commande integer	date_livraison date	entreprise character varying	id_livraison text
1	2021-11-15	KAEDE	1
3	2021-11-17	KUROMU	livraison non programmée
5	2021-11-22	KUROMU	3
4	2021-11-18	KASUMI	2
2	2021-11-15	KASUMI	1

FIGURE 13 – les commandes en cours pour Novembre/Décembre avec leurs livraisons programmées

```
SELECT chauffeur.nom as chauffeur, ordre_mission.id_ordre, ordre_mission.date_chargement
FROM ordre_mission INNER JOIN chauffeur ON (ordre_mission.id_chauffeur = chauffeur.id_chauffeur)
WHERE date_part('month', ordre_mission.date_chargement) = '12'
AND LTRIM(RTRIM(LOWER(chauffeur.nom))) LIKE 'henry le roc'h';
```

FIGURE 14 – Requête pour avoir les livraisons prévues en décembre pour le chauffeur Henry

Transformation d'une commande en un ordre de mission (message 7) :

Requête exécutée pour transformer la commande de 17 Novembre de la part de la société KUROM en ordre de mission affecté à Jacques WEBER.

```
do $$
DECLARE
idOrdre integer;
idChauffeur integer;
idQuai integer;
BEGIN
SELECT MAX(id_ordre) INTO idOrdre FROM ordre_mission;
idOrdre:= idOrdre + 1;
SELECT id_quai INTO idQuai FROM quai_chargement WHERE LTRIM(RTRIM(LOWER(adresse_quai))) LIKE '%nice%';
SELECT id_chauffeur INTO idChauffeur FROM chauffeur WHERE LTRIM(RTRIM(LOWER(nom))) LIKE 'weber';
INSERT INTO ordre_mission VALUES
(idOrdre, idQuai, idChauffeur, CAST('2021-11-17 07:00:00' as timestamp));
END;$$
```

FIGURE 15 – Requête pour transformer la commande du 17 novembre en ordre de mission

On vérifie que l'ordre de mission a bien été créer à l'aide de la requête qui suit. Étant donné qu'il n'y a pas beaucoup d'ordres, je sélectionne tout :

```
SELECT id_ordre, adresse_quai, CONCAT(chauffeur.nom, ' ', chauffeur.prenom) as chauffeur, ordre_mission.date_chargement
FROM ordre_mission INNER JOIN chauffeur ON (ordre_mission.id_chauffeur= chauffeur.id_chauffeur)
INNER JOIN quai_chargement ON (ordre_mission.id_quai = quai_chargement.id_quai);
```

FIGURE 16 – Liste des ordre des missions

L'exécution de la requête précédente montre que l'ordre pour Jacques WEBER a bien été créé :

id_ordre integer	adresse_quai character varying	chauffeur text	date_chargement timestamp without time zone
1	5 allée Beltegeuse Soumoulou France	DENT Arthur	2021-11-15 06:00:00
2	5 allée Beltegeuse Soumoulou France	CE'NEDRA Alexandra	2021-11-15 08:00:00
3	15 rue des rochers Metz France	EMOUCHET David	2021-11-22 08:00:00
4	10 boulevard des marins Nice France	WEBER Jacques	2021-11-17 07:00:00

FIGURE 17 – Vérification de la création de l'ordre de mission de Jacques WEBER

Message 8 : Le message 8 devrait être redirigé vers les personnes concernées.

Prise en considération des types des camions (message 11) :

L'arrivée de ce message à exigé l'ajout d'une nouvelle colonne dans la table camion qui est *camion.type*. Pour ajouter cette nouvelle colonne, j'ai exécuté la requête suivante :

```
ALTER TABLE camion
ADD "type" VARCHAR;
```

FIGURE 18 – Ajout de la colonne type de camion

Dans ce même message, on déclare l'arrivée d'un nouveau chauffeur. Ainsi je l'ai ajouté dans la base donnée. Pour le moment, il ne dispose pas de camion attribué, ainsi l'ajout se fait comme suit :

```
INSERT INTO chauffeur VALUES
(7, 'Pierre', 'KIMOUS', NULL);

SELECT * FROM chauffeur;
```

