

База даних для роботи фірми  
з виробництва віконних систем

Виконав: Бурець В.В.

## **Зміст**

1.1 Постановка задачі обліку на виробництві віконних систем	4
1.2 Опис предметної області	5
1.3. Специфікація вимог для системи управління вироництва віконних систем	11
1.3.1. Вступ	11
1.3.2. Загальний опис	12
1.3.3. Характеристики системи	14
4.3.2. Вимоги зовнішніх інтерфейсів	21
4.3.3. Інші нефункціональні вимоги	23
2.1. Концептуальне моделювання предметної області	25
2.1.1. Опис сутностей	25
2.1.2. Опис зв'язків між сутностями	33
2.2. Логічне проектування БД	36
2.3. Проектування типових запитів і транзакцій	39
3.1. Реалізація доступу до бази даних	42
3.2. Реалізація функціональних характеристик системи	44
3.2.1. Реєстрація замовлення	44
3.2.2. Сортування даних	45
3.2.3. Фільтрація даних	46
3.2.4. Додавання персони	47
3.2.3. Редагування даних персони	50
3.2.5. Додавання продукту до замовлення	51
3.2.6. Додавання продукту до замовлення	53

3.2.7. Додавання нового матеріалу на склад	54
3.2.8. Додавання відомостей про заробітну плату працівникам	56
3.2.9. Створення заявки на потребу нового матеріалу для постачання.....	57
3.2.10. Додавання нового працівника	58
3.2.11. Обрахунок вартості продукту та замовлення	60
3.2.12. Створення нової поставки товарів	61
3.2.13. Оплата замовлення	62
3.3. Опис роботи програми	63
Висновки .....	69
Використана література.....	70
Додаток А(Скрипт створення бази даних).....	71
Додаток Б(Скрипти створення транзакцій, функцій).....	73
Додаток В(Результат перевірки схеми БД в Erwin Data Modeler).....	87

## РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ОБЛІКУ НА ВИРОБНИЦТВІ ВІКОННИХ СИСТЕМ

### 1.1 Постановка задачі обліку на виробництві віконних систем

#### **Вступ**

Під час виробництва віконних систем дуже часто виникають труднощі у зберіганні інформації про нові замовлення, облік матеріалів, заробітну плату працівників.

#### **Аналіз та опис предметної області**

Ціллю будь-якого виробництва є продукт. Тобто нам потрібна програмна система яка б дозволяла вести облік матеріалів для виготовлення продукту та вести облік замовлень даного продукту. Продуктом є віконні системи. Тобто виробництво вікон є *предметною областю*

Ціллю роботи будь-якого виробництва є виготовлення продукту, отже виготовлення віконних систем, співпраця усіх його відділів і буде нашою **предметною областю**.

База даних для фірми з виробництва віконних систем призначена для інженерів та інших співробітників фірми і повинна забезпечувати доступ до інформації про замовлення, матеріали на складі, тощо, що в свою чергу суттєво скоротить час на обробку і пошук необхідної інформації.

Якщо коротко будь-яке виробництво складається із декількох основних процесів:

- прийняття замовлення від клієнта;
- виконання робіт згідно із замовленням;
- нарахування заробітної плати працівникам

*Прийняття замовлення* - отримання інформації про замовлений продукт та інформація про замовника, а також укладання з ним угоди/домовленості щодо кінцевого результату, фінансових затрат. А також вирішення ряду технічних питань, характеру робіт.

*Виконання робіт* - безпосереднє виробництво.

*Нарахування заробітної плати працівникам* – нарахування заробітної плати працівникам за виконаний обсяг робіт.

## 1.2 Опис предметної області

Основною діяльністю підприємства із виробництва віконних систем є їх виробництво. У більшості підприємств цієї галузі кожен продукт є унікальним і виготовляється під замовлення. Відповідно діяльність підприємства складається із двох основних складових:

1. Прийняття замовлення з уточненням всіх особливостей виконання замовлення;
2. Виготовлення замовлення
3. Нарахування заробітної плати відповідно до обсягу виконаних робіт.

Головними дійовими особами є:

1. Робітник цеху;
2. Бухгалтер;
3. Менеджер;

Працівники, менеджери, бухгалтери чи інші відповідальні особи, які мають справу із виробництвом. Їхня мета – автоматизувати та пришвидшити процеси, збільшити продуктивність підприємства та отримувати детальні звіти про всі процеси, які дадуть можливість бачити реальну картину та поточний стан бізнесу.

Відповідно до статусу працівника при автентифікації йому надається певні права доступу.

*Робітник цеху* складає список необхідних матеріалів, має доступ до панелі керування виробничим процесом.

*Бухгалтер* нараховує зарплату, має можливість змінити схему нарахування зарплати.

*Менеджер* - працівник фірми, який реєструє нове замовлення та працює із замовником. А також може додати новий продукт до заданого списку на складі.

Менеджери співпрацюють із замовниками і складають замовлення. Для створення замовлення менеджер вносить необхідну інформацію до програми. Це контактні дані замовника та продукт, який замовляє він. Продукт формується на основі вимог замовника. Це матеріали із якого виготовляти віконну систему, фурнітура, скло, розміри та інші характеристики.

Вартість замовлення обраховується шляхом сумування вартості всіх матеріалів і вартості роботи робітників, які були залучені до виконання замовлення. До отриманої суми додається 30% отриманої суми це і є вартість замовлення.

Важливим етапом роботи виробництва є своєчасний моніторинг наявності та закупівля усіх необхідних для виготовлення матеріалів.

Основними дійовими особами на цьому етапі є робітник цеху, менеджер закупівель та постачальник. Під час виготовлення продукту робітник цеху складає список всіх необхідних матеріалів та витратних матеріалів, які йому необхідні для проведення робіт. У випадку відсутності необхідних ресурсів робітник цеху звертається до менеджера закупівель, який напряду працює з постачальниками.

В обов'язки менеджера закупівель входить безпосередня робота з постачальником. Менеджер має доступ до інформації про постачальників, які співпрацюють з фірмою.

Також в обов'язки менеджера входить пошук та внесення в БД інформації про нових постачальників товарів та перелік товарів для виготовлення, які вони надають.

Після виконання замовлення відбувається нарахування заробітної плати працівникам залученим у виконання замовлення відповідно до обсягу виконаних робіт. Обрахування сум виконується таким чином: 30% від вартості замовлення розподіляються між працівниками. Відповідно із отриманого прибутку менеджер отримує 30%, робітник цеху 65%, бухгалтер 5%.

#### Постановка завдання

Програмний продукт, для якого призначена наступна база даних, буде використовуватися фірмами спеціалізованими на виробництві вікон. Доступ до бази даних будуть мати тільки працівники компанії.

#### Користувачі:

- Менеджер по роботі з клієнтами – особа, що безпосередньо контактує з клієнтами, формує перелік зазначених клієнтом побажань. Зокрема визначає особливості кожної віконної системи. Уточнює вимоги щодо кожного конкретного замовлення. Та здійснює оформлення замовлення. Надає клієнтам інформацію про орієнтовні терміни виконання робіт та їх вартість. особа, яка може додати новий продукт до заданого списку на складі.
- Робітник цеху – особа, що керує виробництвом віконних систем. Зокрема дає завдання кожному робітнику цеху згідно із замовленням. Має доступ до панелі керування виробничим процесом. Може додавати нових робітників на виробництво. Може створити список необхідних матеріалів необхідних для виготовлення.
- Бухгалтер – може обчислити заробітну плату для всіх працівників фірми.

### Сутності:

- **Робітник** – містить персональну та контактну інформацію про працівника, який бере участь у виготовленні продукту(вікна) і вартість його роботи погодинно.
- **Продукт(віконна система)** – містить детальну інформацію про продукт, час за який буде виготовлений цей виріб, а також про робітника, який виготовив цей виріб. До характеристик продукту відносяться його ширина, висота, колір, матеріали(профіль, фурнітура, скло) з яких він виготовлений.
- **Профіль** – містить інформацію(країну походження, назву, ціну, кількість на складі, категорію товару) про профіль(віконний матеріал), який буде використаний для виробництва вікна.
- **Скло** - містить інформацію(країну походження, назву, ціну, наявність контурного малюнку, кількість на складі, категорію товару) про скло(віконний матеріал), який буде використаний для виробництва вікна.
- **Фурнітура** - містить інформацію(країну походження, назву, ціну, кількість на складі, категорію товару) про фурнітуру(віконний матеріал), яка буде використана для виробництва вікна.
- **Замовлення** – містить інформацію про замовника, замовлений товар, час замовлення,.
- **Колір** – містить список доступних кольорів виробу.
- **Країна** - містить список країн походження кожного товару.
- **Тип** – містить список типів матеріалів.
- **Постачальник** – містить назву компанії, адресу, контактну інформацію(номер телефона, E-mail), перелік наявних поварів.

### Бізнес-процеси:

- Реєстрація нового клієнта;
- Додавання матеріалів до списку;
- Оформлення замовлення;
- Облік усіх замовлень;
- Облік працівників та їх роботи;
- Облік матеріалів;



**Бізнес-правила:**

- Працівник може працювати лише над одним замовленнями.
- На кожне замовлення укладається договір «Про виготовлення віконної системи».
- Над одним замовленням може одночасно працювати кілька працівників.
- Терміни виконання замовлення можуть змінюватися.
- Вартість виконання замовлення може змінюватися.
- Зарплата працівника залежить від кількості відпрацьованих ним годин.

**Функціонал ПЗ**

1. Створення/оновлення/перегляд/видалення інформації про працівників.
2. Створення/оновлення/перегляд/видалення інформації про клієнтів.
3. Створення/оновлення/перегляд/видалення інформації про постачальників матеріалів.
4. Перегляд історії виготовлених віконних систем.
5. Облік матеріалів на складі.
6. Ведення фінансових записів.

**Вхідні дані:**

- Інформація про замовника:
  - Ім'я
  - Прізвище
  - Номер телефону
- Інформація про матеріали:
  - Назва
  - Ціна

- країна походження
- Тип(ціновий клас)
- Інформація про продукт:
  - Тип профілю
  - Тип фурнітури
  - Тип скла
  - Колір
  - Ціна
  - Робітник який складав
- Інформація про замовлення
  - Час замовлення
  - Замовник
  - Номер замовлення
- Інформація про працівника:
  - Ім'я
  - Прізвище
  - Вартість роботи

**Вихідні дані:**

- Рахунок за виготовлений товар
- Нарахована працівникам зарплатня

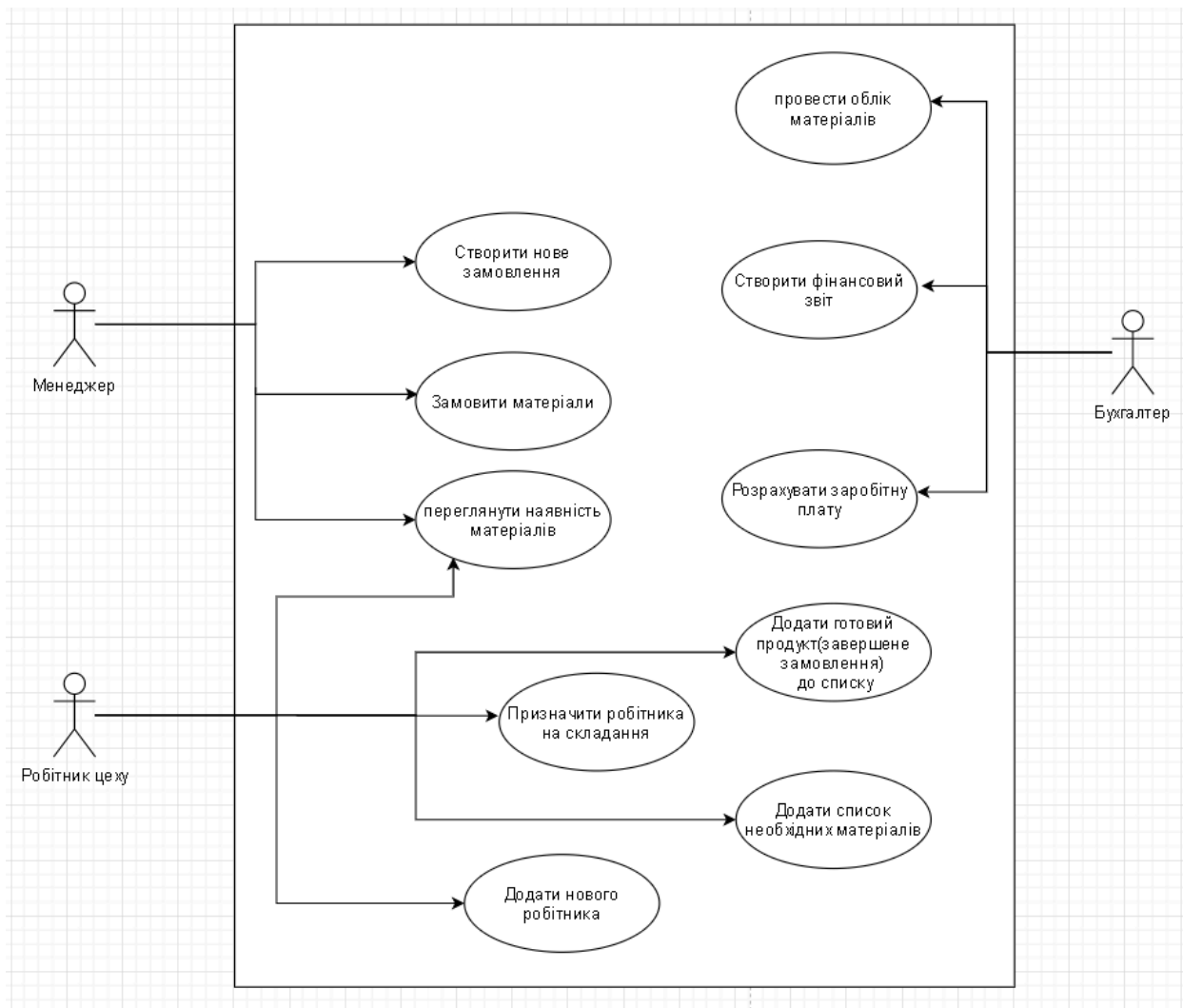


Рис. 1. Діаграма прецедентів для заданої системи

### 1.3. Специфікація вимог для системи управління виробництва віконних систем

#### 1.3.1. Вступ

##### 1.3.1.1. Призначення, мета

WindowFactory – програмний продукт для управління виробництвом, закупівлею, користувацькими замовленнями та персоналом для виробництва.

### 1.3.2. Загальний опис

#### 1.3.2.1. Перспективи продукту

Даний продукт призначений для автоматизації обліку руху коштів(нарахування зарплати, облік закупівель, оплата замовлень, а також облік складу) на виробництві. Зокрема на виробництві вікон. Хоч і на ринку уже присутні схожі продукти, але на місцевому (Україна та Східна Європа) Factory Management Studio має можливість стати новим та потужним (поєднує управління декількох бізнес-процесів) програмним забезпеченням.

#### 1.3.2.2. Характеристики продукту

Цей програмний продукт дає можливість відстежувати всі закупівельні замовлення, створювати замовлення на виготовлення (із врахуванням виконавців: менеджера, працівника, матеріали). Також є можливість керувати персоналом та такими їхніми показниками: кількістю відпрацьованих годин, кількістю завершених замовлень, кількістю вихідних днів та заробітною платою.

Важливою підсистемою цього продукту є управління користувацьким замовленнями. Даний продукт автоматично відстежує необхідні продукти для замовлення.

#### 1.3.2.3. Припущення та залежності

Система буде мати залежності від фреймворку для розробки (ASP.NET та бази даних SQL SERVER), від додаткових бібліотек користувацького інтерфейсу, хмарних сховищ та інших сервісів.

#### 1.3.2.4. Середовище функціонування

Даний програмний продукт є застосунком, який може бути розгорнутий на платформах Windows. Для коректної роботи потрібно встановити Microsoft

SQL Server, щоб працювати з базою даних.

*Таблиця 1.1.*

Вимоги до платформи

Процесор	32-х бітний або 64-х бітний процесор з тактовою частотою 1 ГГц або більше.
Пам'ять	1 Gb (рекомендовано не менше 2 Gb для 64-розрядних систем із можливістю подальшого збільшення об'єму відповідно до збільшення бази даних).
Операційна система	ОС Windows (Windows 7 – Windows 10) або Linux 4.
Додаткові програмні компоненти	Microsoft .NET Core 2.1, а також Microsoft SQL Server 2012 і вище для організації бази даних, а також роботи з системою керування базою
Інтернет	Наявність доступу до мережі інтернет є бажаною для своєчасного оновлення компонентів системи.

#### 1.3.2.5. Обмеження проектування та реалізації

Продукт повинен бути реалізований впродовж 6 місяців після старту розробки. Система є кросплатформною і не повинна вимагати наявності додаткових програмних компонентів крім Microsoft .NET, а також Microsoft SQL Server.

#### 1.3.2.6. Документація користувача

Для користувача буде доступна документація, в якій буде представлено приклади використання основних функцій даного програмного продукту та

контактні дані (email) для технічної підтримки.

#### 1.3.2.7. Припущення та залежності

Для коректної роботи продукту потрібно мати встановленим екземпляр Microsoft SQL Server версії 2012 і вище, а також потрібно встановити Microsoft .NET Core 2.1 або новішої версії.

