Ремінник Олег Олегович

Розробка та дослідження інформаційної системи для обліку діяльності станції технічного обслуговування автомобілів

Керівник: канд. техн. наук, доц. Чихіра І.В.

АНОТАЦІЯ

Дипломна робота складається з пояснювальної записки та графічної частини (ілюстративний матеріал – слайди).

Об’єм графічної частини дипломної роботи становить 11 слайдів.

Об’єм пояснювальної записки складає 127 друкованих сторінок формату А4 (210×297), об’єм додатків – 16 друкованих сторінок формату А4.

Дипломна робота складається з восьми розділів, в яких нараховується 24 рисунків та 8 таблиць з даними.

В роботі використано 21 літературних джерел

В даному дипломному проекті було розроблено автоматизовану систему керування клієнт-серверною базою даних на базі сервера InterBase.

В результаті була створена база даних по обліку об’єктів, а також програма по управлінню базою даних, яка дозволяє виконувати різноманітні дії: введення, корегування даних, пошук, побудову звітів.

Основна задача розробленої системи – автоматизація збору та збереження інформації. Вона заміняє книги обліку об’єктів цивільної оборони. Впровадження даної системи дозволяє значно скоротити паперовий обіг документів, пришвидшити обробку інформації.

Ключові слова: БАЗА ДАНИХ, INTERBASE, АВТОМАТИЗОВАНА СИСТЕМА, DELPHI

.

ЗМІСТ

ВСТУП

Метою даного дипломного проекту є розробка та дослідження інформаційної системи для обліку діяльності станції технічного обслуговування автомобілів.

Виходячи із сучасних вимог, пропонованих до якості роботи фінансової ланки товариства, не можна не відзначити, що ефективна робота його цілком залежить від рівня оснащення офісу компанії електронним устаткуванням, таким, як комп'ютери, програмним забезпеченням, засобами зв'язку, копіювальними пристроями.

У цьому ряді особливе місце займають бази даних і інше програмне забезпечення, зв'язане з їх використанням як інструмент для діловодства і раціоналізації фінансової праці. Їхнє використання дозволяє скоротити час, необхідний на підготовку конкретних маркетингових і виробничих проектів, зменшити непродуктивні витрати при їхній реалізації, виключити можливість появи помилок у підготовці бухгалтерського, технологічного й іншого видів документації, що дає для малого підприємства прямий економічний ефект.

Зрозуміло, для розкриття всіх потенційних можливостей, що несе в собі використання баз даних, необхідно застосовувати в роботі комплекс програмних і апаратних засобів, максимально відповідних поставленим задачам. Тому в даний час велика потреба малих підприємств у комп'ютерних програмах, що підтримують і погоджують роботу управлінської і фінансової ланок компанії, а також в інформації про способи оптимального використання наявного в компанії комп'ютерного устаткування.

1 АНАЛІТИЧНА ЧАСТИНА

1.1. Аналіз відомих технічних рішень з питань автоматизації обліку діяльності станції технічного обслуговування автомобілів

При огляді вітчизняної та зарубіжної літератури простежується тенденція до повної автоматизації та комп'ютеризації на підприємствах бухгалтерських операцій, складського обліку тощо. Для автоматизації обліку підприємством переважно використовують бази даних та різноманітні системи управління базами даних (СУБД).

Однією з найпоширеніших СУБД є система Microsoft Access 2000, XP. Ця система дозволяє створювати бази даних, виконувати операції з записами у таблицях, систематизовувати наявну інформацію. Проте використання на підприємстві СУБД Microsoft Access 2000 не завжди оправдане, оскільки через свою багатофункціональність вона є досить складною у користуванні. Крім того, ціна ліцензії на використання цієї СУБД досить висока, що досить відчутно для підприємств малого бізнесу.

На деяких підприємствах для автоматизації обліку на підприємстві використовують програму 1С Підприємство, проте облік підприємства – не головне призначення даної програми, вона також передбачає автоматизацію операцій, пов'язаних із управлінням приватним підприємством, що не завжди доцільно, тому що на багатьох підприємствах для цього уже використовується інше спеціалізоване програмне забезпечення. Отже, використання програми 1С Підприємство не вирішує двох основних проблем: складність в управлінні та висока ціна.

До цього часу часто використовуються не досконалі бази даних написаних раніше, це наприклад АРМ-ОК-94, однак вона також має ряд недоліків, які вважаються критичними і вимагають виправлення, а саме:

программа не розподіляє права доступу до даних, тобто будь-хто, хто має доступ до комп'ютера з встановленою програмою, може безперешкодно

редагувати дані БД; програма використовує велику кількість файлів для збереження даних, які зберігалися на сервері, при цьому при одночасній роботі декількох користувачів швидкість помітно падає; інтерфейс програми абсолютно не розрахований на довготривалу роботу оператора ЕОМ – надто контрастні кольора діалогових вікон і вікон вводу інформації, а також видача звукового сигналу після вдалого вводу інформації сильно втомлювала користувачів.

В залежності від архітектури СУБД розрізняють файл-серверні та клієнт-серверні СУБД. Усі частини файл-серверної СУБД розміщуються на комп'ютері користувача бази даних. Якщо до однієї бази даних звертаються декілька користувачів одночасно, кожен комп'ютер користувача повинен мати свою копію локальної СУБД.

На відміну від цього, значна частина програмно-апаратних засобів клієнт-серверної СУБД централізована і розташована на одному досить потужному комп'ютері (сервері), в той час, коли комп'ютери користувачів несуть порівняно невелику частину СУБД, яка називається клієнтом.

Локальні (файл-серверні) СУБД можуть працювати в мережі, але можуть і не використовувати її, в той час як клієнт-серверні СУБД обов'язково працюють у комп'ютерній мережі. Безумовною перевагою клієнт-серверних систем є можливість централізованого управління доступом до бази даних. В таких системах база даних у значній мірі захищена як від випадкових, так і навмисних спотворень, в них простіше реалізувати цілісність даних.

Багато організацій використовують файл-серверні бази даних для підтримки своїх робочих процесів. Часто це системи на одного, двох користувачів, що виконані з використанням dbf – орієнтованих засобів обробки: Clipper, Dbase, FoxPro, Paradox, Access. Зазвичай використовується ряд таких баз, незалежно одна від одної.

Якщо інформація, що зберігається в таких базах даних, представляє інтерес не тільки для безпосередніх користувачів, то для її подальшого

поширення використовуються паперові звіти та довідки, що створені базою даних.

З появою локальних мереж, що підключені до таких мереж як Інтернет, створенням внутрікорпоративних мереж, з'являється можливість з будь-якого робочого місця організації отримати доступ до інформаційного ресурсу мережі.

Але, при спробі використовувати існуючі бази даних виникають проблеми, що пов'язані з вимогою до однорідності робочих місць, завантаженням мережі (доступ здійснюється напряму до файлів бази даних), файлового сервера та неможливістю віддаленої роботи (наприклад, співробітників у відрядженні).

Архітектура файл-сервер неефективна якнайменше у трьох випадках:

1. При виконанні запиту до бази даних, що розміщується на файловому сервері, насправді здійснюється запит до локальної копії даних на комп'ютері користувача. Тому перед виконанням запиту, дані у локальній копії у повному обсязі оновлюються з реальної бази даних. Так, якщо таблиця бази даних складається з 10 000 записів, а для виконання запиту потрібно лише 10 записів, то все одно клієнту передадуться усі 10000 записів. Таким чином, не потрібно мати занадто багато користувачів та запитів від них, щоб серйозно завантажити мережу, що, звичайно, не може не вплинути на її швидкодію.

2. Цілісність бази даних забезпечується клієнтськими програмами. Це потенційне джерело помилок, що порушує фізичну та логічну цілісність даних, оскільки різні клієнти можуть здійснювати контроль цілісності по-різному або не проводити такого контролю взагалі. Найбільш ефективніше управляти базою даних з єдиного місця та по єдиним законам. Тому безпека при роботі в архітектурі файл-сервер не є високою і завжди присутнім є елемент невизначеності.

3. В такій архітектурі важко забезпечити секретність даних та їх захист: таблиці зберігаються на сервері у вигляді звичайних файлів, тому будь-хто, хто має доступ до каталогу (каталогам) мережевого сервера, де зберігається база даних, може змінювати таблиці, копіювати їх та інше.

Перерахованих недоліків позбавлені СУБД, що побудовані за архітектурою клієнт-сервер, характерною особливістю якої є перенесення обчислювального навантаження на сервер бази даних (SQL-сервер) та максимальне розвантаження клієнта від обчислювальної роботи, а також забезпечення безпеки даних як від зловмисних, так і просто помилкових змін.

Як і в архітектурі файл-сервер, база даних розміщується на мережевому сервері, але програма клієнта не має можливості прямого доступу до бази даних. Доступ до бази даних регулюється спеціальною програмою – сервером бази даних (SQL-сервером).

В архітектурі клієнт-сервер клієнт формує запит до сервера на мові запитів (SQL – Structured Query Language – структурована мова запитів), що є промисловим стандартом для реляційних баз даних. SQL належить до мови високого рівня, за допомогою якої користувач вказує, які дані необхідно отримати, не уточнюючи процедуру їх отримання.

SQL-сервер забезпечує інтерпретацію запита, його виконання, формування результату та видачу цього результату клієнтові. При цьому ресурси клієнтського комп’ютера не беруть участь у фізичному виконанні запиту: комп’ютер клієнта лише відсилає запит до серверної бази даних та отримує результат, після чого інтерпретує його необхідним чином і представляє користувачу.

Так як клієнтському додатку надсилається результат виконання запиту, по мережі передаються лише ті дані, які в дійсності потрібні клієнтові. В кінцевому результаті знижується навантаження на мережу. Крім того, SQL-сервер оптимізує отриманий запит таким чином, щоб він був виконаний за мінімально можливий час. Усе це підвищує швидкодію системи та знижує час очікування результату запиту.

Переваги архітектури клієнт-сервер:

1. Більша частина обчислювальних процесів відбувається на сервері, що знижує вимоги до обчислювальних потужностей комп’ютера клієнта;

збільшення обчислювальних потужностей одного сервера еквівалентно одночасному збільшенню потужностей усіх клієнтських місць;

2. Знижується завантаження мережі за рахунок відсилання сервером клієнту тільки тих даних, які він запитував; наприклад, якщо необхідно зробити в таблиці обсягом 100 000 записів вибірку, результатом якої будуть лише 2 записи, сервер виконає запит та перешле клієнтові тільки потрібні йому 2 записи;

3. База даних на сервері представляє собою, як правило, єдиний файл, в якому зберігаються таблиці, обмеження цілісності та інші компоненти бази даних; зламати, викрасти або зіпсувати таку базу даних значно важче; збільшується захищеність бази даних від введення неправильних значень, оскільки сервер бази даних проводить автоматичну перевірку відповідності значень, що вводяться, обмеженням, які накладені у базі даних, та автоматично виконує необхідні дії; крім цього, сервер відслідковує рівні доступу для кожного користувача та блокує спроби виконання заборонених для користувача дій. Усе це дозволяє говорити про значно вищий рівень забезпечення безпеки бази даних;

4. Сервер реалізує управління транзакціями та запобігає спробам одночасної зміни одних і тих самих даних;

5. Безпека системи зростає за рахунок переносу більшої частини правил на сервер.

Дані в серверній базі даних фізично зберігаються на диску у вигляді одного великого файлу, що у поєднанні з паролями та привілеями, які назначаються кожному користувачу, суттєво підвищує захист даних від намірів пошкодження та крадіжок.

Для реалізації архітектури застосовують так звані промислові сервери баз даних, такі як InterBase, Oracle, Informix, Sybase SQL Server, DB2, MS SQL Server.

1.2 Обгрунтування актуальності розробки інформаційної системи

Таким чином, за наявної ситуації на ринку програмного забезпечення більшість приватних підприємств малого бізнесу не можуть використовувати СУБД відомих виробників і тому проводять складський облік без використання інформаційних технологій або за допомогою морально застарілого программного забезпечення. За даних умов досить актуальною є розробка інформаційної системи, яка б не потребувала великих фінансових затрат на обслуговування.

Система управління підприємством побудована відповідно до лінійно-ієрархічним принципом. На кожному рівні чітко визначені зони відповідальності та зони підпорядкування. Інформаційний процес, спрямований на отримання науково-технічної, планової, контрольної, облікової і аналітичної інформації, в інформаційних системах уніфікований і базується на електронно-обчислювальної техніки. Підвищення ефективності використання інформаційних систем досягається шляхом наскрізної побудови і сумісності інформаційних систем, що дозволяє усунути дублювання і забезпечити багаторазове використання інформації, установити певні інтеграційні зв'язки, обмежити кількість показників, зменшити обсяг інформаційних потоків, підвищити ступінь використання інформації.

Важливу роль відіграють способи реєстрації інформації, обробки, накопичення і передачі, систематизоване зберігання інформації та видача її в необхідній формі, виробництво нової числової та іншої інформації. В сучасних умовах в великих організаціях створені і ефективно діють інформаційні системи, які обслуговують процес підготовки і прийняття бухгалтерських та управлінських рішень, і вирішальні наступні завдання: обробка даних, обробка інформації [9].

Для визначення ефективності внутрішньо фірмової системи управління на багатьох підприємствах в обліку та звітності використовують показники:

- відношення одержуваного прибутку до витрат на технічні засоби та забезпечення функціонування внутрішньо фірмової системи інформації.

Основними принципами і цілями внутрішньо фірмових систем інформації є:

- визначення вимог до змісту інформації і до її характеру, в залежності від цілеспрямованості;
- вироблення системи зберігання, використання та надання інформації в централізованому і децентралізованому управлінні;
- визначення потреб у технічних засобах (в тому числі, в комп'ютерній техніці) на підприємстві в цілому;
- розробка програмного забезпечення, створення і використання банків даних;
- автоматизована обробка вводиться і поточної інформації і видача інформації з бухгалтерського обліку та відділів технічного оснащення;
- автоматизація адміністративно-управлінської праці на основі використання комп'ютерної техніки.

Важливими завданнями внутрішньо фірмової системи управління є:

- координація діяльності по збору і обробці даних фінансових звітів на вищому рівні управління і у виробничих відділеннях з метою підвищення якості та своєчасності надходження фінансової інформації по підприємству в цілому;
- визначення основних напрямків системи збору, обробки та зберігання первинних даних;
- визначення основних напрямків розвитку технології обробки інформації.

Оснащення електронною технікою дозволяє економити управлінські та накладні витрати, забезпечує ефективне внутрішньо фірмове планування.

Розвиток систем телекомунікацій і, зокрема, технологій локальних обчислювальних мереж, дозволило об'єднати всі технічні засоби обробки бухгалтерської інформації в єдину внутрішньо фірмову інформаційну мережу [2].

1.3 Відомості про об'єкт автоматизації

Призначення інформаційної системи - автоматизація роботи таких підрозділів ТОВ "Автосервіс":

- диспетчерської служби (формування замовлень, облік робіт і облік комплектації замовлень);
- служба матеріально-технічного постачання (прийом заявок на закупівлю запчастин і витратних матеріалів, облік наявності товару на складі);
- ремонтна служба (прийом і виконання замовлень).

Процес прийому і виконання замовлення можна коротко описати по наступних кроків :

- визначення, чи є клієнт постійним замовником компанії (постійні замовники отримують знижки на виконання робіт);
- попереднє визначення сутності замовлення (замовлень). Залежно від потреби клієнта диспетчер звертається до майстрів ремонтної служби, служби сервісу;
- визначення виконавця замовлення. Диспетчер при цьому керується журналом замовлень. Він звертається до того майстру, який, на думку диспетчера, найменш зайнятий. Однак в разі фактичної зайнятості цього майстра, диспетчер переадресує заявку іншому;
- аналіз майстром несправності, попереднє діагностування і оцінка тривалості робіт і їх вартості, а також наявності необхідної комплектації на складі. Диспетчер бере участь в перевірці наявності на складі комплектуючих;
- оформлення замовлення за заданою формою, оформлення приймально-здавального акту передачі транспортного засобу виконавцю, оформлення довіреності на проведення технічних випробувань транспортного засобу; фіксація отримання передоплати; заповнення журналу замовлень;
- видача замовлення, оформлення диспетчером замовлення-наряду та приймально-здавального акту передачі транспортного засобу замовнику.

2 ТЕХНОЛОГІЧНА ЧАСТИНА

2.1 Найменування та область застосування

В даній дипломній роботі розроблено інформаційну систему для автоматизації складського обліку на базі серверу InterBase.

Інформаційна система для складського обліку – це база даних, в якій розміщується інформація про товар, та програмне забезпечення для роботи з нею.

Програмне забезпечення для роботи з базою даних – це прикладні програми, що розміщуються на робочих станціях мережі підприємства і виконують усі необхідні операції з таблицями бази даних. В даному випадку для створення прикладних програм будемо використовувати середовище програмування Borland Delphi 6 Enterprise, яке передбачає роботу з сервером баз даних InterBase і надає усі необхідні засоби для швидкого створення прикладних програм для роботи під управлінням операційних систем Windows.

База даних являє собою сукупність таблиць, зв’язаних між собою певними відношеннями. База даних повинна задовольняти такі умови:

- файли бази повинні розміщуватися лише на сервері підприємства;
- всі операції з базою даних повинні проводитися на робочих станціях підприємства;
- структура бази даних повинна забезпечувати швидкий доступ до даних, відсутність дублювання даних та цілісність даних;
- база даних повинна бути захищена від несанкціонованого доступу.

Для цих умов найбільше підходить мережева реляційна база даних з архітектурою „клієнт-сервер”. В якості серверу баз даних будемо використовувати сервер InterBase.

Припустимо, що товариство для якого розробляється дана інформаційна система, складається з чотирьох підрозділів: трьох магазинів і складу. Підприємство від постачальників отримує продукцію.

Для здійснення цієї операції розроблювана інформаційна система містить інформацію про постачальників та про товар. Після поступлення до підприємства товар може бути проданий цим же підрозділом, або переміщено в інший підрозділ, який має потребу у товарах даного виду. Продаж здійснюється гуртовим, дрібно гуртовим або роздрібним покупцям, причому ціни для кожної групи покупців різні. Після продажу товар може бути повернено на підприємство. Продаж товару може проводитися за накладними, інформація про які також зберігається.

Отже, база даних розроблюваної інформаційної системи містить таку інформацію: перелік наявної продукції по підрозділах на підприємстві (групи товарів, назви, заводи-виробники, кількість, ціни тощо), список постачальників, покупців товару, інформацію про накладні (номер, підрозділ, з якого було здійснено продаж, покупець, тип та кількість одиниць товару, ціна тощо), інформацію про рух товару (тип операції, вид та кількість товару, підрозділи, які здійснили операцію, дата тощо).

2.2. Структура бази даних інформаційної системи

База даних забезпечує збереження інформації, а також зручний і швидкий доступ до даних. Вона являє собою сукупність даних різноманітного характеру, організованих по певних правилах. Інформація в базі даних повинна бути:

- несуперечливою;
- ненадлишковою;
- цілісною.

Реляційна база даних складається із пов’язаних між собою таблиць. Кожна таблиця містить інформацію про об’єкти одного типу, а сукупність всіх таблиць утворює єдину базу даних. Таблиці знаходяться в каталозі на жорсткому диску у окремих файлах і схожі на окремі документи, їх можна переміщати і копіювати звичайним методом, наприклад, за допомогою Провідника Windows. Однак, на відміну від документів, таблиці бази даних

підтримують багатокористувацький режим доступу, тобто можуть одночасно використовуватися декількома прикладними програмами.

Для однієї таблиці створюється декілька файлів, що містять дані, індекси, ключі тощо. Головним з них є файл з даними, ім’я цього файлу співпадає з іменем таблиці, яке задається при її створенні. Імена всіх інших файлів таблиці призначаються автоматично – всі файли мають одинакове ім’я, і різні розширення, що вказують на вміст відповідного файлу.

Кожна таблиця бази даних складається з стрічок і стовбців для збереження даних про однотипні об’єкти інформаційної системи. Стрічка таблиці називається записом, стовбець – полем. Кожне поле повинне мати унікальне ім’я в межах таблиці. Поле містить дані одного з допустимих типів. При введені значення в поле таблиці автоматично проводиться перевірка відповідності типу значення і типу поля. У випадку, коли ці типи не співпадають, а перетворення типу значення неможливе, генерується виключна ситуація.

Основу таблиці складає опис її полів, кожна таблиця повинна мати хоча б одне поле. Поняття структури таблиці є більш широким і включає в себе:

- описання полів;
- ключ;
- індекси;
- обмеження на значення полів;
- обмеження посилальної цілісності між таблицями;
- паролі.

З таблицею в цілому можна виконати такі операції:

- створення (визначення структури);
- зміна структури (реструктуризація);
- перейменування;
- знищення.

Ключ являє собою комбінацію полів, дані в яких однозначно визначають кожен запис в таблиці. Простий ключ складається з одного поля. Ключ забезпечує:

- однозначну ідентифікацію записів таблиці;
- прискорення виконання запитів до бази даних;
- встановлення зв’язків між окремими таблицями;
- використання обмежень посилальної цілісності.

Інформація про ключ зберігається в окремому файлі або разом з даними таблиці. Для кожного значення ключа існує унікальне посилання, що вказує на розміщення відповідного запису в таблиці (в головному її файлі). Тому при пошуку записів виконується не послідовний перегляд всієї таблиці, а прямий доступ до запису на основі впорядкованих значень ключа.

Проектування бази даних починається з визначення всіх об’єктів, інформація про які повинна зберігатися, і виділенні атрибутів (характеристик) цих об’єктів. Атрибути всіх об’єктів зводяться в одну таблицю, яка є вихідною. Потім ця таблиця послідовно приводиться до нормальних форм у відповідності з їх вимогами. На практиці зазвичай використовують три нормальних форми.

Перша нормальна форма передбачає виконання таких вимог:

- поля містять неподільну інформацію;
- в таблиці відсутні групи полів, що повторюються.

До другої нормальної форми ставляться наступні вимоги:

- таблиця повинна заловільняти вимоги першої нормальної форми;
- будь-яке неключове поле повинне однозначно ідентифікуватися ключовими полями.

Вимоги третьої нормальної форми:

- таблиця повинна задовільняти вимоги другої нормальної форми;
- жодне з неключових полів не повинно однозначно ідентифікуватися значеннями іншого неключового поля.

Виходячи з вищеописаних вимог, база даних розроблюваної інформаційної системи включає в себе такі таблиці:

1. Таблиця постачальників POSTACH. Таблиця містить такі поля:
   - CODE типу integer – ключове поле, яке містить унікальний код постачальника;
   - NAME стрічкового типу – ім’я (назва) постачальника;
   - ADDRES стрічкового типу – містить адресу постачальника;
   - TEL стрічкового типу – телефон постачальника;
   - NOTE стрічкового типу – поле, яке містить додаткову інформацію про постачальника.

2. Таблиця підрозділів підприємства MAGAZYN, яка містить дані про підрозділи підприємства, містить такі поля:
   - CODE цілочисельного типу integer – ключове поле, яке містить унікальні коди кожного підрозділу;
   - NAME стрічкового типу – містить назви підрозділів;
   - NOTE стрічкового типу – поле, яке містить додаткову інформацію про підрозділи.

3. Таблиця покупців POCUPETS з інформацією про клієнтів підприємства, містить такі поля:
   - CODE типу integer – ключове поле, яке містить унікальний код покупця;
   - NAME стрічкового типу – ім’я (назва) покупця;
   - ADDRES стрічкового типу – містить адресу покупця;
   - TEL стрічкового типу – телефон покупця;
   - NOTE стрічкового типу – поле, яке містить додаткову інформацію про клієнта.

4. Таблиця груп товарів TOVGRUP містить такі поля:
   - CODE цілочисельного типу integer – ключове поле, яке містить унікальні коди кожної групи товару;
   - NAME стрічкового типу – містить назву групи;

- NOTE стрічкового типу – поле додаткової інформації про групи товарів.

5. Таблиця виробників товарів ZAVOD:

- CODE цілочисельного типу integer – ключове поле з унікальними кодами виробників товарів;
- NAME стрічкового типу – назва виробника;
- NOTE стрічкового типу – додаткова інформація про виробника.

6. Таблиця товарів TOVAR, що містить дані про наявний у підрозділах підприємства товар:

- CODE цілочисельного типу integer – ключове поле, яке містить унікальні коди товарів;
- IDGROUP типу integer – поле, яке містить унікальні ідентифікатори груп товарів (береться з таблиці TOVGRUP);
- NAME стрічкового типу – назва товару;
- MANUF цілочисельного типу – поле, яке містить унікальні коди виробників товарів (з таблиці ZAVOD);
- PR типу integer – купівельна (прихідна) ціна товару;
- PRA типу integer – роздрібна ціна товару;
- PRB типу integer – дрібногуртова ціна товару;
- PRC типу integer – гуртова ціна товару;
- M, MA, MB, MC стрічкового типу – поля грошових одиниць відповідно для купівельної, роздрібної, дрібно гуртової та гуртової цін;
- NOTE стрічкового типу – поле, яке містить додаткову інформацію про товар.

7. Таблиця BASE, яка містить дані про наявність товару по підрозділах:

- CODE цілочисельного типу – ключове поле;
- IDMAG цілочисельного типу – ідентифікатор підрозділу підприємства, відповідає полю CODE таблиці MAGAZYN;

- IDTOV цілочисельного типу – ідентифікатор товару, відповідає полю CODE таблиці TOVAR;
- NUMOF цілочисельного типу – кількість одиниць наявного товару в даному підрозділі;
- DEN типу дата – дата поступлення товару в даний підрозділ.

8. Таблиця MOVE містить інформацію про переміщення товару, вона складається з таких полів:

- CODE – ключове поле;
- TYP цілочисельного типу – містить цифру, яка вказує на тип переміщення товару. Переміщення може бути чотирьох типів: прихід товару, переміщення між підрозділами підприємства, продаж, повернення товару;
- VID цілочисельного типу – інформація про те, звідки поступив товар. Якщо тип руху – прихід, то в полі VID міститься код постачальника, який співпадає з значенням поля CODE цього постачальника в таблиці POSTACH, для внутрішнього переміщення або продажу товару в цьому полі записується код підрозділу підприємства (значення поля CODE таблиці MAGAZYN), а для руху типу повернення – код покупця, який повернув товар (поле CODE таблиці POCUPETS);
- KUDY цілочисельного типу – інформація про те, куди перемістився товар, аналогічно полю VID, залежить від типу переміщення;
- IDTOV цілочисельного типу – код товару, який здійснив переміщення (поле CODE таблиці TOVAR);
- DEN типу дата – дата переміщення товару;
- NOTE стрічкового типу – додаткова інформація про рух товару.

9. Таблиця KURS, в якій фіксується динаміка зміни курсу долара відносно гривні. Містить такі поля:

- DOLLAR числового типу – курс долара відносно гривні;
- DEN типу дата – дата внесення інформації.

10. Таблиця TEMP. Служить для зберігання тимчасової інформації про товар і складається з таких полів:

- ID, CODE, CODE1, NUMOF, IDTOV, IDZAVOD, IDGROUP цілочисельного типу;
- NAME, NOTE стрічкового типу;
- PR, PRA, PRB, PRC десяткового типу (числа з фіксованою цілою і дробовою частиною);
- M, MA, MB, MC – логічного типу.

Усі таблиці бази даних знаходяться в одному файлі Sklad.odb, який розміщений на сервері корпоративної мережі підприємства. Крім таблиць у цьому ж файлі містяться також тригери і генератори, призначення яких буде розглянуто в наступному розділі.

3 КОНСТРУКТОРСЬКА ЧАСТИНА

3.1. Організація роботи інформаційної системи

У своєму складі ТОВ "Автосервіс" має чотири підрозділи – диспетчерська служба (формування замовлень, облік робіт), ремонтна служба, матеріально-технічне постачання та склад.

Підрозділи підприємства з’єднані мережею, у кожному з яких є персональний комп’ютер. Центральний офіс обладнаний серверною кімнатою. На сервері розміщена база даних, а віддалені користувацькі станції у підрозділах підприємства. Користувач вносить зміни у базу даних зі свого робочого місця.

У роботів використано віддалену базу даних. Співробітник підприємства завантажує з робочого місця прикладну програму, що розміщена на сервері для роботи з базою даних. У даному випадку робота буде здійснюватись з копією бази. Даний тип організації роботи місить назву “файл-серверна” архітектура. Використання даної архітектури дозволяє обслуговувати декількох клієнтів одночасно та зменшити загрузку мережі.

Однак архітектура "файл-сервер" має і суттєві недоліки:

- для роботи з даними використовується навігаційний метод доступу, при цьому по мережі циркулюють великі розміри даних. В результаті мережа перевантажується, знижується її швидкодія,
  що призводить до поганої погіршення роботи бази даних;
- необхідна синхронізація роботи окремих користувачів, пов’язана з блокування в таблицях тих записів, які редагуються іншим користувачем;
- прикладні програми не тільки опрацьовують дані, але і керують самою базою даних. У зв’язку з тим, що управління здійснюється з різних комп’ютерів, виникають проблеми з організацією контролю доступу, дотримання конфіденційності і підтримання цілісності БД.