FIGURE 19 – Ajout du nouveau chauffeur dans la table chauffeur

On remarque que le chauffeur à bien été ajouté à la table chauffeur :

id_chauffeur [PK] integer	nom character varying	prenom character varying	immatriculation character varying
1	DENT	Arthur	AC-543-AG
2	LE ROC'H	Henry	BA-865-PF
3	DUPUIS	Nathalie	AH-126-GG
4	WEBER	Jacques	BA-921-AA
5	EMOUCHET	David	AD-671-KA
6	CE'NEDRA	Alexandra	CN-225-AB
7	Pierre	KIMOUS	[null]

FIGURE 20 – Résultat de l'ajout du nouveau chauffeur

Bilan d'activité de Henry le ROC'H (message 12) :

Ci-dessous la requête exécutée pour récupérer toutes les livraisons effectués par Henry le ROC'H.

```
SELECT id_ordre, adresse_quai, CONCAT(chauffeur.nom, ' ', chauffeur.prenom) as chauffeur, ordre_mission.date_chargement
FROM ordre_mission INNER JOIN chauffeur ON (ordre_mission.id_chauffeur= chauffeur.id_chauffeur)
INNER JOIN quai_chargement ON (ordre_mission.id_quai = quai_chargement.id_quai)
WHERE EXTRACT(YEAR FROM ordre_mission.date_chargement) = '2021' AND LTRIM(RTRIM(LOWER(chauffeur.prenom))) = 'henry'
AND LTRIM(RTRIM(LOWER(chauffeur.nom))) = 'le roc'h';
```

FIGURE 21 – Requête pour avoir le bilan d'activité de Henry le ROC'H

La requête renvoie un tableau vide ce qui signifie que Henry le ROC'H n'a pas effectué des livraison pour l'année en cours.

Extension des activités de l'entreprise : utilisation des camions frigorifiques de gros gabarit message (13) :

L'arrivée de ce message impose l'ajout d'une nouvelle colonne *certification* dans la table chauffeur. La colonne ajoutée est de type **VARCHAR**. Cette donnée contiendra les certifications que le chauffeur a obtenu pour la conduite des nouveaux camions de gros gabarit.

Pour l'ajouter on exécute la requête d'ajout comme pour les messages 3 et 11.

Certificat des camions frigorifiques et double essieu (message 14 15) :

Suite à la réception des certifications des deux conducteurs David et Alexandra, j'ai exécuté les requêtes suivantes qui permettent en premier lieu d'ajouter le type de certification **regulier** pour tous les chauffeurs. Et ensuite, les certifications particulières pour les deux chauffeurs qui les ont envoyées.

```

UPDATE chauffeur
SET certification = 'regulier';

UPDATE chauffeur
SET certification = 'regulier, frigorifiques, double essieu'
WHERE chauffeur.nom = 'CE' 'NEDRA' AND chauffeur.prenom = 'Alexandra';

UPDATE chauffeur
SET certification = 'regulier, frigorifiques'
WHERE chauffeur.nom = 'EMOUCHET' AND chauffeur.prenom = 'David';

SELECT * FROM chauffeur;

```

FIGURE 22 – Requêtes pour la modification de la donnée certification

Les résultats de la modification sont comme suit :

id_chauffeur [PK] integer	nom character varying	prenom character varying	immatriculation character varying	certification character varying
1	DENT	Arthur	AC-543-AG	regulier
2	LE ROC'H	Henry	BA-865-PF	regulier
3	DUPUIS	Nathalie	AH-126-GG	regulier
4	WEBER	Jacques	BA-921-AA	regulier
7	Pierre	KIMOUS	[null]	regulier
6	CE'NEDRA	Alexandra	CN-225-AB	regulier, frigorifiques, double essieu
5	EMOUCHET	David	AD-671-KA	regulier, frigorifiques

FIGURE 23 – Résultat de la prise en considération des certifications de chauffeurs

Changement de la réglementation : plus de palettes de 200kg (message 16) : Aucun changement n'a été effectué avec l'arrivée de ce message. En effet, dans ma modélisation, je ne prends pas en considération la taille des palettes. Car c'est une information qui ne peut pas être gérée au préalable. Ce n'est qu'au moment du chargement qu'on décide quelle palette utiliser pour charger la marchandise.