#### 1.3.3. Характеристики системи

##### 1.3.3.1. Реєстрація користувача

##### 1.3.3.1.1. Опис і пріоритет

Перед початком роботи в системі користувача адміністратор повинен мати можливість зареєструвати його.

Пріоритет – високий.

Надання користувачу можливість керувати та відслідковувати вхідні замовлення.

Пріоритет характеристики – високий.

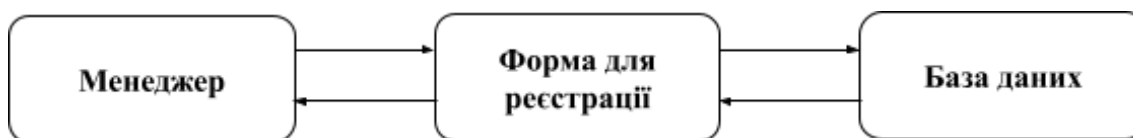
##### 1.3.3.1.2. Послідовності дія/відгук

«Вхідні замовлення» – виведення списку всіх поточних вхідних замовлень.

Позначення замовлення як виконаного – перенесення у статус «Виконано», архівування та оновлення списку продуктів на складах.

«Скасування замовлення» – відміна всіх змін, спричинених цим замовленням.

«Розрахувати зарплату» - обрахування заробітної плати для заданого робітника чи списку робітників за встановлений період.



#### 1.3.3.1.3. Функціональні вимоги

REQ-1.1 Забезпечення продовження роботи при введенні неприпустимих даних.

REQ-1.3. Сортування доступних готових продуктів при натисканні на поле із назвою параметра таблиці. При першому натисканні відбувається сортування за зростанням параметра, при повторному натисканні на те ж поле сортування відбувається за спаданням параметра.

REQ-1.4. Пошук продуктів (матеріалів для виготовлення вікон) за вказаними параметрами. У діалоговому вікні додавання продуктів повинен бути випадаючий список, в якому користувач обирає параметр для пошуку, та рядок пошуку для введення значення шуканого параметра.

REQ-1.5: Забезпечення можливості автоматично формувати замовлення (коли є всі необхідні продукти/вироби) та сповіщати користувача.

REQ-1.6: Можливість автоматично генерувати замовлення на виготовлення певних продуктів (при нестачі) з обов'язковим підтвердженням користувача.

REQ-1.7: Відслідковування статусу замовлень: «Створено», «У ході виконання», «Скасовано замовником», «Скасовано постачальником» та «Виконано». Для кожного статусу при необхідності мати можливість вносити пояснення.

REQ-1.8: Можливість генерувати автоматичні звіти за різними критеріями: кількість замовлень із описами, замовлення за статусом виконання, замовлення за певний період часу.

### 1.3.3.2. Управління персоналом

#### 1.3.3.2.1. Опис і пріоритет

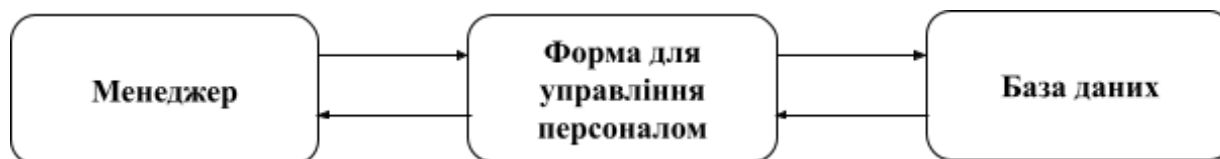
Ця характеристика програмної системи передбачає надання користувачу можливість керувати процесом відслідковування виконаної роботи працівників, ефективно створювати завдання для певної групи працівників, керувати процесом виплати заробітної плати. Пріоритет характеристики – високий.

#### 1.3.3.2.2. Послідовності дія/відгук

2. Вибір поточних виконуваних завдань певною групою працівників– вивід списку всіх замовлень на виготовлення із статусом «У ході виконання».

3. Вибір завершених завдань певною групою працівників – вивід списку всіх замовлень на виготовлення із статусом «Виконано».

4. Вибір опису певного працівника – вивід детальної інформації про нього із списком виконуваних завдань, опціональним списком виконаних, історія заробітної плати та кількість відпрацьованих годин за поточний місяць чи інший період у межах одного року.





#### 4.3.1.1.1. Функціональні вимоги

REQ-2.1: Можливість отримання даних про всіх працівників у вигляді таблиці.

REQ-2.2: Можливість отримання даних про конкретний продукт та отримання детального опису у вигляді категорій(категорія: властивість N: значення) на відповідній сторінці веб-сайту.

REQ-2.3: Можливість модифікації контракту працівника, а саме зміна контактної інформації, типу зайнятості, оплати праці.

REQ-2.4: Можливість додавати нових чи звільняти вже існуючих працівників.

REQ-2.5: Можливість генерувати автоматичні звіти за різними критеріями: кількість виконаних замовлень певною групою працівників чи конкретним, кількість відпрацьованих годин за місяць, заробітна плата за поточний рік.

#### 4.3.1.2. Управління виробничим процесом

##### 4.3.1.2.1. Опис і пріоритет

Ця характеристика програмної системи передбачає надання користувачу можливість керувати процесом виготовлення вікон. Пріоритет характеристики – високий.

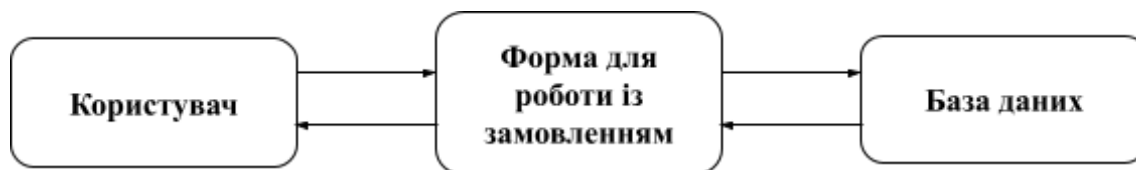
##### 4.3.1.2.2. Послідовності дія/відгук

Вибір поточних виконуваних завдань – вивід списку всіх замовлень на виготовлення із статусом «У ході виконання».

Вибір завершених завдань – вивід списку всіх замовлень на виготовлення із статусом «Виконано».

Вибір скасованих завдань – вивід списку всіх замовлень на виготовлення із статусом «Скасовано».

Натиск на кнопку створити замовлення – відкривання вікна із можливістю заповнення даних про нове замовлення, підтвердження або скасування створення.



#### 4.3.1.2.3. Функціональні вимоги

REQ-3.1: Створення нових замовлень на виготовлення певного продукту у певній кількості та призначення відповідальної бригади працівників.

REQ-3.2: Забезпечення можливості фільтрувати замовлення відповідно до статусу виконання.

REQ-3.3: Можливість генерувати автоматичні звіти за різними критеріями: кількість замовлень із описами, замовлення за статусом виконання, замовлення за певний період часу.

REQ-3.4: Можливість обрахунку заробітної плати.

#### 4.3.1.2.4. Управління закупівлею матеріалів

#### 4.3.1.2.5. Опис і пріоритет

Ця характеристика програмної системи передбачає надання користувачу можливість керувати процесом замовлення необхідних матеріалів для процесу виробництва. Вона тісно пов'язана із характеристикою номер 3 – *Управління виробничим процесом*. Пріоритет характеристики – високий.

#### 4.3.1.2.6. Послідовності дія/відгук

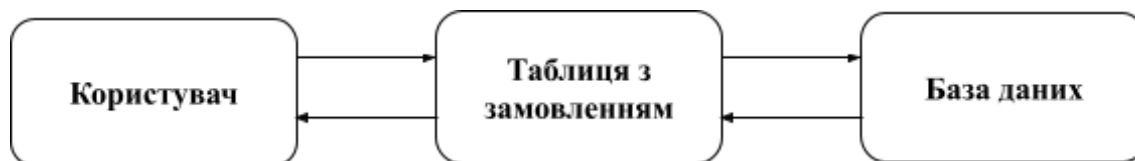
Закінчення матеріалів на складах – сповіщення про необхідність замовлення нових матеріалів.

Фільтрування списку необхідних матеріалів – виведення матеріалів за пріоритетом замовлення, ціною, кількістю та іншими характеристиками.

Створення нового замовлення – автоматичне формування деталей замовлення за попередньою конфігурацією (необхідна кількість певного матеріалу на певний період часу) та додавання у список створених для подальшого відстеження та перенесення у статус «Виконано».

Позначення замовлення як виконаного – перенесення у статус «Виконано», архівування та оновлення список матеріалів на складах.

Скасування замовлення – відміна всіх змін спричинених цим замовленням.



#### 4.3.1.2.7. Функціональні вимоги

REQ-4.1: Якщо користувач намагається закупити товар, якого зараз немає у наявності або списати товар, який не повинен бути списаним програмний продукт повинен сповістити користувача про помилку і продовжити свою роботу. Забезпечення продовження роботи при введенні неприпустимих даних.

REQ-4.2: Забезпечення можливості створювати нові замовлення на матеріали.

REQ-4.3: Можливість конфігурувати процес створення нових замовлень для автоматичного заповнення даних новоствореного замовлення (постачальник та кількість).

REQ-4.4: Можливість перегляду та фільтрування списку необхідних матеріалів (які закінчуються).

REQ-4.5: Відслідковування статусу замовлень за номером: «Створено», «У ході виконання», «Скасовано замовником», «Скасовано постачальником» та «Виконано». Для кожного статусу при необхідності мати можливість вносити пояснення.

REQ-4.6: Можливість генерувати автоматичні звіти за різними критеріями: кількість замовлень із описами, замовлення за статусом виконання, замовлення за певний період часу.

REQ-4.7: Можливість створення звітності, що до заробітної платні робітників фірми.

4.3.1.2.8. Облік матеріалів (матеріалів та виготовлених продуктів)

4.3.1.2.9. Опис і пріоритет

Ця характеристика програмної системи передбачає надання користувачу можливість оглядати всю інформацію про наявний інвентар (матеріалів та виготовлених продуктів), кількість предметів, опис. Пріоритет характеристики – високий.

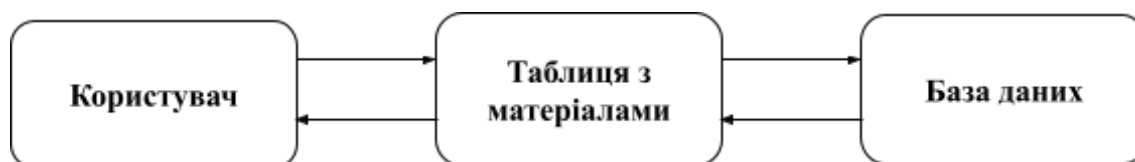
4.3.1.2.10. Послідовності дія/відгук

Використання чек боксів та інших засобів для фільтрування – виведення таблиці із урахуванням фільтрів.

Введення ключових слів у текстове поле пошуку – виведення елементів таблиці, що відповідають критерію пошуку.

Додавання нового предмету до списку – оновлення бази даних, оновлення таблиці.

Видалення предмету зі списку – оновлення бази даних, оновлення таблиці.



#### 4.3.1.2.11. Функціональні вимоги

REQ-5.1: Забезпечення продовження роботи при введенні неприпустимих даних.

REQ-5.2: Забезпечення можливості додавати та видаляти записи у таблицю обліку матеріалів.

REQ-5.3: Можливість поділу таблиці за певним типом продукту чи матеріалу – виведення інформації тільки про нього.

REQ-5.4: Можливість сортування таблиць за обраним критерієм.

REQ-5.5: Можливість виведення таблиці двома способами – загальна інформація (назва, короткий опис, кількість) та детальна інформація (назва, короткий опис, кількість, постачальник, відповідальний).

REQ-5.6: Можливість формувати звіт про присутні матеріали чи продукти на складі.

#### 4.3.2. Вимоги зовнішніх інтерфейсів

##### 4.3.2.1. Користувацькі інтерфейси

GUI повинен містити головну сторінку, сторінку автентифікації та панелі керування: панель керування персоналом, вхідними замовленнями, вихідними замовленнями (закупівля), управління персоналом та керування

процесом виробництва. Всі процеси повинні бути пов'язані між собою та оновлювати дані у реальному часі.

На кожній із панелей повинні бути мінімальні інструменти для виконання тих чи інших завдань (відповідно до розділу) для простої роботи користувача. Дотримання евристики про попередження та допомогу в усуненні помилок, евристики про сповіщення поточного стану та документація, а саме інтерактивні FAQ для кожного розділу відповідно.

Для професійних користувачів надати можливість використовувати гарячі клавіші для виконання часто виконуваних завдань (наприклад, створення замовлення).

#### 4.3.2.2. Апаратні інтерфейси

Програмна система передбачає використання клавіатури та миші. Необхідне підключення до мережі Інтернет.

Комп'ютер на якому буде виконуватись дана програма повинен мати наступні апаратні характеристики:

- 32-х бітний або 64-х бітний процесор з тактовою частотою 1 ГГц, або більше;
- 1 ГБ ОЗП (x86) або ж 2 ГБ ОЗП (x64).

#### 4.3.2.3. Програмні інтерфейси

Для коректної роботи програми потрібні наступні програмні компоненти:

- ОС Windows (Windows 7 – Windows 10) або Linux 4+;
- SQL Server 2012 і вище;
- .NET Core 2.1 і вище.

#### 4.3.2.4. Комунікаційні інтерфейси

Для коректної роботи ПК повинен мати порт RJ-45 або ж модуль Wi-Fi, а локальна мережа підтримувати стандарт Fast Ethernet.

#### 4.3.3. Інші нефункціональні вимоги

##### 4.3.3.1. Вимоги продуктивності

Основними вимогами до продуктивності системи є:

- керування змінами даних в програмі, швидка заміна застарілих даних;
- при заміні даних синхронізація з користувачем має бути оперативною;
- час відображення і відповіді сервера не має перевищувати 2-х секунд;
- коректне та швидке використання паралельного використання сервера кількома користувачами .

##### 4.3.3.2. Вимоги надійності

Система повинна коректно обробляти ситуації, коли з'єднання з сервером було розірване під час транзакції.

У разі збою програми, вона повинна безпечно зберегти дані. У разі неполадок з боку апаратного забезпечення повинна бути реалізована можливість переносу програмного забезпечення на інший пристрій без втрати даних. Під час використання продукту необхідно створювати резервні копії, які зберігаються на сервері.

##### 4.3.3.3. Вимоги безпеки

Авторизація будується на класах користувачів. Автентифікація повинна

відбуватись за допомогою логіну і пароллю.

#### 4.3.3.4. Атрибути якості програмного продукту

- автоматизація основних бізнес-процесів дозволяє уникнути зловживань і помилок персоналу;
- захист комерційної інформації, розмежування користувачів за рівнем доступу до інформації;
- навчання роботі із системою повинно займати мінімальну кількість часу;
- система повинна бути стабільною та ефективно використовувати апаратні ресурси.



## РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ

### 2.1. Концептуальне моделювання предметної області

#### 2.1.1. Опис сутностей

Таблиця Person(Персона) відображає сутність, яка описує основні характеристики людини, тобто учасника, або користувача системи.

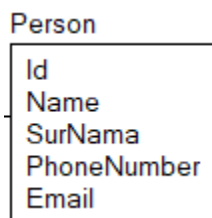


Рис. 2.1. Таблиця «Персона»

Таблиця містить такі поля:

- Id(PK) – унікальний номер (ідентифікатор);
- SurName – прізвище;
- Name – ім'я;
- PhoneNumber – номер телефону;
- Email – адреса електронної пошти.

Таблиця WorkerType(ТипРобітника) описує типи працівників фірми.

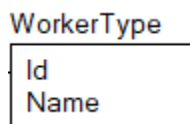
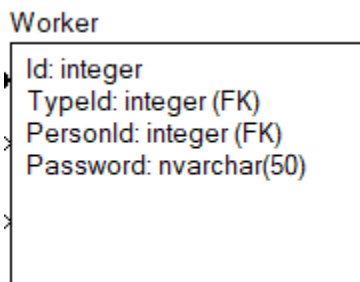


Рис. 2.2. Таблиця «Працівник»

Містить такі поля:

- Id(PK) – унікальний номер (ідентифікатор);
- Name – ідентифікатор типу працівника (посилання на таблицю WorkerType)

Таблиця Worker(Робітник) описує працівника фірми.

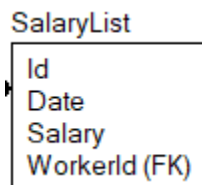


*Рис. 2.3. Таблиця «Працівник»*

Містить такі поля:

- Id(PK) – унікальний номер (ідентифікатор);
- Password – унікальний пароль входу в систему цього користувача;
- Typeid(FK) – ідентифікатор типу працівника (посилання на таблицю WorkerType).;
- PersonId(FK) – ідентифікатор працівника (посилання на таблицю Person).

Таблиця SalaryList(Список заробітної плати) є списком нарахованої заробітної плати працівникам.



*Рис. 2.4. Таблиця «Відділення»*

Містить такі поля:

- Id(PK) – унікальний номер (ідентифікатор);
- Date– місяць зарахування заробітної плати;
- Salary – нарахована заробітна плата;
- WorkerId(FK) – ідентифікатор працівника (посилання на таблицю Worker).

Таблиця Order(Замовлення) містить дані про замовлення.