Через ці недоліки ми не будемо використовувати архітектуру "файл-сервер".

В мереженій архітектурі "клієнт-сервер" база даних розміщується на комп'ютері-сервері мережі. Прикладна програма, що здійснює роботу з цією БД, знаходиться на комп'ютері користувача (в даному випадку – у підрозділі підприємства), її також називають програмою-клієнтом.

Клієнт і сервер взаємодіють наступним чином. Клієнт формулює і відсилає запит (SQL-запит) серверу, на якому розміщена БД. Сервер виконує запит і видає клієнту в якості результатів необхідні дані. Таким чином, вся обробка запиту виконується на віддаленому сервері. До переваг такої архітектури відносяться слідуючі фактори:

- для роботи з даними використовується реляційний метод доступу, що зменшує навантаження на мережу;
- прикладна програма напряму не керує базою, управлінням займається лише сервер. У зв'язку з цим можна забезпечити високий ступінь захисту даних;
- в прикладній програмі відсутній код, пов'язаний з управлінням БД, тому сама програма спрощується.

При роботі в архітектурі "клієнт-сервер" прикладна програма повинна:

- виконувати з'єднання з сервером і відключення від нього;
- формувати і відсилати запити серверу, отримуючи від нього результати запиту;
- виконувати обробку отриманих даних.

Таким чином, для бази даних розроблюваної системи будемо використовувати архітектуру "клієнт-сетвер". Для цієї мети на комп'ютері-сервері підприємства необхідно встановити сервер баз даних InterBase 6.x.

3.2. Створення бази даних

Для створення віддаленої бази даних InterBase зручно використовувати програму IBConsole, яка входить в стандартний комплект поставки серверу.

Процес створення БД починаємо з виклику команди Database\Create Database, що призводить до появи вікна Create Database (рис. 1).

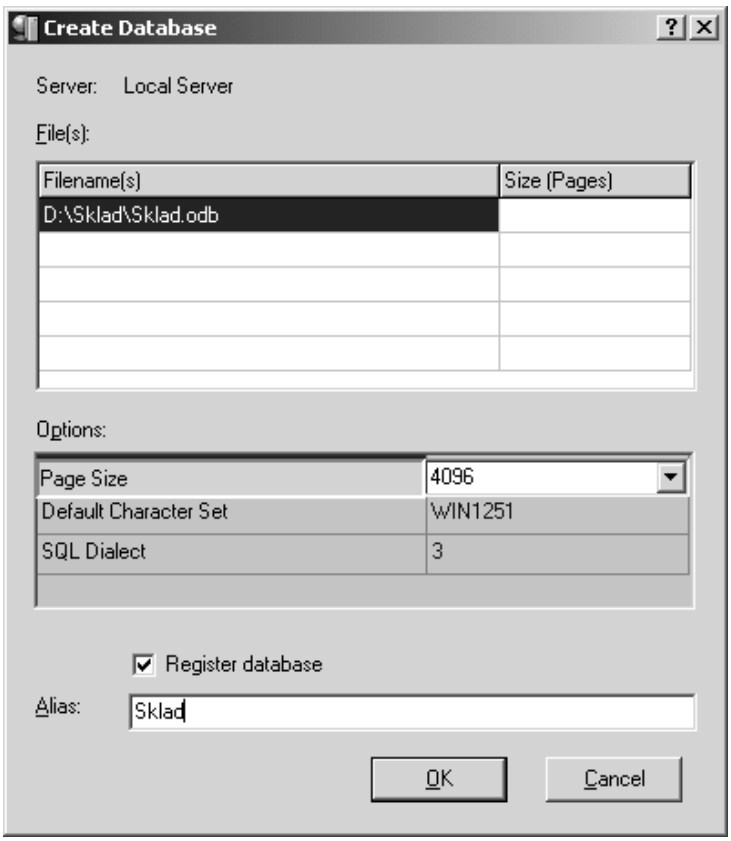

Рисунок 3.1 Вікно створення бази даних Create Database.

Для нової БД необхідно вказати її псевдонім, файли і параметри. Псевдонім, що задається в полі Alias, служить для ідентифікації БД при роботі всередині серверу InterBase. Задаємо псевдонім БД Sklad.

В списку File(s) перераховуються файли створюваної бази даних і їх розміри (в сторінках), оскільки наша БД складається з одного файлу, то вказуємо його ім’я і каталог, в якому він розміщується. Розмір файлу по замовчуванню становить 75 сторінок, тому задавати його повторно не обов’язково.

Список параметрів Options містить такі параметри:

- Page Size – розмір сторінки, залишаємо значення по замовчуванню – 4096;
- Default Character Set – встановлення набору, що використовується по замовчуванню для кодування символів. У Windows для

символів, що включають кирилицю, застосовується варіант Windows 1251 коду ANSI (встановлюємо значення WIN1251);

- SQL Dialect – діалект мови запитів SQL, встановлюємо значення 3.

Після натиснення кнопки ОК в каталозі D:\Sklad буде створено файл бази даних Sklad.odb. Проте новостворена БД не містить ні таблиць, ні даних. Для створення таблиць будемо використовувати мову структурованих запитів SQL (Structured Query Language). SQL здійснює дії з таблицями та їхніми даними. Запущена програма на мові SQL називається SQL-запитом.

Мова SQL має декілька стандартів, з яких найбільш поширеними серед виробників програминх продуктів є стандарти SQL-89 і SQL-92. “Стандарт SQL-92”, який підтримується “Американським національним інститутом стандартів” ANSI, також називають стандартом ANSI або ANSI/ISO. Є декілька стандартів і різноманітна їх інтерпретація призвели до появи багатьох діалектів мови SQL, які в більшій чи меншій мірі відрізняються один від одного.

В інтераутивному режимі IB Console виконує команди , які написані на мові SQL. Виконання SQL-запитів з показом результатів виконуєтсья у вікні Interactive SQL, що викликається за допомогою команди Tools/Interactive SQL (рис. 2). В заголовку вікна відображається ім’я файлу бази даних, а в стрічці стану – його повне ім’я. У вікні Inetractive SQL можна виконувати різноманітні операції з БД, включаючи створення і знищення БД і її таблиць тощо. SQL-запити можна набирати і виконувати в ручному режимі. Команди вводяться зверху екрана, а результати показуються знизу.

Виконання оператора здійснюється натисненням кнопки Execute Query на інструментальній панелі.

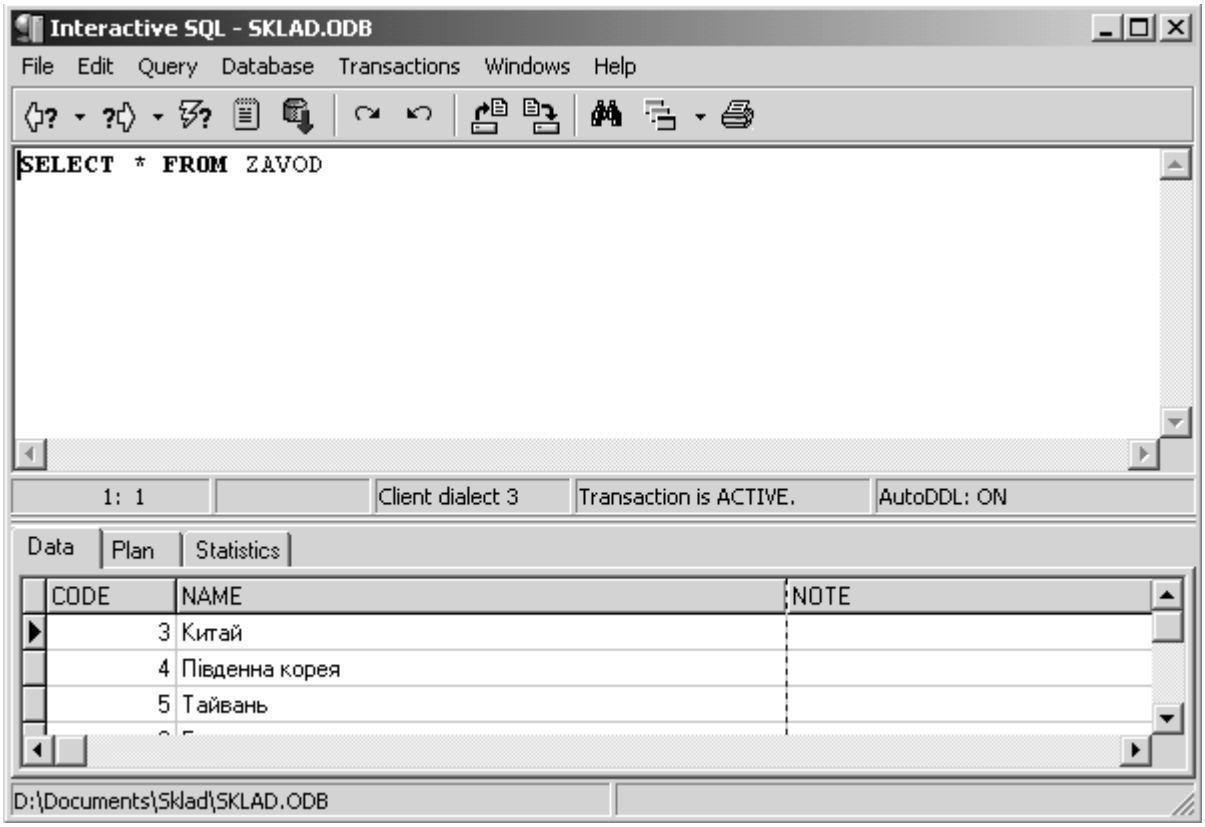

Рисунок 3.2 Вікно Interactive SQL при виконанні запиту в інтерактивному режимі.

В даному випадку вікно Inetractive SQL консолі IBConsole використовується лише для створення таблиць бази даних, генераторів та тригерів, тобто для створення БД. Запити під час роботи інформаційної системи формуються програмою-клієнтом.

Для створення таблиці використовуємо оператор мови SQL CREATE TABLE. Розглянемо створення таблиці постачальників POSTACH. У верхній частині вікна Interactive SQL розміщуємо SQL-запит:

```
CREATE TABLE POSTACH
( CODE      INTEGER NOT NULL,
  NAME      VARCHAR(50)  NOT NULL,
  NOTE      VARCHAR(50) ,
  ADDRES  VARCHAR(150) ,
  TEL         VARCHAR(30)
  PRIMARY KEY (CODE)  );
```

Після оператору CREATE TABLE пишемо назву таблиці – POSTACH, після чого в дужках перераховуємо поля, які повинна містити таблиця – CODE, NAME, NOTE, ADRES, TEL, та описуємо ключове поле – операнд PRIMARY

KEY. Після імені поля таблиці потрібно описати тип поля і можна задати обмеження поля. В даному випадку поле CODE цілочисельного типу (INTEGER), усі інші поля – стрічкового типу (VARCHAR(N), де N – максимальна довжина стрічки). Обмеження поля NOT NULL вказує на те, що дане поле не може містити порожній запис, це стосується полів CODE та NAME, оскільки поле CODE – ключове і містить унікальний код постачальника, а поле NAME містить ім’я постачальника, що є також обов’язковим для всіх записів.

У розроблюваній базі даних використовуються такі типи полів:

- INTEGER – значення цілого числа, від - 2 147 483 648 до 2 147 483 647;
- VARCHAR(N) – стрічка символів довжиною до N символів (не більше 32 767);
- CHAR(1) – стрічка символів довжиною 1, заміняє логічний тип;
- DECIMAL(M, N) – десяткове число з цілою (не більше M цифр) і дробовою (N цифр) частинами;
- DATE – значення дати, від 01.01.0100 до 11.12.5941.

Тексти SQL-запитів для створення інших таблиць БД подані в додатках.

Серед типів стовбців таблиць InterBase відсутній автоінкрементний тип, при заданні якого значення записів поля автоматично заповнюються унікальними цілочисельними значеннями, тому для забезпечення унікальних записів використовуються генератори і тригери.

Генератор повертає унікальне цілочисельне значення. За допомогою мови SQL-серверу можна створити генератор і встановити йому початкове значення. Генератор створюєтсья стлідуючим оператором:

CREATE GENERATOR <ім’я генератора>;

Початкове значення задається командою:

SET GENERATOR <ім’я генератора> TO <початкове значення>;

Початкове значення являє собою ціле число, починаючи з якого формується числовий ряд. Звернення до генератора виконується з допомогою функції

GEN_ID (<ім’я генератора>, <крок>);

Ця функція повертає значення, збільшене на цілочисельний крок відносно попереднього значення генератора.

Таким чином, для кожної таблиці розроблюваної бази даних необхідно створити генератор, значення якого будуть заноситися у ключові поля таблиць. Для кращої ясності імена генераторів будемо отримувати з імен відповідних таблиць з приставкою GEN. Наприклад, для створення генератора для таблиці BASE у верхній частині вікна Interactive SQL програми IBConsole напишемо такий текст:

```
CREATE GENERATOR GENBASE;
SET GENERATOR GENBASE TO 1;
```

Після натиснення кнопки Execute SQL буде створено генератор GENBASE, який має початкове значення, рівне 1. Аналогічно необхідно створити генератори для кожної з таблиць бази даних.

Список всіх створених генераторів та їх поточне значення можна побачити у вікні IBConsole, клацнувши мишею у лівій частині цього вікна на піктограмі Generators (рис. 3.3).

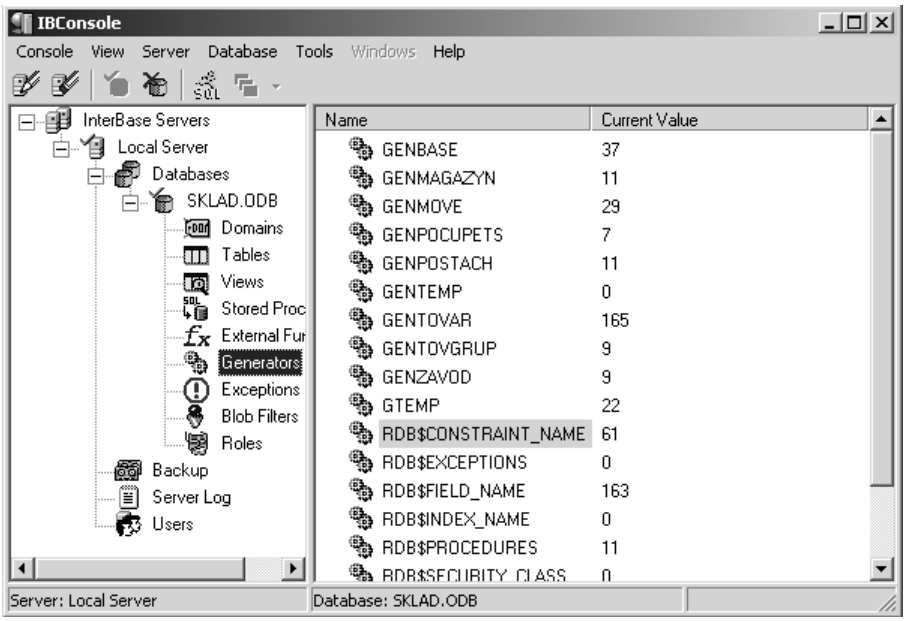

Рисунок 3.3 Список створених генераторів у вікні програми IBConsole.

Тригер, як і генератор, являє собою процедуру, що знаходиться на сервері баз даних, але, на відміну від генератора, тригер викликається автоматично при модифікації записів бази даних, тобто при зміні стовбців або при їх знищенні чи додаванні. Тригер не можна викликати з прикладної програми клієнта. Тригер може викликатися при редагуванні, додаванні або знищенні записів до і/або після цих подій. В нашому випадку тригер зручно використовувати для заповнення ключових полів таблиць значеннями відповідних генераторів. Оскільки для кожної з таблиць БД необхідно створити відповідний тригер, то імена тригерів будемо будувати із імені таблиці з приставкою CODE. Розглянемо текст SQL-запиту для створення тригеру для таблиці TOVAR.

```
CREATE TRIGGER CODETOVAR FOR TOVAR
        ACTIVE
        BEFORE INSERT
                AS
                BEGIN
                        NEW.CODE = GEN_ID(GENTOVAR, 1);
                END
```

Створення тригера виконується оператором CREATE TRIGGER, після якого пишеться ім’я тригера, операнд FOR та ім’я таблиці, для якої створюється тригер. Операнд ACTIVE визначає активність тригера відразу після створення. Операнди BEFORE INSERT вказують на те, що тригер буде викликатися перед вставкою нового запису. Оператором NEW.CODE = GEN_ID(GENTOVAR, 1) ключовому полю CODE присвоюється унікальне значення генератора GENTOVAR, яке рівне попередньому значенню цього генератора, збільшеному на 1.

Аналогічно створюємо тригери для інших таблиць бази даних. Таким чином, за допомогою генераторів і тригерів у таблицях БД значення ключових полів будуть автоматично заповнюватися унікальними значеннями після вставки нових записів. Тексти SQL-запитів для створення генераторів і тригерів подано в додатках.

3.3. Створення клієнтської програми

Візуальне середовище Delphi досить популярне : від неспеціалістів до системних програмістів, які займаються розробкою складних прикладних програм і інформаційних систем.

Delphi дозволяє швидко і зручно розробляти ефективні прикладні програми, включаючи програми для роботи з базами даних. Система має розвинуті можливості по створенню користувацького інтерфейсу, широкий набір функцій, методів і властивостей для вирішення прикладних розрахункових задач. В системі наявні розвинуті засоби відладки, що полегшує розробку прикладних програм.

Традиційно Delphi відносять до систем швидкої розробки програмного забезпечення. Разом з тим, ця система володіє практично усіма можливостями сучасних СУБД, таких як Microsoft Access і Visual FoxPro. Вона дозволяє зручно створювати прикладні програми з допомогою інструментальних програмних засобів, візуально готувати запити до баз даних, а також безпосередньо писати SQL-запити до баз даних. Враховуючи ці фактори, середовище програмування Delphi якнайкраще підходить для створення програмного забезпечення розроблюваної інформаційної системи.

3.3.1. Створення головного вікна програми

Для того щоб почати розробку нової програми, необхідно викликати команду File\New\Application середовища програмування, після чого з’явиться порожнє головне вікно програми.

Головне вікно на етапі проектування має вигляд, показаний на рис. 4.

У властивість стрічкового типу Name форми встановлюємо значення 'BaseF', для встановлення заголовку вікна служить властивість Caption, іконка вікна зберігається у властивості Icon типу Ticon.

На форма головного вікна розміщені такі компоненти:

- головне меню MainMenu1;

- інструментальна панель ToolBar1;
- кнопки інструментальної панелі ToolButton;
- випадний список вибору підрозділу підприємства cbMagazyn;
- компонент для табличного представлення даних DBGrid1;
- компонент для відображення поточного стану програми StatusBar1;
- набір зображень для пунктів головного меню і інструментальних кнопок ImageList1;
- компонент для впорядкування подій у програмі ActionManager1;
- невізуальні компоненти для доступу до бази даних та для формування наборів даних IBDatabase1, IBTransaction1, cbQuery, IBQuery1, DataSource1.

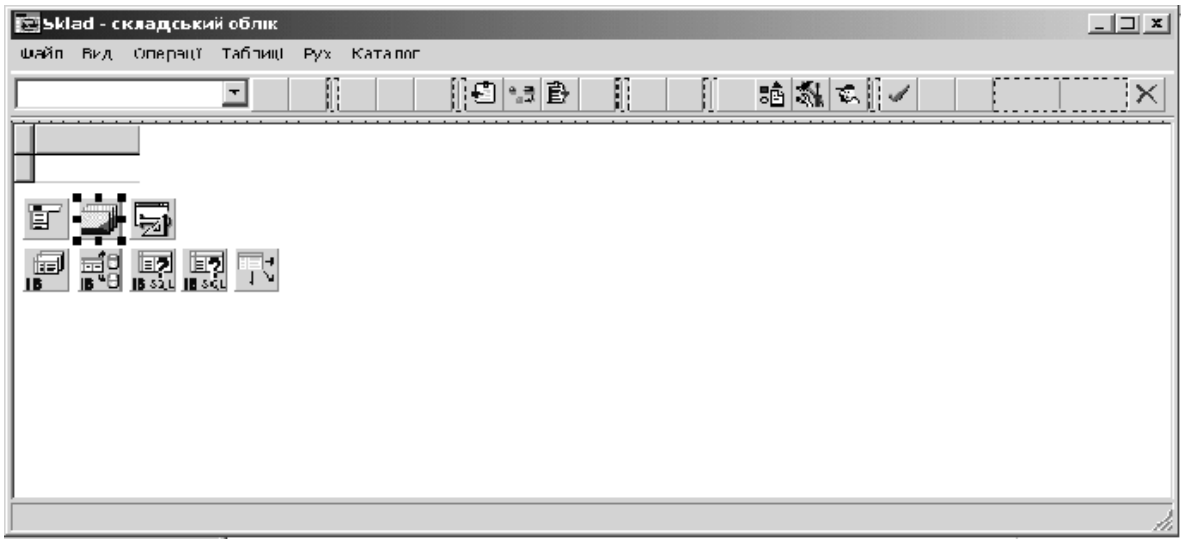

Рисунок 3.4 Головне вікно програми на етапі проектування.

Для створення головного меню програми використовуємо компонент MainMenu1 класу TMainMenu. Після розміщення компонента на формі необхідно створити його опції, для чого командою контекстного меню компонента Menu Designer викликаємо однойменне вікно для створення пунктів головного меню (рис. 3.5). Для створення нової опції головного меню необхідно перейти у вікно Інспектора об’єктів, ввести текст у стрічці Caption і натиснути Enter. Кожна опція може розкриватися у список підопцій. У назвах опцій можна вказувати символ «&» перед тим символом, який визначає

клавішу швидкого вибору опції (акселератор). Ця клавіша (або комбінація клавіш) визначається властивість ShortCut опції головного меню.

Головне меню програми складається з таких пунктів:

- Файл – містить команди для настройки параметрів програми, друку документів тощо;
- Вид – команди для сортування даних в наборі даних та для налаштування параметрів фільтрації при відображенні даних;
- Операції – команди для виконання основних операцій з товаром – прихід, внутрішнє переміщення, продаж – та для роботи з записами в наборі даних;
- Таблиці – команди для перегляду і зміни таблиць бази даних, які містять дані про підрозділи підприємства, групи товарів, виробників, постачальників, покупців та динаміку курсу долара;
- Рух – команди для відображення руху товару на підприємстві та встановлення параметрів цього відображення;
- Каталог – команди для формування та встановлення параметрів каталогів.

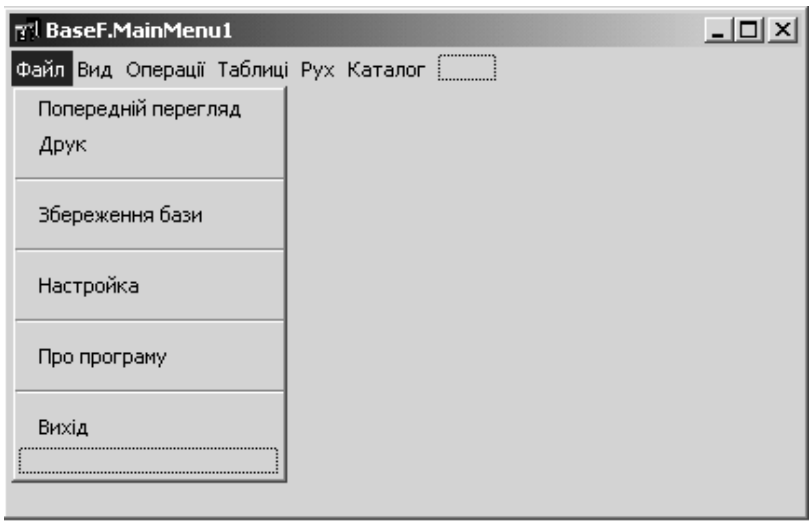

Рисунок 3.5 Створення пунктів головного меню програми за допомогою вікна Menu Designer

Більшість з команд головного меню для зручності дублюються відповідними кнопками інструментальної панелі.

Для того, щоб компонент DBGrid1 зайняв усе вільне місце на формі головного вікна, встановлюємо його властивість Align у значення alClient.

3.3.2. З'єднання з базою даних

Перед початком роботи програми необхідно під'єднатися до бази даних на сервері. Для з'єднання з базою даних служить компонент IBDatabase1, у властивості якого DatabaseName стрічкового типу знаходиться ім'я файлу БД та шлях доступу до нього. Властивість Params типу Tstrings визначає параметри з'єднання, в тому числі і ім'я користувача USER NAME та його пароль PASSWORD. Ці дані можна ввести на етапі проектування (рис. 6), проте тоді будь-який користувач системи буде мати змогу запустити клієнтську програми минувши парольний захист, тому внесення імені користувача та паролю відбувається при роботі програми.

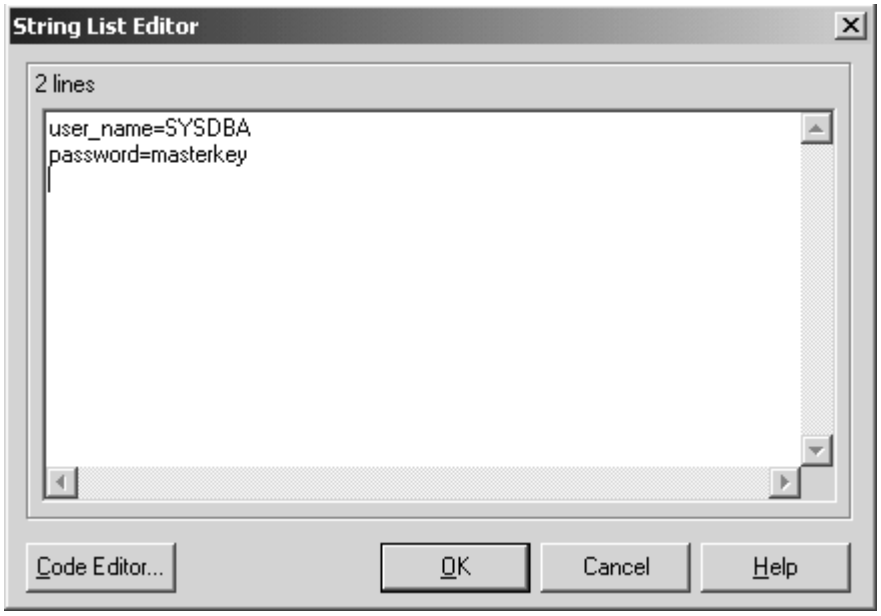

Рисунок 3.6 Вікно для встановлення параметрів з'єднання з базою даних на етапі проектування програми.

З'єднання з базою даних відбувається за допомогою вікна AboutF, яке показується в модальному режимі перед появою головного вікна, для чого в обробнику події OnShow форми BaseF розміщуємо код

AboutF.ShowModal;

Для того, щоб полоси заголовку вікна AboutF не було видно, встановлюємо властивість цього вікна BorderStyle у значення bsNone. На порожній формі AboutF розміщуємо компонент Image1, у властивість Picture класу TBitmap вносимо відповідне зображення, яке буде з’являтися при старті програми. Також на формі розміщуємо компоненти класу TLabel для відображення статичної інформації про програму та розробника. Для вводу імені користувача та паролю використовуємо компоненти Edit1 та Edit2 класу TEdit. Оскільки пароль буде вводитися в компоненті Edit2, то його властивість PasswordChar вводимо сивол "*", яким при роботі програми буде замінятися кожен введений символ у компоненті. Загальний вигляд форми AboutF на етапі проектування показано на рис. 7. Оскільки на формі відсутні кнопки для підтвердження вводу або закриття вікна, то необхідно передбачити реакцію компонентів Edit1 та Edit2 на натискання клавіш клавіатури. Натискання клавіші Enter буде означати підтвердження вводу (віртуальний код клавіші 13), клавіша Escape (віртуальний код 27) – відміна вводу і закриття вікна. Для виконання цього використовуємо обробник події OnKeyDown відповідних компонентів, у параметрах якого передається віртуальних код натисненої клавіші.

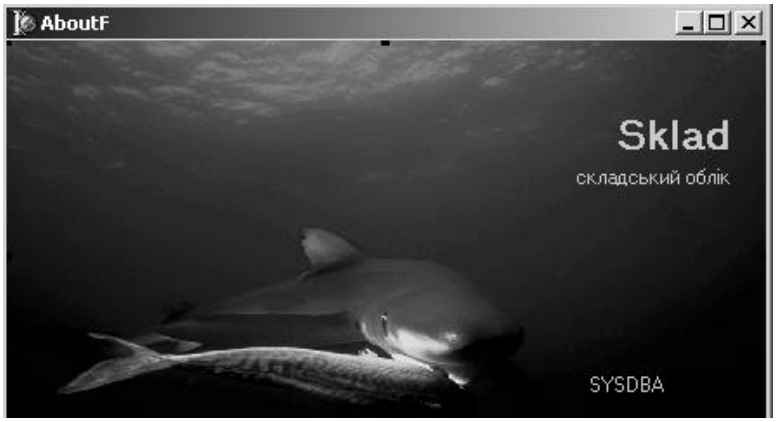

Рисунок 3.7 Вигляд вікна AboutF на етапі проектування.