Le listing des produits demandé par Jacques WEBER (message 19) : La demande de Jacques WEBER n'est pas professionnelle et en contradiction avec la loi. Les données de l'entreprise ne devraient pas être divulguées à des parties externes. Sans l'accord de l'entreprise.

0.2.6 PL/SQL

Après l'ajout des données des quantités des produits sur chaque quai, on ajoute les triggers et fonctions triggers.

Fonction trigger sur l'insertion d'une ligne de livraison qui déduit la quantité de produit à livrer :

Ce trigger est lancé lorsqu'une commande est passée en ordre de mission.

Ci-dessous le code pour le créer :

```
DECLARE
    produit_qte CURSOR FOR SELECT commande_en_ordre.id_commande, comporte.id_produit, comporte.quantite
    FROM ordre_mission INNER JOIN commande_en_ordre ON (ordre_mission.id_ordre=commande_en_ordre.id_ordre)
    INNER JOIN commande ON (commande_en_ordre.id_commande=commande.id_commande)
    INNER JOIN comporte ON (commande.id_commande = comporte.id_produit)
    INNER JOIN produit ON (comporte.id_produit=produit.id_produit)
    WHERE ordre_mission.id_ordre = NEW.id_ordre;

    quai_id INTEGER;
    ordre_id INTEGER;

    commande_id INTEGER;
    produit_id INTEGER;
    quantite_commande FLOAT;
    qte_quai FLOAT;

BEGIN
    quai_id := NEW.id_quai;
    ordre_id := NEW.id_ordre;
    OPEN produit_qte;
    LOOP
        FETCH produit_qte INTO commande_id, produit_id, quantite_commande;
        EXIT WHEN NOT FOUND;
        UPDATE quai_produit
        SET qte = qte - quantite_commande
        WHERE quai_produit.id_produit = produit_id AND quai_produit.id_quai = quai_id;
    END LOOP;
    CLOSE produit_qte;
    RETURN NEW ;
END ;

CREATE TRIGGER gestion_quantite_insertion
    AFTER INSERT
    ON commande_en_ordre
    FOR EACH ROW
```

FIGURE 24 – Requêtes pour la création d'un trigger d'insertion

Fonction trigger pour la mise à jour d'une ligne de livraison qui déduit la quantité de produit dans le quai de chargement :

Ce trigger est lancé lorsqu'on change la quantité d'un produit qui appartient à une commande déjà passée en livraison.

Ci-dessous le code pour le créer :


```

CREATE OR REPLACE FUNCTION gere_quantite_produit_update() RETURNS TRIGGER language 'plpgsql' AS
$BODY$
DECLARE
quai_id INTEGER;
BEGIN
    IF OLD.quantite != NEW.quantite THEN
        IF OLD.id_commande IN (SELECT id_commande FROM commande_en_ordre) THEN
            SELECT ordre_mission.id_quai INTO quai_id
            FROM ordre_mission INNER JOIN commande_en_ordre
            ON (ordre_mission.id_ordre = commande_en_ordre.id_ordre);

            UPDATE quai_produit
            SET quai_produit.qte = quai_produit.qte + OLD.quantite - NEW.quantite
            WHERE quai_produit.id_quai = quai_id AND OLD.id_produit = quai_produit.id_produit;
        END IF;
    END IF;
    RETURN NEW ;
END ;
$BODY$

CREATE TRIGGER gestion_quantite_update
AFTER UPDATE
ON commande
FOR EACH ROW
EXECUTE PROCEDURE gere_quantite_produit_update();

```

FIGURE 25 – Requêtes pour la création d'un trigger de gestion des mises à jour des quantités produit dans une livraison.

Pour tester le bon fonctionnement de ce trigger, j'ai modifié la quantité initiale du produit d'Id **5** dans la commande d'id **4** en passant de **100** kg en **50** kg. Le quai de chargement pour cette commande est le quai d'id 1, dans ce quai le produit d'id **5** avait pour quantité initial **400** (figure :26). Ainsi la quantité prévue après la mise à jour de la quantité commandée était 450.

id_produit integer	id_quai integer	qte real
5	1	400

FIGURE 26 – Quantité initiale du produit 5 dans le quai de chargement 1

En exécutant la requête de mise à jour (figure : 27) on vérifie bien qu'on obtient la quantité prévue.

```
UPDATE comporte
SET quantite = 50
WHERE id_commande = 4 AND id_produit = 5;
```

FIGURE 27 – Requête de mise à jour

id_produit	id_quai	qte
integer	integer	real
5	1	450

FIGURE 28 – Quantité du produit 5 dans le quai de chargement après la mise jour de la commande.