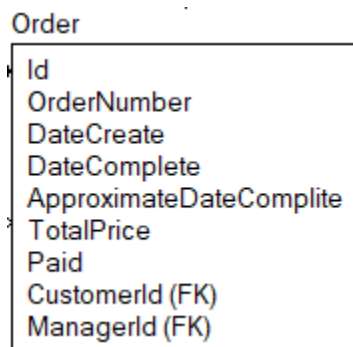


Рис. 2.5. Таблиця «Замовлення»

Містить поля:

- Id(PK) – унікальний номер навчальної групи (ідентифікатор);
- TotalPrice – назва групи;
- OrderNumber – номер замовлення;
- DateComplete – дата завершення замовлення;
- DateCreate – дата створення замовлення;
- Paid – частина оплаченої загальної вартості;
- ApproximateDateComplete – приблизна дата виконання замовлення (приблизно тиждень);
- CustomerId(FK) – ідентифікатор замовника (посилання на таблицю Person);
- ManagerId(FK) – ідентифікатор менеджера, який оформив замовлення (посилання на таблицю Worker).

Таблиця Payment(Оплата) містить дані про проведені оплати замовлень.

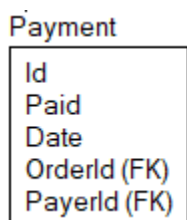


Рис. 2.6. Таблиця «Курсант»

Містить поля:

- Id(PK) – унікальний номер (ідентифікатор);
- Paid – сума оплати;
- Date – дата проведення операції;
- OrderId(FK) – ідентифікатор замовлення (посилання на таблицю Order);
- PayerId(FK) – ідентифікатор платника, який здійснив оплату (посилання на таблицю Person).

Таблиця Product(продукт) являє собою вікно, яке потрібно виготовити.

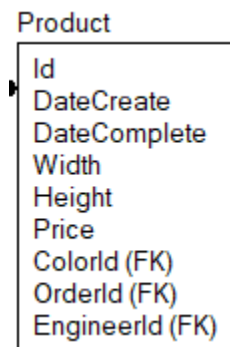


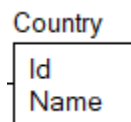
Рис. 2.7. Таблиця «Продукт»

Містить такі поля:

- Id(PK) – унікальний номер (ідентифікатор);
- DateCreate – дата створення продукту;
- DateComplite – дата завершення виготовлення замовлення;
- Width – ширина вікна;
- Height – висота вікна;
- Price – вартість вікна;
- ColorId(FK) – ідентифікатор кольору (посилання на таблицю Color).
- OrderId(FK) – ідентифікатор замовлення, для якого виготовляється вікно(посилання на таблицю Order).

- EngineerId – ідентифікатор робітника, який виготовляє вікно (посилання на таблицю Worker).

Таблиця Country(країна) являє собою список країн

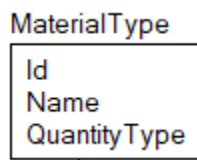


*Рис. 2.8. Таблиця «країна»*

Містить поля:

- Id(PK) – унікальний номер (ідентифікатор);
- Name – назва країни;

Таблиця MaterialType(тип матеріалу) описує типи матеріалів, які будуть використовуватися у виробництві вікон систем.



*Рис. 2.9. Таблиця «Тип матеріалу»*

Містить такі поля:

- Id(PK) – унікальний номер (ідентифікатор);
- Name – назва типу матеріалу(наприклад фурнітура, скло, профіль).
- QuantityType – величина вимірювання матеріалу.

Таблиця Material (матеріал) описує матеріал, з якого виготовляються вікна.

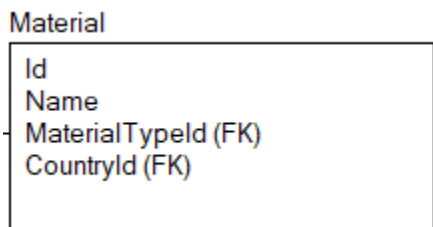


Рис. 2.10. Таблиця «Матеріал»

Містить наступні поля:

- Id(PK) – унікальний номер (ідентифікатор);
- Name – назва(марка) матеріалу;
- MaterialTypeId(FK) – ідентифікатор типу матеріалу (посилання на таблицю MaterialType).

Таблиця Storage(Склад) Матеріали, які знаходяться на складі.

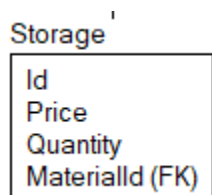


Рис. 2.11. Таблиця «Склад»

Містить такі поля:

- Id(PK) – унікальний номер (ідентифікатор);
- Price – вартість матеріалу;
- Quantity – кількість матеріалів на складі;
- MaterialId(FK) – ідентифікатор матеріалу (посилання на таблицю Material).

Таблиця MaterialList(список матеріалів) описує список матеріалів для виготовлення кожного вікна.



Рис. 2.12. Таблиця «Список матеріалів»

Містить такі поля:

- Id – унікальний номер (ідентифікатор);
- Quantity – кількість матеріалів взятих із складу;
- Date – дата і час проведення операції;
- ProductId – ідентифікатор продукту, для якого призначено матеріал (посилання на таблицю Product);
- MaterialId – ідентифікатор матеріалу, який буде використано для виготовлення вікна (посилання на таблицю Material);

Таблиця Company(Компанія), являє собою список компаній, які займаються постачанням матеріалів.

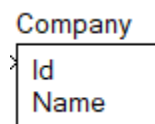


Рис. 2.13. Таблиця «Компанія»

Містить такі поля:

- Id – унікальний номер (ідентифікатор);
- Name – назва компанії.

Таблиця Offers(Пропозиції) описує дані про пропозиції матеріалів, які постачаються замовниками.



Рис. 2.14. Таблиця «Пропозиції»

Містить такі поля:

- Id – унікальний номер (ідентифікатор);
- Quantity – кількість матеріалів;
- Price – ціна матеріалу від постачальника;
- DeliveryDate – теоретична дата доставки;
- MaterialId – ідентифікатор матеріалу (посилання на таблицю Material)
- SupplierId – ідентифікатор постачальника, який постачає матеріал (посилання на таблицю Person)
- CompanyId – ідентифікатор компанії, до якої належить постачальник (посилання на таблицю Company)

Таблиця SupplyItem (Матеріал для постачання) описує матеріал, який постачає постачальник.

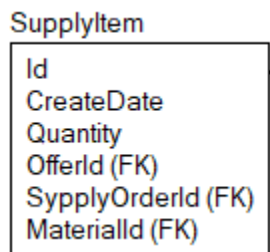


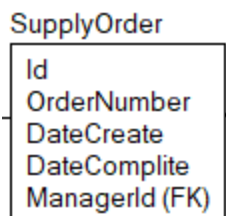
Рис. 2.15. Таблиця «Матеріал для постачання»



Містить наступні поля:

- Id – унікальний номер (ідентифікатор);
- CreateDate– дата створення позиції про потрібний матеріал;
- Quantity – кількість матеріалу, який потрібен;
- OfferId(FK) – ідентифікатор пропозиції постачальника, яка була вибрана для постачання (посилання на таблицю Offers);
- SypplyOrderId(FK) – ідентифікатор замовлення, (посилання на таблицю SypplyOrder);
- MaterialId(FK) - ідентифікатор матеріалу, який потрібно доставити (посилання на таблицю Material);

Таблиця SupplyOrder(Поставка товарів) описує замовлення матеріалів які повинні бути доставлені.



*Рис. 2.16. Таблиця «Поставка товарів»*

Містить такі поля:

- Id – унікальний номер платежу з виплати заробітної плати (ідентифікатор);
- OrderNumber – номер поставки;
- DateCreate – дата створення поставки;
- DateComplite – дата завершення поставки;
- ManagerId(FK) – ідентифікатор менеджера, який приймає замовлення (посилання на таблицю Worker)

### **2.1.2. Опис зв'язків між сутностями**

Між сутностями було встановлено наступні зв'язки:

Зв'язок між таблицями Person і Worker – один до одного. Тому що це частина таблиці Worker. Працівник має одні персональні дані.

Зв'язок між таблицями Worker і SalaryList – один до багатьох. Робітник може мати заробітну плату декілька місяців. Тобто мати відношення до декількох рядків в таблиці SalaryList.

Зв'язок між таблицями Worker і WorkerType – багато до одного. Багато робітників можуть мати один тип зайнятості.

Зв'язок між таблицями Person і Order – один до багатьох. Поле CustomerId(замовник) посилається на таблицю Person. Один замовник може мати багато замовлень.

Зв'язок між таблицями Worker і Order – один до багатьох. Поле ManagerId(менеджер) посилається на таблицю Worker. Один менеджер може обробляти декілька замовлень.

Зв'язок між таблицями Order і Product – один до багатьох. Одне замовлення може мати декілька продуктів.

Зв'язок між таблицями Order і Payment – один до багатьох. Одне замовлення може бути оплачене не відразу, а через декілька разів. Тобто таблиця Order може посилатися на таблицю Payment Декілька разів.

Зв'язок Person і Payment – один до багатьох. Одна персона може виконати декілька оплат.

Зв'язок між таблицями Worker і Product – один до одного. Один робітник може виготовляти один продукт.

Зв'язок між таблицями Product і MaterialList – один до багатьох. Один продукт завжди посилається на декілька рядків із таблиці MaterialList(список

матеріалів). Тобто кожен продукт посилається на 3 рядки із списку матеріалів

Зв'язок між таблицями Color і Product – один до багатьох. Один колір може бути у декількох продуктів.

Зв'язок між таблицями MaterialList і Storage – один до одного. Один рядок зі списку потрібних матеріалів може посилатися лише на один рядок із таблиці Storage.

Зв'язок між таблицями Material і Storage – один до багатьох. Один матеріал може посилатися на декілька рядків із таблиці Storage через відмінність у ціні та інших характеристиках.

Зв'язок між таблицями Person і Offer – один до багатьох. Поле Supplierid(постачальник) в таблиці Offer посилається на таблицю Person. Один постачальник може мати багато пропозицій в таблиці Offers.

Зв'язок між таблицями Company і Offers – один до багатьох. Декілька пропозицій можуть належати одній компанії.

Зв'язок між таблицями SupplyItem і Offers – один до одного. Поле OfferId(пропозиція яка має бути вибрана із списку пропозицій) в таблиці SupplyItem посилається на таблицю Offers. В таблиці Матеріал поставки може бути лише одне посилання на вибрану пропозицію.

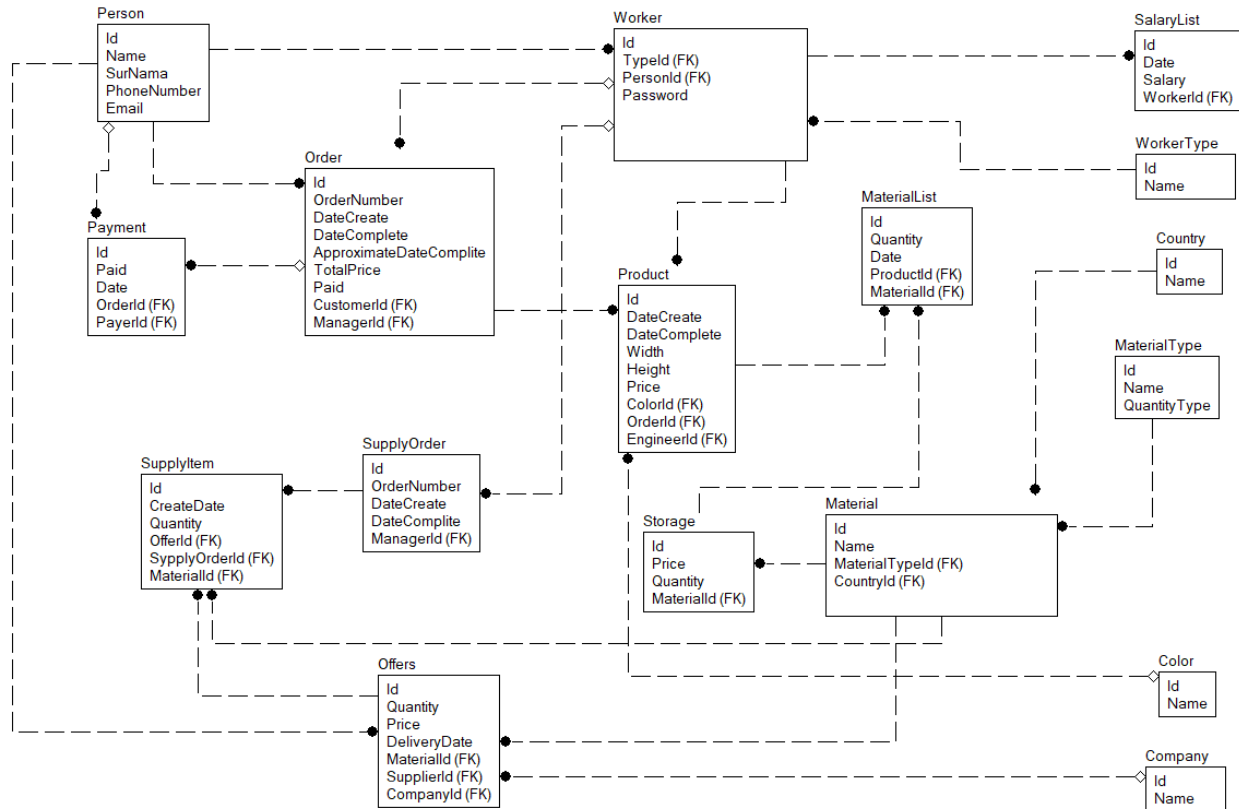
Зв'язок між таблицями Material і Offers – один до багатьох. Один матеріал може бути в декількох пропозиціях.

Зв'язок між таблицями SupplyOrder і SupplyItem – один до багатьох. Одна поставка може мати декілька матеріалів для постачання.

Зв'язок між таблицями SupplyOrder і Worker – один до одного. Поле ManagerId(менеджер, який має прийняти поставку) в таблиці SupplyOrder посилається на таблицю Worker. Одну поставку може приймати лише один менеджер.

## 2.2. Логічне проектування БД

В рамках даної роботи було створено логічну (рис. 2.16) та фізичну (рис. 2.17) моделі БД для роботи виробництва вікон.



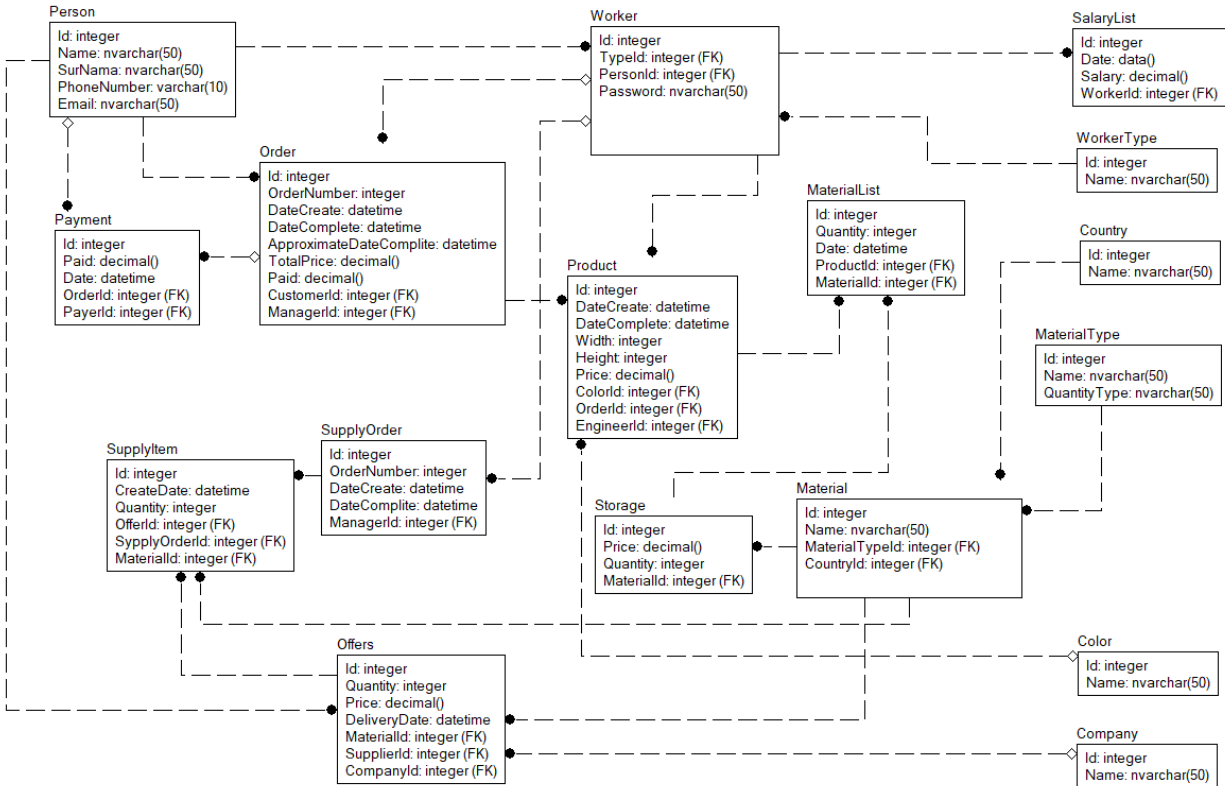
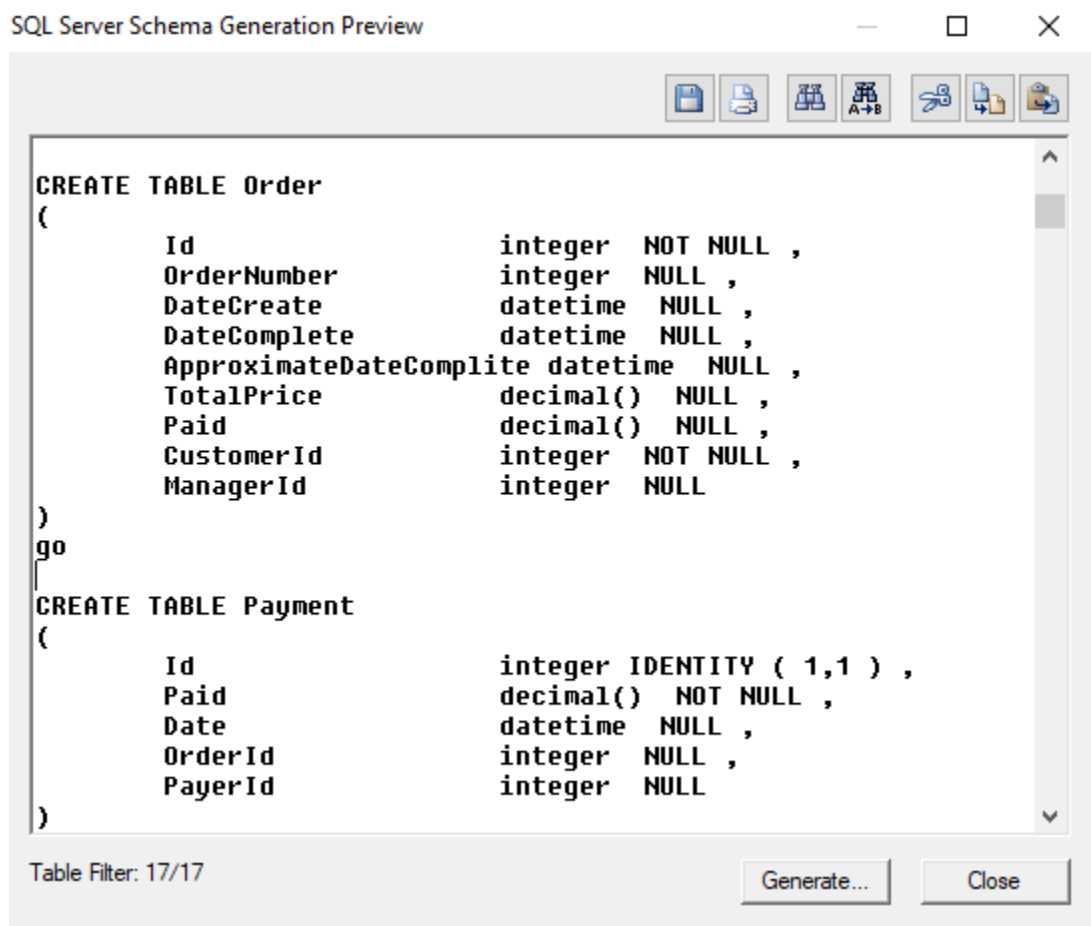