Код обробника OnKeyDown має такий вигляд:

```
procedure TAboutF.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
```

```
begin
if Key = 13 then ConnectDB; // якщо натиснено Enter, то викликається
                            // процедура ConnectDB для з'єднання з БД
if Key = 27 then            // натиснено Escape
  begin
  Close;                    // закрити вікно AboutF
  BaseF.Close;              // закрити головне вікно програми
  end;
end;
```

Процедура ConnectDB у властивість Params компоненту IBDatabase1, який розміщений на формі головного вікна, вносить ім'я користувача з компоненту Edit1 та його пароль (компонент Edit2), після чого робиться спроба з'єднатися з базою даних:

```
BaseF.IBDatabase1.Open;
```

Якщо спроба виявилася вдалою, то закриваємо вікно AboutF:

```
if BaseF.IBDatabase1.Connected then AboutF.Close;
```

Після закриття вікна AboutF управління знову передається головному вікну програми.

Для відображення даних з БД у головному вікні використовується набір даних використовується компонент IBQuery1, у властивість якого Database вводимо ім'я компонента IBDatabase1. З'єднання візуального компоненту DBGrid1 з набором даних виконується за допомогою джерела даних DataSource1, у властивості якого DataSet вибираємо ім'я компонента IBQuery1, після чого встановлюємо властивість компонента DBGrid1 DataSource у значення DataSource1. Таким чином, після відкриття набору даних IBQuery1 компонент DBGrid1 буде відображувати дані, які в ному містяться. Встановлення значень описаних властивостей компонентів IBQuery1, DataSource1 та DBGrid1 виконуємо на етапі проектування програми.

Для формування SQL-запитів програмою використовується властивість SQL типу TStrings компонентів класу TIBQuery. Після вводу тексту запиту виконання самого запиту виконується або при відкритті набору даних (IBQuery1.Open), або викликом методу набору даних ExecuteSQL.

3.3.3. Зчитування даних з таблиць

Під час роботи програми часто виникає потреба отримати швидкий доступ до назви і коду підрозділу, групи товарів, виробника, постачальника або покупця. З цією метою для зберігання імен використовуємо змінні List1 ... List5 типу TStringLists, а для зберігання відповідних кодів – Lcod1 … Lcod5 відповідно. Для заповнення цих списків створюємо процедуру Lists, в якій спочатку резервується пам’ять для всіх списків, а потім заповнюються самі списки. Резервування пам’яті виконуємо таким чином:

```
List1 := TStringList.Create; Lcod1 := TStringList.Create; //Підрозділи
List2 := TStringList.Create; Lcod2 := TStringList.Create; //Групи товарів
List3 := TStringList.Create; Lcod3 := TStringList.Create; //Виробники
List4 := TStringList.Create; Lcod4 := TStringList.Create; //Покупці
List5 := TStringList.Create; Lcod5 := TStringList.Create; //Постачальники
```

При виході з програми (закритті головного вікна програми) у процедурі FormClose вивільняємо пам’ять, зарезервовану для списків, та закриваємо базу даних:

```
IBDatabase1.Close;
List1.Free; Lcod1.Free;
List2.Free; Lcod2.Free;
List3.Free; Lcod3.Free;
List4.Free; Lcod4.Free;
List5.Free; Lcod5.Free;
```

Для заповнення списків даними з відповідних таблиць БД, необхідно сформувати SQL-запити їх результати помістити у змінні типу TStringLists. Спочатку очищуємо властивість SQL компонента cbQuery від текстів попередніх запитів:

```
cbQuery.SQL.Clear;
```

тоді записуємо в цю властивість текст нового SQL-запиту:

cbQuery.SQL.Add('SELECT name, code FROM MAGAZYN');

В даному запиті використовується оператор SELECT – один з найважливіших операторів мови SQL. Від використовується для відбору записів, що задовільняють складним критеріям пошуку. Результат виконання SQL-запиту, заданого оператором SELECT, є вибірка записів, що відповідають заданим у ньому умовам. Оскільки в даному випадку нам потрібні всі записи з таблиць, то умов ми не задаємо. Після оператору SELECT перераховуються поля, записи яких необхідно отримати, в нашому випадку це поле name – для списку List1, та поле code – для списку Lcode1. Після переліку полів пишеться слово FROM і перераховуються таблиці, з яких робиться вибірка (в даному випадку – з таблиці MAGAZYN). Таким чином, при виконанні описаного SQL-запиту отримаємо набір даних із назв підрозділів підприємства і їх кодів. Виконання запиту здійснюється при відкритті набору даних:

cbQuery.Open;

Після цього заповнюємо списки List1 та Lcod1 відповідними даними:

```
for i:=1 to cbQuery.RecordCount do
  begin
  List1.Add(cbQuery.FieldByName('name').AsString);
  Lcod1.Add(cbQuery.FieldByName('code').AsString);
  cbQuery.Next;
  end;
```

Властивість RecordCount компонента cbQuery вказує на кількість записів у наборі даних, доступ до стрічкових полів виконується функцією

cbQuery.FieldByName('name').AsString

яка повертає вміст поточного запису поля name.

Після заповнення списків закриваємо набір даних:

cbQuery.Close;

Для внесення у списки інформації з інших таблиць виконуємо аналогічні дії, як для таблиці MAGAZYN, але в тексті SQL-запиту вказуємо інші імена таблиць.

У головному вікні програми повинні відображатися дані про товар, даявний у вибраному підрозділі підприємства. Підрозділ вибирається у випадному списку cbMagazyn класу TComboBox. Для того, щоб відключити можливість ручного вводу символів у цей список, властивість компонента Style встановлюємо у значення csDropDownList.

На початку роботи програми необхідно створити пункти випадного списку cbMagazyn, які б відповідали назвам підрозділів. Для цього виконуємо операцію:

```
CBmagazyn.Items := List1;
```

в якій у властивість Items компонента записується вміст списку з іменами підрозділів List1.

Компонент IBQuery1 призначений для формування записів з бази даних у головному вікні. Оскільки текст SQL-запиту при цьому змінюватися майже не буде, то його можна задати на етапі проектування і під час роботи програми не змінювати. Цей текст має такий вигляд:

```
SELECT BASE.CODE, BASE.IDTOV, MAGAZYN.NAME,
TOVGRUP.NAME, TOVAR.NAME, ZAVOD.NAME, BASE.NUMOF,
TOVAR.PR, TOVAR.M, TOVAR.PRA, TOVAR.MA, TOVAR.PRB,
TOVAR.MB, TOVAR.PRC, TOVAR.MC, BASE.DEN,
TOVAR.NOTE            /* перелік полів */
FROM BASE, MAGAZYN, TOVGRUP, TOVAR, ZAVOD  /* таблиць */
WHERE (BASE.IDMAG = :cb)        /* задання умов відбору */
AND (BASE.IDMAG = MAGAZYN.CODE)
AND (TOVAR.CODE = BASE.IDTOV)
AND (TOVGRUP.CODE = TOVAR.IDGROUP)
AND (ZAVOD.CODE = TOVAR.MANUF)
```

У цьому запиті крім переліку полів для результуючого набору даних та таблиць задаються ще умови вибірки записів, які перераховуються після слова WHERE і розділяються словами AND, що вказує на обов’язкове виконання всіх умов запиту. Проте текст запиту буде залежати від вибраного у списку cbMagazyn підрозділу. Для налаштування статичного SQL-запиту під час

виконання програми зручно використовувати параметри. Параметр – це спеціальна змінна, перед іменем якої в тексті запиту ставиться двокрапка. Задання параметрів виконується на етапі проектування програми, для цього необхідно викликати вікно Editing IBQuery1.Params відповідного компонента, клацнувши на його властивості Params у інсекторі об’єктів. Таким чином створюємо параметр cb для компонента IBQuery1. Під час роботи програми перед виконанням SQL-запиту цього компонента необхідно його параметру присвоїти значення (код підрозділу):

```
IBQuery1.ParamByName('cb').AsInteger :=
StrToInt(Lcod1.Strings[cbMagazyn.ItemIndex]);
```

Після чого виконуєто SQL-запит:

```
IBQuery1.Open;
```

Таким чином, при запуску програми після під’єднання до бази даних у головному вікні будути відображатися дані про товар, наявний у вибраному у випадному списку підрозділі. Для того, щоб набір даних обновлювався при виборі іншого підрозділу, необхідно в процедурі TBaseF.cbMagazynChange, яка викликається при зміні стану списку, закрити набір даних, присвоїти параметру cb значення іншого вибраного підрозділу і знову виконати запит, відкривши набір повторно:

```
procedure TBaseF.cbMagazynChange(Sender: TObject);
begin
IBQuery1.Close;
IBQuery1.ParamByName('cb').AsInteger :=
                StrToInt(Lcod1.Strings[cbMagazyn.ItemIndex]);
OpenIBQuery;
end;
```

У розроблюваній програмі передбачається щоденне фіксування курсу долара відносно гривні. При запуску програми необхідно перевіряти, чи в таблиці DOLLAR вже є запис із сьогоднішньою датою, і у випадку відсутності запропонувати користувачеві ввести відповідні дані.

3.3.4. Вставка записів в таблиці бази даних

Для вводу користувачем даних щодо курсу долара створюємо форму DollarF, вигляд якої показано на рис. 3.8

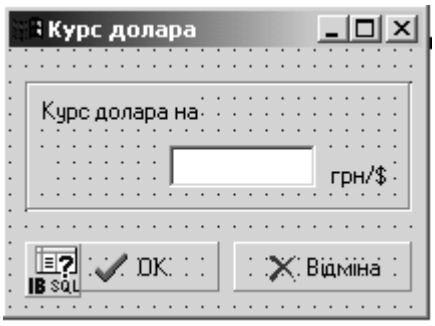

Рисунок 3.8 Вигляд форми для вводу курсу долара.

На формі поряд з іншими компонентами розміщений компонент IBQuery1, який служить для роботи з таблицею DOLLAR. В обробнику події OnClick кнопки з написом "Відміна" розміщуємо команду Close, яка призводить до закриття вікна без зміни будь-яких параметрів. Натисканням кнопки "ОК" властивість Tag форми DollarF встановлюється у значення 1, після чого вікно також закривається:

```
DollarF.Tag := 1;
Close;
```

Перед виконанням цих дій виконується заміна у введеному тексті, який міститься у стрічці st, коми або крапки на стандартний для даної операційної системи розділовий знак між цілою і дробовою частиною числа:

```
st := Edit1.Text;
for i:=1 to length(st) do
 if (st[i]='.') or (st[i]=',') then st[i]:=DecimalSeparator;
```

після чого здійснюється перевірка правильності вводу числа шляхом спроби перетворення введеного тексту у дійсне число:

```
try
  StrToFloat(st);
  except
  MessageDlg('Недобре введено число !', mtError, [mbOk],0);
  exit;
  end;
```

Для видачі відповідного повідомлення користувачу у випадку помилки вводу використовуємо стандартну процедуру MessageDlg, в якій вказуємо текст повідомлення, його тип та кількість і тип кнопок.

При появі головного вікна програма формує SQL-запит до бази даних, яким робиться вибірка записів з таблиці DOLLAR, у полі дати яких міститься сьогоднішня дата, яку отримуємо за допомогою функції Date. Текст запиту вносимо у властивість SQL компоненту IBQuery1:

```
DollarF.IBQuery1.SQL.Clear;
DollarF.IBQuery1.SQL.Add('select * from kurs where den =
                                                '+#39+DateToStr(Date)+#39);
DollarF.IBQuery1.Open;
```

Якщо при використанні оператора SELECT у SQL-запиті необхідно отримати певні записи у всіх полях, то список полів можна заміняти символом "*". Якщо набір даних після виконання даного запиту виявиться порожнім, що свідчить про відсутність запису із сьогоднішньою датою, то показуємо форму DollarF у модальному режимі:

```
if DollarF.IBQuery1.IsEmpty then DollarF.ShowModal;
```

Якщо дане вікно було закрите кнопкою "ОК", про що свідчить значення властивості Tag форми, то вставляємо введене користувачем число і сьогоднішню дату у таблицю DOLLAR:

```
f DollarF.Tag = 1 then begin
   s := DateToStr(Date);
   DollarF.IBQuery1.Close;
   DollarF.IBQuery1.SQL.Clear;
```

```
        DollarF.IBQuery1.SQL.Add('insert into kurs (den, dollar) values
('+#39+s+#39+', '+#39+st+#39+')');
        DollarF.IBQuery1.ExecSQL;
        end;
```

Вставка записів виконується з допомогою оператора INSERT, який добавляє до таблиць одну або декілька записів. В результаті виконання цього оператора до таблиці, ім’я якої вказано після слова INTO, додається один запис. Для доданого запису заповняються поля, перераховані в списку (в даному випадку – DEN, DOLLAR). Значення полів беруться із списку, розміщеного після слова VALUES. Списки полів і списки значень повинні відповідати один одному по кількості “елементів” і по типу. Порядок полів і значень може відрізнятися від порядку полів в таблиці.

3.3.5. Використання механізму подій

Більшість дій, виконуваних в програмі, можна виконати командами головного меню або відповідними кнопками на інструментальній панелі. Крім того, деякі команди для зручності при користуванні програмою варто винести у контекстні меню відповідних компонентів. Для централізації виконання цих дій у програмі використовуємо компонент ActionManager1 класу TActionManager. Цей компонент дозволяє створювати об’єкти класу TAction, для яких можна задати піктограму, заголовок, текст оперативної підказки тощо. Дію, яка відповідає даному об’єкту, описуємо в обробнику події OnExecute об’єкту. Для створення об’єкту необхідно викликати вікно Editing BaseF.ActionManager1 командою Customize контекстного меню компонента (рис.3.9).

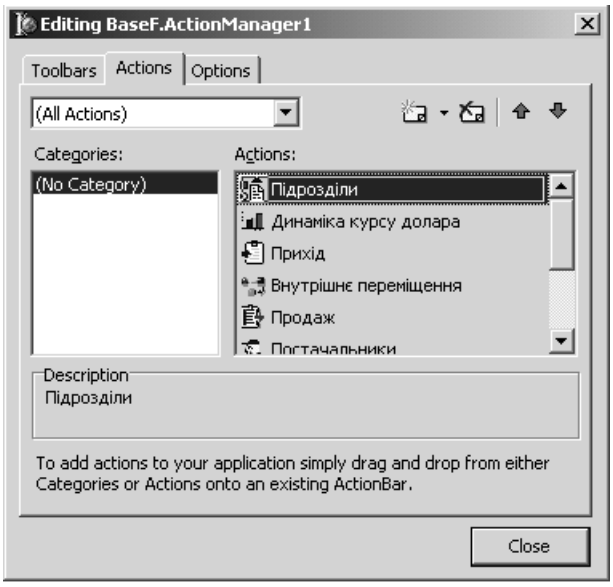

Рисунок 3.9 Вікно для створення об’єктів класу TAction.

Кнопкою New Action у вікні або натисканням клавіші Insert на клавіатурі створюється новий об’єкт, а кнопкою Delete – знищується виділений об’єкт.

Після виконання описаних дій властивостям Action пунктів меню та інструментальних кнопок присвоюємо імена відповідних об’єктів класу TAction, після чого пунктам меню та кнопкам присвоюються аналогічні до об’єктів піктограми, заголовки та оперативні підказки.

Таки чином, для перегляду і зміни даних про підрозділи підприємства (команда Підрозділи меню Таблиці) створюємо об’єкт класу TAction, присвоюємо йому ім’я Tochka (властивість Name), заголовок і підказку "Підрозділи" (властивості Caption та Hint) та відповідну піктограму. Усі піктограми зберігаються у компоненті ImageList1, зв’язок якого з компонентом ActionManager1 встановлюється через властивість Images останнього. Для присвоєнню об’єкту TAction відповідної піктограми, присвоюємо властивості ImageIndex об’єкту номер цієї піктограми у компоненті ImageList1. Редагування списку піктограм виконується за допомогою вікна BaseF.ImageList1 ImageList, що викликається командою контекстного меню компонента ImageList Editor.

3.3.6. Виконання операцій за таблицями груп товарів, виробників, постачальників та клієнтів.

Для перегляду і зміни інформації про підрозділи підприємства, групи товарів, виробників, постачальників і покупців служить вікно TableF, вигляд якого на етапі проектування показаний на рис. 3.10.

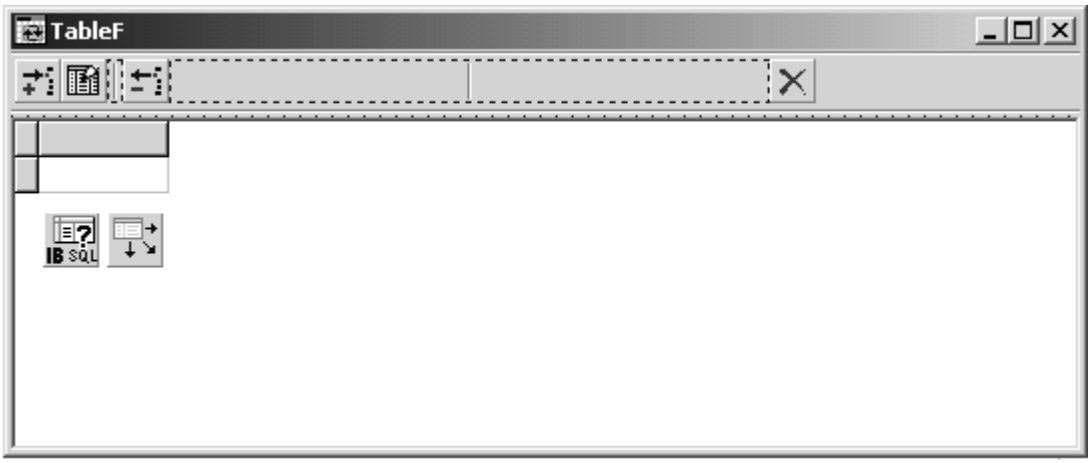

Рисунок 3.10 Вигляд вікна TableF на етапі проектування.

На формі розміщені інструментальна панель ToolBar1 з кнопками для додавання нового запису до таблиці, зміни, видалення записів та закриття вікна, компонент DBGrid1 для табличного відображення даниз з БД, та компоненти IBQuery1 та DataSource1 для доступу до бази даних.

Так, в обробнику події OnExecute об’єкту Tochka встановлюємо заголовок вікна TableF "Підрозділи" і показуємо вікно в модальному режимі:

```
procedure TBaseF.TochkaExecute(Sender: TObject);
begin
TableF.Caption := 'Підрозділи';
TableF.ShowModal;
end;
```

Аналогічно створюємо об’єкти Groups, Zavod, Postach і Pocupets для виведення інформації про групи товарів, виробників, постачальників і покупців. Обробники подій OnExecute цих об’єктів відрізняються лише тим, що вінку TableF присвоюються відповідні заголовки.

В обробнику події OnShow вікна TableF стрічковій змінній s1 в залежності від заголовку вікна присвоюється ім’я таблиці, дані з якої необхідно відобразити у компоненті DBGrid1. Після того формується SQL-запит для вибірки всіх записів з цієї таблиці:

```
IBQuery1.SQL.Clear;
IBQuery1.SQL.Add('select * from ' + s1);
IBQuery1.Open;
```

Доступ до полів набору даних виконується за опомогою методу набору FieldByName(name), де name – ім’я поля в таблиці бази даних. Цей метод являє собою вункцію, яка повертає об’єкт поля TField, властивості якого можна змінювати під час виконання програми. Наприклад, властивість об’єкту DisplayLabel стрічкового типу містить заголовок стовбця таблиці при відображенні його візуальними компонентами, логічна властивість Visible вказує на те, чи буде взагалі дане поле відображатися цими компонентами:

```
IBQuery1.FieldByName('Name').DisplayLabel := 'Назва';
IBQuery1.FieldByName('Note').DisplayLabel := 'Додатково';
IBQuery1.FieldByName('Code').Visible := false;
```

Таблиці покупців і постачальників товару містять крім назви і коду також поля Address і Tel, в яких відображаються адреси і телефони, тому для цих таблиць необхідно передбачити коректне відображення заголовків стовбців:

```
if (TableF.Caption = Покупці') or (TableF.Caption = 'Постачальники') then
    begin
    IBQuery1.FieldByName('Addres').DisplayLabel := 'Адреса';
    IBQuery1.FieldByName('Tel').DisplayLabel := 'Телефон';
    end;
```

При закритті вікна TableF необхідно закрити набір даних:

```
IBQuery1.Close;
```

Управління таблицями у цьому вікні здійснюється за допомогою кнопок на інструментальній панелі. Для відображення інструментальної панелі та кнопок до неї служить компонент ToolBar1 класу TToolBar. Після розміщення компоненту на формі він відображається як порожня панель у верхній частині вікна. Для того, щоб на ній розмістити кнопку класу TToolButton, необхідно вибрати з контекстного меню компонента команду New Button, команда New Separator цього ж меню вставляє на панель сепаратор, який призначений для функціонального виділення на інструментальній панелі груп елементів і являє собою різновидність кнопок класу TToolButton.

Описаним методом створюємо на панелі інструментальні кнопки для управління таблицями, сепаратор і четверту кнопку для закриття вікна TableF. Для цієї четвертої кнопки в обробнику події OnClick необхідно написати всього одну команду – Close, яка і закриє вікно.

Для додавання і зміни записів таблиці командою File\New\Form створюємо нову форму NewF, вигляд якої показаний на рис. 3.11.

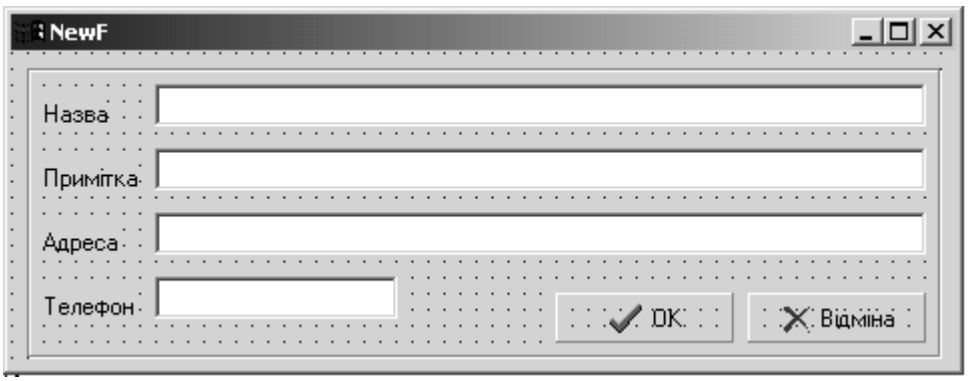

Рисунок 3.11. Вигляд вікна NewF на етапі проектування.

На новій формі розміщуємо чотири компоненти класу TLabel для відображення статичного тексту і присвоємо їх властивостям Caption відповідно стрічки "Назва", "Примітка", "Адреса" і "Телефон". Для вводу або зміни даних використовуємо відповідно чотири елементи класу TEdit. Для закриття вікна служать дві кнопки класу TSpeedButton з написами "ОК" і "Відміна". При натисненні кнопки "Відміна" перед закриттям вікна

користувачу видається повідомлення, в якому запитується, чи зберегти внесені зміни або введені дані:

```
if Edit1.Text <> '' then begin
i := MessageDlg('Закрити без збереження ?',mtInformation,[mbYES,mbNO],0);
  if i = 7 then exit;
  end;
```

У діалоговому вікні функції MessageDlg розміщуємо дві кнопки – Yes і No. При натисненні кнопки Yes діалогове вікно закривається і у його властивість ModalResult, значення якої повертає функція MessageDlg, вміщується значення 6, при натисненні кнопки No модальний результат рівний 7 і вікно TableF не закривається.

При появі на екрані вікна TableF в залежності від його заголовку на формі NewF приховуються або показуються поля для вводу адреси та телефону. Тоді значення властивості Tag вікна NewF встановлюється у відповідне значення від 1 до 5, оскільки у даному вікні може відображатися одна з п’яти таблиць:

```
if TableF.Caption = 'Підрозділи' then begin
    NewF.Caption := 'Новий підрозділ';
    NewF.Label3.Visible := False;
    NewF.Label4.Visible := False;
    NewF.Edit3.Visible := False;
    NewF.Edit4.Visible := False;
    NewF.Tag := 1;
    end;
```

Натиснення кнопки ToolButton1 інструментальної панелі (вставка нового запису) призводить до показу вікна NewF у модальному режимі. При натисненні кнопки ToolButton2 (редагування запису) у поля вводу форми NewF вводяться значення поточного запису в таблиці, після чого вікно також показується:

```
NewF.Edit1.Text := IBQuery1.FieldByName('name').AsString;
NewF.Edit2.Text := IBQuery1.FieldByName('note').AsString;
```

```
if (TableF.Caption = 'Покупці') or (TableF.Caption = 'Постачальники') then
  begin
  NewF.Edit3.Text := IBQuery1.FieldByName('addres').AsString;
  NewF.Edit4.Text := IBQuery1.FieldByName('tel').AsString;
  end;
NewF.ShowModal;
```

При натисненні кнопки "ОК" вікна NewF спочатку відбувається перевірка наявності введених або змінених даних, після чого в залежності від значення властивості Tag форми стрічковим змінним s1, s2 і s3 присвоюються відповідно назва таблиці, перелік полів та перелік значень для формування SQL-запиту:

```
if NewF.Tag = 1 then begin
   s1 := 'magazyn';
   s2 := 'name, note';
   s3 := #39+Edit1.Text+#39+', '+#39+Edit2.Text+#39;
   end;
```

Тоді цілочисельній змінній *i* присвоюється значення поля CODE поточного запису:

```
i := TableF.IBQuery1.FieldByName('code').AsInteger;
```

після чого формується і виконується SQL-запит для зміни запису:

```
TableF.IBQuery1.SQL.Add('update '+s1+' set name = '+#39+Edit1.Text+#39+
      ', note = '+#39+Edit2.Text+#39);
TableF.IBQuery1.SQL.Add(' where code = '+IntToStr(i));
   TableF.IBQuery1.ExecSQL;
```

або вставки нового запису:

```
TableF.IBQuery1.SQL.Add('insert into '+s1+' ('+s2+')');
TableF.IBQuery1.SQL.Add('values ('+s3+')');
   TableF.IBQuery1.ExecSQL;
```

Редагування записів таблиці бази даних виконується оператором UPDATE, після виконання якого для всіх записів, що задовільняють умову

відбору WHERE, змінюються значення полів. Імена полів і нові значення вказуються після слова SET.

Після виконання SQL-запиту вікно NewF закривається командою Close.

Кнопка ToolButton3 інструментальної панелі форми TableF служить для знищення поточного запису з таблиці БД. Перед знищенням користувачу видається відповідне повідомлення:

```
if MessageDlg('Знищити даний запис ?',mtWarning,[mbYES,mbNO],0) = 6
        then    begin  ...
```

Значення модального результату 6 свідчить про те, що діалогове вікно було закрите кнопкою Yes.

Після видачі попередження змінним i та s1 присвоюються відповідно значення поля Code та ім’я таблиці i формується SQL-запит:

```
i := IBQuery1.FieldByName('code').AsInteger;
IBQuery1.Close;
IBQuery1.SQL.Clear;
IBQuery1.SQL.Add('delete from '+s1+' where code = '+IntToStr(i));
IBQuery1.ExecSQL;
```

Для знищення запису або групи записів використовується оператор DELETE, в результаті виконання якого з таблиці, ім’я якої вказано після слова FROM, знищуються всі записи, які задовільняють умовам критерію відбору.

3.3.7. Виконання основних операцій з товаром на підприємстві

При поступленні нового товару на підприємство в базу даних необхідно внести відповідну інформацію. Для цього створюємо вікно AddF, яке також використовується при здійсненні внутрішнього переміщення або продажу товару. Загальний вигляд форми цього вікна показано на рис. 3.12.

Оскільки вікно може виклакатися і командами головного меню програми, і кнопками інструментальної панелі, то для його виклику при виконанні різніх операцій створюємо три об’єкти класу TAction: Add, VnutrMove, Prodaj і

присвоюємо їм заголовки – відповідно "Прихід", "Внутрішнє переміщення" і "Продаж". Перед показом вікна AddF у його властивість Tag записуємо чило, яке буде свідчити про виконувану операцію.