Fonction trigger pour la mise à jour des quantités des produits des livraisons supprimées :

Ce trigger est lancé lorsqu'un ordre de mission est supprimé (i.e suppression d'une livraison).

```

CREATE OR REPLACE FUNCTION gere_quantite_produit_surpression() RETURNS TRIGGER language 'plpgsql' AS
$BODY$
DECLARE
    produit_qte CURSOR FOR SELECT commande_en_ordre.id_commande, comporte.id_produit, comporte.quantite
    FROM ordre_mission INNER JOIN commande_en_ordre ON (ordre_mission.id_ordre=commande_en_ordre.id_ordre)
    INNER JOIN commande ON (commande_en_ordre.id_commande=commande.id_commande)
    INNER JOIN comporte ON (commande.id_commande = comporte.id_produit)
    INNER JOIN produit ON (comporte.id_produit=produit.id_produit)
    WHERE ordre_mission.id_ordre = OLD.id_ordre;

    quai_id INTEGER;
    ordre_id INTEGER;

    commande_id INTEGER;
    produit_id INTEGER;
    quantite_commande FLOAT;
    qte_quai FLOAT;

BEGIN
    quai_id := OLD.id_quai;
    ordre_id := OLD.id_ordre;
    OPEN produit_qte;
    LOOP
        FETCH produit_qte INTO commande_id, produit_id, quantite_commande;
        EXIT WHEN NOT FOUND;
        UPDATE quai_produit
        SET qte = qte + quantite_commande
        WHERE quai_produit.id_produit = produit_id AND quai_produit.id_quai = quai_id;
    END LOOP;
    CLOSE produit_qte;
    RETURN NEW ;
END ;
$BODY$;

CREATE TRIGGER gestion_quantite_surpression
    AFTER DELETE
    ON commande_en_ordre
    FOR EACH ROW
    EXECUTE PROCEDURE gere_quantite_produit_surpression();

```

FIGURE 29 – Requêtes de création du trigger de suppression

0.2.7 JAVA

Cette section du projet fournit un POC suite à la demande de Tatsuya Siba pour la récupération des données nécessaires pour la génération d'un bordereau de livraison et ceci à travers JAVA.

La figure suivante montre le code utilisé pour se connecter à la base de données avec le driver java postgresql. Et la requête employée pour récupérer les données du bordereau de livraison.

```

try {
    // Connection à la base de donnée
    Class.forName("org.postgresql.Driver");
    Connection connect = DriverManager.getConnection("jdbc:postgresql://localhost:5432/BOFURI_DATABASE", "admin", "0630199901");

    // Requête à exécuter dans la base pour récupérer les données de bordereau
    String query = "SELECT DISTINCT ordre_mission.id_ordre, entreprise.nom as entreprise, adresse_depot as depot, CONCAT(chauffeur.nom, ' ', '\n' +
        " chauffeur.prenom) as chauffeur, commande_en_ordre.date_livraison_ordre, \n" +
        " adresse_quai as quai_chargement, date_chargement, produit.nom as produit, comporte.quantite\n" +
        " FROM ordre_mission INNER JOIN quai_chargement ON (ordre_mission.id_quai=quai_chargement.id_quai)\n" +
        " INNER JOIN commande_en_ordre ON (ordre_mission.id_ordre=commande_en_ordre.id_ordre)\n" +
        " INNER JOIN chauffeur ON (ordre_mission.id_chauffeur = chauffeur.id_chauffeur)\n" +
        " INNER JOIN commande ON (commande.id_commande = commande_en_ordre.id_commande)\n" +
        " INNER JOIN livree_a ON (commande.id_commande = livree_a.id_commande)\n" +
        " INNER JOIN depot ON (livree_a.id_depot = depot.id_depot)\n" +
        " INNER JOIN entreprise ON (commande.id_entreprise = entreprise.id_entreprise)\n" +
        " INNER JOIN comporte ON (comporte.id_commande = commande.id_commande)\n" +
        " INNER JOIN produit ON (comporte.id_produit = produit.id_produit)\n" +
        " WHERE ordre_mission.id_ordre = ?";

    PreparedStatement stmt = connect.prepareStatement(query);
    stmt.setInt(1, id_ordre);
    ResultSet res = stmt.executeQuery();

    while (res.next()) {
        System.out.println("Chauffeur : " + res.getString("chauffeur"));
        System.out.println("Date de chargement : " + res.getString("date_chargement"));
        System.out.println("Quai de Chargement : " + res.getString("quai_chargement"));
        break;
    }

    // itérer sur toutes les commandes dans l'ordre
    System.out.println("Nom      Adresse                               Date Livraison      Nom Produit  Quantité Livrée");
    while (res.next()) {
        System.out.println(res.getString("entreprise")+" "+res.getString("depot")+" "+res.getString("date_livraison_ordre")+" "+res.getString("produit")+" "+res.getString("quantite"));
    }
    stmt.close();
    connect.close();
} catch (java.lang.ClassNotFoundException e) {
    System.err.println("ClassNotFoundException : " + e.getMessage());
} catch (SQLException ex) {
    System.err.println("SQLException : " + ex.getMessage());
}