Рис. 2.17. Фізична модель

Після моделювання я засобами Erwin Data Modeler згенерував скрипт створення бази даних. Для цього я вибрав фізичну модель та перейшов в меню Tools-> Forward Engineering -> Schema generation. Вікні Forward engineering schema generation я налаштував необхідні параметри генерації та вибрав пункт Preview.



*Рис. 2.18. Скрипт створення бази даних*

Після генерації скрипта я відмодифікував його згідно своїх потреб та переніс його у середовище SQL Server Management Studio та з його допомогою створив базу даних.

Детальніше із кодом створення бази даних можна ознайомитися в Додатку А

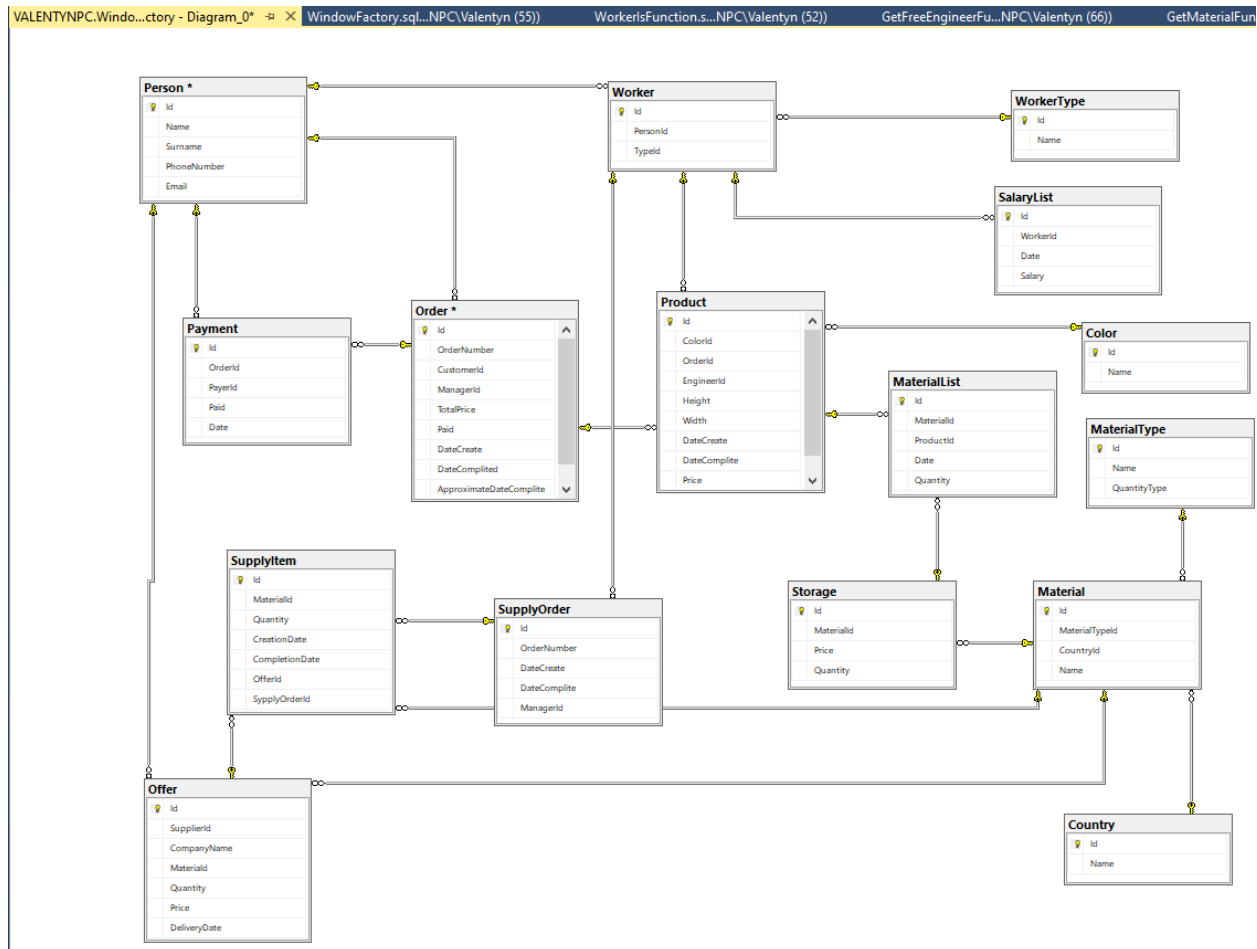


Рис. 2.19. Діаграма бази даних в SQL Server 2019

Були додані наступні обмеження:

- Усі первинні ключі не можуть бути рівні NULL (обмеження SQL).
- На більшість атрибуту з вторинними ключами було накладено обмеження з умовою - не рівні NULL.
- У всіх таблицях, де потрібно, ідентифікатори мають властивість identity, тобто є автоінкрементальними.
- Значення поля PassWord в таблиці Worker повинне бути інікальним.

### 2.3. Проектування типових запитів і транзакцій

В ході проектування системи управління базою даних було, спроектовано ряд запитів, а саме:

- Процедура додавання нового матеріалу;
- Додавання нового продукту;
- Додавання заробітної плати робітнику;
- Додавання нового матеріалу для постачання;
- Процедура додавання нового працівника;
- Процедура обрахунку вартості замовлення;
- Процедура обрахунку вартості продукту;
- Створення нової поставки товарів;
- Процедура створення списку необхідних матеріалів для виготовлення продукту;
- Функція для отримання списку вільних інженерів;
- Процедура додавання нової персони;
- Процедура додавання нового замовлення;
- Функція для отримання списку матеріалів;
- Функція для перевірки чи робітник є потрібного напрямку (інженер, менеджер...).

Детальніше скрипти створення кожного запиту можна переглянути в Додатку В

Індекси в даній системі будуть використовуватися за замовчуванням середовища SQL Server та створені некластерні індекси у таблицях Person, Payment, Order, Product, SupplyOrder, Offer.

```
CREATE NONCLUSTERED INDEX [ix_person] ON [dbo].[Person]
(
    [Name] ASC,
    [SurName] ASC,
    [PhoneNumber] ASC,
    [Email] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

CREATE NONCLUSTERED INDEX [ix_Payment] ON [dbo].[Payment]
(
    [Date] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

CREATE NONCLUSTERED INDEX [ix_OrderNumber] ON [dbo].[Order]
(
    [OrderNumber] ASC,
```



```

[CustomerId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

```

```

CREATE NONCLUSTERED INDEX [ix_OrderId] ON [dbo].[Product]
(

```

```

[OrderId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

```

```

CREATE NONCLUSTERED INDEX [ix_OrderNumber] ON [dbo].[SupplyOrder]
(

```

```

[OrderNumber] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

```

```

CREATE NONCLUSTERED INDEX [ix_Price] ON [dbo].[Offer]
(

```

```

[Price] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

```

## **РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ РОБОТИ З БАЗОЮ ДАНИХ**

### **3.1. Реалізація доступу до бази даних**

Для реалізації доступу до даних БД обрано платформу ADO.NET Entity Framework, яка дозволяє розробникам створювати додатки для доступу до даних, що працюють з концептуальною моделлю додатку, а не безпосередньо з реляційною схемою зберігання. Її метою є зменшення обсягу коду і зусиль по обслуговуванню додатків, орієнтованих на обробку даних.

ADO.NET Entity Framework має вбудований в Visual Studio графічний редактор і підтримує три підходи до створення об'єктної моделі та бази даних:

- Code First – дозволяє визначити модель за допомогою класів C# або VB.Net, створює базу даних і додає в неї таблиці на основі коду та дозволяє оновлювати базу даних за допомогою міграцій.
- Database First – дозволяє реконструювати модель на основі існуючої бази даних.
- Model First – дозволяє створити нову модель за допомогою конструктора Entity Framework, а потім сформувати схему бази даних на основі моделі. Модель зберігається в EDMX-файлі, основі якого автоматично формуються класи в додатках, з якими відбувається взаємодія.

Таким чином, платформа ADO.NET Entity Framework характеризується наступними перевагами, які й обумовили її вибір для реалізації доступу до даних:

- додатки можуть працювати з концептуальною моделлю в термінах предметної області, в тому числі з успадкованими типами, складними елементами і зв'язками;

- додатки звільняються від жорстких залежностей від конкретного ядра СУБД або схеми зберігання;
- зіставлення між концептуальною моделлю і схемою, специфічною для конкретного сховища, можуть змінюватися без зміни коду програми;
- розробники мають можливість працювати з узгодженою моделлю об'єктів додатки, яка може бути порівняна з різними схемами зберігання, які, можливо, будуть реалізовані в різних системах управління даними;
- кілька концептуальних моделей можуть бути співставлені з єдиною схемою зберігання;
- підтримка інтегрованих в мову запитів (LINQ) забезпечує під час компіляції перевірку синтаксису запиту щодо концептуальної моделі.

Зв'язок з сутностями БД у цьому проекті було здійснено за допомогою утиліти Scaffold-DbContext у Visual Studio. Діаграма компонент на рис. 3.1.

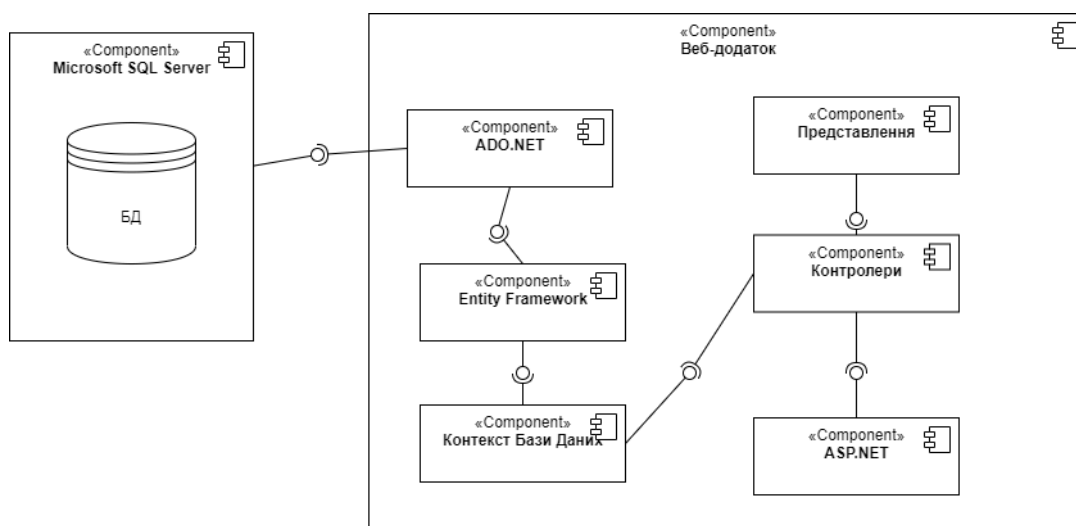


Рис. 3.1. Діаграма компонент

ADO.NET Entity Framework дозволяє легко підключити прикладний додаток до наявної бази даних. З підходом Database First це робиться наступним чином: з допомогою *NuGet Package Manager Console* вводиться команда

*ScaffoldDataSource=ValentynPC;InitialCatalog=WindowFactory;Integrate  
d Security=True Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models,*

де “ *WindowFactory* ” – назва бази даних, “*Models*” – назва папки, у якій містяться класи моделі.

### 3.2. Реалізація функціональних характеристик системи

#### 3.2.1. Реєстрація замовлення

Для додавання нового замовлення менеджер фірми повинен увійти в програмну систему, перейти до сторінки додавання нового замовлення. Після введення даних відбувається їх валідація на клієнтській частині. Якщо дані введено коректно, то вони надсилаються на сервер, який додає нову особу до бази даних. На стороні бази даних теж проводить перевірка на можливість вносити такі дані. Інакше працівнику виводиться помилка, яка вказує на неправильно заповнені поля. Також у випадку якщо замовник є новим, тоді буде проведено додавання нової персони в таблицю *Person*, в іншому випадку проводить пошук ідентифікатора замовника та додавання його до даних замовлення. Також відбувається перевірка на те що робітник дійсно є менеджером.

Фрагмент програми:

```
public virtual int AddOrder(string customerId, Nullable<int> managerId)
{
    var customerIdParameter = customerId != null ?
        new ObjectParameter("CustomerId", customerId) :
        new ObjectParameter("CustomerId", typeof(string));

    var managerIdParameter = managerId.HasValue ?
        new ObjectParameter("ManagerId", managerId) :
```

```

        new ObjectParameter("ManagerId", typeof(int));

        return
        ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("AddOrder",
        customerIdParameter, managerIdParameter);

```

### Процедура додавання нового замовлення

```

DROP PROCEDURE IF EXISTS dbo.AddOrder
GO
CREATE PROCEDURE dbo.AddOrder

--Customer Data
@CustomerId nvarchar(50),
--Manager Data
@ManagerId int
--

AS

DECLARE @ManagerTypeId int, @res bit
SELECT @ManagerTypeId = Id FROM WorkerType

WHERE WorkerType.Name = 'manager'

--SELECT @ManagerTypeId

@ManagerId)

SET @res = dbo.WorkerIs(@ManagerTypeId,

= 0)

IF(dbo.WorkerIs(@ManagerTypeId, @ManagerId)

BEGIN
    PRINT 'Wrong ManagerId'
    RETURN
END

DECLARE @CurrentOrderNumber int

SELECT @CurrentOrderNumber =

FROM [Order]

IF NOT(ISNUMERIC(@CurrentOrderNumber) = 1)
    SET @CurrentOrderNumber = 1

--VALUES(Id, OrderNumber, CustomerId,
ManagerId, TotalPrice, Paid, DateCreate, DateComplited, ApproximateDateComplite)
INSERT INTO [Order]
(CustomerId, OrderNumber, ManagerId)
VALUES(
    @CustomerId,
    @CurrentOrderNumber,
    @ManagerId)

EXEC dbo.AddOrder 1, 5

}

```

### 3.2.2. Сортування даних

Для сортування даних у одній з таблиць програмної системи необхідно клацнути на заголовок того поля, за яким необхідно посортувати. Для

сортування у зворотньому порядку необхідно натиснути на те саме поле повторно.