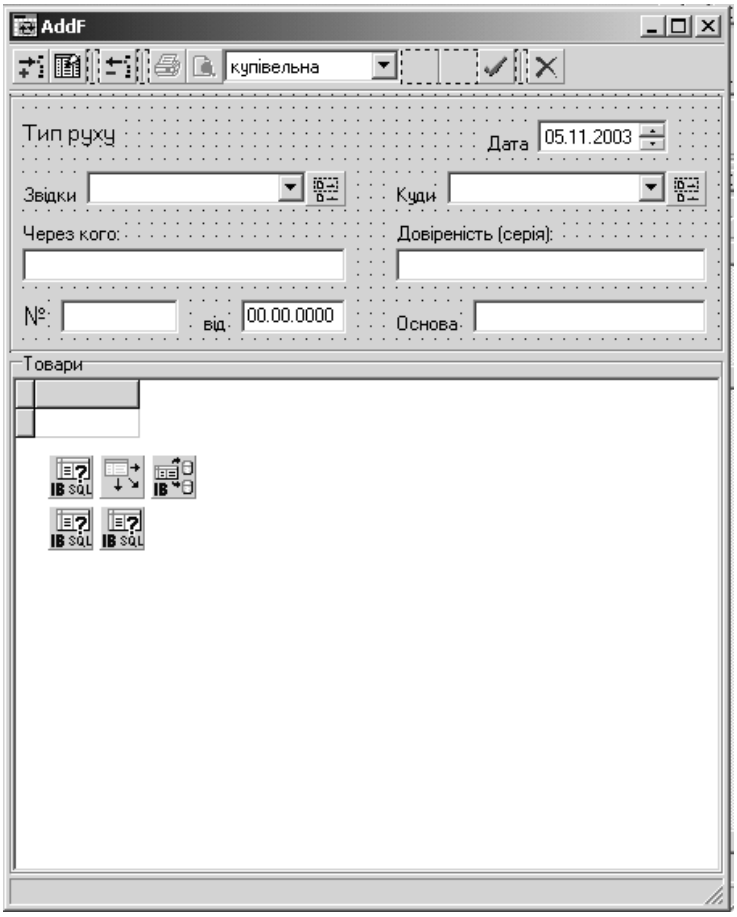

Рисунок 3.12. Вигляд форми вікна AddF.

Таким чином, у обробнику події OnExecute об’єкту Add пишемо такий код:

```
AddF.Tag := 1; //Прихід товару
AddF.Show;
```

Оскільки при відкритому вікні AddF необхідно зберегти можливість доступу до головного вікна програми, то дане вікно показуємо в немодальному режимі, а для того, щоб воно не розміщувалося на задній план при активному

головному вікні, властивість FormStyle форми AddF встановлюємо у значення fsStayOnTop, при якому вікно розміщується на передньому плані навіть при втраті активності.

На формі AddF розміщуємо такі основні компоненти:

- DBGrid1 – таблиця для відображення переліку товарів, з якими виконуються операції;
- ToolBar1 – інструментальна панель;
- ToolButton1..11 – кнопки інструментальної панелі для виконання операції з товаром та сепаратори для виділення функціональних груп кнопок;
- Label1..7 – компоненти для відображення статичних написів;
- Dtp1 класу TDateTimePicker – календар для відображення та вводу дати здійснення операції;
- cbVid та cbKudy класу TComboBox – випадні списки для вибору покупців, постачальників товару або підрозділів підприємства, між якими здійснюється операція;
- sb1, sb2 класу TSpeedButton – кнопки для виклику вікон перегляду і редагування таблиць постачальників, підрозділів або покупців;
- LEdit1, LEdit2 класу TLabeledEdit, Edit1, Edit3 та MaskEdit1 – поля для вводу супутньої інформації щодо виконуваної операції;
- StatusBar1 – панель для відображення допоміжної інформації;
- Query1..3 – набори даних для виконання операцій з базою даних;
- DataSource1 – джерело даних;
- IBTransaction1 – невізуальний компонент, який використовується для управління транзакціями.

Транзакція являє собою виконання послідовності операцій. При цьому можливі дві ситуації:

- Успішно завершені всі операції. В цьому випадку транзакція рахується успішною, і всі зміни в базі даних, які були проведені в

рамках транзакції окремими операціями, підтверджуються. В результаті БД перезходить з одного цілісного стану в інший.

- Невдало завершена хоча б одна операція. При цьому вся транзакція рахується неуспішною, і результати виконання всіх операцій (навіть успішно виконаних) відміняються. В результаті проходить повернення БД в стан, в якому вона знаходилася до початку транзакції.

При появі вікна на екрані в його процедурі-обронику події OnShow аналізується властивість Tag форми і в залежності від неї виконуються наступні дії, наприклад:

```
case Tag of
  1: begin
    Caption := 'Прихід товару';
    cbVid.Items := List5;  //Постачальники
    cbKudy.Items := List1; //Підрозділи
    cbKudy.ItemIndex := BaseF.cbMagazyn.ItemIndex;
    sb1.Action := BaseF.Postach;
    sb2.Action := BaseF.Tochka;
    ToolButton1.Enabled := true;
    ToolButton2.Enabled := true;
    end;
```

Так, якщо виконується операція прихід товару (властивість Tag містить значення 1), то встановлюється відповідний заголовок вікна (властивість форми Caption), властивість Items випадного списку cbVid заповнюється значеннями списку List5 – перелік постачальників, а ця ж властивість компоненту cbKudy заповнюєтся списком підрозділів List1, оскільки при поступленні товару рух здійснюється від постачальників до підрозділів підприємства. При внутрішньому переміщенні обидва компоненти заповнюються списками підрозділів, а при продажі – спсками підрозділів, у яких здійснюється продаж, і списками покупців (List4).

Для зручності біля компонентів класу TComboBox розміщуємо дві кнопки TSpeedButton, які дозволяють викликати вікна для перегляду і

редагування відповідних списків шляхом зміни таблиць POSTACH, MAGAZYN і POCUPETS бази даних. При появі на екрані вікна AddF властивостям Action цих кнопок присвоюються імена відповідних об’єктів TAction – BaseF.Postach, BaseF.Tochka або BaseF.Pocupets.

Далі значенню властивості Date компонента Dtp1 присвоюється сьогоднішня дата:

dtp1.Date := Date;

після чого виконується початок транзакції:

IBTransaction1.StartTransaction;

Для здійснення операцій з товаром використовуємо таблицю TEMP бази даних. Наприклад, при поступленні на підприємство нового товару у цю таблицю заноситься уся необхідна інформація про нові товари, при закритті вікна ця інформація розподіляється між іншими таблицями БД, після чого робиться очищення таблиці TEMP і підтвердження транзакції.

Для внесення інформації в таблицю TEMP при поступленні нового товару використовується вікно TovarF, вигляд якого показано на рис. 3.13.

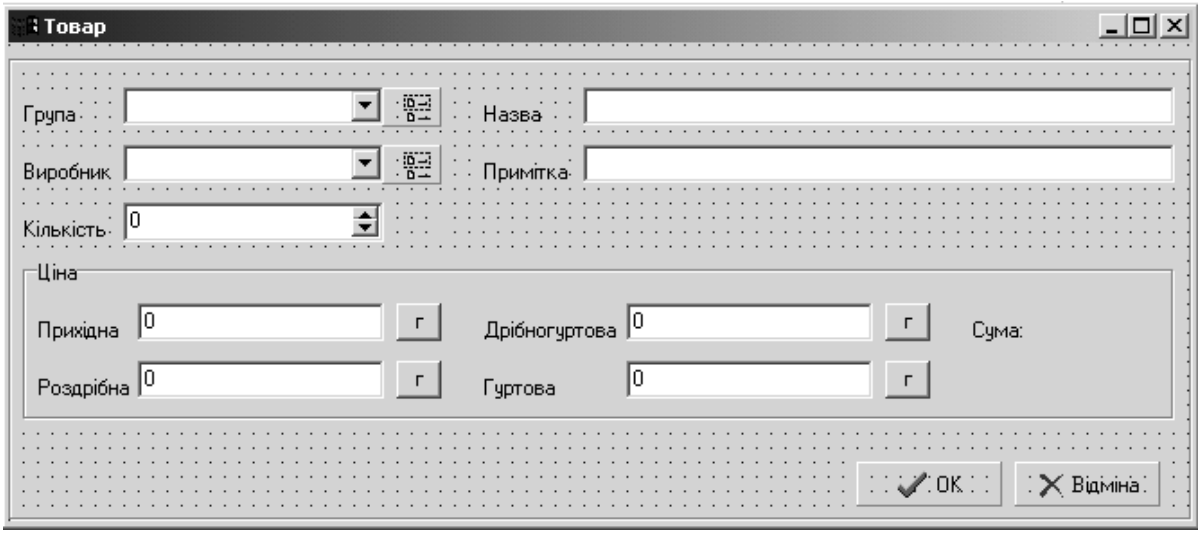

Рисунок 3.13. Вигляд вікна TovarF на етапі проектування.

Виклик вікна TovarF в модальному режимі здійснюється кнопкою ToolButton1 інструментальної панелі вікна AddF:

```
TovarF.ShowModal;
```

На формі розміщені компоненти для вводу даних про товар. Для вооду назви і додаткової інформації про товар служать компоненти Edit1 і Edit2, для вибору групи товарів і виробників використовуємо випадні списки cb1 і cb2 класу TComboBox, кількість товару вводимо за допомогою компоненту SpinEdit1, ввід даних про ціну товару виконується за допомогою компонентів Edit3..6, а зміна грошових одиниць виконується кнопками Button1 … Button4.

При зміні значення кількості товару, яке міститься у властивості Value компоненту SpinEdit1 автоматично змінюється заголовок статичного напису Label10, який показує суму товару, що рівна добутку кількості товару і прихідної ціни. Це досягається за допомогою наступного коду, розміщеного в обробнику події OnChange компонента, що виникає при зміні властивості Value:

```
try
Label10.Caption := 'Сума:  ' + IntToStr(SpinEdit1.Value *
                   StrToInt(Edit3.Text)) + '  ' + Button1.Caption;
except end;
```

У властивості Caption кнопок Button1..4 міститься символ "г" або "$", який визначає грошову одиницю – гривню або американський долар. Зміна цієї властивості при виконанні програми (зміна грошової одиниці) виконується натисненням відповідної кнопки:

```
procedure TTovarF.Button2Click(Sender: TObject);
begin
if Button2.Caption = 'г' then Button2.Caption := '$' else Button2.Caption := 'г';
end;
```

На формі також розміщені кнопки "ОК" і "Відміна" класу TSpeedButton. При натисненні кнопки "Відміна" відбувається закривання вікна без збереження введеної інформації, проте перед цим видається відповідне попередження:

```
if (cb1.Text <> '') or (Edit1.Text <> '') then
   if MessageDlg('Закрити без збереження ?',mtWarning,[mbYes,mbNo],0) =
7 then exit;
Close;
```

Натисненням кнопки "ОК" після перевірки правильності введення нові дані підтверджуються. Перевірка введення текстової інформації здійснюється наступним чином:

```
if cb1.Text = '' then begin
   MessageDlg('Потрібно вказати групу товару ',mtWarning,[mbOK],0);
   exit;   end;
```

Для перевірки правильності введення числової інформації використовуємо захищений блок try ... except … end. Порядок виконання операторів такий: спочатку виконуються оператори секції try … except, якщо оператори виконані без виникнення виключної ситуації, робота захищеного блоку на цьому припиняється, і управління отримує оператор, що стоїть після end. У випадку виконання частини try виникла виключна ситуація, управління отримує відповідний обробник в секції except. Таким чином, у секції try ... except робимо спробу перетворення введеного тексту у числовий тип:

```
try
  StrToFloat(Edit3.Text);
  except
  MessageDlg('Недобре введено прихідну ціну',mtError,[mbOK],0);
  end;
```

Якщо при перетворенні виникла помилка, то користувачу видається повідомлення. Після виконаних перевірок вікно TableF закривається і управління знову передається вікну AddF.

Компонент Query2, розміщений на формі AddF, служить для формування набору даних, які містяться в таблиці TEMP. Оскільки текст SQL-запиту при виконанні програми змінюватися не буде, то його можна внести у властивість SQL компоненту ща не етапі проектування. Він має такий вигляд:

```
SELECT TOVGRUP.NAME, TEMP.NAME, ZAVOD.NAME,
TEMP.NUMOF, TEMP.PR, TEMP.M, TEMP.PRA, TEMP.MA, TEMP.PRB,
TEMP.MB, TEMP.PRC, TEMP.MC, TEMP.NOTE, TEMP.ID
FROM TOVGRUP, ZAVOD, TEMP
WHERE (TOVGRUP.CODE = TEMP.IDGROUP) AND (ZAVOD.CODE =
TEMP.IDZAVOD)
```

Оскільки назви груп товарів та виробників не містяться безпосередньо в таблиці TEMP, то в тексті запиту передбачаємо вибірку цих назв з таблиць відповідно TOVGRUP і ZAVOD. Таблиця TOVGRUP зв’язана з таблицею TEMP полями CODE і IDGROUP, а таблиці ZAVOD і TEMP – полями CODE і IDZAVOD. Ці зв’язки показуються в умові відбору WHERE.

Отже, якщо вікно TovarF було закрито кнопкою "ОК", то необхідно вставити нову інформацію в таблицю TEMP. Для цієї мети використовуємо набір даних Query1:

```
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add(Ins);
Query1.ExecSQL;
```

Текст SQL-запиту формується за допомогою функції Ins, у якій змінній s1 присвоюється код групи товарів, який вибирається із списку Lcod2:

```
s1 := Lcod2.Strings[TovarF.cb1.ItemIndex];
```

змінній s2 – назва товару:

```
s2 := #39+TovarF.Edit1.Text+#39;
```

а змінній s3 – код виробника товару із списку Lcod3:

```
s3 := Lcod3.Strings[TovarF.cb2.ItemIndex];
```

Після цього формується текст запиту:

```
Result := 'insert into temp (idgroup, name, idzavod, pr, m, pra, ma, prb, mb,
prc, mc, '+'note, numof) values ('+s1+', '+s2+', '+s3+', '+TovarF.Edit3.Text+',
'+#39+Tovarf.Button1.Caption+#39+', '+TovarF.Edit4.Text+',
'+#39+Tovarf.Button2.Caption+#39+', '+TovarF.Edit5.Text+',
'+#39+Tovarf.Button3.Caption+#39+', '+TovarF.Edit6.Text+',
```

```
'+#39+Tovarf.Button4.Caption+#39+', '+ #39+TovarF.Edit2.Text+#39+',
'+IntToStr(TovarF.SpinEdit1.Value)+')';
```

Після вставки в таблицю TEMP набір даних Query2 обновляється для візуального відображення внесених змін:

```
Query2.Close;
Query2.Open;
```

По замовчуванню для кожного фізичного поля при відкритті набору даних автоматично створюється об’єкт типу TFiled, а всі поля в наборі даних є динамічними і доступними. Для створення статичних (стійких) полів використовується спеціальний Редактор полів. У випадку, якщо хоча б одне поле набору даних є статичним, динамічні поля більше створюватися не будуть. Таким чином, в наборі даних будуть доступні тільки статичні поля, а всі інші рахуються відсутніми. Задати склад статичних полів можна за допомогою Редактора полів на етапі розробки програми.

Для запуску редактора полів з контекстного меню компонента Query2 вибираємо пункт Fields Editor, при цьому з’явиться вікно AddF.Query2, показане на рис. 3.14.

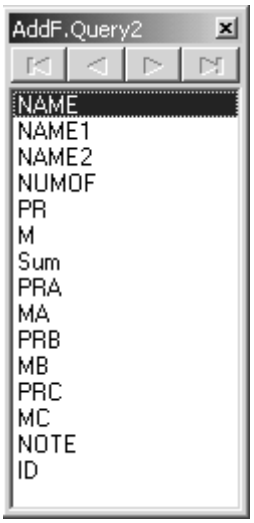

Рисунок 3.14 Вікно Редактора полів компонента Query2.

За допомогою Редактора полів можна виконати слідуючі операції:

- створити нове статичне поле;
- знищити статичне поле;
- змінити порядок розміщення статичних полів.

Крім того, для будь-якого вибраного в редакторі статичного поля з допомогою Інспектора об’єктів можна задавати або змінювати властивості цього поля (об’єкту типу TField) і визначення обробників його подій.

Для створення статичних полів викликаємо контекстне меню Редактора полів і вибираємо пункт Add Fields, в результаті чого з’явиться вікно з переліком усіх доступних полів. Після створення статичних полів для кожного з них задаємо ширину при відображенні в компоненті DBGrid1 – властивість DisplayWidth, та текст заголовку стовбця – властивість DisplayLabel.

У наборі даних Query2 також використовуємо обчислювальне статичне поле – поле суми, яке рівне добутку значень ціни товару та його кількості. Для виконнаня цих обчислень використовуємо обробник події OnCalcFields компоненту Query2, який має такий вигляд:

```
procedure TAddF.Query2CalcFields(DataSet: TDataSet);
begin
Query2sum.AsInteger := Query2pr.AsInteger * Query2numof.AsInteger;
end;
```

Для редагування записів таблиці TEMP бази даних також використовуємо вікно TovarF, яке в цьому випадку викликається кнопкою ToolButton2 інструментальної панелі вікна AddF. Перед викликом вікна TovarF у відповідні поля вводу в цьому вікні заносяться значення полів поточного запису набору даних Query2:

```
TovarF.Edit1.Text := Query2name1.AsString;
TovarF.Edit2.Text := Query2note.AsString;
TovarF.Edit3.Text := Query2pr.AsString;
TovarF.Edit4.Text := Query2pra.AsString;
TovarF.Edit5.Text := Query2prb.AsString;
TovarF.Edit6.Text := Query2prc.AsString;
TovarF.SpinEdit1.Value := Query2numof.AsInteger;
```

Також вибираються відповідні значення з випадних списків груп товарів та виробників:

```
for i := 0 to List2.Count-1 do
  if List2.Strings[i] = Query2name.AsString then TovarF.cb1.ItemIndex := i;
for i := 0 to List3.Count-1 do
  if List3.Strings[i] = Query2name2.AsString then TovarF.cb2.ItemIndex := i;
```

Аналогічно встановлюються значення Caption кнопок Button1 ... Button4, після чого вікно TovarF показується в модальному режимі. Якщо дане вікно буде закрито кнопкою "ОК", то спочатку витираємо з таблиці поточний запис, а тоді вставляємо новий запис, як при додаванні нового товару:

```
Query1.SQL.Clear;
Query1.SQL.Add('delete from temp where (id = '+Query2id.AsString+')');
Query1.ExecSQL;
Query1.SQL.Clear;
Query1.SQL.Add(Ins);
Query1.ExecSQL;
```

Для знищення записів з таблиці TEMP використовуємо кнопку ToolButton4 інструментальної панелі вікна AddF. Перед видаленням користувачу видається відповідне повідомлення, і у випадку його згоди формується і виконується SQL-запит:

```
if MessageDlg('Видалити запис ?',mtWarning,[mbYes,mbNo],0) = 6 then
  begin
  Query1.Close;
  Query1.SQL.Clear;
  Query1.SQL.Add('delete from temp where (code = '+Query2id.AsString+')');
  Query1.ExecSQL;
  end;
```

При виконанні операції переміщення товару між підрозділами або продажу товару необхідно передбачити можливість вибору товару з переліку наявної продукції у головному вікні програми. Вибір товару в цьому вікні

виконується подвійним клацанням миші на потрібному записі при відкритому вікні AddF. Для забезпечення виконання цих функцій в обробнику події OnDblClick компонента DBGrid1, розміщеного на формі BaseF, розміщуємо такий код:

```
if AddF.Visible and ((AddF.Tag = 2) or (AddF.Tag = 3)) then
                                                    AddF.TovarForMove;
```

Цей обробник викликається при подвійному клацанні мишею на компоненті. У ньому спочатку перевіряємо чи відкрите вікно AddF, тоді за допомогою властивості Tag цього вікна визначаємо тип операції, яка виконується в даний момент, після чого викликаємо процедуру TovarForMove модуля AddU.

У цій процедурі спочатку робиться перевірка, чи вибраний у головному вікні товар вже є у таблиці TEMP. Для цього компонентом Query3 формуєтьяс відповідний SQL-запит до БД:

```
Query3.SQL.Add('select numof from temp where (code = ' +
    BaseF.IBQuery1CODE.AsString + ')');
  Query3.Open;
```

Якщо після виконання цього запиту набір даних виявиться порожнім, що свідчить про відсутність даного товару в таблиці TEMP, то за допомогою статичного напису на формі TovarF відображується вся доступна кількість товару і для компонента SpinEdit1 встановлюється обмеження на максимальне введене число, рівне кількості товару:

```
if Query3.IsEmpty then  begin
    TovarF.Label11.Caption := '(доступно -
                                    '+BaseF.IBQuery1NUMOF.AsString+' )';
    TovarF.SpinEdit1.MaxValue := BaseF.IBQuery1NUMOF.AsInteger;
    j := 0                  end
```

Якщо ж набір даних не порожній, то, окрім описаних дій, змінній j типу integer присвоюється кількість вибраного товару у таблиці TEMP, після чого ця кількість порівнюється із загальною кількістю товару:

```
            else          begin
      j := Query3.Fields[0].AsInteger;
      if BaseF.IBQuery1NUMOF.AsInteger = j then
        ShowMessage('У даному підрозділі вибраного товару більше немає
              в наявності');
      exit;                    end;
```

Якщо виявиться, що кількість товару в таблиці TEMP рівна кількості товару у даному підрозділі, то після видачі відповідного повідомлення процедура завершує свою роботу.

Після виконання описаних дій, у поля вводу вікна TovarF вводяться дані про вибраний товар, як при редагуванні записів таблиці TEMP, крім того, всі ці поля робляться недоступними для виключення можливості зміни даних, що може призвести до суперечливості інформації у БД:

```
TovarF.cb2.Enabled := false;
TovarF.Edit1.Enabled := false;  ...
```

Далі у процедурі вікно TovarF показується в модальному режимі для того, щоб користувач ще раз міг переглянути характеристику товару і ввести його кількість. Після закриття цього вікна, якщо в таблиці TEMP знищується запис із даними про аналогічний товар, якщо такий запис був взагалі, після чого виконується вставка нового запису:

```
Query3.SQL.Clear;
Query3.SQL.Add('insert into temp (idgroup, name, idzavod, pr, m, pra, ma, '+
  'prb, mb, prc, mc, note, idtov) select idgroup, name, manuf, pr, m, pra, ma, '+
  'prb, mb, prc, mc, note, code from tovar where (code = ' +
  BaseF.IBQuery1IDTOV.AsString + ')');
 Query3.ExecSQL;
```

Оскільки після вставки цього запису поле кількості товару NUMOF у таблиці TEMP залишається порожнім, то виконуємо редагування щойно вставленого запису для внесення кількості товару:

```
Query3.SQL.Clear;
    Query3.SQL.Add('update temp set numof = ' +
      IntToStr(TovarF.SpinEdit1.Value + j) +
      ', code = ' + BaseF.IBQuery1CODE.AsString + ' where (idtov = ' +
      BaseF.IBQuery1IDTOV.AsString + ')');
    Query3.ExecSQL;
```

Тепер необхідно змінити дані таблиці BASE, оскільки частину товару переноситься у інший підрозділ або продається. Для цього змінній j знову присвоюємо кількість вибраного товару у таблиці TEMP:

```
Query3.SQL.Clear;
    Query3.SQL.Add('select numof from temp where (code = ' +
       BaseF.IBQuery1CODE.AsString + ')');
    Query3.Open;
    j := Query3.Fields[0].AsInteger;
```

Якщо виявиться, що число j рівне загальній кількості товару у даному підрозділі, то з таблиці BASE знищується запис із цим товаром, при цьому текст SQL-запиту буде мати вигляд:

```
'delete from base where (code = ' + BaseF.IBQuery1CODE.AsString + '
```

В іншому випадку редагуємо запис таблиці BASE запитом такого змісту:

```
'update base set numof = ' + IntToStr(BaseF.IBQuery1NUMOF.AsInteger -
                TovarF.SpinEdit1.Value) + ' where (code = ' +
                 BaseF.IBQuery1CODE.AsString + ')'.
```

На цьому процедура TovarForMove закінчує свою роботу.

Після закінчення формування переліку товарів, з якими проводиться операція поступлення, продажу або переміщення між підрозділами, вікно AddF можна закрити за допомогою кнопки ToolButton9, яка перед закриттям підтверджує поточну транзакцію, початок якої виконано при появі даного вікна

на екрані. Якщо ж вікно закрити будь-яким іншим методом, в тому числі й за допомогою кнопки ToolButton11 інструментальної панелі, транзакція підтверджена не буде і всі зміни, проведені із таблицями бази даних, будуть відмінені. Це досягається за допомогою коду, розміщеного в обробнику події OnClose форми AddF, який виникає при закритті вікна:

```
if IBTransaction1.InTransaction then IBTransaction1.Rollback;
BaseF.IBQuery1.Close;
```

Метод RollBack компоненту класу TIBTransaction відміняє поточну транзакцію, а метод Commit – підтверджує її. Перевірити, чи активна в даний момент транзакція, можна за допомогою логічної властивості InTransaction компоненту IBTransaction1. Таким чином, якщо транзакція активна, тобто кнопка "ОК" (ToolButton9) натиснута не була, всі зміни, зроблені з БД, відміняються.

При натисканні кнопки "ОК" відбуваються наступні дії. Спочатку формується набір даних з таблиці TEMP:

```
Query3.SQL.Clear;
Query3.SQL.Add('select * from temp');
Query3.Open;
```

Тоді у циклі з визначеною кількістю повторень опрацьовується кожен запис з набору:

```
for i := 1 to Query3.RecordCount do
  begin
  ...
```

Якщо відбувається прихід товару або переміщення товару між підрозділами підприємства, то необхідно вставити або змінити відповідні записи у таблицю BASE, при продажі товару цей крок можна пропустити, що здійснюємо з допомогою оператора переходу:

```
if Tag = 3 then goto mitka;
```

Для того, щоб виконувати операції з таблицею BASE, необхідно отримати код товару, з яким виконується операція. Якщо такий товар уже є на підприємстві, або був раніше, то у таблиці TOVAR будуть міститися дані про нього, в тому числі і його код. Для отримання цього коду використовуємо процедуру Seltovar:

```
AddF.Query1.SQL.Clear;
AddF.Query1.SQL.Add('select code from tovar
    where (idgroup = '+AddF.Query3.FieldByName('idgroup').AsString+
    ') and (name = '+#39+AddF.Query3.FieldByName('name').AsString+#39+
   .....
    ') and (note = '+#39+AddF.Query3.FieldByName('note').AsString+#39+')');
AddF.Query1.Open;
```

Якщо після виконання цієї процедури набір даних Query1 виявиться порожнім, що свідчить про відсутність даного товару, то виконуємо вставку запису у цю таблицю з даними про товар, після чого знову виконуємо процедуру Seltovar:

```
if Query1.IsEmpty then   begin
        Query1.Close;
        Query1.SQL.Clear;
        Query1.SQL.Add('insert into tovar (idgroup, name, manuf, pr, pra, prb,
          prc, '+'m, ma, mb, mc, note) select idgroup, name, idzavod, pr, pra, prb,
          prc, '+'m, ma, mb, mc, note from temp where (id =
         +Query3.FieldByName('id').AsString+')');
        Query1.ExecSQL;
        Seltovar;    end;
```

Після виконаних дій отримаємо набір даних Query1, у якому міститься одне поле і один запис, у якому і міститься потрібний код товару. Цей код ми присвоюємо цілочисельній змінній id:

```
id := Query1.Fields[0].AsInteger;
```

Тепер необхідно перевірити, чи є товар з даним кодом у підрозділі, куди цей товар постуає. Для цієї мети використовуємо процедуру Selbase:

```
procedure Selbase;
begin
AddF.Query1.Close;
AddF.Query1.SQL.Clear;
AddF.Query1.SQL.Add('select code, numof from base where (idmag = '+
 Lcod1.Strings[AddF.cbKudy.ItemIndex]+') and (idtov = '+IntToStr(id)+')');
AddF.Query1.Open;
end;
```

У цій процедурі формується набір даних Query1 шляхом вибірки з таблиці BASE записів з кодом поточного підрозділу та з кодом даного товару. У випадку, якщо після виконання процедури Selbase набір виявиться порожнім, виконуємо вставку нового запису в цю таблицю:

```
if Query1.IsEmpty then   begin
        Query1.Close;
        Query1.SQL.Clear;
        Query1.SQL.Add('insert into base (idmag, idtov, numof, den) values ('+
          Lcod1.Strings[cbKudy.ItemIndex]+', '+IntToStr(id)+', '+
          Query3.FieldByName('numof').AsString+',
'+#39+DateToStr(dtp1.Date)+#39+')');
        Query1.ExecSQL; end
```

Якщо ж набір даних Query1 містить якісь записи, тобто якщо у підрозділі уже є товар з таким кодом, то лише змінюємо його кількість:

```
else   begin
        j := Query1.FieldByName('code').AsInteger;
        k := Query1.FieldByName('numof').AsInteger;
        Query1.Close;
        Query1.SQL.Clear;
        Query1.SQL.Add('update base set numof =
          '+IntToStr(k+Query3.FieldByName('numof').AsInteger)+
          ' where (code = '+IntToStr(j)+')');
        Query1.ExecSQL; end;
```

Для зміни кількості у змінній k запам’ятовуємо кількість наявного у підрозділі товару, після чого збільшуємо значення цієї змінної на кількість товару у таблиці TEMP і записуємо нове значення k у таблицю BASE.