```

FIGURE 30 – Code pour récupérer les données du bordereau de livraison

L'exécution du code précédent donne le résultat suivant, on vérifie que c'est bien les mêmes données fournies dans l'énoncé.

```

Chauffeur : EMOUCHET David
Date de chargement : 2021-11-22 08:00:00
Quai de Chargement : 15 rue des rochers Metz France
Nom      Adresse                               Date Livraison      Nom Produit  Quantité Livrée
KUROMU 1 rue de la braderie Lille France 2021-11-22 16:00:00 Orange      400

```

FIGURE 31 – Bordereau de livraison généré

0.2.8 NoSQL avec MongoDB

Cette partie du projet concerne la création d'une base de données NoSQL avec mongoDB pour la gestion des fiches des collaborateurs.

Le travail réalisé se divise en 3 partie :

- La création de la base de données **ficheCollaborateur** en utilisant **mongosh**.
- La définition d'un utilisateur pour cette base avec un mot de passe.
- Création d'un POC d'utilisation avec python en exploitation le package **py-mongo**. Le code pour le POC est fournie en pièce jointe.

```
CONNECTION_URL= "mongodb://127.0.0.1:27017/"
FICHES =[ fiche1, fiche2, fiche3, fiche4, fiche5, fiche6]

def connect(connection_url):
    client = MongoClient(connection_url)
    db = client["ficheCollaborateur"]
    fiches = db["fiches"]
    return fiches

if __name__ == "__main__":

    # Connection à la base de donnée:
    fiche = connect(CONNECTION_URL)

    # # Insertion des fiches:
    # fiche.insert_many(FICHES)

    # Le nombre total de fiches:
    print("Nombre de documents: ", fiche.count_documents({}))

    # Nombre de fiches de chaque collaborateur:
    print("Nombre de fiches de Alice DUMOND",
        | fiche.count_documents({"nomCollaborateur": "DUMOND", "prenomCollaborateur": "Alice"}))

    print("Nombre de fiches de Pascal LELIEVRE",
        | fiche.count_documents({"nomCollaborateur": "LELIEVRE", "prenomCollaborateur": "Pascal"}))

    print("Nombre de fiches de Laetitia DUPOND",
        | fiche.count_documents({"nomCollaborateur": "DUPOND", "prenomCollaborateur": "Laetitia"}))

    # Fiche de Jacques REY:
    ficheClient = fiche.find({"nomClient": {'$regex': '\s*(?i)rey\s*'}, "prenomClient": {'$regex': "\s*(?i)Jacques\s*"}})
    print("Fiche de Jaques REY: ")
    for f in ficheClient:
        | pprint(f)

    # Fiche d'entreprise "Primeur & co"
    ficheClient = fiche.find({"societe": {'$regex': "\s*(?i)primeur\s*&\s*(?i)co"}})
    print("Fiche de la société Primeur & co: ")
    for f in ficheClient:
        | pprint(f)
```