Фрагмент коду:

```
const getCellValue = (tr, idx) => tr.children[idx].innerText ||
tr.children[idx].textContent;

const comparer = (idx, asc) => (a, b) => ((v1, v2) =>
  (v1 !== '' && v2 !== '' && !isNaN(v1) && !isNaN(v2)) ? v1 - v2 :
  v1.toString().localeCompare(v2)
)(getCellValue(asc ? a : b, idx), getCellValue(asc ? b : a, idx));

document.querySelectorAll('th')
  .forEach(th => th.addEventListener('click', (() => {
    const table = document.querySelector('tbody');
    Array.from(table.querySelectorAll('tr:nth-child(n+1)'))
      .sort(comparer(Array.from(th.parentNode.children).indexOf(th), this.asc =
!this.asc))
      .forEach(tr => table.appendChild(tr));
  })));
```

### 3.2.3. Фільтрація даних

Для фільтрації даних необхідно перейти до деталей запису, що містить масив тих даних. Наприклад, з деталей про групу можна перейти до відфільтрованого списку студентів.

Фрагмент коду:

```
class OrderFilter : FilterBase<OrderModel>
{
  public OrderFilter AddPriceFilter(decimal from, decimal to)
  {
    AddFilter("AddPriceFilter", (model) => model.TotalPrice >= from &&
model.TotalPrice < to);
    return this;
  }

  public OrderFilter AddNumberFilter(int from, int to)
  {
    AddFilter("AddQuantityFilter", (model) => model.OrderNumber >= from &&
model.OrderNumber < to);
    return this;
  }

  public OrderFilter AddDateByCreateFilter(DateTime from, DateTime to)
  {
    AddFilter("AddQuantityFilter", (model) => model.DateCreate >= from &&
model.DateCreate < to);
    return this;
  }

  public OrderFilter AddDateByCompliteFilter(DateTime from, DateTime to)
```

```

    {
        AddFilter("AddQuantityFilter", (model) => model.DateComplite >= from &&
model.DateComplite < to);
        return this;
    }

    public OrderFilter AddPaidFilter()
    {
        AddFilter("AddNameFilter", (model) => ( model.Paid == model.TotalPrice));
        return this;
    }

```

### 3.2.4. Додавання персони

Якщо замовник є новим, тоді буде проведено додавання нової персони в таблицю Person, в іншому випадку проводиться оновлення даних в таблиці та проводиться пошук ідентифікатора персони та додавання його до даних замовлення.

```

    public virtual int AddPerson(string personName, string personSurName, string
personPhoneNumber, string personEmail)
    {
        var personNameParameter = personName != null ?
            new ObjectParameter("PersonName", personName) :
            new ObjectParameter("PersonName", typeof(string));

        var personSurNameParameter = personSurName != null ?
            new ObjectParameter("PersonSurName", personSurName) :
            new ObjectParameter("PersonSurName", typeof(string));

        var personPhoneNumberParameter = personPhoneNumber != null ?
            new ObjectParameter("PersonPhoneNumber", personPhoneNumber) :
            new ObjectParameter("PersonPhoneNumber", typeof(string));

        var personEmailParameter = personEmail != null ?
            new ObjectParameter("PersonEmail", personEmail) :
            new ObjectParameter("PersonEmail", typeof(string));

        return
            ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("AddPerson",
personNameParameter, personSurNameParameter, personPhoneNumberParameter,
personEmailParameter);
    }

```

### Процедура додавання нової персони

```

DROP PROCEDURE IF EXISTS dbo.AddPerson
GO

```

```

CREATE PROCEDURE dbo.AddPerson
(
    @PersonName nvarchar(50),
    @PersonSurName nvarchar(50),
    @PersonPhoneNumber nvarchar(10),
    @PersonEmail nvarchar(50)
)
AS
BEGIN

    DECLARE @PersonId int = 0
    DECLARE @uperr int
    DECLARE @maxerr int

    SELECT @PersonId = Person.Id
                                FROM Person
                                WHERE @PersonName = Person.Name AND
                                    @PersonSurName = Person.Surname

    IF (@PersonId != 0)
        BEGIN

            IF EXISTS (SELECT *
                        FROM Person
                        WHERE @PersonName = Person.Name AND
                            @PersonSurName = Person.Surname AND
                            @PersonPhoneNumber = Person.PhoneNumber AND
                            @PersonEmail = Person.Email)
                BEGIN
                    PRINT 'Person exist in table'
                    RETURN @PersonId
                END
            ELSE
                PRINT 'Person exist in table Try update person data. ' + CHAR(13) + ' CustomerId = ' +
CAST(@PersonId AS nvarchar);

            BEGIN TRAN

                UPDATE Person
                SET PhoneNumber = @PersonPhoneNumber,
                    Email = @PersonEmail
                WHERE @PersonId = Person.Id

            BEGIN

```



```

SET @uperr = @@error
IF @uperr > @maxerr
SET @maxerr = @uperr

-- If an error occurred, roll back
IF @maxerr <> 0
    BEGIN
        ROLLBACK
        PRINT 'did bot updated ' + CHAR(13) + ' Transaction
rolled back'

    END
ELSE
    BEGIN
        COMMIT
        PRINT 'updated successfully ' + CHAR(13) + '
Transaction committed'

    END
END

RETURN @PersonId

END

ELSE
    BEGIN
        --Person don't exist in table
        --Insert person

        PRINT 'Person did not exist in table ' + CHAR(13) + ' Try to Insert in table'

        BEGIN TRAN

        DECLARE @InsertedPersonId int = (SELECT MAX(Id) + 1 FROM Person)

        INSERT INTO Person
        (Id, [Name], Surname, PhoneNumber, Email)
        VALUES (
            @InsertedPersonId,
            @PersonName,
            @PersonSurName,
            @PersonPhoneNumber,
            @PersonEmail
        )

        BEGIN

```

```

SET @uperr = @@error
IF @uperr > @maxerr
SET @maxerr = @uperr

-- If an error occurred, roll back
IF @maxerr <> 0
    BEGIN
        ROLLBACK
        PRINT 'did bot updated ' + CHAR(13) + ' Transaction rolled back'
    END
ELSE
    BEGIN
        COMMIT
        PRINT 'updated successfully ' + CHAR(13) + ' Transaction committed'
    END
END

RETURN @InsertedPersonId

END
END

```

### 3.2.3. Редагування даних персони

Для зміни даних персони потрібно перейти на сторінку з даними про персону. Якщо потрібно змінити загальну інформацію, то потрібно натиснути кнопку редагування і на сторінці, яка відкриється, в формі змінити дані на нові значення. Для редагування доступні прізвище, ім'я, номер телефону, електронна адреса особи. Потім потрібно натиснути на кнопку “Редагувати дані” і підтвердити операцію. Далі будуть оновлені дані в відповідній таблиці бази даних. Якщо все пройде успішно, то з’явиться спливаюче повідомлення про успіх, в іншому випадку буде видалено поле з яким є проблеми. Також можна скасувати операцію.

Фрагмент коду для редагування даних абонента:

```

public UserDatum UpdateUser(int userId, string newName, string newSurname, string
phoneNumber, string Email)
{
    BaseUserInfo user = _context.BaseUserInfos.FirstOrDefault(u => u.UserId ==
userId);
    if (user != null)
    {
        user.Name = newName;
        user.Surname = newSurname;
        user.Email = Email;
        user.phoneNumber = phoneNumber;
        _context.BaseUserInfos.Update(user);
        int count = _context.SaveChanges();
        if (count > 0)
    }
}

```

```

        {
            return _context.UserData.FirstOrDefault(u => u.UserId == userId);
        }
    }

    return null;

```

### 3.2.5. Додавання продукту до замовлення

Викликається сторед процедура додавання нового продукту

```

public virtual int AddProduct(Nullable<int> orderNumber, Nullable<int> engeneerId,
    Nullable<int> colorId, Nullable<int> height, Nullable<int> width, Nullable<int>
    profileId, Nullable<int> glassId, Nullable<int> furnitureId)
{
    var orderNumberParameter = orderNumber.HasValue ?
        new ObjectParameter("OrderNumber", orderNumber) :
        new ObjectParameter("OrderNumber", typeof(int));

    var engeneerIdParameter = engeneerId.HasValue ?
        new ObjectParameter("EngeneerId", engeneerId) :
        new ObjectParameter("EngeneerId", typeof(int));

    var colorIdParameter = colorId.HasValue ?
        new ObjectParameter("ColorId", colorId) :
        new ObjectParameter("ColorId", typeof(int));

    var heightParameter = height.HasValue ?
        new ObjectParameter("Height", height) :
        new ObjectParameter("Height", typeof(int));

    var widthParameter = width.HasValue ?
        new ObjectParameter("Width", width) :
        new ObjectParameter("Width", typeof(int));

    var profileIdParameter = profileId.HasValue ?
        new ObjectParameter("ProfileId", profileId) :
        new ObjectParameter("ProfileId", typeof(int));

    var glassIdParameter = glassId.HasValue ?
        new ObjectParameter("GlassId", glassId) :
        new ObjectParameter("GlassId", typeof(int));

    var furnitureIdParameter = furnitureId.HasValue ?
        new ObjectParameter("FurnitureId", furnitureId) :
        new ObjectParameter("FurnitureId", typeof(int));

    return
        ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("AddProduct",
        orderNumberParameter, engeneerIdParameter, colorIdParameter, heightParameter,
        widthParameter, profileIdParameter, glassIdParameter, furnitureIdParameter);
}

```

## Процедура додавання нового продукту

```

CREATE PROCEDURE dbo.AddProduct
    @OrderNumber int,
    --Engineer Data
    @EngineerId int,
    --Product Data
    @ColorId int,
    @Height int,
    @Width int,

    @ProfileId int,
    @GlassId int,
    @FurnitureId int
    --
AS

    DECLARE @EngineerTypeId int, @res bit
    SELECT @EngineerTypeId = Id FROM WorkerType
    WHERE WorkerType.Name = 'engineer'
    --SELECT @EngineerTypeId

    SET @res = dbo.WorkerIs(@EngineerTypeId,
    @EngineerId)

    IF(dbo.WorkerIs(@EngineerTypeId, @EngineerId) = 0)
    BEGIN
        PRINT 'Wrong EngineerId'
        RETURN
    END

    DECLARE @ProductId int
    SELECT @ProductId = MAX(Id) + 1
    FROM Product

    PRINT ISNUMERIC(@ProductId)
    IF NOT(ISNUMERIC(@ProductId) = 1)
        SET @ProductId = 1

    PRINT @ProductId

    PRINT 'INSERT VALUES(Id, ColorId, OrderId, EngineerId,
    Height, Width, DateComplite, Price)'
    SET IDENTITY_INSERT Product ON
    INSERT INTO Product
    (Id, ColorId, OrderId, EngineerId, Height, Width)
    VALUES(
        @ProductId,
        @ColorId,
        @OrderNumber,
        @EngineerId,
        @Height,
        @Width
    )

    PRINT 'INSERT VALUES(Id, MaterialId, ProductId, [Date],
    Quantity)'

    INSERT INTO MaterialList
    (MaterialId, ProductId)
    VALUES(
        @FurnitureId,
        @ProductId
    )

    INSERT INTO MaterialList
    (MaterialId, ProductId)
    VALUES(
        @GlassId,
        @ProductId
    )

    INSERT INTO MaterialList
    (MaterialId, ProductId)
    VALUES(
        @ProfileId,
        @ProductId
    )

    PRINT 'Complited'
    GO

```

### 3.2.6. Додавання продукту до замовлення

Потрібно вибрати матеріали зі списку. Далі їх ідентифікатори передаються в сторед процедуру. Викликається сторед процедура додавання вибраних матеріалів

```
public virtual int SetMaterialList(Nullable<int> productId, Nullable<int>
materialId, Nullable<int> quantity)
{
    var productIdParameter = productId.HasValue ?
        new ObjectParameter("ProductId", productId) :
        new ObjectParameter("ProductId", typeof(int));

    var materialIdParameter = materialId.HasValue ?
        new ObjectParameter("MaterialId", materialId) :
        new ObjectParameter("MaterialId", typeof(int));

    var quantityParameter = quantity.HasValue ?
        new ObjectParameter("Quantity", quantity) :
        new ObjectParameter("Quantity", typeof(int));

    return
        ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("SetMaterialList",
        productIdParameter, materialIdParameter, quantityParameter);
}
```

Процедура створення списку необхідних матеріалів для виготовлення продукту

DROP PROCEDURE IF EXISTS dbo.SetMaterialList	ROLLBACK TRAN
GO	RETURN 0
	END
CREATE PROCEDURE dbo.SetMaterialList	
@ProductId int,	-----
@MaterialId int,	DECLARE @MaterialListId int
@Quantity int	
AS	SELECT @MaterialListId = Id
BEGIN TRAN	FROM MaterialList
	WHERE ProductId = @ProductId AND
	MaterialId = @MaterialId
DECLARE @CurrentQuantity int	
SELECT @CurrentQuantity = Quantity	
FROM Storage as S	PRINT 'Material Id'
WHERE S.MaterialId = @MaterialId	PRINT @MaterialListId
IF(@Quantity > @CurrentQuantity)	IF (ISNUMERIC(@MaterialListId) = 1)
BEGIN	BEGIN
PRINT 'Not enough material in the storage'	

```

PRINT 'UP DATE Material'
@Quantity
)
Quantity in Material List'
UPDATE MaterialList
PRINT 'Update Storage'
SET Quantity = @Quantity
UPDATE Storage
WHERE MaterialId =
SET Quantity = Quantity -
@MaterialId AND
@Quantity
ProductId =
WHERE Storage.MaterialId =
@ProductId
@MaterialId
END
ELSE
PRINT 'Complited'
BEGIN
END
PRINT 'Insert Materials in
Material List'
SELECT @CurrentQuantity = Quantity FROM Storage
--Id, ColorId, OrderId, WHERE MaterialId = @MaterialId
EngineerId, Height, Width, DateComplite, Price
IF( @CurrentQuantity > 0)
INSERT INTO MaterialList
COMMIT TRAN
(MaterialId, ProductId, Quantity)
ELSE
VALUES(
ROLLBACK
@MaterialId,
@ProductId,
GO;

```

### 3.2.7. Додавання нового матеріалу на склад

При додаванні нового матеріалу відбувається перевірка чи такий матеріал вже існує в базі даних, якщо він існує тоді оновлюється його кількість на складі, якщо такого матеріалу іще немає тоді додається новий.

```

public virtual int AddMaterial(Nullable<int> typeId, string name,
Nullable<decimal> price, Nullable<int> quantity, Nullable<int> countryId)
{
    var typeIdParameter = typeId.HasValue ?
        new ObjectParameter("TypeId", typeId) :
        new ObjectParameter("TypeId", typeof(int));

    var nameParameter = name != null ?
        new ObjectParameter("Name", name) :
        new ObjectParameter("Name", typeof(string));

    var priceParameter = price.HasValue ?
        new ObjectParameter("Price", price) :
        new ObjectParameter("Price", typeof(decimal));

    var quantityParameter = quantity.HasValue ?
        new ObjectParameter("Quantity", quantity) :
        new ObjectParameter("Quantity", typeof(int));

    var countryIdParameter = countryId.HasValue ?
        new ObjectParameter("CountryId", countryId) :

```

```

        new ObjectParameter("CountryId", typeof(int));

        return
        ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("AddMaterial",
        typeIdParameter, nameParameter, priceParameter, quantityParameter, countryIdParameter);
    }

```

## Процедура додавання нового матеріалу

```

DROP PROCEDURE IF EXISTS dbo.AddMaterial
GO
CREATE PROCEDURE dbo.AddMaterial
(
    @TypeId int,
    @Name nvarchar(50),
    @Price decimal,
    @Quantity int,
    @CountryId int
)
AS
DECLARE @MaterialId int

SELECT @MaterialId = Id
FROM Material
WHERE Material.MaterialTypeId = @TypeId AND
      Material.CountryId = @CountryId AND
      Material.[Name] = @Name

IF NOT (ISNUMERIC(@MaterialId) = 1)
BEGIN
    PRINT 'Material do not exist in
table Material'

    INSERT INTO Material
(MaterialTypeId, CountryId,
[Name])
VALUES
(@TypeId, @CountryId, @Name)

    SELECT @MaterialId = Id
    FROM Material
    WHERE Material.MaterialTypeId
= @TypeId AND
      Material.[Name] = @Name AND
      Material.CountryId = @CountryId

    IF (ISNUMERIC(@MaterialId) = 1)
    BEGIN
        PRINT 'Material exist in Table
Storage with index'

        PRINT 'Update quantity of
material in Table Storage'

        UPDATE Storage
        SET Quantity = Quantity +
@Quantity
        WHERE Id = @StorageId
    END
    ELSE
    BEGIN
        PRINT 'Material do not exist in
Table Storage'

        PRINT 'Insert new material in
Table Storage'

        INSERT Storage
        (Price, Quantity, MaterialId)
        VALUES

```

```

(@Price,          @Quantity,          END
@MaterialId)      PRINT 'Material data updated'

```

### 3.2.8. Додавання відомостей про заробітну плату працівникам

Зчитуються дані з форми та передаються в сторед процедуру.

```

public virtual int AddSalary(Nullable<int> workerId, Nullable<System.DateTime>
date, Nullable<decimal> salary)
{
    var workerIdParameter = workerId.HasValue ?
        new ObjectParameter("WorkerId", workerId) :
        new ObjectParameter("WorkerId", typeof(int));

    var dateParameter = date.HasValue ?
        new ObjectParameter("Date", date) :
        new ObjectParameter("Date", typeof(System.DateTime));

    var salaryParameter = salary.HasValue ?
        new ObjectParameter("Salary", salary) :
        new ObjectParameter("Salary", typeof(decimal));

    return
        ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("AddSalary",
workerIdParameter, dateParameter, salaryParameter);
}