Після описаних операторів розміщуємо мітку:

mitka:

Тобто, якщо виконується продаж товару, то програма відразу перейде до цього місця, оскільки усі необхідні зміни в базі даних при цій операції були зроблені ще до натискання кнопки "ОК".

Далі залишається лише зробити відмітку про виконану операцію, тобто вставити новий запис у таблицю MOVE:

```
Query1.Close;
Query1.SQL.Clear;
case Tag of
1: Query1.SQL.Add('insert into move (typ, vid, kudy, idtov, den) values (1, '+
    Lcod5.Strings[cbVid.ItemIndex]+', '+Lcod1.Strings[cbKudy.ItemIndex]+
    ', '+IntToStr(id)+', '+#39+DateToStr(dtp1.Date)+#39+')');
. . . . . .
4: Query1.SQL.Add('insert into move (typ, vid, kudy, idtov, den) values (4, '+
    Lcod4.Strings[cbVid.ItemIndex]+', '+Lcod1.Strings[cbKudy.ItemIndex]+
    ', '+IntToStr(id)+', '+#39+DateToStr(dtp1.Date)+#39+')');
       end;
Query1.ExecSQL;
```

Усі описані дії виконуються виконуються у циклі, тобто для кожного запису у таблиці TEMP. Після закінчення роботи циклу, очищуємо цю таблицю:

```
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('delete from temp');
Query1.ExecSQL;
```

Тоді підтверджуємо активну транзакцію, щоб зроблені з базою даних зміни вступили в силу, та закриваємо вікно AddF:

```
if IBTransaction1.InTransaction then IBTransaction1.Commit;
Close;
```

Після цього управління знову передається головному вікну програми.

3.3.8. Перегляд даних про рух товару

У програмі необхідно передбачити можливість перегляду таблиці руху товарів. Для цієї мети використовуємо вікно MoveF, загальний вигляд якого на етапі проектування зображено на рис. 3.15.

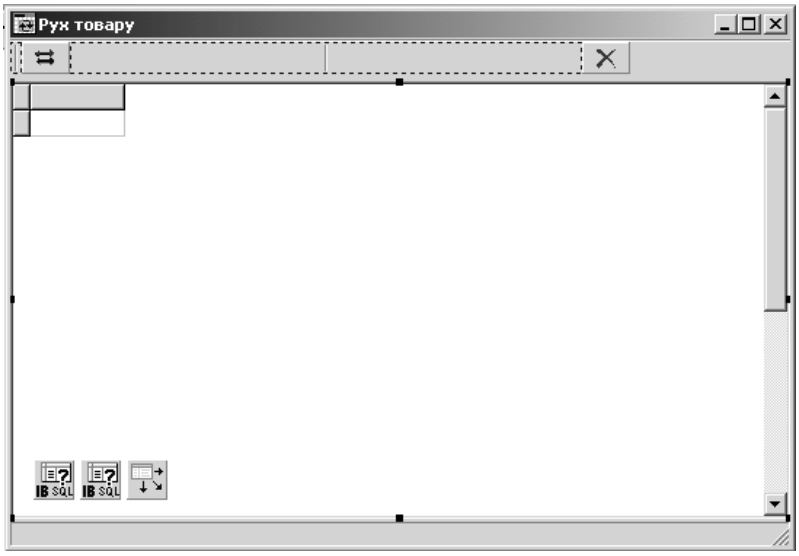

Рисунок 3.15 Вигляд вікна MoveF на етапі проектування.

На формі розміщений стандартний для подібних цілей набір компонентів: панель інструментів ToolBar1 з кнопками ToolButton1, панель стану StatusBar1, два набори даних класу TIBQuery, джерело даних DataSource1 та сітка DBGrid1.

3.3.9. Сортування записів у таблицях

Сортування записів набору даних виконується за допомогою команд меню Вид головного меню програми. Це меню дозволяє посортувати записи за часом надходження, за групами товарів або за виробником.

Сортування являє собою впорядкування записів по зростанню або спаданню значень полів. Список полів, по яких виконується сортування, вказується в операнді ORDER BY в тексті SQL-запиту. Порядок полів в цьому операнді визначає порядок сортування: спочатку записи впорядковуються по значеннях поля, вказаного в цьому списку першим, тоді записи, що мають одинакове значення першого поля, впорядковуються по другому полю і т.д.

Поля в списку позначаються іменами або номерами, які відповідають номерам в списку полів після слова SELECT.

По замовчуванню сортування проходить в порядку збільшення значень полів. Для встановлення аберненого порядку сортування по якому-небудь полю потрібно вказати після імені цього поля параметр DESC.

Таким чином, для сортування по полю виробника записів набору даних IBQuery1, які відображаються у головному вікні програми, необхідно до тексту SQL-запиту цього компонента (властивість SQL) дописати стрічку

ORDER BY ZAVOD.NAME,

після чого закрити і знову відкрити набір даних. Записи у ньому після повторного відкриття будуть відсортовані по полю виробника. Проте при спробі повторного сортування до тексту SQL-запиту знову допишеться така ж стрічка (з іншим іменем поля), тому перед сортуванням необхідно перевірити, чи міститься в тексті запиту операнд ORDER BY і, якщо такий операнд наявний, знищити його:

```
if IBQuery1.SQL.Strings[IBQuery1.SQL.Count - 2] = ' order by' then
  begin
  IBQuery1.SQL.Delete(IBQuery1.SQL.Count - 2);
  IBQuery1.SQL.Delete(IBQuery1.SQL.Count - 1);
  end;
```

Після чого можна виконувати сортування:

```
IBQuery1.Close;
IBQuery1.SQL.Add(' order by');
IBQuery1.SQL.Add(' zavod.name');
IBQuery1.Open;
```

Сортування записів по інших полях набору даних виконується аналогічно.

Таким чином, у розділі було поетапно описано роботу інформаційної системи автоматизації складського обліку взагальному, створення бази даних для цієї системи та робзробку клієнтської програми за допомогою середовища програмування Borland Delphi 6.

4 СПЕЦІАЛЬНА ЧАСТИНА

У даному дипломному проекті при розробці прикладних програм для інформаційної системи автоматизації складського обліку використовується середовище програмування Borland Delphi 6 Enterprise.

4.1 Інтегроване середовище розробки Delphi.

Створення прикладних програм Delphi виконується в інтегрованому середовищі розробки IDE (Integrated Development Environment). IDE служить для організації взаємодії з програмістом і включає в себе ряд вікон, що містять різноманітні керуючі елементи. За допомогою засобів інтегрованого середовища користувач може зручно проектувати інтерфейсну частину прикладної програми, а також писати програмний код і пов’язувати його з керуючими елементами. При цьому вся робота по створенню прикладної програми, включаючи від лагодження, проходить в інтегрованому середовищі розробки.

Інтегроване середовище розробки Delphi 6 являє собою багатовіконну систему. Вигляд інтегрованого середовища розробки (інтерфейс) може відрізнятися в залежності від налаштувань після завантаження інтерфейс Delphi 6 виглядає так, як показано на рис. 16 і включає п’ять вікон:

- головне вікно (Delphi 6 – Project 1);
- вікно Дерева об’єктів (Object Tree View);
- вікно Інспектора об’єктів (Object Inspector);
- вікно Конструктора форми (Form1);
- вікно Редактора коду (Unit1.pas);
- вікно Провідника коду (Exploring Unit1.pas).

На екрані, крім вказаних вікон, можуть бути присутні і інші вікна, що відображаються при виклику відповідних засобів, наприклад, Редактора зображень (Image Editor). Вікна Delphi можна переміщувати, змінювати їх розміри і забирати з екрану (крім головного вікна), а також з'єднувати між собою.

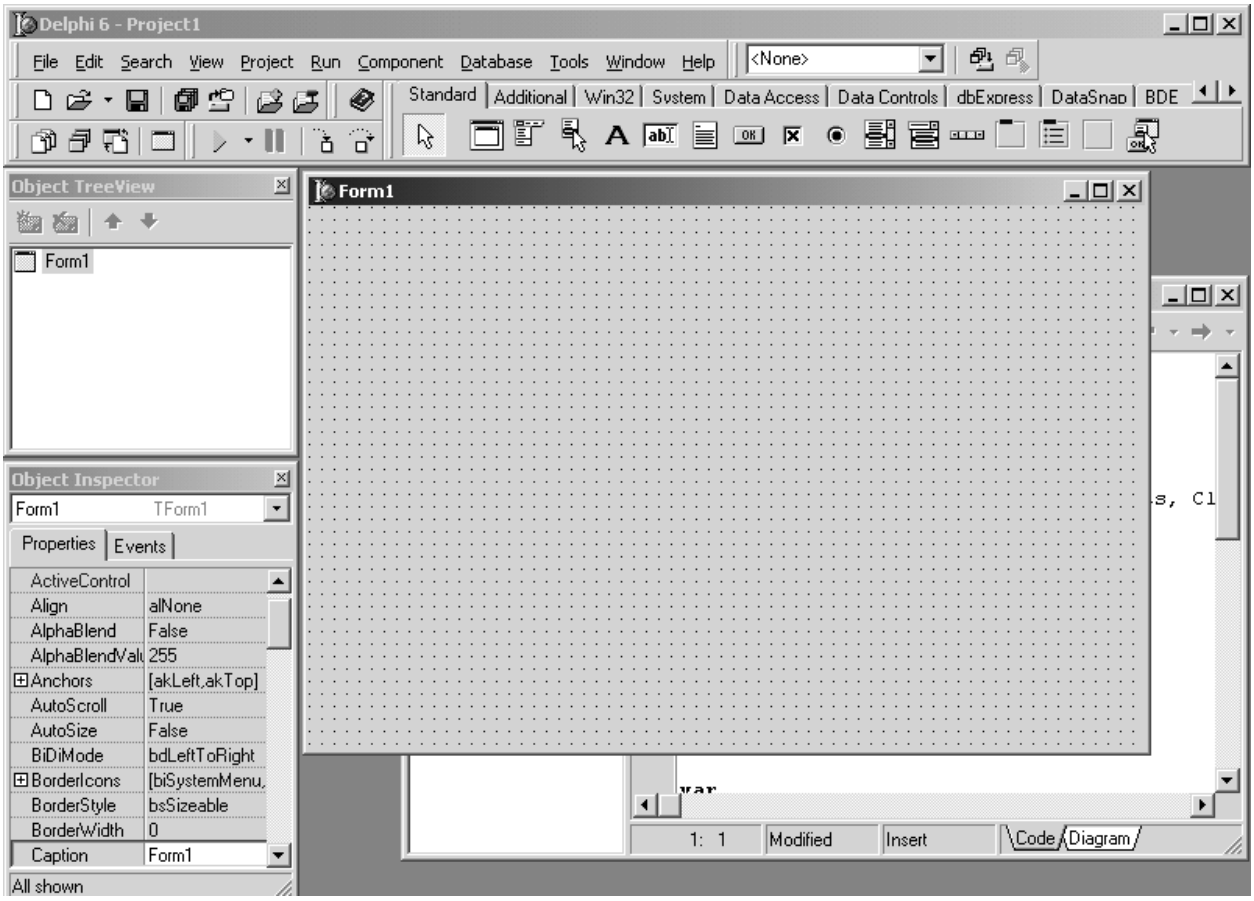

Рисунок 4.1 Вигляд інтегрованого середовища розробки.

Не дивлячись на наявність багатьох вікон, Delphi є однодокументним середовищем і дозволяє одночасно працювати лише з однією прикладною програмою (проектом). Назва проекту виводиться в стрічці заголовку головного вікна у верхній частині екрану.

При мінімізації головного вікна відбувається мінімізація всього інтерфейсу Delphi і, відповідно, всіх відкритих вікон, при закритті головного вікна робота з Delphi припиняється. Головне вікно Delphi включає:

- головне меню;

- панелі інструментів;
- Палітру компонентів.

4.2 Головне меню програми

Головне меню містить широкий набір команд для доступу до функцій Delphi.

Панелі інструментів знаходяться під головним меню в лівій частині головного вікна і містять п’ятнадцять кнопок для виклику найбільш часто використовуваних команд головного меню, наприклад, File/Open (Файл/Відкрити) або Run/Run (Виконання/Виконати).

Викликати багато команд головного меню можна також за допомогою комбінацій клавіш, що вказуються праворуч від назви відповідної команди. Наприклад, команду Run/Run можна викликати за допомогою клавіші <F9>, а команду View/Units (Перегляд/Модулі) – за допомогою комбінації клавіш <Ctrl>+<F12>.

Всього є шість панелей інструментів:

- Standart (Стандартна);
- View (Перегляд);
- Debug (Відлагодження);
- Custom (Користувацька);
- Desctop (Робочий стіл);
- Internet (Інтернет).

Відображенням панелей інструментів і налаштувань кнопок на них можна керувати. Ці дії виконуються за допомогою контекстного меню панелей інструментів. За допомогою контекстного меню можна також керувати видимістю Component Palette (Палітри компонентів).

Більш широкі можливості по налаштуванню панелей інструментів і головного меню надає показане на рис. 17 діалогове вікно Customize (Індивідуальні налаштування), яке викликається однойменною командою контекстного меню панелей інструментів. За його допомогою можна

приховувати або відображати потрібну панель інструментів, змінювати склад кнопок на ній, а також вибирати режим відображення підказок для кнопок.

Палітра компонентів знаходиться під головним меню в правій частині головного вікна і містить велику кількість компонентів, що розміщуються в створюваних формах. Компоненти являються свого роду будівельними блоками, з яких конструюються форми прикладної програми. Всі компоненти розбиті на групи, кожна з яких в Палітрі компонентів розміщується на окремій сторінці, а самі компоненти представлені піктограмами.

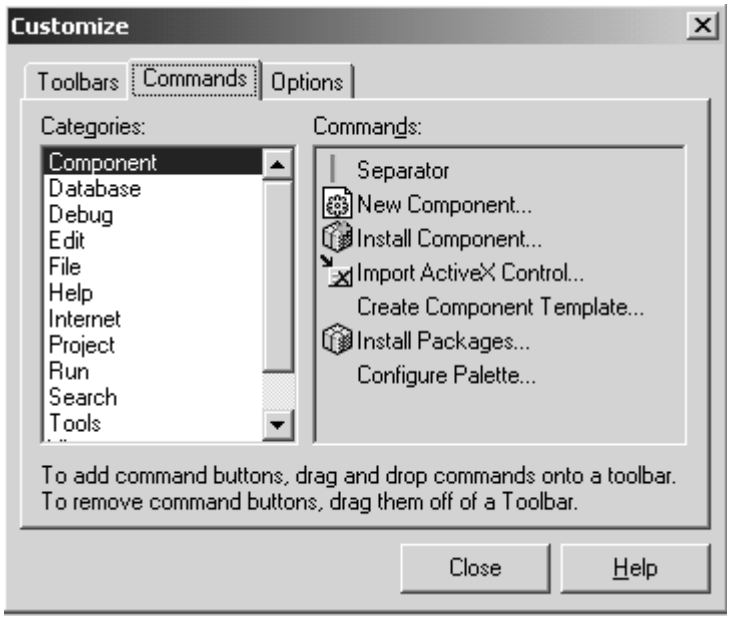

Рисунок 4.2 Діалогове вікно налаштувань панелей інструментів.

4.3 Вікно палітри компонентів.

Палітру компонентів можна налаштовувати за допомогою діалогового вікна Palette Properties (Властивості Палітри). Це вікно (рис. 18) викликається командою Properties (Властивості) контекстного меню Палітри компонентів або командою Components/Configure Palette (Компонент/Налаштування Палітри) головного меню. Вікно дозволяє виконувати такі операції, як знищення, додавання окремих компонентів і переміщення їх на інше місце, а також додавання, знищення або переміщення сторінок компонентів.

В лівому списку діалогового вікна Palette Properties містяться назви сторінок, в правому – назви компонентів вибраної сторінки. За допомогою кнопок, розміщених під списками, задаються наступні дії:

- додати (Add), знищити (Delete) або перейменувати (Rename) сторінку;
- переставити сторінку або компонент на позицію вище (Move Up) або нижче (Move Down);
- приховати компонент (Hide).

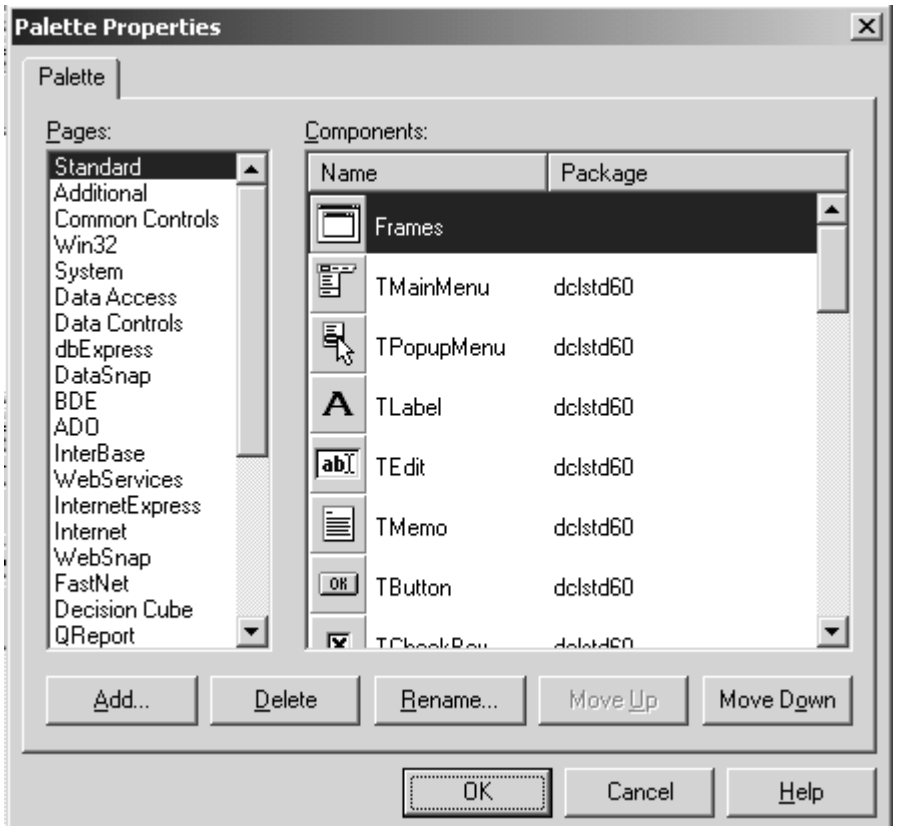

Рисунок 4.3 Діалогове вікно властивостей Палітри компонентів.

Вікно Конструктора форми спочатку знаходиться в центрі екрану і має заголовок Form1. В ньому виконується проектування форми, в процесі якого на форму з Палітри компонентів розміщуються необхідні компоненти. При цьому проектування полягає в візуальному конструюванню форми, а дії користувача схожі на роботу в середовищі простого графічного редактора. Сам Конструктор форми під час її проектування залишається ніби „за кадром”, і користувач має

справу безпосередньо з формою, тому часто вікно Конструктора також називають Вікном форми або Формою.

Вікно Редактора коду (заголовок Unit1.pas) після запуску системи програмування знаходиться під вікном Конструктора форми і майже повністю перекривається ним. Редактор коду являє собою звичайний текстовий редактор, з допомогою якого можна редагувати текст модуля і інші текстові файли прикладної програми, наприклад, файл проекту. Кожний редагований файл знаходиться у вікні Редактора на окремій сторінці, доступ до якої здійснюється клацанням на відповідному значку. На початку у вікні Редактора коду на сторінці Code міститься лише одна закладка Unit1 вихідного коду модуля форми Form1 розроблюваної прикладної програми.

В Delphi 6 редактор коду підтримує також перегляд і редагування інших елементів прикладної програми. Для цього використовуються сторінки:

- Diagram – відображення і налаштування взаємозв’язків між елементами баз даних;
- Preview – перегляд документа HTML у вікні броузера;
- HTML Script – перегляд документу HTML і тексту JavaScript, згенерованих за допомогою компонентів в модулі SitePageProducer;
- HTML Result – перегляд документу HTML, згенерованого на основі HTML-шаблону.

У вікні Редактора коду завжди присутня сторінка Code, а інші сторінки вимагають відповідного налаштування.

Перемикання між вікнами Конструктора форми і Редактора коду зручно виконувати з допомогою клавіші <F12>.

Вікно Провідника коду (Exploring Unit1.pas) розміщується ліворуч від вікна Редактора коду. В ньому у вигляді дерева відображаються всі об’єкти модуля форми, наприклад змінні і процедури (рис. 19). У вікні Провідника коду можна зручно переглядати об’єкти прикладної програми і швидко переходити до потрібних об’єктів, що особливо важливо для великих модулів. Виклик вікна Провідника коду виконується по команді Code Explorer меню View (Вид).

Для налаштування Провідника коду використовується показане на рис. 20 вікно Explorer Options (Параметри провідника), яке викликається командою Properties (Властивості) контекстного меню. За допомогою цього вікна

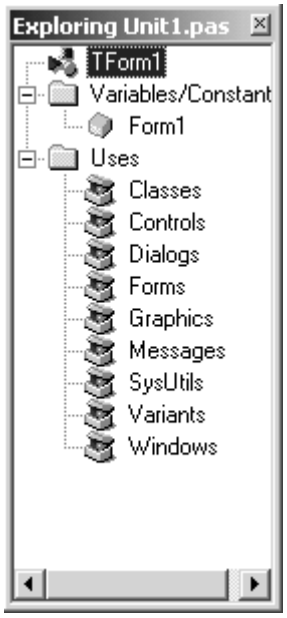

Рисунок 4.4 Вікно Провідника коду.

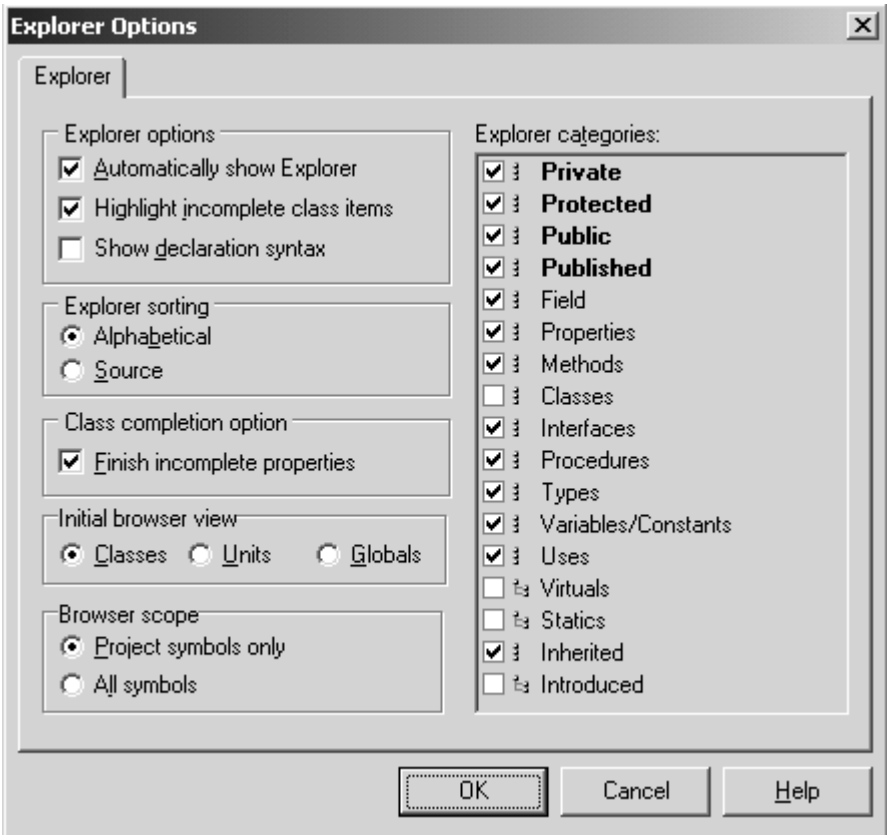

Рисунок 4.5 Вікно параметрів Провідника коду.

можна, наприклад, керувати відображенням об’єктів категорій, що містяться в списку Explorer categories (Категорії перегляду). Щоб вікно Провідника коду по замовчуванню було відсутнім, потрібно скинути прапорець Automatically show Explorer (Автоматично показувати Провідник).

Вікно дерева об’єктів після запуску системи знаходиться під Головним вікно і відображає дерево видну структуру об’єктів поточної форми. Це вікно зручно використовувати у випадку форм, що служать для обробки баз даних, так як воно дозволяє змінювати зв’язки між компонентами, наприклад, пере назначати таблиці джерело даних іншої таблиці.

Вікно інспектора об’єктів (рис. 4.6) знаходиться під вікном Дерева об’єктів в лівій частині екрану і відображає властивості і події об’єктів для поточної форми Form1. Його можна викликати на екран командою View/Object Inspector (Перегляд/Інспектор об’єктів) або натисненням клавіші <F11>.

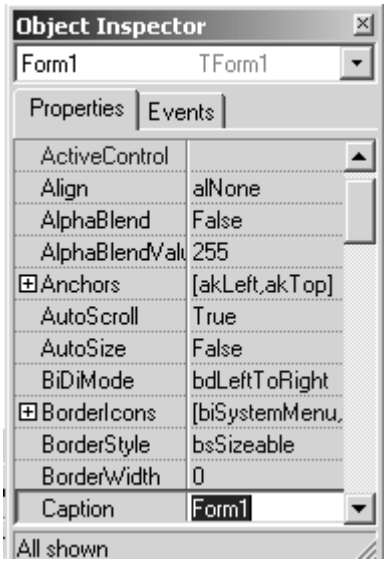

Рисунок 4.6 Вікно Інспектора об’єктів.

Вікно Інспектора об’єктів має дві сторінки: Properties (Властивості) і Events (Події).

Сторінка Properties відображає інформацію про поточний (вибраний) компонент у вікні Конструктора форми і при проектуванні форми дозволяє зручно змінювати властивості компонентів.

Сторінка Events визначає процедуру, яку компонент повинен виконати при виникненні вказаної події. Якщо для якої-небудь події така процедура задана, то в процесі виконання прикладної програми при виникненні цієї події процедура викликається автоматично.

В конкретний момент часу Інспектор об’єктів відображає властивості і події поточного (вибраного) компоненту, ім’я і тип якого відображаються в списку під заголовком вікна Інспектора об’єктів. Компонент, розміщений на формі, можна викликати клацанням миші на ньому або вибором у списку Інспектора об’єктів. Кожний компонент має свій набір властивостей і подій, які визначають його особливості.

Delphi підтримує технологію Dock-вікон, які можуть з’єднуватися одне з одним за допомогою миші.

5. НАУКОВО-ДОСЛІДНА ЧАСТИНА

5.1. Характеристика об’єкту дослідження

Досліджуваним об’єктом являється NGN мережа, її особливості та характеристики.

Мережа майбутнього покоління (NGN) - це мережа з пакетною комутацією, здатна забезпечити користувачів послугами, включаючи послуги зв'язку, і здатна використовувати колективну широкосмугову мережу, технології транспортування, основаних на QoS, в яких функції, пов'язані з послугами, не залежать від базових технологій транспортування. Вона дає користувачам необмежений доступ до різних послуг провайдерів. Вона підтримує узагальнену мобільність, яка дозволить постійно і повсюдно забезпечення послугами користувачів.

NGN (Мережа майбутнього покоління) характеризується наступними фундаментальними аспектами:

- Технологія з пакетною комутацією;
- Поділом контрольних функцій, виклик / сеанс і додаток / послуга;
- Поділ на віртуальні мережі та забезпечення відкритими інтерфейсами;
- Підтримкою широкого ряду послуг, програм та механізмів, заснованих на стандартних (уніфікованих) сервісно-конструюються блоках (включаючи послуги реального часу, потокові, не реального часу і мультимедійні послуги);
- Можливістю широкосмугового доступу з наскрізною передачею з заданою якістю QoS надаваних послуг і інваріантністю по відношенню до різних схем кодування;
- Забезпеченням між мережевого обміну і взаємодія з діючими мережами за допомогою відкритих інтерфейсів;
- Підтримкою мобільного зв'язку;
- Наданням користувачам необмеженого доступу до різних послуг;
- Різноманітністю схем ідентифікації в IP мережах;

- Уніфікованими характеристиками сервісних послуг для коректного використання її кінцевим користувачем
- Об'єднаними послугами між статичними і рухомими мережами;
- Незалежністю сервісних функцій від базових технологій транспортування;
- Відповідність всім чинним вимогам, що стосуються аварійного зв'язку, безпеки, приватності і т.д.