FIGURE 32 – POC python

Le script ci-dessous permet de :

- Se connecter à la base de données mongoDB.
- Ajouter des documents(fiches clients) à la base de données.
- Calculer le nombre des documents. Calculer le nombre des documents par collaborateur. Fournir les fiches correspondants à des critères de recherche.

```

Nombre de documents: 6
Nombre de fiches de Alice DUMOND 2
Nombre de fiches de Pascal LELIEVRE 2
Nombre de fiches de Laetitia DUPOND 2
Fiche de Jaques REY:
{'_id': ObjectId('61844086c80d1faabb5123c2'),
 'email': 'Jacques.Rey@fruitcompany.com',
 'nomClient': 'REY',
 'nomCollaborateur': 'DUMOND',
 'prenomClient': 'Jacques',
 'prenomCollaborateur': 'Alice',
 'resultat': 'Attente avis',
 'societe': 'My Fruit Company',
 'tel': '+33 6 65 89 56 34'}
Fiche de la société Primeur & co:
{'_id': ObjectId('61844086c80d1faabb5123c4'),
 'date': '12/03/2019',
 'email': 'primerandco@gmail.com',
 'nomClient': 'DIMITRIEVSKI',
 'nomCollaborateur': 'LELIEVRE',
 'prenomClient': 'Vladimir',
 'prenomCollaborateur': 'Pascal',
 'resultat': 'A relancer',
 'societe': 'Primeur & co'}
{'_id': ObjectId('61844086c80d1faabb5123c7'),
 'date': '4/05/2019',
 'nomClient': 'MARKOVA',
 'nomCollaborateur': 'DUPOND',
 'prenomClient': 'Alexandra',
 'prenomCollaborateur': 'Laetitia',
 'resultat': 'A recontacter en fin d'année',
 'societe': 'Primeur & co',
 'tel': '06 87 81 20 74'}

```

FIGURE 33 – Résultat de l'exécution du script de la figure 32

0.3 Les difficultés rencontrées.

Le montage de ce projet a connu certaines difficultés, étant donné que les consignes changeaient avec l'arrivée d'un message. Ce qui a exigé dans quelques cas de repartir du modèle conceptuel et donc refaire toutes le projet. Ce qui était pas si mal, vu que j'ai utilisé **SQL Power Architect** qui est un logiciel qui automatise la génération du code SQL pour la création de la base de données.

Par contre, la difficulté c'était de rechanger à chaque fois la manière avec laquelle j'entre les données dans les requêtes d'insertion, car lorsque la table change (exemple : sup-

pression d'une colonne) il faut changer aussi la requête d'insertion correspondante en supprimant la colonne en question.

Une autre difficulté était liée à la modélisation des adresses dans la base de données, ma première approche c'était de considérer une table adresse avec ses attributs correspondants (numéro de rue, nom de la rue, ville, pays...) et de l'associer aux tables concernées par une adresse, comme les tables entreprise, depot et quai de chargement. Cette première approche fonctionnait mais il m'a créé un soucis dans les requêtes qui devenaient très lourdes à écrire quand on voulait récupérer une adresse. Pour s'en passer de cette perte de temps, j'ai refais le modèle conceptuel en ajoutant tout simplement des attributs adresse dans les tables concernés, comme ça toute l'adresse est renseignée comme une seule chaîne de caractère dont la récupération est directe et beaucoup moins lourde.

0.4 Analyse critique

Dans cette partie je citerai les différentes améliorations que j'estime dans le futur.

- La partie gestion des camions devrait être amélioré en employant des *triggers* qui renseigneront automatiquement la disponibilité d'un camion dans la table camion lorsque le celui-ci est en utilisation par un chauffeur (ceci correspond à une insertion dans la table utilise_camion) ou lorsqu'un chauffeur déclare la fin d'utilisation d'un camion (ceci correspond à une suppression dans la table utilise_camion).

0.5 Conclusion

Ce TP m'a permis de se familiariser avec les notions de base concernant les bases de données relationnelles et non relationnelles, et les méthodes d'accès à ces bases de données en utilisant de la programmation par Java et python.

Mais le plus important, c'est que le TP traite un problème réaliste d'une entreprise fictive, ce qui met en évidence le travail d'un ingénieur au sein de la DSI.