```

### Додавання заробітної плати робітнику

```

DROP PROCEDURE IF EXISTS dbo.AddSalary
GO
CREATE PROCEDURE dbo.AddSalary
@WorkerId int,
@Date date,
@Salary decimal
AS
If NOT EXISTS(SELECT * FROM Worker WHERE Worker.Id = @WorkerId)
BEGIN
    PRINT 'Worker not found'
    RETURN
END

INSERT INTO SalaryList
(WorkerId, [Date], Salary)
VALUES (@WorkerId, @Date, @Salary)
GO;

```



3.2.9. Створення заявки на потребу нового матеріалу для постачання  
Зі списку матеріалів вибирається матеріал. Далі цей матеріал і кількість передаються в сторед процедуру

```
public virtual int AddToSupplyItem(Nullable<int> materialId, Nullable<int>
quantityNeed)
{
    var materialIdParameter = materialId.HasValue ?
        new ObjectParameter("MaterialId", materialId) :
        new ObjectParameter("MaterialId", typeof(int));

    var quantityNeedParameter = quantityNeed.HasValue ?
        new ObjectParameter("QuantityNeed", quantityNeed) :
        new ObjectParameter("QuantityNeed", typeof(int));

    return
        ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("AddToSupplyItem",
        materialIdParameter, quantityNeedParameter);
}
```

### Додавання нового матеріалу для постачання

```
DROP PROCEDURE IF EXISTS dbo.AddToSupplyItem
GO
CREATE PROCEDURE dbo.AddToSupplyItem
    @MaterialId int,
    @QuantityNeed int
AS
DECLARE @SupplyItemId int
SELECT @SupplyItemId = Id
FROM SupplyItem as S
WHERE S.MaterialId = @MaterialId

IF(ISNUMERIC(@SupplyItemId) = 1)
    BEGIN
        PRINT 'Updated row SupplyItem'
        UPDATE SupplyItem
        SET Quantity = Quantity + @QuantityNeed
    END
ELSE
    BEGIN
        PRINT 'Added row into SupplyItem'
        INSERT INTO SupplyItem
            (Quantity, MaterialId)
        VALUES
            (@QuantityNeed, @MaterialId)
```

```
END
GO;
```

### 3.2.10. Додавання нового працівника

Зчитуються дані з форми і передаються в збережину процедуру. У збереженій процерурі відбувається перевірка на правильність введених даних і їх допустимість на додавання до бази даних. Потім відбувається перевірка чи цей працівник вже існує в БД. Якщо він існує тоді відбувається оновлення даних про нього, якщо ні – додавання нового працівника.

#### Процедура додавання нового працівника

```
DROP PROCEDURE IF EXISTS AddWorker
GO
CREATE PROCEDURE dbo.AddWorker
(
    @PersonName nvarchar(50),
    @PersonSurName nvarchar(50),
    @PersonPhoneNumber nvarchar(10),
    @PersonEmail nvarchar(50),
    @WorkerType nvarchar(50),
    @PassWord nvarchar(50)
)
AS
BEGIN
    BEGIN TRAN
    DECLARE @uperr int
    DECLARE @maxerr int
    DECLARE @PersonId int
    EXEC @PersonId= dbo.AddPerson @PersonName, @PersonSurName, @PersonPhoneNumber, @PersonEmail;

    IF(ISNUMERIC (@WorkerType) = 1)
        BEGIN
            PRINT 'you entered wrong data about worker type' + CHAR(13) + 'rollback trun'
            ROLLBACK
            RETURN
        END

    DECLARE @CheckPassWord int

    SELECT @CheckPassWord = Id
    FROM Worker
    WHERE [PassWord] = @PassWord

    IF(ISNUMERIC (@CheckPassWord) = 1)
        BEGIN
            PRINT 'you entered wrong data about worker password!!! Password must be unique!!!' + CHAR(13) + 'rollback
trun'
            ROLLBACK
            RETURN
        END

    DECLARE @WorkerTypeId int = 0

    SELECT @WorkerTypeId = w.Id
```

```

FROM WorkerType as w
WHERE w.[Name] = @WorkerType

IF(@WorkerTypeid = 0)
    BEGIN
        INSERT INTO WorkerType([Name])
            VALUES (@WorkerType)

        SELECT @WorkerTypeid = w.Id
            FROM WorkerType as w
    END

IF EXISTS(SELECT * FROM Worker WHERE PersonId = @PersonId)
    BEGIN
        IF EXISTS(SELECT *
            FROM Worker AS w
            WHERE PersonId = @PersonId AND
                w.Typeid IN (
                    SELECT Id
                    FROM WorkerType
                    WHERE [Name] = @WorkerType)
            )
            BEGIN
                PRINT 'data about worker is exist in table' + CHAR(13) + 'Commit trun'
                COMMIT
            END
        ELSE
            BEGIN
                UPDATE Worker
                SET Typeid = @WorkerTypeid,
                    [PassWord] = @PassWord
                WHERE PersonId = @PersonId

                UPDATE Worker
                SET Typeid = @WorkerTypeid
                WHERE PersonId = @PersonId

                BEGIN
                    SET @uperr = @@error
                    IF @uperr > @maxerr
                        SET @maxerr = @uperr

                    -- If an error occurred, roll back
                    IF @maxerr <> 0
                        BEGIN
                            ROLLBACK
                            PRINT 'did bot updated ' + CHAR(13) + ' Transaction
rolled back'

                            RETURN
                        END
                    ELSE
                        BEGIN
                            COMMIT
                            PRINT 'updated successfully ' + CHAR(13) + '
Transaction committed'

                            END
                        END
                    END
                END
            END
        ELSE
            BEGIN
                PRINT 'data about worker is NOT exist in table'
                INSERT INTO Worker (Typeid, PersonId, [PassWord])
                VALUES (@WorkerTypeid, @PersonId, @PassWord);
            END
    END

```

```

BEGIN
    SET @uperr = @@error
    IF @uperr > @maxerr
        SET @maxerr = @uperr

    -- If an error occurred, roll back
    IF @maxerr <> 0
        BEGIN
            ROLLBACK
            PRINT 'did bot inserted ' + CHAR(13) + ' Transaction rolled back'
            RETURN
        END
    ELSE
        BEGIN
            COMMIT
            PRINT 'inserted successfully ' + CHAR(13) + ' Transaction committed'
        END
    END
END
END

```

### 3.2.11. Обрахунок вартості продукту та замовлення

Для обрахунку вартості замовлення потрібно натиснути на кнопку „обрахувати” далі в обробнику кнопки відбувається виклик збереженої процедури яка за вказаним ідентифікатором замовлення відбувається орахунок вартості продуктів, які входили до замовлення. Потім за знайденими вартостями продуктів знаходиться загальна вартість замовлення.

#### Процедура обрахунку вартості замовлення

```

DROP PROCEDURE IF EXISTS dbo.CalculateOrderPrice
GO
CREATE PROCEDURE dbo.CalculateOrderPrice
    @OrderId int
AS
    DECLARE @TotalPrice decimal = 0--, @ProductId int = 1

    SELECT @TotalPrice = @TotalPrice + p.Price
    FROM Product AS p
    WHERE p.OrderId = @OrderId

    PRINT 'Total price of Order is'
    PRINT @TotalPrice
    --SELECT * FROM vMaterial

    IF (@TotalPrice != 0)
        BEGIN
            PRINT 'TotalPrice is correct'

            UPDATE [Order]
            SET TotalPrice = @TotalPrice
            WHERE [Order].Id = @OrderId

            END
        ELSE
            BEGIN
                PRINT 'TotalPrice calculated incorrectly'

                RETURN
            END
        GO;

```

## Процедура обрахунку вартості продукту

```

DROP PROCEDURE IF EXISTS dbo.CalculateProductPrice
GO

CREATE PROCEDURE dbo.CalculateProductPrice
    @ProductId int
AS
    DECLARE @TotalPrice decimal = 0--, @ProductId int = 1
    SELECT @TotalPrice = @TotalPrice + s.Price
    FROM Storage AS s
    WHERE s.Id IN (
        SELECT MaterialId
        FROM MaterialList
        WHERE MaterialList.ProductId =
@ProductId
    )

    PRINT 'Total price is'
    GO;

PRINT @TotalPrice
--SELECT * FROM vMaterial

IF(@TotalPrice != 0)
    BEGIN
        PRINT 'TotalPrice is correct'
        UPDATE Product
        SET Price = @TotalPrice
        WHERE Product.Id = @ProductId
    END
ELSE
    BEGIN
        PRINT 'TotalPrice calculated
incorrectly'
    END
RETURN
END

```

### 3.2.12. Створення нової поставки товарів

У відповідному вікні потрібно вибрати товари, які потрібно передати на поставку та передати список в процедуру.

В процедурі по чергово перебирають та додаються до списку поставки товари які передані в процедуру

### Створення нової поставки товарів

```

DROP Type ItemList
GO

CREATE TYPE ItemList AS TABLE
(
    Id int Identity(1, 1) PRIMARY KEY,
    ItemId int
)
Go

DROP PROCEDURE IF EXISTS dbo.CreateSupplyOrder
GO

CREATE PROCEDURE dbo.CreateSupplyOrder
    @Items ItemList READONLY,
    @ManagerId int
AS
    DECLARE @ManagerTypeId int, @res bit
    SELECT @ManagerTypeId = Id FROM WorkerType
    WHERE WorkerType.Name = 'manager'
    --SELECT @EngineerTypeId

```

```
SET @res = dbo.WorkerIs(@ManagerTypeId,
@ManagerId)
```

```
IF(dbo.WorkerIs(@ManagerTypeId, @ManagerId) = 0)
BEGIN
    PRINT 'Wrong ManagerId'
    RETURN
END
```

```
DECLARE @CurrentOrderNumber int, @CurrentOrderId
int
```

```
SELECT @CurrentOrderNumber =
MAX(SupplyOrder.OrderNumber) + 1
FROM SupplyOrder
```

```
IF NOT(ISNUMERIC(@CurrentOrderNumber) = 1)
BEGIN
    SET @CurrentOrderNumber = 0
END
```

```
INSERT INTO SupplyOrder
(OrderNumber, ManagerId)
VALUES
(@CurrentOrderNumber, @ManagerId)
```

```
SELECT @CurrentOrderId = Id
FROM SupplyOrder AS s
```

```
WHERE s.OrderNumber = @CurrentOrderNumber
```

```
DECLARE @ItemId int
```

```
DECLARE OrderCursor CURSOR FOR
SELECT
ItemId
FROM @Items
```

```
OPEN OrderCursor
```

```
FETCH NEXT FROM OrderCursor
INTO
@ItemId
```

```
WHILE @@FETCH_STATUS = 0
BEGIN
```

```
UPDATE SupplyItem
SET SupplyItem.SypplyOrderId = @CurrentOrderId
WHERE SupplyItem.Id = @ItemId
```

```
END
FETCH NEXT FROM OrderCursor
INTO
@ItemId
```

```
CLOSE OrderCursor
DEALLOCATE OrderCursor
```

### 3.2.13. Оплата замовлення

У відповідному вікні потрібно вибрати замовлення. Далі дані про оплату передаються в процедуру для збереження оплати. В процедурі відбувається перевірка чи вибране замовлення існує в базі даних. Якщо воно не існує тоді відбувається відкат транзакції та повідомлення про помилку

### Процедура оплати замовлення

```

DROP PROCEDURE IF EXISTS dbo.Pay
GO
CREATE PROCEDURE dbo.Pay
    @PersonId int,
    @OrderId int,
    @Paid decimal
AS
BEGIN TRAN

    If NOT EXISTS(SELECT * FROM [Order] WHERE
[Order].Id = @OrderId)
        BEGIN
            PRINT 'Order not found'
            ROLLBACK
            RETURN
        END

    BEGIN TRY

        PRINT 'insert new payment'
        INSERT INTO Payment
        (OrderId, PayerId, Paid)
        VALUES (@OrderId, @PersonId, @Paid)

        DECLARE @CurrentPaid decimal

        SELECT @CurrentPaid = Paid
        FROM [Order]

        WHERE [Order].Id = @OrderId

        IF NOT(ISNUMERIC(@CurrentPaid) = 1)
        BEGIN
            UPDATE [Order]
            SET Paid = 0
        END

        PRINT 'Udate payment in Order table'
        UPDATE [Order]
        SET Paid = Paid + @Paid
        WHERE [Order].Id = @OrderId

        COMMIT TRAN

    END TRY

    BEGIN CATCH

        PRINT 'Cannot add new payment'
        ROLLBACK

    END CATCH

    --end of procedure

GO;

```

### 3.3. Опис роботи програми

При вході в систему користувач повинен авторизуватися за допомогою логіну(прізвища) та паролю, які йому видав менеджер персоналом.

Прізвище

Пароль

Рис. 3.1. Сторінка входу в систему

[Матеріали](#) / [Постачальники](#) / [Замовлення](#) / [Працівники](#)

Рис. 3.2. Меню вибору головних сторінок

Для того щоб додати нове замовлення нам потрібно натиснути на кнопку „ додати замовлення”. Відповідно після цього з’являється відповідне вікно, куди потрібно внести дані про замовлення. Та потрібно додати вікна які нам потрібно виготовити.

Прізвище: <input type="text" value="Петров"/>	Профіль <input type="text" value="someProf"/>
Ім'я: <input type="text" value="Петро"/>	Фурнітура <input type="text" value="bestFur"/>
Телефон: <input type="text" value="+380943532462"/>	Скло <input type="text" value="GoodGlass"/>
Пошта: <input type="text" value="Petya@gmail.com"/>	Ширина <input type="text" value="150"/>
	Висота <input type="text" value="100"/>
<input type="button" value="Завершити замов"/>	
<input type="button" value="Замовлені вікна"/>	<input type="button" value="Add"/>



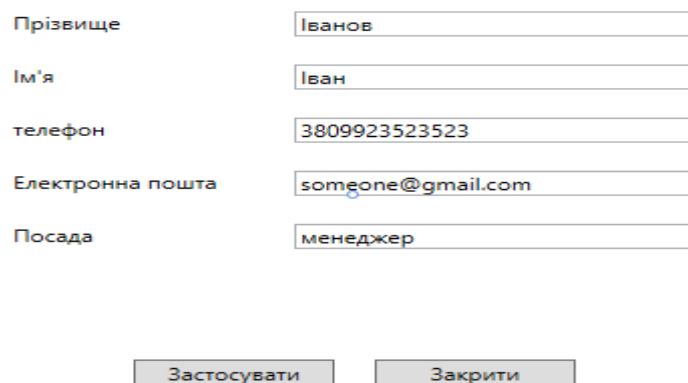
Рис. 3.3. Додавання нового замовлення

Для того щоб задати яку кількість кожного матеріалу нам потрібно для виготовлення кожного вікна нам потрібно натиснути кнопку „додати матеріали”. Після цього з’являється відповідне вікно. Куди можна внести необхідну кількість.

Назва	<input type="text" value="someProfile"/>
Тип	<input type="text" value="profile"/>
Кількість	<input type="text" value="4"/>
Ціна	<input type="text" value="400"/>

Рис. 3.4. Задавання кількості матеріалів

Для того щоб зареєструвати нового користувача необхідно натиснути кнопку „Додати користувача”. Після цього з’явить відповідне вікно для внесення даних.



Прізвище	Іванов
Ім'я	Іван
телефон	3809923523523
Електронна пошта	someone@gmail.com
Посада	менеджер

Застосувати    Закрити

Рис. 3.5. Додавання нового користувача

Для того щоб переглянути замолення потрібно натиснути на потрібне із списку замовлень. Після цього з'явиться відповідне вікно. Де можна ознайомитися із всією необхідною інформацією.

Order

**Замовлення № 123****Від: 15.05.2021 До 22.05.2021****Укладач Степан Микитич****Інженер складання: Смирнов****Оплачена вартість : 200****Загальна вартість: 400****Позиції замовлення:**

#	Запланована дата	Дата виконання
1	20.05.2021	19.05.2021
2	19.05.2021	20.05.2021

**Позиція №2**

Код	Висота	Ширина	Ціна	відомо
2	150	100	300	відомо

Рис. 3.6. Перегляд характеристики та особливостей замовлення

У відомостях можна переглянути які платіжки було проведено що до оплати замовлення.

[Додати новий](#)

Курсант-платник	Сума	Дата оплати
O'Hare George	13329.00	08.04.2021 02:25:10
O'Hare George	1000.00	30.04.2021 15:23:00
O'Hare George	1000.00	30.04.2021 15:23:00
Jeaffreson Seth	13217.00	26.04.2021 05:37:24
McGilroy Willabella	11049.00	22.03.2021 07:50:00
Le Lievre Chrissie	9843.00	07.03.2021 09:33:52
Gillingwater Christalle	12984.00	03.04.2021 11:43:20

Рис. 3.7. Перегляд інформації про оплату замовлення

Щоб переглянути інформацію про працівників фірми, необхідно перейти на вкладку „Працівники”, після чого з’явиться відповідне вікно. Де можна вибирати працівників із списку зліва та переглядати інформацію про них справа. Також можна редагувати дані працівників.