5.2. Програма і методика теоретичних і експериментальних досліджень.

Методика досліджень полягає у аналізі використовуваних NGN мережею протоколів, пошуку їхніх переваг та недоліків, та узагальнення отриманих даних. Теоретичні та експериментальні дослідження здійснюються шляхом використання програм емуляції протоколів. Результатом проведених досліджень являтиметься підбір протоколів функціонування що краще задовольнятимуть вимоги новоствореної мережі.

Протоколи, що використвуються в мережах NGN

MGCP - Протокол управління медіа шлюзами, служить для управління медіа шлюзами та надання доступу пакетним терміналам MGCP з боку гнучкого комутатора.

H.248/MEGACO - Протокол управління медіа шлюзами, служить для управління медіа шлюзами та надання доступу пакетним терміналам H.248 з боку гнучкого комутатора.

SIP - Протокол ініціювання сесій, служить для взаємозв'язку між гнучкими комутаторами або серверами додатків SIP, а також для доступу мультимедійних пакетних терміналів SIP.

SIP-T - Протокол розширення SIP, служить для прозорої передачі сигналізації ISUP.

SIP-I - Протокол SIP-I забезпечує розширення стандартного протоколу SIP для можливості транспортування повідомлень протоколу ISUP через мережу шляхом вставки (інкапсуляції) у повідомлення протоколу SIP.

BICC - Протокол BICC використовується для взаємодії гнучких комутаторів між собою і підтримує надання вже існуючих послуг ТМЗК / ISDN в пакетних мережах, а також підтримує взаємодію наявних вузлів комутації TDM за допомогою гнучких комутаторів і взаємодію вузлів комутації TDM через пакетну мережу на базі будь-яких транспортних технологій (IP, ATM або ін.).

H.323 - Протокол обробки викликів і мультимедійного зв'язку IP, служить для взаємозв'язку між гнучким комутатором і перетворювачем GK, шлюзом GW або пристроєм конференц зв'язку MCU у звичайній мережі H.323, а також для доступу мультимедійних пакетних терміналів H.323.

Сімейство протоколів SIGTRAN:

Протокол SCTP служить для надання послуги надійної передачі пакетних даних для протоколів адаптації сигналізації мережі з комутацією каналів SCN (Switched Circuit Network) на базі IP.

Протокол M2UA служить для взаємозв'язку між гнучким комутатором і медіа шлюзами з функціями вбудованого шлюзу сигналізації.

Протокол M3UA служить для взаємозв'язку між гнучким комутатором і шлюзами сигналізації.

Протокол V5UA служить для взаємозв'язку між гнучким комутатором і медіа шлюзами з функціями вбудованого шлюзу сигналізації, що підтримує інтерфейс V5.

Протокол IUA служить для взаємозв'язку між гнучким комутатором і медіа шлюзами з функціями вбудованого шлюзу сигналізації DSS1.

Сімейство протоколів SS7:

Протокол MTP служить для взаємозв'язку між гнучким комутатором і мережею сигналізації SS7, щоб гнучкий комутатор міг взаємодіяти з кінцевими пунктами SP або транзитними пунктами STP в мережі сигналізації SS7.

Протокол TUP служить для взаємодії між гнучким комутатором і мережею ТМЗК, щоб гнучкий комутатор міг підтримувати сполучні лінії TUP через транзитні медіа шлюзи TMG для взаємозв'язку зі станціями ТМЗК.

Протокол ISUP служить для взаємодії між гнучким комутатором і мережею ТМЗК, щоб гнучкий комутатор міг підтримувати сполучні лінії ISUP через шлюзи TMG для взаємозв'язку зі станціями ТМЗК.

Протокол SCCP служить для підтримки протоколу INAP, щоб гнучкий комутатор міг взаємодіяти з пунктами управління послугами SCP в інтелектуальній мережі IN через мережу сигналізації SS7.

Протокол TCAP служить для забезпечення додатків гнучкого комутатора і пунктів управління послугами SCP рядом функцій і процедур, що не залежать від додатків, щоб гнучкий комутатор міг підтримувати додатки, віднесених до послуг інтелектуальної мережі IN.

Протокол INAP служить для визначення інформаційних потоків між функціональними об'єктами інтелектуальної мережі IN, щоб гнучкий комутатор міг підтримувати функцію комутації послуг SSF (Service Switching Function), функцію управління викликами CCF (Call Control Function), функцію спеціалізованих ресурсів SRF (Specialized Resource Function) і функцію доступу до управління викликами CCAF (Call Control Access Function), а також діяти в якості вузла комутації послуг SSP в стандартній архітектурі інтелектуальної мережі IN.

DSS1 - Сигналізація абонентської мережі ISDN служить для взаємодії між гнучким комутатором і серверами NAS або станціями УПАТС, щоб гнучкий комутатор міг підтримувати інтерфейси первинного доступу (PRI; Primary Rate Interface) через медіа шлюзи.

V5 - Сигналізація абонентської мережі служить для взаємодії між гнучким комутатором і мережею доступу V5 або контролерами базових станцій, щоб гнучкий комутатор міг підтримувати інтерфейси V5.1/V5.2 через медіа шлюзи.

IPSec - Служить для забезпечення безпеки зв'язку між гнучким комутатором і шлюзами, що перебувають під її управлінням, такими, як медіа шлюзи, інтегровані пристрої доступу IAD.

SNMP - Служить для підтримки взаємозв'язку між гнучким комутатором і пристроями мережі.

FTP - Служить для підтримки взаємозв'язку між гнучким комутатором і білінговими центрами.

FTAM - Служить для підтримки взаємозв'язку між гнучким комутатором і білінговими центрами.

5.3. Обробка результатів досліджень

Далі проведені дослідження відкритих протоколів, управління мережею NGN, та їхнє порівняння.

Протоколи управління медіашлюзами

В якості протоколів управляння медіашлюзами може використовуватись Megaco/H.248, та MGCP.

H.248 призначений для вирішення проблем встановлення зв'язку між шлюзами (MG), які перетворюють комутацією телефонних каналів у пакетний трафік, і контролерами медіа-шлюзів (MGC). Media Gateway Controller (MGC), використовуючи H.248 вказує інструкції одному, або кільком шлюзам (MG) для підключення потоків, що надходять ззовні пакетних мереж даних у пакетні потоки, або потоки з використанням фіксованих комірок, такі як протокол передачі даних в реальному часі. [6]

H.248/Megaco завдяки його концепції Master – Slave не описує створення з’єднань між доменами, або між контролерами Media Gateway. H.248/Megaco використовується для зв'язку елементів включно, до шлюзів і не становить повної картини системи. Архітектура вимагає з’єднання точка-точка для зв'язку між контролерами Media Gateway.

Пристрій, який обробляє функції управління викликами управління позиціонується, як інтелектуальний Media Gateway Controller (MGC) пристрій,

який обробляє медіа - відносно не інтелектуальний Media Gateway (MG). H.248 визначає протоколи контролерів Media Gateway для управління медіа шлюзами аби підтримувати мультимедійні потоки через IP мережі та телефонної мережі загального користування (PSTN).

Завдяки типу пристроїв призначених для управління, та низькому рівню його керуючої структури, H.248 як правило, розглядається в якості доповнення до H.323 і SIP. У той час як Media Gateway Controller (MGC) буде використовувати H.248/Megaco для управління, та встановлення зв'язку з численними медіа-шлюзами (MG), інші протоколи, такі як SIP та H.323 використовуватимуться в одному Media Gateway Controller (MGC) для зв'язку з іншим контролером медіа-шлюзу (MGC). З точки зору SIP, поєднання MGC і MG, розглядаються разом, як шлюз SIP.

Архітектура, методології та програмні інтерфейси Media Gateway Control Protocol описані в RFC 2805. [8]

MGCP mater-slave протокол, який дозволяє пристроям управління викликами, таким як Call Agent взяти під контроль визначений порт Media Gateway. У контексті MGCP - Media Gateway Controller, є Call Agent. Це надає протоколу перевагу централізованого управління шлюзом і забезпечує в значній мірі масштабовані рішення IP-телефонії. Розподілена система складається з Call Agent, принаймні, одного Media Gateway (MG), який виконує перетворення сигналів середовища, і принаймні один шлюз сигналізації (SG) при підключенні до PSTN (перетворення з TDM голосового зв'язку у VoIP).

MGCP передбачає архітектуру управління викликами, у точках крайніх вузлів мережі (кінцеві точки, шлюзів), де обмежене інтелектуальне управління і присутній «інтелект» у ядрі Call Agent. MGCP припускає, що Call Agents, будуть синхронізуватися один з одним для відправки когерентних команд, та відповідати на запити шлюзфів під їх контролем.

Call Agent використовує MGCP щоб надати шлюзу інформацію, які події повинні бути повідомлені Call Agent, як повинні бути взаємопов'язані кінцеві точки, і які сигнали повинні бути активовані на кінцевих точках.

MGCP також дозволяє Call Agent перевіряти поточний стан кінцевих точок Media Gateway.

Медіа-шлюз використовує MGCP для повідомлення про події, що відносяться, до Call Agent.

Хоча будь-який шлюз сигналізації, як правило, на тому ж фізичному комутаторі, що і медіа-шлюз, насправді такої необхідності немає. Call Agent не використовує MGCP для управління шлюзом сигналізації, а швидше, SIGTRAN протоколи використовуються для транспортної сигналізації між шлюзами та Call Agent.

Табл. 5.1.Основні відмінності між Megaco/H.248 та MGCP.

Megaco/H.248	MGCP
Виклик здійснюється терміналами	Виклик представлений кінцевими пристроями
Виклики включають будь-яку комбінацію мультимедіа та конференцій	Виклики включать з'єднання точка-точка, та багато точкові з'єднання
Синтаксис текстовий, або двійковий	Синтаксис двійковий
Транспортний рівень TCP або UDP	Транспортний рівень UDP
Стандартний протокол для управління медіа шлюзом	IETF статус - інформаційний. MGCP не є інтернет стандартом жодного роду
Визначений стандарт ITU (сформований IETF і ITU)	Керований промисловістю. Багато компаній мають свої власні реалізації MGCP

Протокол Megaco використовує спрощений процес встановлення з'єднання. За допомогою емулятора протоколу Megaco проаналізую процес базового встановлення виклику. Опис дій пов'язаних із типом повідомлення представлені в табл. 5.2.

Табл. 5.2. Процес встановлення зв’язку Megaco

Індекс	Тип повідомлення	Опис дії
0	Отримано	Очікування поки ініціатор активує лінію
1	Надіслано	Відповідь до ініціатора
2	Надіслано	Надсилання DigitMap (XXX) до ініціатора
3	Отримано	Відповідь від ініціатора
4	Отримано	Очікування поки ініціатор набери 3 знаки
5	Надіслано	Відповідь до ініціатора
6	Надіслано	Запит SDP від ініціатора
7	Отримано	Отримано SDP від ініціатора
8	Надіслано	Надіслано запит SDP до терміналу
9	Отримано	Отримано SDP від терміналу
10	Надіслано	Надіслано віддалений SDP до ініціатора
11	Отримано	Відповідь від ініціатора
12	Отримано	Очікування поки термінал активує лінію
13	Надіслано	Відповідь до терміналу
14	Надіслано	Надання терміналу al/on
15	Отримано	Відповідь від терміналу
16	Надіслано	Надсилається SendReceive, Signals{} ініціатору
17	Отримано	Відповідь від ініціатора
18	Надіслано	Надсилається SendReceive до терміналу
19	Отримано	Відповідь від терміналу
20	Надіслано	Надіслано SendReceive до ініціатора
21	Отримано	Відповідь від ініціатора
22	Отримано	Очікування поки термінал активує лінію
23	Надіслано	Відповідь до терміналу
24	Надіслано	Надіслано Subtract до терміналу
25	Отримано	Відповідь від терміналу
26	Отримано	Очікування поки ініціатор активує лінію
27	Надіслано	Відповідь до ініціатора
28	Надіслано	Надіслано Subtract до ініціатора
29	Отримано	Відповідь від ініціатора
30	Надіслано	Оснащення всіх ліній al/off
31	Отримано	Відповідь від ініціатора

Сигнальні протоколи доступу:

До сигнальних протоколів доступу відносяться SIP та H.323.

SIP (англ. Session Initiation Protocol - протокол встановлення сеансу) - протокол передачі даних, який описує спосіб встановлення і завершення користувацького інтернет-сеансу, що включає обмін мультимедійним вмістом.

У моделі взаємодії відкритих систем SIP є мережевим протоколом прикладного рівня.

Протокол описує, яким чином клієнтська програма може запросити початок з'єднання в іншого, можливо, фізично віддаленого клієнта, що знаходиться в тій же мережі, використовуючи його унікальне ім'я. Протокол визначає спосіб узгодження між клієнтами про відкриття каналів обміну на основі інших протоколів, які можуть використовуватися для безпосередньої передачі інформації (наприклад, RTP). Допускається додавання або видалення таких каналів протягом встановленого сеансу, а також підключення та відключення додаткових клієнтів (тобто допускається участь в обміні більше двох сторін - конференц-зв'язок). Протокол також визначає порядок завершення сеансу. [9]

Рекомендація H.323 поділяє передачу даних на чотири складових, кожна з яких описана у відповідних додаткових рекомендаціях:

- Сигналізація - формує з'єднання і управляє його статусом, описує тип переданих даних.
- Управління потоковим мультимедіа - передача даних за допомогою транспортних протоколів реального часу (RTP).
- Додатки передачі даних (факсимільні сесії тощо) - передача в рамках відповідних стандартів, таких як T.120 і T.38.
- Комунікаційні інтерфейси - взаємодія пристроїв на фізичному, канальному, мережевому рівнях. [10]

Сигналізація H.323 грунтується на рекомендації Q.931, застосовуваної в ISDN. Найбільш поширені види сигналізації H.225.0 і H.245.

Табл. 5.4. Основні відмінності між SIP та H.323

SIP	**H.323**
Набір послуг що підтримують протоколи, приблизно однаковий	
Хороша підтримка мобільності	Мобільність підтримується, але менш гнучка
Зручна розширюваність, проста сумісність з попередніми версіями	Розширюваність присутня, але існує ряд складностей
Хороша підтримка масштабованості мережі	
Одна транзакція для встановлення зв’язку	Необхідно кілька транзакцій, для встановлення зв’язку
Простий протокол, текстовий формат повідомлень	Складний протокол, двійкове представлення повідомлень.
Сумісність практично відсутня	Майже повна сумісність, стандарти мають чіткий набір специфікацій

Табл. 5.5. Процес встановлення з’єднання за допомогою SIP протоколу

Індекс	Тип повідомлення	Опис дії
0	Надіслано	Від ініціатора надсилається SIP/SDP запрошення
1	Отримано	SIP сервер повертає повідомлення 100 Trying, Та надсилає запрошення кінцевому пристрою
2	Отримано	Ініціатор отримує повідомлення про прогрес сесії (Status: 183)
3	Отримано	Ініціатор отримує підтвердження запиту (Status: 200)
4	Надіслано	Ініціатор надсилає підтвердження
5	Двосторонній зв'язок	Встановлюється RTP/RTCP потік даних
6	Отримано	Кінцевий пристрій надсилає повідомлення розриву зв’язку (SIP: BYE)
7	Надіслано	Підтвердження розриву зв’язку (Status: 200 OK)

Протоколи сигналізації

Протокол Сигналізації SS7 (Signaling System 7) поділяє інформацію, необхідну для налаштування і управління з'єднанянми в комутованій телефонній мережі загального користування (PSTN) на окремі мережі з комутацією пакетів (мережі сигналізації). [11] Він використовує сигнальні одиниці повідомлень (MSUs), Стан лінії сигнальних одиниць (LSSUs) і сигнальних одиниць заповнення інформацією (FISUs) в якості сигнальних одиниць. Основні протоколи включають MTP (Message Transfer Part - Рівень 1 до 3), SCCP (Signaling Connection Control Part) і ISUP (ISDN користувацької частини).

В результаті впровадження нових послуг є попит на збільшення пропускної здатності мережі SS7. SS7 високої швидкості з'єднання (HSL) призначений виконувати цю вимогу, та використовує всю T1 (або E1) пропускну здатність, а не тільки 56 Кбіт/с або 64 Кбіт/с для проведення сигналізації SS7. HSL технологія може бути реалізована або за допомогою unchannelized HDLC (за допомогою MTP2), ATM (ATM рівень замінює MTP1 і 2), або SIGTRAN.

За допомогою SS7 (C7) Protocol Analyzer проаналізовано структуру повідомлень та стек SS7.

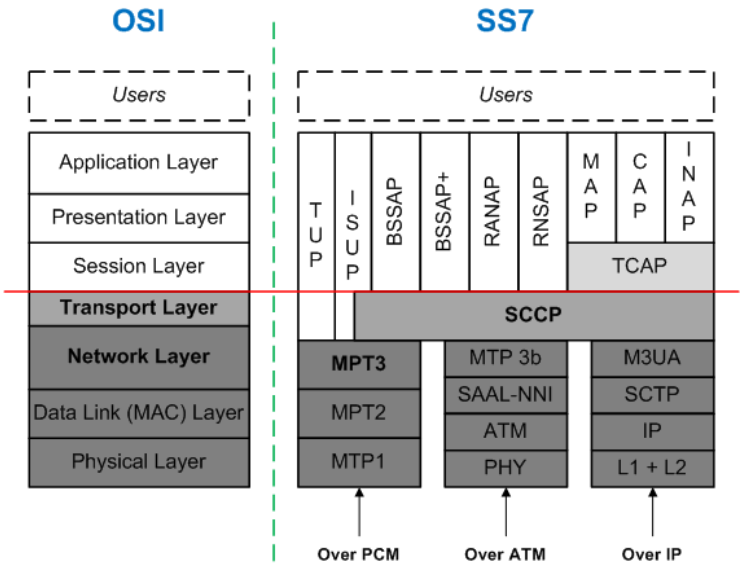

Рис.5.1. Загальна структура стеку моделі SS7 в порівнянні з OSI моделлю.

В межах мережі SS7 використовується три типи повідомлень або сигнальних юнітів:

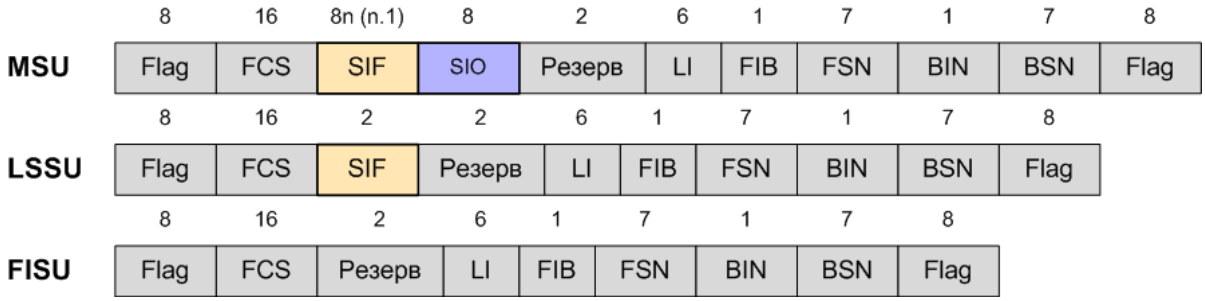

Рис.5.2. типи повідомлень SS7

SIGTRAN - це назва групи телекомунікаційних протоколів, створених для взаємодії традиційної телефонії та VoIP. Назва утворена від слів signaling і transport було дана робочою групою Internet Engineering Task Force (IETF), яка розробляє специфікації для сімейства протоколів, що забезпечують надійне обслуговування дейтаграм та адаптацію рівня користувача для Загальноканальна Системи № 7 (SS7) і ISDN. SIGTRAN протоколи - це розширення сім'ї протоколів SS7. Вони підтримують ті ж програми та парадигми управління викликами, як SS7, але використовують Інтернет-протокол (IP) для адресації і передаються по SCTP. [12]

5.4. Аналіз і узагальнення отриманої інформації

Концепція NGN не віддає перевагу якійсь одній мережевій структурі з використанням єдиного протокольного стека для усього розмаїття інфокомунікаційних послуг, у зв'язку з чим завжди існує необхідність забезпечення взаємодії програмно-апаратних компонент різних виробників, які підтримують протоколи та інтерфейси різних стандартів.

Результати досліджень протоколів мали на мені порівняти конкуруючі технології, та вибрати конкретне рішення для мережі АСУ верхнього рівня , що розробляється.

Для управління пристроями використовуватиметься протокол Megaco, зв’язано це з тим, що він здатен використовувати на траспортному рівні протокол TCP, що в свою чергу гарантуватиме доставку повідомлень в автоматизованій системі управління.

У якості сигнального протоколу використовуватиметься SIP. Відсутність сумісності компенсуватиметься, використанням протоколу по-усій розробленій мережі. Великою перевагою даного протоколу являється його незалежність стосовно транспортних технологій.

Необхідною і важливою в умовах еволюції мереж є можливість введення нових версій протоколів та забезпечення сумісності різних версій одного протоколу. Розширюваність (extensibility) протоколу забезпечується:

- узгодженням параметрів;
- стандартизацією кодеків;
- модульностью архітектури.

Протокол SIP досить просто забезпечує сумісність різних версій. Поля, що не зрозумілі обладнанню, просто ігноруються. Це зменшує складність протоколу, а також полегшує обробку повідомлень та впровадження нових послуг.

Рис.5.3. Структура повідомлень протоколу SIP

Протокол SIP використовує текстовий формат повідомлень, подібно протоколу HTTP. Це полегшує синтаксичний аналіз і генерацію коду, дозволяє реалізувати протокол на базі будь-якої мови програмування, полегшує експлуатаційне управління, дає можливість ручного введення деяких полів, полегшує аналіз повідомлень. Назва заголовків SIP-повідомлень ясно вказує їх призначення.

У разі необхідності, в організації IANA (Internet Assigned Numbers Authority) можуть бути зареєстровані нові заголовки. Для реєстрації в IANA відправляється запит з ім'ям заголовка і його призначенням. Назва заголовка вибирається таким чином, щоб вона говорила про його призначення. Зазначеним чином розробник може впроваджувати нові послуги.

Вихід за межі мережі здійснюватиметься залежно від обставин, та може включати уже існуючі лінії мереж, зокрема з використанням протоколів SS7, SIP, SIP-T, BICC

6 ОБҐРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

6.1 Економічне обґрунтування розробки та впровадження програми

6.1.1 Розрахунок витрат на розробку програмного забезпечення Сумарні витрати на розробку програмного забезпечення $S_{рп}$ визначаються за формулою:

$$S_{рп} = (1+\rho_н)\cdot\left\{\sum_i t_{роі}\cdot B_{ро}\cdot\left[(1+\omega_д)(1+\omega_с)+\omega_н\right]+t_{мо}\cdot e_г\right\}, \quad (6.1)$$

де $\rho_н$ – норматив рентабельності, що враховує прибуток установи, яка розробляє дану програму, долі одиниці;

$t_{роі}$ – час, що витрачається на розробку даної програми працівником *і-ої* кваліфікації, люд.-міс.;

$B_{роі}$ – основна заробітна плата розробника і-ої кваліфікації, грн/міс.;

$\omega_д$ – коефіцієнт, що враховує додаткову заробітну плату розробникам програми, в долях від основної заробітної плати;

$\omega_с$ – коефіцієнт, що враховує нарахування органам соціального захисту на заробітну плату, в долях від основної та додаткової заробітної плати;

$\omega_н$ – коефіцієнт, що враховує накладні витрати установи, в якій розробляється ця програма, в долях до основної заробітної плати розробника;

$t_{мо}$ – машинний час ЕОМ, необхідний для налагоджування даної програми, машино-год;

$e_г$ – експлуатаційні витрати, що припадають на 1 годину машинного часу.

Значення коефіцієнтів, як правило, рівні:

$\omega_д = 0{,}12$; $\omega_с = 0{,}35$; $\omega_н = 0{,}6$; $\rho_н = 0{,}14$.

Приймемо $t_{роі} = 2$ люд.-міс., а $B_{роі} = 350$ грн.

Експлуатаційні витрати, що припадають на 1 годину машинного часу можуть бути визначені за витратами електроенергії, оскільки програма налагоджувалась в дисплейному залі:

$$e_г = P_{сп} \cdot C_{год}, \tag{6.2}$$

де $P_{сп}$ - споживана потужність ЕОМ, Вт;

$C_{год}$ - вартість 1 кВт/год електроенергії.

Оскільки $P_{сп} = 0{,}045$ кВт, $C_{год} = 0{,}18$ грн., то згідно (5.2):

$e_г = 0{,}045 \cdot 0{,}18 = 0{,}0081$ *грн/год.*

Необхідний час налагодження програми становить 30 машино-годин.

Отже, сумарні витрати на розробку програмного забезпечення згідно (5.1) становитимуть:

$$S_{рп} = (1 + 0{,}14) \cdot (2 \cdot 350 \cdot ((1 + 0{,}12) \cdot (1 + 0{,}35) + 0{,}6) + 30 \cdot 0{,}0081) = 1685{,}65 \text{ грн}$$

Вартість розробленої програми обчислюється наступним чином:

$$z_n = \frac{S_{рп}}{n_n}, \tag{6.3}$$

де z_n - вартість, за якою продається програма, грн.,

n_n — кількість установ, що придбають дану програму. На даний час зацікавленість програмою, яка розробляється проявили дві організації (управління по справам цивільної оборони міста Кам’янець-Подільського та приватна фірма “ОІЮМ”), отже $n_n = 2$.

Отже, згідно (5.3):

$z_n = 842{,}82$ *грн.*

6.1.2 Розрахунок капітальних вкладень

Додаткові капітальні вкладення $\Delta K_{д2/1}$, пов'язані з впровадженням розробленої системи визначаються за наступною формулою:

$$\Delta K_{д2/1} = K_{ЕОМ} \frac{T_{м.в2}}{T_{пол}} + z_n, \tag{6.4}$$

де $K_{ЕОМ}$ - капітальні вкладення в ЕОМ та інші складові системи;

$T_{м.в2}$ - машинний час ЕОМ необхідний користувачу для тих задач, які він розв'язує за допомогою розробленої програми машино-год/рік;

$T_{пол}$ - корисний річний фонд роботи цієї ЕОМ (без врахування простоїв в ремонті);

z_n — ціна нової програми, грн.

Капітальні вкладення в ЕОМ та інші складові системи визначаються за наступною формулою:

$$K_{ЕОМ} = \sum_i C_{сер.і} + \sum_i C_{ком.і}, \tag{6.5}$$

де $C_{сер.і}$ - вартість обладнання, грн;

$C_{ком.і}$ - вартість комунікацій, грн. Вартість основного обладнання (грн.):

- Комп'ютер Pentium-III 500, HDD 20 Gb, 128MB RAM, FDD 3,5”, 15”SVGA Samtron – 55E;

$C_{сер.і} = 2500$ грн.

Вартість комунікацій

$C_{ком.і} = 240$ *грн.*

Значення капітальних вкладень становить:

$K_{ЕОМ} = 2500 + 240 = 2740$ грн.

Корисний річний фонд роботи ЕОМ визначається за наступною формулою:

$$T_{пол} = Д_р \cdot (1 - k_n) \cdot t_р, \tag{6.6}$$

де $Д_р$ - дійсний річний фонд часу, дні;

k_n — коефіцієнт, що враховує профілактичні роботи та плановий ремонт ($k_n = 0{,}1$);

$t_р$ - тривалість робочої зміни, год ($t_р = 8$).

Тоді корисний фонд часу ЕОМ згідно (5.6) становить:

$T_{пол} = 364 \cdot (1 - 0{,}1) \cdot 8 = 2620{,}8$ *год/рік.*

Машинний час ЕОМ необхідний користувачу для вирішення задачі з допомогою ЕОМ обчислюється за наступною формулою:

$$T_{м.в2} = Д_р \cdot (1 - k_п) \cdot t_з, \tag{6.7}$$

де $t_з$ - час, який витрачає користувач на вирішення задачі на ЕОМ, год.

Отже, машинний час ЕОМ становить:

$T_{м.в2} = 364 \cdot (1 - 0{,}1) \cdot 1 = 328$ *год/рік.*

Додаткові капітальні вкладення згідно (5.4) становлять:

$\Delta K_{д2/1} =$ 1185,73 грн.

6.1.3 Розрахунок експлуатаційних витрат

Економія витрат пов'язаних з експлуатацією програми $\Delta E_{е2/1}$ визначається за формулою:

$$\Delta E_{е2/1} = (1 + \omega_с) \cdot (1 + \omega_д) \cdot \sum_i B_{оі} - (T_{м.в2} \cdot e_г + \frac{z_п}{T_с}), \tag{6.8}$$

де $B_{оі}$ — основна заробітна плата і-того робітника, який розв'язував цю задачу вручну, грн/рік; $i = 2$, $B_{оі} = 3600$;

$T_с$ - термін служби програми, роки.

Приймемо $T_с = 5$ років.

Економія експлуатаційних витрат згідно (5.8) становить:

$\Delta E_{е2/1} = (1 + 0{,}35) \cdot (1 + 0{,}12) \cdot 3600 \cdot 2 - (328 \cdot 0{,}0081 + 1685{,}65 / 5) =$ *10643,96 грн.*

6.1.4 Розрахунок зведених економічних показників

Термін окупності додаткових капітальних вкладень визначається за наступною формулою:

$$\tau_{2/1} = \frac{\Delta K_{д2/1}}{\Delta E_{е2/1}}. \tag{6.9}$$

Отже, згідно (5.9) отримаємо:

$\tau_{2/1} =$ 1185,73/*10643,96 = 0,11 роки.*

Грошовий річний ефект, який отримує користувач при застосуванні системи визначається за формулою:

$$\Delta W_{p.e.2/1} = \Delta E_{e2/1} - \varepsilon_н \cdot \Delta K_{д2/1}, \quad (6.10)$$

де

$$\varepsilon_н = \frac{1}{\tau_н}.$$

При $\tau_{2/1} < \tau_н = 2{,}5$ року використання розробленого алгоритму є економічно ефективним. Тому $\varepsilon_н = 1/2{,}5.$

Тоді грошовий річний ефект згідно (5.10) становитиме:

$\Delta W_{p.e.2/1} =$ *10643,96* - 1185,73/2,5 = 10107,06 *грн.*

В таблиці 5.1 перелічені зведені економічні показники проектованого програмного забезпечення.

Таблиця 6.1 - Зведені економічні показники розробки

Показник	Розмірність	Значення
Витрати на розробку програмного забезпечення	грн.	1387,47
Капітальні вкладення	грн.	689,78
Економія експлуатаційних витрат	грн/рік	10643,96
Термін окупності	Рік	0,11
Річний грошовий економічний ефект	грн./рік	10107,06

З вищенаведених розрахунків видно, що розробка та впровадження даної системи є економічно доцільним.

6.2 Побудова сіткового графа

Порядок побудови сіткових графіків визначається прийнятою технологією і організацією робіт. Сіткові графіки тільки відображають існуючу або проектовану черговість і взаємозв'язок виконання робіт.

Всі вихідні дані для розробки сіткового графіку представлені в таблицях робіт і подій.

Таблиця 6.2 - Перелік подій мережевого графа

	Подія
0.	Отримання технічного завдання на дипломне проектування
1.	Ознайомлення з завданням дипломного проекту
2.	Ознайомлення з технічною літературою
3.	Аналіз підібраного матеріалу
4.	Встановлення програмного забезпечення
5.	Розгляд методів розв’язку задачі
6.	Розгляд програмних засобів, що дозволяють реалізувати дану задачу
7.	Вибір і обгрунтування методу розв’язку задачі
8.	Вибір мови програмування для реалізації завдання
9.	Створена база даних
10.	Створені SQL запити
11.	Визначення програмних модулів
12.	Написання програми
13.	Відлагодження усіх модулі програми
14.	Тестування програми
15.	Виправлення помилок
16.	Аналіз отриманих результатів
17.	Написання пояснювальної записки
18.	Завершення роботи

В таблиці 6.3 представлено види робіт та їх тривалості.

Таблиця 6.3 — Види робіт та їх тривалість

№	Роботи	Тривалість
0-1	Отримання технічного завдання	1
1-2	Ознайомлення з завданням дипломного проекту	4
2-3	Ознайомлення з технічною літературою	11
3-4	Аналіз підібраного матеріалу	10
	Встановлення програмного забезпечення	
3-5	Розгляд методів розв’язку задачі	2
4-7	Аналіз точності розглянутих методів	3
5-6	Розгляд програмних засобів, що дозволяють реалізувати дану задачу	1
7-8	Вибір і обгрунтування методу розв’язку задачі	6
6-7	Вибір мови програмування для реалізації завдання	3
8-9	Створення загальної блок-схеми алгоритму	4
9-10	Визначення програмних модулів	5
9-14	Написання програми	9
10-11	Відлагодження усіх модулів програми	5
11-12	Тестування програми	3
12-13	Виправлення помилок	4
13-15	Аналіз отриманих результатів	4
14-15	Написання пояснювальної записки	10

Сітковий граф представлений в додатку А.

6.3 Розрахунок параметрів сіткового графіка

Для отриманого сіткового графіка знаходимо критичний шлях і розраховуємо ранній час, пізній час і резерв часу подій.

Оскільки в нашому випадку сітковий графік має розгалужені ділянки, тому для знаходження критичного шляху знайдемо максимальні тривалості процесів на розгалужених ділянках:

$L_{0\text{-}1\text{-}2\text{-}3\text{-}4\text{-}7\text{-}8\text{-}9\text{-}10\text{-}11\text{-}12\text{-}13\text{-}15}$ = 60 (днів),

$L_{0\text{-}1\text{-}2\text{-}3\text{-}5\text{-}7\text{-}8\text{-}9\text{-}10\text{-}11\text{-}12\text{-}13\text{-}15}$ = 55 (днів),

$L_{0\text{-}1\text{-}2\text{-}3\text{-}4\text{-}7\text{-}8\text{-}9\text{-}14\text{-}15}$ = 57 (днів),

$L_{0\text{-}1\text{-}2\text{-}3\text{-}5\text{-}6\text{-}7\text{-}8\text{-}9\text{-}14\text{-}15}$ =52 (днів).

Отже, критичний шлях буде рівний: 60

У випадку, коли директивний термін $T_{дир}$ виконання комплексу робіт є менш тривалий, ніж критичний шлях $T_{кр}$, виникають проблеми. Вони полягають у тому, що необхідно здійснити прискорене виконання комплексу робіт на величину

$$\Delta T_к = T_{дир} - T_{кр} \tag{6.11}$$

Покриття дефіциту часу можна здійснити за рахунок прискорення вибраних критичних робіт.

6.4 Висновки

В результаті проведення економічного обгрунтування розробки та впровадження програми було визначено такі економічні показники, як витрати на розробку програмного забезпечення, капітальні вкладення, економія експлуатаційних витрат, термін окупності спроектованого програмного забезпечення та річний грошовий економічний ефект від впровадження програмного забезпечення.

Аналіз розрахованих значень цих показників дає можливість зробитивисновок, що впровадження даної програми є економічно доцільним.

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

7.1. Вплив шуму на організм людини і розроблення заходів по зниженню рівня шуму.

Деякі ВДТ є потенційними джерелами цілого ряду звуків, що містять як коливання, які можна почути, так і коливання ультразвукового діапазону. Цей шум справляє негативний вплив на функціональний стан користувачів.

Відомо, що шум несприятливо діє на людину, особливо при тривалому впливі. У користувача, діяльність якого пов'язана з переробкою інформації, що часто супроводжується елементами творчості, це виражається у зниженні розумової працездатності (наприклад, швидкість обробки тексту зменшується на 10— 15%, зростає кількість помилок), у прискоренні розвитку зорового втомлення, зміні відчуття кольорів, підвищенні витрати енергії (на 17%), появі головного болю, розвитку безсоння, послабленні уваги та ін.

Шум може бути фактором, що сприяє розвитку стресу. Відзначено взаємозв'язок між скаргами на шум від ВДТ, з одного боку, та емоційними порушеннями і поганим настроєм — з другого. Вплив шуму на вегетативну нервову систему може проявлятися при рівнях, близьких до припустимого, і призводити до порушення периферійного кровообігу за рахунок спазму капілярів шкіри та слизових оболонок, а також до інших негативних наслідків.

Для вимірювання шуму застосовують різні шумо-вимірювачі, частотні аналізатори та інші прилади. Частотні аналізатори служать для виділення будь-якої смуги частот для подальшої направленої корекції шуму як за об'єктивними показниками, так і згідно з суб'єктивним сприйняттям користувача. Вимірювання шуму на робочих місцях здійснюється згідно з ГОСТ 12.1.050-86 та ГОСТ 23941-79.

Нормованими параметрами шуму на робочих місцях є рівні середньоквадратичних звукових тисків (дБ) та рівні звуку (дБА), що вимірюються по шкалі «А» шумовимірювача. Останні найбільш близькі до фізіологічного сприйняття людиною.

Згідно з ГОСТ 12.1.003-83 шум у приміщенні, де виконують роботу, пов'язану з виробленням концепцій, створенням нових програм, викладацькою роботою, творчістю, не повинен перевищувати 40 дБА. Праця керівників виробництва, пов'язана з контролем групи людей, що виконують переважно розумову роботу, не повинна супроводжуватися шумом вище 50 дБА. Висококваліфікована розумова робота, що вимагає зосередженості, може проводитись у приміщеннях, де рівень шуму не перевищує 55дБА. Під час виконання розумової роботи за особистим графіком з інструкцією (операторська та близькі до неї види діяльності) і точних зорових робіт рівень шуму не повинен перевищувати 65 дБА.

Сумарний вплив численних джерел шуму у приміщенні у результаті багаторазового відбиття звукових хвиль може значно перевищити енергію прямого звуку від тих же джерел. Шум від окремих приладів не повинен перевищувати фоновий більше ніж на 5 дБ.

Найчастіше рівні акустичного випромінювання, які виходять від ВДТ, охоплюють діапазон частот від 6,3 до 40 кГц. Домінуючими є частоти від 16 до 40 кГц, пов'язані з частотою горизонтальної розгортки. Шум, можливо, виникає у осерді перетворювача горизонтальної розгортки. Не виключено, що всередині ВДТ існують вторинні джерела шуму.

Рівні звукового тиску на відстані приблизно 50 см від багатьох ВДТ у напрямі максимуму випромінювання знаходяться у межах від 30 до 68 дБ (середнє значення — 51 дБ). В діапазоні 16 - 20 кГц максимальний зареєстрований рівень склав 61 дБ (середнє значення 53 дБ).

Основними заходами боротьби з шумом є усунення або ослаблення причин шуму в самому його джерелі у процесі проектування, використання засобів звукопоглинання, раціональне планування виробничих приміщень.

7.2. Заходи, що покращують умови праці оператора

Впровадженню режимів праці та відпочинку оператора ЕОМ повинна передувати робота щодо наукового обґрунтування тривалості та порядку проведення перерв, заснована на урахуванні змісту праці та факторів, які обумовлюють умови праці. Кількість, тривалість та структура перерв, а також їх «наповнення» визначаються характером прояву втоми та відповідають періодам зниження працездатності (пов'язаних з роботою, що виконується, та підвладних впливу добової періодичної зміни фізіологічних та психофізіологічних функцій людини).

Тривалість безперервної роботи за ВДТ без регламентованої перерви не повинна перевищувати 2 год. Тривалість обідньої перерви визначається чинним законодавством про працю та правилами внутрішнього трудового розпорядку підприємства (організації, установи).

При 8-го динній робочій зміні регламентовані перерви доцільно встановити:

для I категорії робіт за ВДТ через 2 год від початку зміни та через 2 год після обідньої перерви, кожна тривалістю 10 хв;

для II категорії робіт за ВДТ через 2 год від початку зміни тривалістю 15 хв, через 1,5 та 2,5 год після обідньої перерви тривалістю 15 та 10 хв відповідно або тривалістю 5—10 хв через кожну годину роботи, залежно від характеру технологічного процесу;

для III категорії робіт за ВДТ через 2 год від початку зміни, через 1,5 та 2,5 год після обідньої перерви тривалістю 20 хв кожна або тривалістю 5—15 хв через кожну годину роботи, залежно від характеру технологічного процесу.

Під час роботи за ВДТ у нічну зміну, незалежно від групи та категорії робіт, тривалість регламентованих перерв збільшується на 60 хв.

Після розробки раціонального режиму праці та відпочинку (визначення тривалості перерв на відпочинок, послідовності чергування проміжків праці та відпочинку, проектування змісту відпочинку) на підприємствах здійснюється

експериментальне впровадження нового режиму праці та відпочинку протягом 3—4 місяців. Потім проводяться фізіологічні та соціально-економічні дослідження для виявлення ефективності нового режиму праці та відпочинку.

Навколишнє робоче середовище повинно формуватися у тісній взаємодії з працівниками таким чином, щоб врахувати особливості користувачів з різними фізичними та розумовими якостями. Тому умови роботи мають бути досить варіабельними.

В положенні сидячи основне навантаження припадає на м'язи, що підтримують хребетний стовп та голову. При цьому тиск більшої частини маси тіла припадає на стегна, перешкоджаючи проникненню крові у нижню його частину. У зв'язку з цим при тривалому сидінні час від часу необхідно змінювати фіксовані робочі пози. До того ж при роботі сидячи природне прогинання поперекової ділянки хребетного стовпа уперед змінюється на прогинання назад, що часто є причиною появи болю у попереку.

Для фізіологічно правильно обґрунтованої робочої пози сидячи мають бути забезпечені оптимальні положення частин тіла: корпус випрямлений, зберігаються природний вигин хребта та кут нахилу таза. Необхідно уникати сильних нахилів торса, поворотів голови та крайніх положень суглобів кінцівок.

Літературні дані про оптимальні кути між сусідніми сегментами тіла, що забезпечують зручність пози, неоднозначні. Для пози сидячи частіше за все рекомендують такі значення кутів:

кут, створений положенням осі торса та шиї, змінюється залежно від роботи, що виконується; при значенні його більше 25° виникають хворобливі відчутгя у задній частині шиї; ближчим до оптимального вважається кут, що наближається до 15°;

вимоги до значення кута, утвореного положенням осі торса та осі стегна, дещо розходяться; за одними даними, він має бути прямим, тобто, 90°, за іншими—тупим (110-115°);

кут, створений віссю стегна та гомілки, може бути у діапазоні 90—120°, при куті більше 120° можлива рання втома розтягнутих згинаючих м'язів стегон;

вважається, що для кута, створеного віссю гомілки та підошви ступні, оптимальне значення 90—100°, можливе його збільшення до 115°;

кращим положенням руки визнано таке, при якому вона звисає вздовж тіла, тобто кут, створений віссю плече-ліктьового сегмента та вертикаллю торса, дорівнює нулю;

при роботі, коли передпліччя підтримуються підлокітниками або площиною столу, рука може утворювати досить великий кут з вертикаллю (до 45°), а коли маса руки утримується плечем і точкою опори служить кисть руки, то максимальне значення кута не повинно перевищувати 35°;

кут, утворений віссю плеча та передпліччя, може бути від 40° при згинанні та до 180° при максимальному витяганні; кут у 90° наближається до оптимального, оскільки згинаючі та розгинаючі м'язи стискаються однаковою мірою, а умови Кровообігу найбільш сприятливі;

кут утворений віссю передпліччя та кистю, рівний 180° вважається краіДим, оскільки при цьому м язи, що приводять у рух кисть, знаходяться у стані рівного скорочення, а кисть є прямим продовженням передпліччя, припустиме латеральне (бічне) відхилення - 10° .

Виходячи з загальних принципів організації робочого місця у нормативно-методичних документах сформульовані вимоги до його конструкції.

Основним обладнанням робочого місця користувача ВДТ є відеомонітор, клавіатура, робочий стіл, стілець (крісло); допоміжним — пюпітр, підставка для ніг, шафи, полиці та ін. Вимоги до них відображені у нормативних документах: ГОСТ 12.2.032-73; ГОСТ 22269-76.

Під просторовою орієнтацією робочого місця розуміється розміщення у певному порядку елементів основного та допоміжного обладнання відносно

одне одного та працюючої людини. Просторова організація робочого місця в основному визначається розмірами та формою сенсорного та моторного простору, формою та параметрами елементів робочого місця та просторовим розташуванням елементів відносно працюючого.

Робочі місця з ВДТ повинні розташовуватися на відстані не менше як 1,5 м від стіни з віконними прорізами, від інших стін — на відстані 1 м, між собою на відстані не менше як 1,5 м. При розміщенні робочих місці» необхідно виключити можливість прямого засвічування екрана джерелом природного освітлення. Джерело природного освітлення (вікно) не повинно також потрапляти у зону прямого спостереження користувача. Відносно світлових отворів робочі місця доцільно розташовувати таким чином, щоб природне світло падало на нього збоку, переважно зліва.

При розміщенні ВДТ на робочому місці потрібно забезпечити простір для користувача величиною не менше як 850 мм з урахуванням виступаючих частин обладнання та застосування (при необхідності) спецодягу. Для стоп має бути передбачено простір по глибині та висоті не менше як 150 мм, по ширині — не менше як 530 мм.

Розташовувати ВДТ на робочому місці необхідно так, щоб поверхня екрана знаходилася на відстані 400— 700 мм від очей користувача. Рекомендується розміщувати елементи робочого місця таким чином, щоб витримувалася однакова відстань очей користувача від екрана, клавіатури, тримача документів.

Залежно від виду роботи та зручності користувача доцільно користуватися можливістю повороту та регулюванням нахилу екрана. Ця вимога тим більш важлива, чим численнішими та різноманітнішими є заплановані випадки застосування ВДТ. Установка рівня екрана над столом та його розташування повинні забезпечуватися за допомогою вторинних пристроїв на робочому місці.

Необхідно стало розташовувати клавіатуру на робочому столі, не допускаючи її хитання. Разом з тим має бути передбачена можливість її

поворотів та переміщень. Положення клавіатури та кут її нахилу повинні відповідати побажанням користувача.

Принтер треба розташовувати так, щоб доступ до нього користувача та його колег був зручним. Конструкція робочого столу повинна забезпечувати можливість оптимального розміщення на робочій поверхні обладнання, що використовується, з урахуванням його кількості, розмірів, конструктивних особливостей та характеру роботи, яка виконується. Корисно мати модульне, рухоме робоче місце. Площа столу залежить від всіх необхідних для роботи компонентів, що розміщуються, та повинна допускати можливість вільного переміщення пристроїв. Поверхня столу має бути матовою з малим відбиттям та тепло ізолюючою.

Якщо конструкція робочого місця передбачає протікання трудового процесу у позі сидячи, то висота робочої поверхні столу повинна регулюватися у межах 680— 800 мм, у середньому вона повинна становити 725 мм. Робочий стіл повинен мати простір для ніг висотою не менше як 600 мм, шириною не менше як 500 мм, глибиною на рівні колін, але не менше як 450 мм та на рівні витягнутої ноги — не менше як 650 мм.

Робоче крісло забезпечує підтримання робочої пози у положенні сидячи, і чим триваліше це положення протягом робочого дня, тим жорсткішими мають бути вимоги до створення зручних та правильних робочих сидінь.

Існує цілий ряд публікацій щодо конструювання різних тигів робочих крісел. Незважаючи на розбіжність думок дослідників у визначенні деяких параметрів виділяють загальні рекомендації конструювання крісла: необхідність реулювання найбільш важливих його елементів — висоти сидіння, висоти спинки сидіння та кута нахилу спинки. Причому процес регулювання має бути нескладним. Не слід надмірно збільшувати кількість регульованих параметрів крісла, оскільки це позначатиметеся на його стійкості. Для надання більшої стійкості та попередження перекидання при відхиленні тіла на спинку крісла у багатьох європейських країнах використовують стільці на п'яти ніжках.

Встановлення правильної висоти сидіння є першочерговим завданням під час організації робочого місця. Цей параметр визначає інші просторові параметри — висоту положення екрана, клавіатури, поверхні для записів тощо.

Висота поверхні сидіння визначається висотою підколінної ямки над підлогою, виміряної у положенні сидячи при куті згинання коліна 90°. Висоту сидіння необхідно регулювати.

Зручність невеликих переміщень у просторі робочої зони, зумовлених характером виробничої діяльності, може бути забезпечена за наявності спеціальних коліщаток на ніжках стільця (звичайних або гальмівних) або шляхом ковзання по поверхні підлоги, що залежить від матеріалу її покриття.

Робоче місце має бути обладнане стійкою підставкою для ніг, параметри якої просто регулюються. Вона має бути розташована по всій ширині ділянки, що відводиться для ніг. Підставка повинна мати ширину не менше як 300 мм, глибину не менше як 400 мм, регулювання по висоті до 150 мм та по куту нахилу опорної поверхні підставки до 20°. Поверхня підставки має бути рифленою, а по передньому краю мати бортик висотою 10 мм.

Дотримання цих правил при організації робочого місця оператора ЕОМ значно покращить продуктивність його праці та зменшить ймовірність виникнення різноманітних професійних захворювань.

7.3 Заходи щодо охорони навколишнього середовища.

При експлуатації використаного у дипломному проекті виробничого обладнання з метою запобігання виникнення аварійних ситуацій, які мо-жуть призвести до забруднення навколишнього середовища, повинні забезпечуватись: - надійність і безпека роботи всього основного і допоміж-ного устаткування; - можливість досягнення номінальної продуктивнос-ті виробничого обладнання;

- економний режим роботи виробничого обладнання, встановлений на основі випробувань і заводських інструкцій;

- регулювальний діапазон навантажень;

- викиди шкідливих речовин в атмосферу в межах допустимих зна-чень.

Поряд з автоматизованим контролем найбільш відповідальних па-раметрів, надмірне відхилення яких від встановленого значення викликає порушення нормального технологічного процесу, передбачаються автома-тичні системи захисту обладнання від пошкоджень.

Пристрої технологічної безпеки повинні бути в повній готовності, але спрацьовувати лише в тому випадку, коли можливості автоматичного та дистанційного керування щодо запобігання відхилень параметрів від вста-новлених значень вичерпані, а оператор не може вчасно на це зреагувати.

Тобто технологічні захисти покликані впливати на об'єкт керування лише у виняткових випадках, зокрема у передаварійному (аварійному) стані, або при несподіваних стрімких зростаннях величин навантажень.

Частіше всього технологічні захисти служать для попередження ава-рій обладнання при відхиленнях параметрів від допустимих значень. Вплив захистів пов'язаний з відкриттям (закриттям) запірних органів, зу-пинкою основного та допоміжного обладнання або відключенням його ре-зерву.

7.4 Розрахунок аерації виробничого приміщення.

Оскільки питання забезпечення нормованих параметрів мікроклімату у виробничому приміщенні є одним із найважливіших у даному випадку із точки зору техніки безпеки та охорони праці, то виконаємо розрахунок аерації виробничого приміщення за умови, що температура зовнішнього повітря становить вище +10 °С, тобто для теплого періоду року.

Середня температура зовнішнього повітря для теплого періоду року (згідно нормативних документів для даного кліматичного поясу) становить: $t_з$ = +20 °С. Густина чистого свіжого зовнішнього повітря $\rho_з$:

$\rho_з$ = 1,199 кг/м3.

Розрахункове значення температури $t_{р.з.}$ в робочій зоні виробничого приміщення визначаємо за виразом:

$t_{р.з.} = t_з + \Delta t_{р.з}$,

де $\Delta t_{р.з.}$ – допустимий перепад температури в робочій зоні.

Значить:

$t_{р.з} = 20 + 8 = 28$ °С.

За рекомендованим нормативним співвідношенням еквівалентних площ для даної категорії виробничих приміщень та використовуваного виробничого обладнання визначаємо коефіцієнт аерації m:

m = 1.

Температура повітря, що видаляється із виробничого приміщення:

$t_{вих.} = t_{р.з.} / m = 28 / 1 = 28$ °С.

Перепад тисків, при якому можлива асиміляція надлишкової теплоти, складає:

$\Delta p_{1,2} = h \cdot (\rho_{вих} - \rho_з)\, g$,

де h – висота виробничого приміщення;

$\rho_{вих}$ - густина забрудненого повітря при $t_{вих}$.

Значить:

h = 4 м;

$\rho_{вих}$ = 1,25 кг / м3 (згідно замірів на базовому підприємстві).

Тоді:

$\Delta p_{1,2} = 4 \cdot (1{,}25 - 1{,}199) \cdot 9{,}8 = 2$ Па.

Для даної категорії виробничих приміщень тиск Δp_1 для забезпечення можливості надходження свіжого зовнішнього повітря у виробниче приміщення визначається за виразом:

$\Delta p_1 = n \cdot \Delta p_{1,2}$,

де n – частина тиску, що витрачається на рух повітря через віконні пройми (для розглядуваних виробничих умов n = 0,15).

Тоді:

$\Delta p1 = 0,15 \cdot 2 = 0,3$ Па.

Тиск $\Delta p2$ для витяжки забрудненого повітря через аераційний ліхтар визначається за формулою:

$\Delta p2 = \Delta p1,2 - \Delta p1 = 2 - 0,3 = 1,7$ Па.

Для визначення площ вентиляційних пройм використаємо формули для визначення тисків:

$$\Delta p_1 = \xi_1 \frac{v_1^2 \rho_з}{2};$$

$$\Delta p_2 = \xi_2 \frac{v_2^2 \rho_{вих}}{2},$$

де $\xi1$, $\xi2$ – коефіцієнти місцевих опорів відповідно для припливних і витяжних пройм:

$\xi1 = 3,2$;

$\xi2 = 5,8$;

v1 – швидкість чистого зовнішнього повітря у припливних проймах;

v2 – швидкість забрудненого повітря у витяжних проймах;

Значення швидкостей v1 і v2 визначаємо відповідно за виразами:

$$v_1 = \sqrt{2\Delta p_1 / (\xi_1 \cdot \rho_н)};$$

$$v_2 = \sqrt{2\Delta p_2 / (\xi_2 \cdot \rho_{вих})}$$

і підставляємо їх у формули для визначення площ відповідно припливних та витяжних пройм.

У результаті отримуємо:

$$F_1 = (G / \rho_з) / (3600 \cdot \sqrt{2\Delta p_1 / (\xi_1 \cdot \rho_з)}) = G / (3600\sqrt{2\Delta p_1 \rho_з / \xi_1});$$

$$F_2 = (G / \rho_{вих}) / (3600 \cdot \sqrt{2\Delta p_2 / (\xi_2 \cdot \rho_{вих})}) = G / (3600\sqrt{2\Delta p_2 \rho_{вих} / \xi_2}),$$

де G – нормовані масові витрати повітря на функціонування аерації для даної категорії виробничих приміщень та даного їх об'єму і забезпечення нормованої кратності повітрообміну, рівної 8.

Тобто сумарні площі становитимуть:

припливних пройм:

$$F_1 = 52000 / (3600 \cdot \sqrt{2 \cdot 0{,}3 \cdot 1{,}199 / 3{,}2}) = 52000 / 1707 = 31 \ \text{м}^2;$$

витяжних пройм:

$$F_2 = 52000 / (3600 \cdot \sqrt{2 \cdot 1{,}7 \cdot 1{,}25 / 5{,}8}) = 52000 / 3082 = 16{,}9 \ \text{м}^2.$$

7.5. Пожежна профілактика

Виробничі умови, системи вентиляції і кондиціонування повітря повинні відповідати протипожежним вимогам будівельних норм.

Повинні бути встановлені терміни проведення профілактичних огля-дів та очищення повітроводів, фільтрів, вогнезатримуючих клапанів, іншого обладнання вентиляційних систем, а також визначений порядок відключення вентиляційних систем і дій обслуговуючого персоналу в разі виникнення пожежі або аварії. Особа, призначена відповідальною за технічний стан та справність вентиляційних систем, зобов'язана забезпечити додержання вимог пожежної безпеки під час їх експлуатації.

На підприємстві відповідно до СНиП 2.04.02-84 та СНиП 2.04.01-85 необхідно передбачати систему протипожежного водопостачання, яка є

джерелом подачі води для пересувної пожежної техніки та установок пожежегасіння. Протипожежний водогін, як правило, об'єднується з господарчо-питтєвим чи виробничим водогоном.

Витрату води на гасіння пожежі при об'єднаному водогоні для спринклерних чи дренчерних установок, внутрішніх пожежних кранів та зов-нішніх гідрантів протягом 1 год. з моменту початку пожежегасіння необхі-дно приймати як суму найбільших витрат, що визначаються у відповідності з вимогами „Інструкції з проектування установок автоматичного пожежегасіння", СНиП 2.04.02-84 та СНиП 2.04.01-85.

При розрахунку протипожежного водопостачання тривалість гасіння пожежі приймається 3 год.; для будівель I та II ступеня вогнестійкості категорій Г та Д - 2 год.

Час роботи пожежних кранів приймається 3 год. При встановленні пожежних кранів на системах автоматичного пожежегасіння час їх роботи необхідно приймати рівним часу роботи систем автоматичного пожежегасіння.