### Працівники

ID	Повне ім'я	Статус	Посада
1	Андрієнко Андрій Андрійович	працює	менеджер

Додати працівника

Редагувати працівника

Звільнити працівника

**Прізвище** Андрієнко

**Ім'я** Андрій

**По батькові** Андрійович

**Адрес** м.ю Львів

**Дата народження** 12.02.2000

**Дата працевлаштування** 20.05.2021

**Контактний телефон:** 095463462:

Рис. 3.7. Перегляд про працівників фірми

Щоб створити поставку товарів, необхідно спочатку зайти в систему як постачальник. Далі у вкладці матеріали необхідно натиснути на кнопку створити поставку, після чого з'явиться відповідне вікно.

У вікні створення поставки можна задати список матеріалів. Зліва відображається інформація про постачальника.

Прізвище:	тип матеріалу
<input type="text" value="Петров"/>	<input type="text" value="профіль"/>
Ім'я:	Ім'я матеріалу
<input type="text" value="Петро"/>	<input type="text" value="bestFur"/>
Телефон:	Вартість
<input type="text" value="+380943532462"/>	<input type="text" value="200"/>
Пошта:	Кількість
<input type="text" value="Petya@gmail.com"/>	<input type="text" value="150"/>
	Країна походження
<input type="button" value="Завершити замов"/>	<input type="text" value="Україна"/>
<input type="text" value="Позиції поставки"/>	<input type="button" value="Add"/>

Рис. 3.8. Створення поставки товарів

## ВИСНОВКИ

У ході виконання даної курсової роботи було створено базу даних для предметної області виробництво віконних систем та додаток для взаємодії з базою, використовуючи технології Microsoft SQL Server для бази даних, ASP.NET для розгортання веб-додатку та Entity Framework для взаємодії з базою даних.

Для розробки я використав такі середовища як Erwin Data Modeler 7.1 для розробки концептуальних та логічних схем бази даних, Microsoft SQL Server Management Studio 2018 для реалізації розроблених схем і Microsoft Visual Studio 2019 для написання та відлагодження додатку. Недоліком розробленої програмної системи є відсутність авторизації та автентифікації.

Уперспективі можна розділити додаток на бекенд з підтримкою REST API, написаному на C# з використанням ASP.NET, та фронтенд, написаний з використанням WPF.

Серед недоліків розробленого програмного продукту можна виділити залежність від конкретної СУБД та застарілість WPF.

У програмному застосунку реалізовано всі функціональні вимоги, які описані в специфікації вимог до ПЗ.

Для цієї програмної системи можна виділити такі перспективи покращення:

- покращення функціоналу для управління працівниками.
- покращення функціоналу системи, що стосується роботи працівника по відношенню до технічної складової (обладнання).
- додавання нової інформації до замовлення.

## СПИСОК ЛІТЕРАТУРИ

1. Entity Framework Core with Existing Database [Електронний ресурс]. – Режим доступу: <https://www.entityframeworktutorial.net/efcore/create-model-for-existing-database-in-ef-core.aspx>.
2. Петкович Д. Microsoft SQL Server 2012. Руководство для начинающих: Пер. с англ. — СПб.: БХВ-Петербург, 2013. — 816 с.
3. Razor Pages with Entity Framework Core in ASP.NET Core [Електронний ресурс]. — Режим доступу: <https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/intro>.
4. Entity Framework Database First. MSDN: Introduction to Entity Framework [Електронний ресурс]. — Режим доступу: [https://msdn.microsoft.com/en-us/library/jj206878\(v=vs.113\).aspx](https://msdn.microsoft.com/en-us/library/jj206878(v=vs.113).aspx).
5. SQL Server Documentation [Електронний ресурс]. – Режим доступу: <https://docs.microsoft.com/ru-ru/sql/sql-server/sql-server-technical-documentation>.

## Додаток А. Скрипт створення БД

```

create table Person(
    Id int NOT NULL primary key,
    [Name] nvarchar(50) DEFAULT('Name'),
    Surname nvarchar(50) NOT NULL,
    PhoneNumber varchar(10) NOT NULL,
    Email nvarchar(50)
);

create table WorkerType(
    Id int identity(1, 1) NOT NULL primary key,
    [Name] nvarchar(50) NOT NULL
);

create table Worker(
    Id int identity(1, 1) NOT NULL primary key,
    PersonId int foreign key references Person(Id),
    TypeId int foreign key references WorkerType(Id)
);

create table SalaryList(
    Id int identity(1,1) NOT NULL primary key,
    WorkerId int foreign key references Worker(Id),
    [Date] date,
    Salary decimal NOT NULL
);

create table [Order](
    Id int identity(1,1) NOT NULL primary key,
    OrderNumber int NOT NULL,
    CustomerId int NOT NULL foreign key references
Person(Id),
    ManagerId int NOT NULL foreign key references
Person(Id),
    TotalPrice decimal,
    Paid decimal,
    DateCreate datetime DEFAULT(CONVERT(varchar,
getdate(), 0)),
    DateComplited datetime,
    ApproximateDateComplite date
DEFAULT(CONVERT(varchar, GETDATE() + 7, 1))
);

create table Payment(
    Id int identity(1,1) NOT NULL primary key,
    OrderId int NOT NULL foreign key references
[Order](Id),
    PayerId int foreign key references Person(Id),
    Paid decimal,
    [Date] datetime DEFAULT GETDATE()
);

create table Color(
    Id int identity(20, 1) NOT NULL primary key,
    [Name] nvarchar(50) NOT NULL
);

create table Product(
    Id int identity(1,1) NOT NULL primary key,
    ColorId int NOT NULL foreign key references Color(Id),
    OrderId int foreign key references [Order](Id),
    EngineerId int NOT NULL foreign key references
Worker(Id),
    Height int,
    Width int,
    DateCreate datetime DEFAULT GETDATE(),
    DateComplite datetime,
    Price decimal
);

create table Country(
    Id int identity(50, 1) NOT NULL primary key,
    [Name] nvarchar(50) NOT NULL
);

create table MaterialType(
    Id int NOT NULL primary key,
    [Name] nvarchar(50) NOT NULL,
    QuantityType nvarchar(50) NOT NULL
);

create table Material(
    Id int identity(1, 1) NOT NULL primary key,
    MaterialTypeId int foreign key references
MaterialType(Id),
    CountryId int foreign key references Country(Id),
    [Name] nvarchar(50)
);

create table Storage(
    Id int identity(1, 1) NOT NULL primary key,
    MaterialId int foreign key references Material(Id),
    Price decimal,
    Quantity int
);

create table MaterialList(
    Id int identity(1, 1) NOT NULL primary key,
    MaterialId int foreign key references Storage(Id),
    ProductId int foreign key references Product(Id),
    [Date] datetime DEFAULT GETDATE(),
    Quantity int
);

create table SupplyOrder(
    Id int identity(1,1) NOT NULL primary key,
    OrderNumber int NOT NULL,
    DateCreate datetime DEFAULT GETDATE(),
    DateComplite datetime,
    ManagerId int foreign key references Worker(Id),
);

create table Offer(
    Id int NOT NULL primary key,
    SupplierId int foreign key references Person(Id),
    CompanyName nvarchar(50),
    MaterialId int foreign key references Material(Id),
    Quantity int,
    Price decimal,
    DeliveryDate datetime
);

create table SupplyItem(
    Id int identity(1, 1) NOT NULL primary key,
    MaterialId int foreign key references Material(Id),

```

```

        Quantity int,
        CreationDate date DEFAULT GETDATE(),
        CompletionDate date DEFAULT GETDATE(),
        OfferId int foreign key references Offer(Id),
        SupplyOrderId int foreign key references
SupplyOrder(Id)
);

--Створення віртуальних таблиць
--Таблиця для представлення матеріалів
DROP VIEW IF EXISTS vMaterial
GO
CREATE VIEW vMaterial
AS

SELECT s.Id,
        mt.Id as 'MaterialTypeId',
        mt.[Name] as 'MaterialType',
        m.[Name],
        s.Price,
        s.Quantity,
        c.[Name] as 'Country',
        mt.QuantityType as 'QuantityType'
FROM Storage as s
JOIN Material as m
        ON s.MaterialId = m.Id
JOIN MaterialType as mt
        ON m.MaterialTypeId = mt.Id
JOIN Country as c
        ON m.CountryId = c.Id

--Віртуальна таблиця для представлення матеріалу для
постачання
USE WindowFactory

DROP VIEW IF EXISTS vSupplyItem
GO
CREATE VIEW vSupplyItem
AS

SELECT s.Id,
        o.Price,
```

```

        s.Quantity,
        s.CreationDate,
        s.OfferId as 'OfferId',
        s.SupplyOrderId,
        m.[Name] as 'MaterialName',
        mt.Id as 'MaterialTypeId',
        m.[Name],
        c.[Name] as 'CountryName',
        mt.[Name] as 'MaterialType',
        mt.QuantityType as 'QuantityType'
FROM SupplyItem as s
JOIN Offer as o
        ON o.Id = s.OfferId
JOIN Material as m
        ON m.Id = s.MaterialId
JOIN MaterialType as mt
        ON mt.Id = m.MaterialTypeId
JOIN Country as c
        ON m.CountryId = c.Id

--Віртуальна таблиця для представлення працівника
фірми
DROP VIEW IF EXISTS vWorker
GO
CREATE VIEW vWorker
AS

SELECT
        w.Id,
        p.[Name],
        p.Surname,
        p.PhoneNumber,
        p.Email,
        wt.[Name] as 'WorkerType'
FROM Worker as w
JOIN Person as p
        ON w.PersonId = p.Id
JOIN WorkerType as wt
        ON w.TypeId = wt.Id
```



SQL Statement	SQL Statement
DROP PROCEDURE IF EXISTS dbo.AddMaterial	END
GO	
CREATE PROCEDURE dbo.AddMaterial	PRINT 'This Material exists in Table Material'
(	
@TypeId int,	DECLARE @StorageId int
@Name nvarchar(50),	
@Price decimal,	SELECT @StorageId = Id
@Quantity int,	FROM Storage
@CountryId int	WHERE Storage.MaterialId = @MaterialId AND
)	Storage.Price = @Price
AS	
DECLARE @MaterialId int	IF (ISNUMERIC(@StorageId) = 1)
	BEGIN
SELECT @MaterialId = Id	PRINT 'Material exist in Table
FROM Material	Storage with index'
WHERE Material.MaterialTypeId = @TypeId AND	PRINT 'Update quantity of
Material.CountryId = @CountryId AND	material in Table Storage'
Material.[Name] = @Name	
	UPDATE Storage
IF NOT (ISNUMERIC(@MaterialId) = 1)	SET Quantity = Quantity +
BEGIN	@Quantity
PRINT 'Material do not exist in	WHERE Id = @StorageId
table Material'	END
	ELSE
INSERT INTO Material	BEGIN
(MaterialTypeId, CountryId,	PRINT 'Material do not exist in
[Name])	Table Storage'
VALUES	PRINT 'Insert new material in
(@TypeId, @CountryId, @Name)	Table Storage'
SELECT @MaterialId = Id	INSERT Storage
FROM Material	(Price, Quantity, MaterialId)
WHERE Material.MaterialTypeId	VALUES
= @TypeId AND	(@Price, @Quantity,
Material.[Name] = @Name AND	@MaterialId)
Material.CountryId =	END
@CountryId	PRINT 'Material data updated'

## Процедура додавання нового замовлення

```

DROP PROCEDURE IF EXISTS dbo.AddOrder
GO
CREATE PROCEDURE dbo.AddOrder

--Customer Data
@CustomerId nvarchar(50),
--Manager Data
@ManagerId int
--
AS

DECLARE @ManagerTypeId int, @res bit
SELECT @ManagerTypeId = Id FROM WorkerType
WHERE WorkerType.Name = 'manager'

--SELECT @ManagerTypeId

SET @res = dbo.WorkerIs(@ManagerTypeId,
@ManagerId)

IF(dbo.WorkerIs(@ManagerTypeId, @ManagerId)
= 0)

BEGIN

PRINT 'Wrong ManagerId'

RETURN

END

DECLARE @CurrentOrderNumber int

SELECT @CurrentOrderNumber =
MAX(OrderNumber) + 1

FROM [Order]

IF NOT(ISNUMERIC(@CurrentOrderNumber) = 1)

SET @CurrentOrderNumber = 1

--VALUES(Id, OrderNumber, CustomerId,
ManagerId, TotalPrice, Paid, DateCreate,
DateComplited, ApproximateDateComplite)

INSERT INTO [Order]

(CustomerId, OrderNumber, ManagerId)

VALUES(

@CustomerId,

@CurrentOrderNumber,

@ManagerId)

EXEC dbo.AddOrder 1, 5

```

## Процедура додавання нової особи

```

DROP PROCEDURE IF EXISTS dbo.AddPerson
GO
CREATE PROCEDURE dbo.AddPerson
(
@PersonName nvarchar(50),
@PersonSurName nvarchar(50),
@PersonPhoneNumber nvarchar(10),
@PersonEmail nvarchar(50)
)
AS
BEGIN

```

```

DECLARE @PersonId int = 0
DECLARE @uperr int
DECLARE @maxerr int

SELECT @PersonId = Person.Id
FROM Person
WHERE @PersonName = Person.Name AND
@PersonSurName = Person.Surname

IF (@PersonId != 0)
    BEGIN

        IF EXISTS (SELECT *
FROM Person
WHERE @PersonName = Person.Name AND
@PersonSurName = Person.Surname AND
@PersonPhoneNumber = Person.PhoneNumber AND
@PersonEmail = Person.Email)
            BEGIN
                PRINT 'Person exist in table'
                RETURN @PersonId
            END
        ELSE
            PRINT 'Person exist in table Try update person data. ' + CHAR(13) + ' CustomerId = ' +
CAST(@PersonId AS nvarchar);

            BEGIN TRAN

                UPDATE Person
                SET PhoneNumber = @PersonPhoneNumber,
                    Email = @PersonEmail
                WHERE @PersonId = Person.Id

            BEGIN
                SET @uperr = @@error
                IF @uperr > @maxerr
                    SET @maxerr = @uperr

                -- If an error occurred, roll back
                IF @maxerr <> 0
                    BEGIN
                        ROLLBACK
                        PRINT 'did bot updated ' + CHAR(13) + ' Transaction
rolled back'

```

```

                                END
                                ELSE
                                BEGIN
                                COMMIT
                                PRINT 'updated successfully ' + CHAR(13) + '
Transaction committed'
                                END
                                END

                                RETURN @PersonId
                                END

ELSE
BEGIN
--Person don't exist in table
--Insert person

PRINT 'Person did not exist in table ' + CHAR(13) + ' Try to Insert in table'

BEGIN TRAN

DECLARE @InsertedPersonId int = (SELECT MAX(Id) + 1 FROM Person)

INSERT INTO Person
(Id, [Name], Surname, PhoneNumber, Email)
VALUES (
    @InsertedPersonId,
    @PersonName,
    @PersonSurName,
    @PersonPhoneNumber,
    @PersonEmail
)

BEGIN
    SET @uperr = @@error
    IF @uperr > @maxerr
    SET @maxerr = @uperr

    -- If an error occurred, roll back
    IF @maxerr <> 0
    BEGIN
        ROLLBACK
        PRINT 'did bot updated ' + CHAR(13) + ' Transaction rolled back'
    END
END

```

```

ELSE
    BEGIN
        COMMIT
        PRINT 'updated successfully ' + CHAR(13) + ' Transaction committed'
    END
END

RETURN @InsertedPersonId

END
END

```

## Додавання нового продукту

```

DROP PROCEDURE IF EXISTS dbo.AddProduct
GO

CREATE PROCEDURE dbo.AddProduct
    @OrderNumber int,
    --Engineer Data
    @EngineerId int,
    --Product Data
    @ColorId int,
    @Height int,
    @Width int,

    @ProfileId int,
    @GlassId int,
    @FurnitureId int
AS
    DECLARE @EngineerTypeId int, @res bit
    SELECT @EngineerTypeId = Id FROM WorkerType
    WHERE WorkerType.Name = 'engineer'
    --SELECT @EngineerTypeId

    SET @res = dbo.WorkerIs(@EngineerTypeId,
    @EngineerId)

    IF(dbo.WorkerIs(@EngineerTypeId, @EngineerId) = 0)
    BEGIN
        PRINT 'Wrong EngineerId'
        RETURN
    END

    DECLARE @ProductId int
    SELECT @ProductId = MAX(Id) + 1
    FROM Product

    PRINT ISNUMERIC(@ProductId)
    IF NOT(ISNUMERIC(@ProductId) = 1)
        SET @ProductId = 1

    PRINT @ProductId

    PRINT 'INSERT VALUES(Id, ColorId, OrderId, EngineerId,
    Height, Width, DateComplite, Price)'
    SET IDENTITY_INSERT Product ON
    INSERT INTO Product
    (Id, ColorId, OrderId, EngineerId, Height, Width)
    VALUES(
        @ProductId,
        @ColorId,
        @OrderNumber,
        @EngineerId,
        @Height,
        @Width
    )

```

```

PRINT 'INSERT VALUES(Id, MaterialId, ProductId, [Date],
Quantity)'

INSERT INTO MaterialList
(MaterialId, ProductId)
VALUES(
    @FurnitureId,
    @ProductId
)

INSERT INTO MaterialList
(MaterialId, ProductId)
VALUES(
    @GlassId,
    @ProductId
)

INSERT INTO MaterialList
(MaterialId, ProductId)
VALUES(
    @ProfileId,
    @ProductId
)

PRINT 'Complited'
GO

```

## Додавання заробітної плати робітнику

```

DROP PROCEDURE IF EXISTS dbo.AddSalary
GO

CREATE PROCEDURE dbo.AddSalary
    @WorkerId int,
    @Date date,
    @Salary decimal
AS
If NOT EXISTS(SELECT * FROM Worker WHERE Worker.Id = @WorkerId)
BEGIN
    PRINT 'Worker not found'
    RETURN
END

INSERT INTO SalaryList
(WorkerId, [Date], Salary)
VALUES (@WorkerId, @Date, @Salary)
GO;

```

## Додавання нового матеріалу для постачання

```

DROP PROCEDURE IF EXISTS dbo.AddToSupplyItem
GO

CREATE PROCEDURE dbo.AddToSupplyItem
    @MaterialId int,
    @QuantityNeed int
AS
DECLARE @SupplyItemId int
SELECT @SupplyItemId = Id

```

```

FROM SupplyItem as S
WHERE S.MaterialId = @MaterialId

IF(ISNUMERIC(@SupplyItemId) = 1)
    BEGIN
        PRINT 'Updated row SupplyItem'
        UPDATE SupplyItem
        SET Quantity = Quantity + @QuantityNeed
    END
ELSE
    BEGIN
        PRINT 'Added row into SupplyItem'
        INSERT INTO SupplyItem
            (Quantity, MaterialId)
        VALUES
            (@QuantityNeed, @MaterialId)
    END
GO;

```

## Процедура додавання нового працівника

```

DROP PROCEDURE IF EXISTS AddWorker
GO
CREATE PROCEDURE dbo.AddWorker
(
    @PersonName nvarchar(50),
    @PersonSurName nvarchar(50),
    @PersonPhoneNumber nvarchar(10),
    @PersonEmail nvarchar(50),
    @WorkerType nvarchar(50),
    @PassWord nvarchar(50)
)
AS
BEGIN
    BEGIN TRAN
    DECLARE @uperr int
    DECLARE @maxerr int
    DECLARE @PersonId int
    EXEC @PersonId= dbo.AddPerson @PersonName, @PersonSurName, @PersonPhoneNumber, @PersonEmail;

    IF(ISNUMERIC (@WorkerType) = 1)
        BEGIN
            PRINT 'you entered wrong data about worker type' + CHAR(13) + 'rollback trun'
            ROLLBACK
            RETURN
        END

    DECLARE @CheckPassWord int

    SELECT @CheckPassWord = Id
    FROM Worker
    WHERE [PassWord] = @PassWord

    IF(ISNUMERIC (@CheckPassWord) = 1)
        BEGIN

```

```

trun'                                PRINT 'you entered wrong data about worker password!!! Password must be unique!!!' + CHAR(13) + 'rollback

                                      ROLLBACK
                                      RETURN

END

DECLARE @WorkerTypeId int = 0

SELECT @WorkerTypeId = w.Id
FROM WorkerType as w
WHERE w.[Name] = @WorkerType

IF(@WorkerTypeId = 0)
    BEGIN
        INSERT INTO WorkerType([Name])
            VALUES (@WorkerType)

        SELECT @WorkerTypeId = w.Id
        FROM WorkerType as w
    END

IF EXISTS(SELECT * FROM Worker WHERE PersonId = @PersonId)
    BEGIN
        IF EXISTS(SELECT *
            FROM Worker AS w
            WHERE PersonId = @PersonId AND
                w.TypeId IN (
                    SELECT Id
                    FROM WorkerType
                    WHERE [Name] = @WorkerType
                )
        )
            BEGIN
                PRINT 'data about worker is exist in table' + CHAR(13) + 'Commit trun'
                COMMIT
            END
        ELSE
            BEGIN
                UPDATE Worker
                SET TypeId = @WorkerTypeId,
                    [PassWord] = @PassWord
                WHERE PersonId = @PersonId

                UPDATE Worker
                SET TypeId = @WorkerTypeId
                WHERE PersonId = @PersonId

                BEGIN
                    SET @uperr = @@error
                    IF @uperr > @maxerr
                        SET @maxerr = @uperr

                    -- If an error occurred, roll back
                    IF @maxerr <> 0
                        BEGIN
                            ROLLBACK
                            PRINT 'did bot updated ' + CHAR(13) + ' Transaction

rolled back'                                RETURN
                                            END
                                ELSE
                                    BEGIN
                                        COMMIT
                                        PRINT 'updated successfully ' + CHAR(13) + '

Transaction committed'                                END

```



```

                                END
                        END
                END
ELSE
        BEGIN
                PRINT 'data about worker is NOT exist in table'
                INSERT INTO Worker (TypeId, PersonId, [PassWord])
                VALUES (@WorkerTypeId, @PersonId, @PassWord);

                BEGIN
                        SET @uperr = @@error
                        IF @uperr > @maxerr
                        SET @maxerr = @uperr

                        -- If an error occurred, roll back
                        IF @maxerr <> 0
                                BEGIN
                                        ROLLBACK
                                        PRINT 'did bot inserted ' + CHAR(13) + ' Transaction rolled back'
                                        RETURN
                                END
                        ELSE
                                BEGIN
                                        COMMIT
                                        PRINT 'inserted successfully ' + CHAR(13) + ' Transaction committed'
                                END
                        END
                END
        END
END
END

```

## Процедура обрахунку вартості замовлення

```

DROP PROCEDURE IF EXISTS dbo.CalculateOrderPrice
GO
CREATE PROCEDURE dbo.CalculateOrderPrice
    @OrderId int
AS
    DECLARE @TotalPrice decimal = 0--, @ProductId int = 1

    SELECT @TotalPrice = @TotalPrice + p.Price
    FROM Product AS p
    WHERE p.OrderId = @OrderId

    PRINT 'Total price of Order is'
    PRINT @TotalPrice
    --SELECT * FROM vMaterial

    IF(@TotalPrice != 0)
        BEGIN
            PRINT 'TotalPrice is correct'

            UPDATE [Order]
            SET TotalPrice = @TotalPrice
            WHERE [Order].Id = @OrderId

            END
        ELSE
            BEGIN
                PRINT 'TotalPrice calculated incorrectly'

                RETURN
            END
        GO;

```

## Процедура обрахунку вартості продукту

```

DROP PROCEDURE IF EXISTS dbo.CalculateProductPrice
GO
CREATE PROCEDURE dbo.CalculateProductPrice

```

```

@ProductId int
AS
DECLARE @TotalPrice decimal = 0--, @ProductId int = 1

SELECT @TotalPrice = @TotalPrice + s.Price
FROM Storage AS s
WHERE s.Id IN (
    SELECT MaterialId
    FROM MaterialList
    WHERE MaterialList.ProductId =
@ProductId
)

PRINT 'Total price is'
PRINT @TotalPrice
--SELECT * FROM vMaterial

IF(@TotalPrice != 0)
BEGIN
    PRINT 'TotalPrice is correct'

    Update price in table Product'

    UPDATE Product
    SET Price = @TotalPrice
    WHERE Product.Id = @ProductId
END
ELSE
BEGIN
    PRINT 'TotalPrice calculated
incorrectly'

    RETURN
END
GO;

```

## Створення нової поставки товарів

```

DROP Type ItemList

CREATE TYPE ItemList AS TABLE
(
    Id int Identity(1, 1) PRIMARY KEY,
    ItemId int
)
GO

DROP PROCEDURE IF EXISTS dbo.CreateSupplyOrder
GO

CREATE PROCEDURE dbo.CreateSupplyOrder
@Items ItemList READONLY,
@ManagerId int
AS
DECLARE @ManagerTypeId int, @res bit
SELECT @ManagerTypeId = Id FROM WorkerType
WHERE WorkerType.Name = 'manager'
--SELECT @EngineerTypeId

SET @res = dbo.WorkerIs(@ManagerTypeId,
@ManagerId)

IF(dbo.WorkerIs(@ManagerTypeId, @ManagerId) = 0)
BEGIN
    PRINT 'Wrong ManagerId'
    RETURN
END

DECLARE @CurrentOrderNumber int, @CurrentOrderId
int

SELECT @CurrentOrderNumber =
MAX(SupplyOrder.OrderNumber) + 1
FROM SupplyOrder

IF NOT(ISNUMERIC(@CurrentOrderNumber) = 1)
BEGIN
    SET @CurrentOrderNumber = 0
END

INSERT INTO SupplyOrder
(OrderNumber, ManagerId)
VALUES
(@CurrentOrderNumber, @ManagerId)

```

```

SELECT @CurrentOrderId = Id
FROM SupplyOrder AS s
WHERE s.OrderNumber = @CurrentOrderNumber

```

```

DECLARE @ItemId int

```

```

DECLARE OrderCursor CURSOR FOR
SELECT
ItemId
FROM @Items

```

```

OPEN OrderCursor

```

```

FETCH NEXT FROM OrderCursor
INTO

```

```

@ItemId

```

```

WHILE @@FETCH_STATUS = 0
BEGIN

```

```

UPDATE SupplyItem
SET SupplyItem.SypplyOrderId = @CurrentOrderId
WHERE SupplyItem.Id = @ItemId

```

```

END
FETCH NEXT FROM OrderCursor
INTO
@ItemId

```

```

CLOSE OrderCursor
DEALLOCATE OrderCursor

```

## Процедура оплати замовлення

```

DROP PROCEDURE IF EXISTS dbo.Pay

```

```

GO

```

```

CREATE PROCEDURE dbo.Pay

```

```

@PersonId int,

```

```

@OrderId int,

```

```

@Paid decimal

```

```

AS

```

```

BEGIN TRAN

```

```

If NOT EXISTS(SELECT * FROM [Order] WHERE
[Order].Id = @OrderId)

```

```

BEGIN

```

```

    PRINT 'Order not found'

```

```

    ROLLBACK

```

```

    RETURN

```

```

END

```

```

BEGIN TRY

```

```

    PRINT 'insert new payment'

```

```

    INSERT INTO Payment

```

```

    (OrderId, PayerId, Paid)

```

```

    VALUES (@OrderId, @PersonId, @Paid)

```

```

    DECLARE @CurrentPaid decimal

```

```

SELECT @CurrentPaid = Paid

```

```

FROM [Order]

```

```

WHERE [Order].Id = @OrderId

```

```

IF NOT(ISNUMERIC(@CurrentPaid) = 1)

```

```

BEGIN

```

```

    UPDATE [Order]

```

```

    SET Paid = 0

```

```

END

```

```

PRINT 'Udate payment in Order table'

```

```

UPDATE [Order]

```

```

SET Paid = Paid + @Paid

```

```

WHERE [Order].Id = @OrderId

```

```

COMMIT TRAN

```

```

END TRY

```

```

BEGIN CATCH

```

```

PRINT 'Cannot add new payment'

```

```

ROLLBACK

```

END CATCH

GO;

--end of procedure

## Процедура створення списку необхідних матеріалів для виготовлення продукту

```

DROP PROCEDURE IF EXISTS dbo.SetMaterialList
GO

CREATE PROCEDURE dbo.SetMaterialList
    @ProductId int,
    @MaterialId int,
    @Quantity int
AS
BEGIN TRAN

    DECLARE @CurrentQuantity int
    SELECT @CurrentQuantity = Quantity
    FROM Storage as S
    WHERE S.MaterialId = @MaterialId

    IF(@Quantity > @CurrentQuantity)
    BEGIN
        PRINT 'Not enough material in the storage'
        ROLLBACK TRAN
        RETURN 0
    END

    -----
    DECLARE @MaterialListId int

    SELECT @MaterialListId = Id
    FROM MaterialList
    WHERE ProductId = @ProductId AND
        MaterialId = @MaterialId

    PRINT 'Material Id'
    PRINT @MaterialListId

    IF (ISNUMERIC(@MaterialListId) = 1)
        BEGIN
            PRINT 'UP DATE Material
            Quantity in Material List'

            UPDATE MaterialList
            SET Quantity = @Quantity
            WHERE MaterialId =
                @MaterialId AND
                ProductId =
                @ProductId
            END
        ELSE
            BEGIN
                PRINT 'Insert Materials in
                Material List'
                --Id, ColorId, OrderId,
                EngineerId, Height, Width, DateComplite, Price
                INSERT INTO MaterialList
                (MaterialId, ProductId, Quantity)
                VALUES(
                    @MaterialId,
                    @ProductId,
                    @Quantity
                )
                PRINT 'Update Storage'
                UPDATE Storage
                SET Quantity = Quantity -
                    @Quantity
                WHERE Storage.MaterialId =
                    @MaterialId

                PRINT 'Complited'
            END

            SELECT @CurrentQuantity = Quantity FROM Storage
            WHERE MaterialId = @MaterialId
            IF (@CurrentQuantity > 0)
                COMMIT TRAN
        
```

ELSE

ROLLBACK

GO;

### Функція для отримання списку вільних інженерів

CREATE FUNCTION dbo.GetFreeEngineer

FROM WorkerType AS w

()

WHERE w.Name = 'engineer'

RETURNS @Engineer table

INSERT INTO @Engineer

(

SELECT w.Id

Id int identity(1,1),

FROM Worker AS w

EngineerId int

WHERE w.TypeId = @TypeId AND w.Id NOT

)

IN(

AS

SELECT EngineerId

BEGIN

FROM Product

)

DECLARE @TypeId int

RETURN

SELECT @TypeId = w.Id

END

### Функція для отримання списку матеріалів

CREATE FUNCTION dbo.GetMaterial

INSERT INTO @Materials

(

SELECT Storage.Id

@MaterialNeed INT

FROM Storage

)

WHERE MaterialId IN(

SELECT Material.Id

RETURNS @Materials table

FROM Material

(

WHERE MaterialTypeId =

Id int identity(1,1),

@MaterialNeed

MaterialId int

)

)

AS

RETURN

BEGIN

END

Функція для перевірки чи робітник є потрібного напрямку (інженер, менеджер...).

CREATE FUNCTION dbo.WorkerIs

(

@WorkerType int,

@Id int

)

```
RETURNS bit
AS
BEGIN
    IF EXISTS (SELECT * FROM Worker WHERE Worker.TypeId = @WorkerType AND Worker.Id = @Id)
        RETURN 'true'
    ELSE
        RETURN 'false'
    RETURN 'false'
END
GO
```

## Додаток В. Результат перевірки схеми БД в Erwin Data Modeler

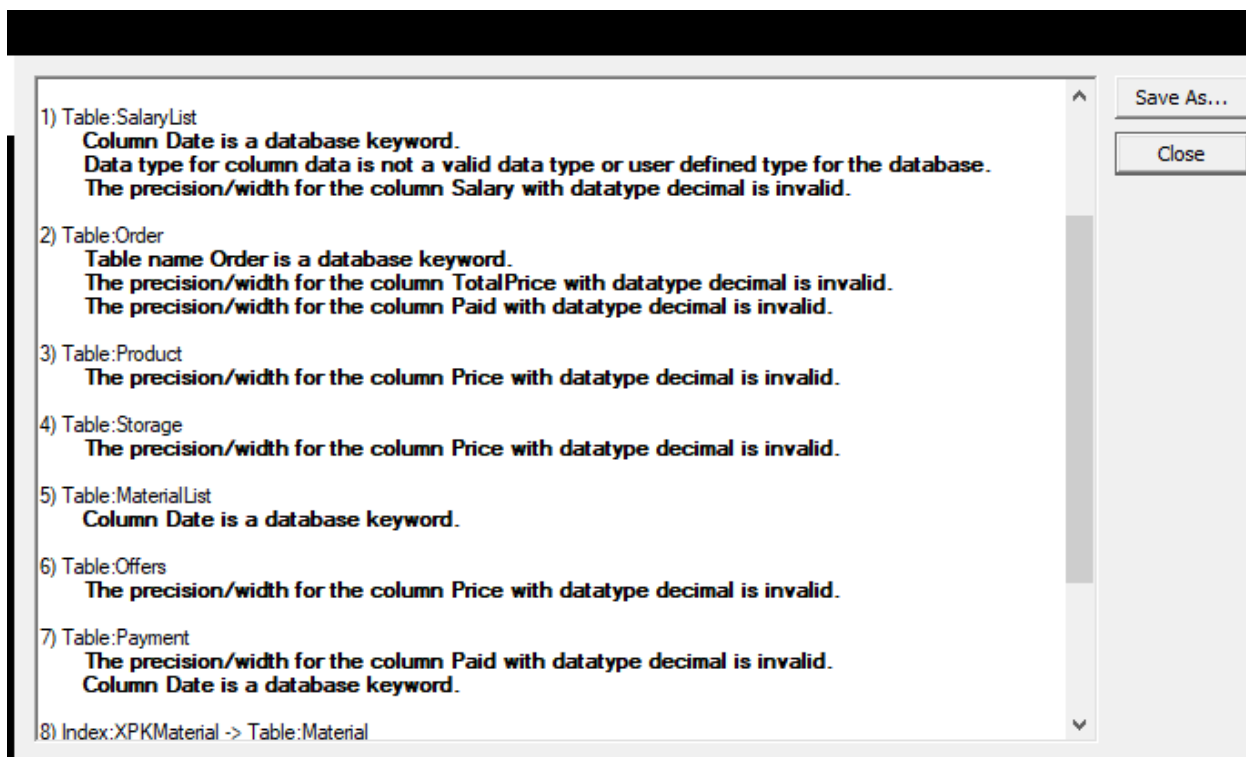


Рис. В.1. Результат перевірки моделі в Erwin Data Modeler