Передбачаємо наступні заходи щодо евакуації персоналу у випадку виникнення пожежі:

а) повинні бути передбачені евакуаційні виходи, число яких прийма-ється згідно вимог протипожежної профілактики не менше двох;

б) ширина шляхів евакуації повинна бути не меншою одного метра, дверей - 0,8 метра;

в) двері повинні відкриватися у напрямі виходу із будівлі;

г) не допускаються на шляхах евакуації перепади висот, більші сорока п'яти сантиметрів і виступи в місцях перепаду висот;

д) не допускається влаштування на шляхах евакуації персоналу приміщень будь-якого призначення, газо- і паропроводів тощо;

є) відстань від найбільш віддаленого робочого місця до евакуаційного виходу не повинна бути більшою двадцяти метрів для даної категорії будівлі;

ж) повинно бути передбачене евакуаційне освітлення.

Згідно з вимогами ГОСТ 12.1.004-91 ССБТ «Пожарная безопасность. Общие требования» всі виробничі приміщення повинні бути забезпечені первинними засобами пожежегасіння, в якості яких можна застосовувати хімічно-пінні вогнегасники ОХП-10, ОП-14, ОП-9М, повітряпопінні вогнегасники ОВА-5, ОШ1-10, ОВП-250 та інші засоби.

8 ЕКОЛОГІЯ

8.1 Актуальність охорони навколишнього середовища

Усе живе в біосфері постійно перебуває під впливом електромагнітного поля (ЕМП) природного походження, в зв'язку з чим у організмів в процесі еволюції виробилися механізми, які дозволяють безболісно переносити середній рівень фонового опромінення, а також окремі пристосувальні можливості, що знижують гостроту реакції на деякі відхилення від норми при зміні ситуації (при грозових розрядах, магнітних бурях тощо). Але перевищення інтенсивності випромінювання над фоновим рівнем, зумовлене роботою радіозасобів, засобів зв'язку викликає несприятливі наслідки і дає підстави твердити про екологічну небезпеку електромагнітного випромінювання практично в усьому діапазоні частот і навіть при дуже малих інтенсивностях. У зв'язку з цим потрібно оцінити ступінь небезпеки, нормувати допустимі рівні опромінення, розробити та вжити необхідних захисних заходів.

Візуальні дисплейні термінали (ВДТ) у даний час є основним засобом взаємозв'язку людини з ЕОМ. Прискорене впровадження ЕОМ (персонального і колективного користування) практично в усі області діяльності веде до появи великої кількості робочих місць із ВДТ. Вони широко поширюються як на різні виробництва в різних системах контролю і керування, так і в різних адміністративно-суспільних будинках, де розміщаються обчислювальні центри організацій й інститутів, читальні і довідкові зали бібліотек, комп'ютерні зали шкіл, технікумів, і, зрештою, офісів і приватних квартир.

Відповідно до діючих нормативів відеотермінал – це пристрій для оперативного вводу і виводу даних з ЕОМ, для відображення на екрані інформації у формі, яка є зручною для користувача.

Відео-дисплейні термінали (ВДТ) є пристроєм для візуального представлення інформації, яка зберігається електронним способом. ВДТ надає можливість швидкого виведення алфавітно-цифрової або графічної інформації. Він складається

з екрана, оснащеного пристроєм обробки, що виводить інформацію, клавіатури управління та введення даних.

Класифікація ВДТ стосовно проблеми їх впливу на здоров'я заснована головним чином на конструктивних особливостях та повних параметрах самого пристрою. Існують ВДТ з плазмовими та рідинно-кристалічними екранами, проте вони ще не мають такого широкого розповсюдження, як ВДТ на основі електронно-променевих трубок (ЕПТ).

8.2 Вимоги до моніторів ВДТ і ПЕОМ

Важливою умовою безпеки людини перед екраном є правильний вибір візуальних параметрів монітора та світлотехнічних умов робочого місця. Робота з моніторами при неправильному підборі яскравості та освітлення екрана, контрастності символів, кольорів символів і фону, при наявності полисків на екрані, мигтіння зображення призводить до зорової втоми, головних болей, до значних фізіологічних та психічних навантажень, до погіршення зору.

В ГОСТ Р 50948-96 і ГОСТ Р 50949-96 та в Санітарних правилах і нормах (СанПіН) встановлені вимоги до двох груп візуальних параметрів:

1) Яскравість, освітлення, кутовий розмір символу та кут спостереження.

2) Нерівномірність яскравості, полиски, мигтіння, відстань між символами, словами, рядками, геометричними спотвореннями, тремтіння зображення.

Відеодисплейний термінал на основі ЕПТ є джерелом випромінювання кількох діапазонів електромагнітного спектра, зокрема мікрохвиль нетеплової інтенсивності. Вважають, що можливі два основних механізми дії мікрохвиль нетеплової інтенсивності. Один з цих механізмів заснований на припущенні, що у результаті резонансного захоплення енергії змінюються структури молекул у клітинах (так званий квантово-біологічний ефект). Другий механізм постулює детектування радіохвиль клітинами та органічними структурами клітин (наприклад, синапсами нервових волокон), що змінює процеси збудження, провідності та обміну речовин у цих клітинних структурах. Обидва механізми можуть впливати на регулюючу

функцію центральної нервової системи, викликаючи різні відхилення у функціональному стані організму.

Електромагнітні випромінювання характеризуються рядом взаємозалежних параметрів. Деякі з цих параметрів (частота, енергія фотонів) пов'язані з діапазоном випромінювання. Електромагнітне поле має електричну та магнітну складові, причому взаємозв'язок їх досить складний. З практичних міркувань це поле можна поділити на “ближнє поле” (менше однієї довжини хвилі від джерела) та “дальнє поле”.

Оскільки ВДТ (точніше монітори) є потенційними джерелами рентгенівських променів, які можуть викликати гінекологічні порушення, ураження шкіри та органу зору, доцільно більш докладно розглянути результати вимірювання цього випромінювання. Потенційним джерелом рентгенівських променів є електронно-променева трубка ВДТ, а конкретно-внутрішня флуоресціююча поверхня екрана. Енергія цих променів обмежена величиною напруги, що використовується для розгону електронів (приблизно до 10-25 кеВ - м'яке рентгенівське випромінювання). Внаслідок обмеженої енергії цих полів рентгенівське випромінювання такого виду ефективно поглинається скляним екраном.

На думку багатьох фахівців, робота з ВДТ не пов’язана з шкідливим радіобіологічним впливом. Припустима потужність дози рентгенівського випромінювання перед екраном на відстані 5 см від його поверхні дорівнює 0,5 мР/г.

Коли говорять про монітори, то не варто забувати про електростатичне поле, які створюють ці пристрої. Сильне електростатичне поле завдає шкідливого впливу на людський організм. Хоча, на відстані 50-60 см від екрана його вплив значно зменшується. Застосування спеціальних фільтрів, які прикривають екран, взагалі дозволяє зменшити його до нуля. Також варто звернути увагу на те, що при роботі монітора електризується не тільки його екран, але й повітря в приміщенні. Повітря набуває позитивного заряду. Позитивно наелектризована молекула кисню не сприймається організмом як кисень, це викликає у користувача кисневе голодування. В теперішній час всі монітори повинні відповідати стандарту MPRII, який обмежує випромінювання моніторів в діапазоні дуже низьких частот (деякі

основні параметри, визначені цим та іншими стандартами наведені в таблиці 8.1.)

Таблиця 8.1. Вимоги до електромагнітних полів монітора

Назва параметра	MPRII	ГОСТ Р 50948-96	СанПіН 2.2.2.542-96
Напруженість ЕМП в 50 см навколо дисплея по електричній складовій, В/м, не більше в діапазоні частот: 5Гц - 2кГц 2 - 400 кГц	25 2,5	25 2,5	
Щільність магнітного потоку в 50 см навколо дисплея, нТл, не більше: в діапазоні частот: 5Гц - 2кГц 250 2 - 400 кГц	250 25		
Поверхневий електростатичний потенціал, В, не більше	500		

Враховуючи вищевикладене, можна зробити висновок про те, що необхідно проводити комплексну оцінку електромагнітної обстановки в робочих приміщеннях з комп'ютерами.

Варто відмітити, що результати досліджень показують невідповідність моніторів, які перевірялись, сучасним вимогам ні по візуальних, ні по емісійних параметрах. Найбільш розповсюджені монітори з діагоналлю 14” мають недостатню частоту зміни епізодів (менше 75 Гц), не захищені від полисків, від накопичення на екрані статичного потенціалу.

Одним з найбільш важливих параметрів зображення є розмір символу, який залежить від відстані між спостерігачем та екраном, а також від алфавіту ін формації, що відображається.

Роздільна здатність (залежить від діаметра одиничної точки) та злитість зображення (залежить від відстані між точками) є незалежними параметрами екрана ВДТ. Взаємний добір цих характеристик з урахуванням чутливості людини до контрасту поліпшує сприймання інформації. У ВСНіПРВЦ передбачено, що мінімальний розмір точки, що світиться, має бути не більше як 0,4 для монохромного ВДТ і не більше як 0,6 - для кольорового.

Тепер діючими нормативними документами (ВСНіПРВЦ) передбачена регламентація тільки деяких параметрів інформації, що пред'являється. До них належать:

- яскравість свічення екрана (не менше як 100 кд/м);
- мінімальний розмір точки свічення (не більше як 0,4 мм для монохромного ВДТ і не менше як 0,6 мм - для кольорового);
- контрастність зображення символу (не менше як 0,8);
- частота регенерації зображення при роботі з позитивним контрастом у режим обробки тексту (не менше як 72 Гц);
- кількість точок на рядку (не менше 640);
- зсув низькочастотного мигтіння зображення (у діапазоні 0,05-1,0 Гц) повинен перебувати у межах 0,1 мм;
- при роботі з текстовою інформацією (у режимі введення даних, редагування тексту та читання з екрана ВДТ) найбільш фізіологічним є пред'явлення чорних знаків на світлому (білому) фоні.

8.3 Електромагнітне забруднення довкілля, його вплив на людину. Шляхи його зменшення.

Антропогенні випромінювання охоплюють усі діапазони. Можливості прямого опромінення радіохвилями визначаються умовами їх розповсюдження, які залежать від довжини хвилі.

На довгих хвилях (10 ... 1 км) електромагнітне поле (ЕМП) створюється хвилею, яка огинає земну поверхню та перешкоди, які на ній знаходяться (будинки, рослинність, нерівності місцевості), і йде між земною поверхнею та нижньою

межею іонізаційного шару атмосфери. Вони майже не поглинаються ґрунтом. Сигнали потужних радіомовних станцій в цьому діапазоні фактично у будь-який час доби вільно розповсюджуються на далекі відстані. Тому станції повинні розглядатися як джерела ЕМП, які відіграють важливу роль в екологічному відношенні.

Середні хвилі (1000 ... 100 м) також достатньо добре огинають земну поверхню, хоча при цьому відхиляються перешкодами, які мають розмір, більший від довжини хвилі, та значно поглинаються ґрунтом. Головну екологічну небезпеку створюють потужні радіомовні станції.

У діапазоні коротких хвиль (100 ... 10 м) радіохвилі дуже сильно поглинаються ґрунтом, але для розповсюдження на велику відстань використовується їх віддзеркалення від земної поверхні та від іоносфери. В цьому діапазоні працюють радіомовні станції та станції зв'язку.

На ультракоротких хвилях (10 ... 1 м), які дуже поглинаються ґрунтом та майже не віддзеркалюються іоносферою, розповсюдження сигналів відбувається практично лише в межах прямої видимості. Для збільшення цієї зони використовують високо розміщені антени та ретранслятори, причому ЕМП утворюється внаслідок інтерференції прямого та віддзеркаленого променів. В цьому діапазоні працюють зв'язкові, радіомовні та телевізійні станції, розташовані, як правило, у місцях великої концентрації населення. ЕМП повинні розглядатися в основному як хвороботворний фактор. На підставі клінічних та експериментальних матеріалів виявлені основні симптоми уражень, які виникають при впливі ЕМП. їх можна класифікувати як радіохвильову хворобу. Ступінь патологій прямо залежить від напруженості ЕМП, тривалості впливу, фізичних особливостей, діапазонів частот, умов зовнішнього середовища, а також від функціонального стану організму, його стійкості до впливу різних факторів, можливостей адаптації.

Поряд з радіохвильовою хворобою як специфічним результатом дії ЕМП спостерігається, завдяки його впливу, загальне зростання захворюваності, а також захворювання окремими хворобами органів дихання, травлення та ін. Це відмічається також і при дуже малій інтенсивності ЕМП, яка незначно перевищує

гігієнічні нормативи. Ймовірно, причиною тут є порушення нервово-психічної діяльності як головної у керуванні усіма функціями організму. Внаслідок дії ЕМП можливі як гострі, так і хронічні ураження, порушення в системах та органах, функціональні зміни в діяльності нервово-психічної, серцево-судинної, ендокринної, кровотворної та інших систем. Активність впливу ЕМП різних діапазонів частот різна: вона значно зростає з ростом частоти та дуже серйозно впливає у НВЧ діапазоні. В даний діапазон входять дециметрові (100 ... 10 см), сантиметрові (10 ... 1 см) та міліметрові (10 ... 1 мм) хвилі.

Вплив НВЧ на біологічні об'єкти останнім часом привертає увагу великої кількості дослідників та висвітлюється у численних наукових доповідях та публікаціях. Є відомості про клінічні прояви дії НВЧ залежно від інтенсивності опромінення.

НВЧ опромінення діє в основному аналогічно хвильовому, але сильніше. Крім того, спостерігаються і деякі особливості. Багато ефектів дії ЕМП пояснюються перетворенням енергії випромінювання у теплову. Оскільки нагрівання зростає пропорційно частоті, явища, пов'язані із нагріванням, на НВЧ проявляються сильніше. Одним із серйозних ефектів, зумовлених НВЧ опроміненням, є ушкодження органів зору. На нижчих частотах такі ефекти не спостерігаються і тому їх треба вважати специфічними для НВЧ діапазону.

Ступінь ушкодження залежить в основному від інтенсивності та тривалості опромінення. Із зростанням частоти, напруженість ЕМП, яка викликає ушкодження зору, – зменшується.

При впливі випромінювання на око спостерігається ушкодження роговиці. Але серед усіх тканин ока найбільшу чутливість має у діапазоні 1 ...10 ГГц кришталик. Сильне ушкодження кришталика зумовлене тепловим впливом НВЧ. При меншій інтенсивності каламутність спостерігається лише у задній ділянці, при великій - по всьому об'єму кришталика. Катарактоутворення пояснюють не лише тепловою дією, але й впливом ряду інших не зовсім встановлених факторів. Велике значення має концентрація поля в середовищі з окремими діелектричними властивостями та об'ємні резонансні ефекти.

Захисні заходи поділяються на організаційні, інженерно-технічні та лікарсько-профілактичні.

Здійснення організаційних та інженерно-технічних заходів покладено передусім на органи санітарного нагляду. У контакті з санітарними лабораторіями підприємств та установ, які використовують джерела електромагнітного випромінювання, вони повинні вживати заходів з гігієнічної оцінки нового будівництва та реконструкції об'єктів, котрі виробляють та використовують радіозасоби, а також нових технологічних процесів та обладнання з використанням ЕМП, проводити поточний санітарний нагляд за об'єктами, які використовують джерела випромінювання, здійснювати організаційно-методичну роботу з підготовки спеціалістів та інженерно-технічний нагляд.

Виключно важливе значення мають інженерно-технічні методи та засоби захисту: колективний (група будинків, район, населений пункт), локальний (окремі будівлі, приміщення) та індивідуальний. Колективний захист спирається на розрахунок поширення радіохвиль в умовах конкретного рельєфу місцевості. Економічно найдоцільніше використовувати природні екрани - складки місцевості, лісонасадження, не житлові будівлі.

При захисті від випромінювання екрана повинне враховуватись затухання хвилі при проходженні через екран (наприклад, лісову смугу), а також явища на верхніх та бічних стінках екрана, які збільшують інтенсивність ЕМП за екраном. Для екранування можна використовувати рослинність.

Локальний захист дуже ефективний і використовується часто. Він заснований на використанні радіозахисних матеріалів, які забезпечують високе поглинання енергії випромінювання у матеріалі та віддзеркалення від його поверхні. Для екранування шляхом віддзеркалення використовують добре провідні металеві листи та сітки. Захист приміщень від зовнішніх випромінювань можна здійснити завдяки обклеюванню стін металізованими шпалерами, захисту вікон сітками, металізованими шторами. Опромінення у такому приміщенні зводиться до мінімуму, але віддзеркалене від екранів випромінювання перерозповсюджується у просторі та потрапляє на інші об'єкти.

Радіопоглинаючі матеріали можуть використовуватись для захисту навколишнього середовища від ЕМП, яке генерується джерелом, що знаходиться в екранованому об’єкті.

Засоби індивідуального захисту використовують лише в тих випадках, коли інші захисні заходи неможливі чи недостатньо ефективні: при переході через зони збільшеної інтенсивності опромінення, при ремонтних та налагоджувальних роботах у аварійних ситуаціях, під час короткочасного контролю та зміни інтенсивності опромінення. У радіочастотному діапазоні засоби індивідуального захисту базуються на принципі екранування людини з використанням відбивання та поглинання ЕМП.

ВИСНОВКИ

Отже, в даній магістерській роботі було проведено проектування і створення інформаційної системи для обліку діяльності станції технічного обслуговування автомобілів.

В пояснювальній записці було описано загальну схему функціонування такої системи, пояснено процес створення бази даних, а також показано основні етапи створення прикладної клієнтської програми.

Проаналізувавши розроблену систему та порівнявши її з аналогічними програмами інших виробників, можна зробити висновки, що використання даної системи приведе до покращення умов праці працівників, підвищить продуктивність праці, зменшить можливість виникнення неточностей або збоїв системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Работа з базами данных в Delph./ - 2-е изд. – СПб.: БХВ-Петербург, 2002. – 624 с.
2. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: учебник для высших учебных заведений. / Под.ред.проф. А.Д.Хомоненко. – СПб.: КОРОНА принт, 2000. – 416 с.
3. Джеймс Р. Грофф, Пол Н. Вайнберг. SQL: полное руководство: пер.с англ. – К.: Издательская группа BHV, 2000. – 608 с.
4. Delphi 6. Учебный курс. – М.: Издательство Молгачева С.В., 2001. – 672с.
5. Фаронов В.В. Delphi 5. Руководство программиста. – М.: "Нолидж", 2001. – 880 с.
6. Расчет экономической эффективности в дипломных и курсовых проектах/ Под общ. ред. Н.Н. Фонталина. – Мн.: Высшая школа, 1984. – 126 с.
7. Безпека і охорона праці на підприємствах машинобудування / Вайнштейн В.Е., Київ: Техніка, 1967. – 304 с.
8. Основи охорони праці. – 4-е вид. Навчальний посібник / Житецький В.Ц., Джигерей В.С., Мельников О.В., Львів: Афіша, 2000. – 350 с.
9. Охорона праці навчальний посібник / Гаврик Є.О., К: Ельга Ніка-центр, 2003. – 280 с.
10. Охорона праці: навчальний посібник / Я.І. Бедрій, Львів: Р.П. “АРІ”, 2000. – 258 с.
11. Практикум з охорони праці. Навчальний посібник / Житецький В.Ц., Джигерей В.С., Сторожук В.М., Львів: Афіша, 2000. – 352с.
12. Білявський Г. О. Падун М. М. Фурдуй Р. С. Основи загальної екології. – Київ, Либідь, 1995. – 368 с.
13. Екологія основи теорії і практикум: Навчальний посібник / Потіш А.Ф., Медвідь В.Г., Гвоздецький В.Г., Львів: Новий світ, 2003. – 296 с.
14. Екологія та охорона навколишнього середовища: Навчальний посібник – 2-ге вид., стереотип Джигирей В.С., К: Знання, 2002. – 203 с.

15. Основи екології та охорона навколишнього середовища (Екологія та охорона природи) Джигирей В.С., Сторожук В.М., Яцюк Р.А., Львів: афіша, 2001. – 272 с.
16. Екологія. Охорона природи: Словник – довідник / Мусієнко М.М., Серебряков В.В., Брайон, К: Знання, 2002. – 550 с.
17. Соціальна екологія: навчальний посібник / За ред. Л.П. Царика, Тернопіль: Підручники і посібники, 2002. – 208 с.
18. Егоров П.Т., Гражданская оборона, - М.: Высшая школа, 1977. – 303 с.
19. Заплатинський В.М., Безпека життєдіяльності людини, - Львів: ЛБК НБУ; Київ: Знання, 2000. – 188 с.
20. Закон “Про цивільну оборону України” (в новій редакції) від 24 березня 1999 року.
21. Положення про Цивільну оборону України (затверджене постановою КМУ від 10 травня 1994 року №299).

ДОДАТОК

Додаток А

Тексти SQL-запитів для створення таблиць та тригерів бази даних

```
CREATE TABLE BASE
(
  CODE        INTEGER NOT NULL,
  IDMAG       INTEGER NOT NULL,
  NUMOF       INTEGER NOT NULL,
  IDTOV       INTEGER,
  DEN         DATE
);
SET TERM ^ ;


CREATE TRIGGER CODEBASE FOR BASE
ACTIVE BEFORE INSERT POSITION 0
AS
  BEGIN
  NEW.CODE = GEN_ID(GENBASE, 1);
  END
 ^

COMMIT WORK ^
SET TERM ;^


CREATE TABLE KURS
(
  DEN         DATE NOT NULL,
  DOLLAR      DECIMAL(5, 3) NOT NULL
);



CREATE TABLE MAGAZYN
(
  NAME        VARCHAR(50) CHARACTER SET WIN1251 NOT NULL,
  NOTE        VARCHAR(50) CHARACTER SET WIN1251,
  CODE        INTEGER NOT NULL
);
SET TERM ^ ;


CREATE TRIGGER CODEMAGAZYN FOR MAGAZYN
ACTIVE BEFORE INSERT POSITION 0
```

```
as
  begin
    new.code = gen_id(GenMagazyn, 1);
  end
^

COMMIT WORK ^
SET TERM ;^

CREATE TABLE MOVE
(
 CODE        INTEGER NOT NULL,
 TYP         INTEGER NOT NULL,
 VID         INTEGER NOT NULL,
 KUDY        INTEGER NOT NULL,
 IDTOV       INTEGER NOT NULL,
 NOTE        VARCHAR(100) CHARACTER SET WIN1251,
 DEN         DATE NOT NULL
);
SET TERM ^ ;

CREATE TRIGGER CODEMOVE FOR MOVE
ACTIVE BEFORE INSERT POSITION 0
AS
 BEGIN
 NEW.CODE = GEN_ID(GENMOVE, 1);
 END
^

COMMIT WORK ^
SET TERM ;^

REATE TABLE "POCUPETS
(
 CODE        INTEGER NOT NULL,
 NAME        VARCHAR(50) CHARACTER SET WIN1251 NOT NULL,
 NOTE        VARCHAR(50) CHARACTER SET WIN1251,
 ADDRES      VARCHAR(150) CHARACTER SET WIN1251,
 TEL         VARCHAR(30) CHARACTER SET WIN1251
);
SET TERM ^ ;

CREATE TRIGGER CODEPOCUPETS FOR POCUPETS
ACTIVE BEFORE INSERT POSITION 0
as
  begin
    new.code = gen_id(GenPocupets, 1);
  end
^

COMMIT WORK ^
SET TERM ;^
```

```
CREATE TABLE POSTACH
(
  CODE        INTEGER NOT NULL,
  NAME        VARCHAR(50) CHARACTER SET WIN1251 NOT NULL,
  NOTE        VARCHAR(50) CHARACTER SET WIN1251,
  ADDRES      VARCHAR(150) CHARACTER SET WIN1251,
  TEL         VARCHAR(30) CHARACTER SET WIN1251
);
SET TERM ^ ;

CREATE TRIGGER CODEPOSTACH FOR POSTACH
ACTIVE BEFORE INSERT POSITION 0
as
   begin
     new.code = gen_id(GenPostach, 1);
   end
 ^

COMMIT WORK ^
SET TERM ;^

CREATE TABLE TEMP
(
  IDGROUP    INTEGER NOT NULL,
  NAME       VARCHAR(200) CHARACTER SET WIN1251 NOT NULL,
  IDZAVOD    INTEGER NOT NULL,
  PR         DECIMAL(15, 2),
  PRA        DECIMAL(15, 2),
  PRB        DECIMAL(15, 2),
  PRC        DECIMAL(15, 2),
  M          CHAR(1) CHARACTER SET WIN1251 NOT NULL,
  MA         CHAR(1) CHARACTER SET WIN1251 NOT NULL,
  MB         CHAR(1) CHARACTER SET WIN1251 NOT NULL,
  MC         CHAR(1) CHARACTER SET WIN1251 NOT NULL,
  NOTE       VARCHAR(100) CHARACTER SET WIN1251,
  NUMOF      INTEGER,
  IDTOV      INTEGER,
  CODE       INTEGER,
  CODE1      INTEGER,
  ID         INTEGER
);
SET TERM ^ ;

CREATE TRIGGER IDTEMP FOR TEMP
ACTIVE BEFORE INSERT POSITION 0
as
  begin
    new.id = gen_id(gtemp, 1);
  end
 ^
```

```
COMMIT WORK ^
SET TERM ;^



CREATE TABLE TOVAR
(
  CODE        INTEGER NOT NULL,
  IDGROUP     INTEGER NOT NULL,
  NAME        VARCHAR(200) CHARACTER SET WIN1251,
  NOTE        VARCHAR(100) CHARACTER SET WIN1251,
  MANUF       VARCHAR(100) CHARACTER SET WIN1251,
  PR          INTEGER,
  PRA         INTEGER,
  PRB         INTEGER,
  PRC         INTEGER,
  M           CHAR(1) CHARACTER SET WIN1251,
  MA          CHAR(1) CHARACTER SET WIN1251,
  MB          CHAR(1) CHARACTER SET WIN1251,
  MC          CHAR(1) CHARACTER SET WIN1251
);
SET TERM ^ ;



CREATE TRIGGER CODETOVAR FOR TOVAR
ACTIVE BEFORE INSERT POSITION 0
AS
  BEGIN
  NEW.CODE = GEN_ID(GENTOVAR, 1);
  END
 ^

COMMIT WORK ^
SET TERM ;^



CREATE TABLE TOVGRUP
(
  CODE        INTEGER NOT NULL,
  NAME        VARCHAR(50) CHARACTER SET WIN1251 NOT NULL,
  NOTE        VARCHAR(50) CHARACTER SET WIN1251
);
SET TERM ^ ;



CREATE TRIGGER CODETOVGRUP FOR TOVGRUP
ACTIVE BEFORE INSERT POSITION 0
```

```
as
  begin
    new.code = gen_id(GenTovgrup, 1);
  end
^

COMMIT WORK ^
SET TERM ;^



CREATE TABLE ZAVOD
(
  CODE          INTEGER NOT NULL,
  NAME          VARCHAR(50) CHARACTER SET WIN1251 NOT NULL,
  NOTE          VARCHAR(50) CHARACTER SET WIN1251
);
SET TERM ^ ;



CREATE TRIGGER CODEZAVOD FOR ZAVOD
ACTIVE BEFORE INSERT POSITION 0
as
  begin
    new.code = gen_id(GenZavod, 1);
  end
^

COMMIT WORK ^
SET TERM ;^
```

Додаток Б

Текст модуля SkladU головного вікна клієнтської програми

```
unit SkladU;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, ComCtrls, StdCtrls, ToolWin, Menus, ExtCtrls,
  ImgList, DBCtrls, ActnList, ActnMan, DB, DBTables, IBDatabase,
  IBCustomDataSet, IBQuery;

type
  TBaseF = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    N9: TMenuItem;
    N10: TMenuItem;
    N11: TMenuItem;
    N12: TMenuItem;
    N13: TMenuItem;
    N14: TMenuItem;
    N15: TMenuItem;
    N16: TMenuItem;
    N17: TMenuItem;
```

N18: TMenuItem;
N19: TMenuItem;
N20: TMenuItem;
N21: TMenuItem;
N22: TMenuItem;
N23: TMenuItem;
N24: TMenuItem;
N25: TMenuItem;
N26: TMenuItem;
N27: TMenuItem;
N28: TMenuItem;
N29: TMenuItem;
N30: TMenuItem;
N31: TMenuItem;
N32: TMenuItem;
N33: TMenuItem;
N34: TMenuItem;
N35: TMenuItem;
ToolBar1: TToolBar;
ToolButton1: TToolButton;
ToolButton2: TToolButton;
ToolButton3: TToolButton;
ToolButton4: TToolButton;
ToolButton5: TToolButton;
ToolButton6: TToolButton;
ToolButton7: TToolButton;
ToolButton8: TToolButton;
ToolButton9: TToolButton;
ToolButton10: TToolButton;
ToolButton11: TToolButton;
ToolButton12: TToolButton;
ToolButton13: TToolButton;
ToolButton14: TToolButton;
ToolButton15: TToolButton;
ToolButton16: TToolButton;

ToolButton17: TToolButton;
ToolButton18: TToolButton;
ToolButton19: TToolButton;
ToolButton20: TToolButton;
ToolButton21: TToolButton;
ToolButton22: TToolButton;
ToolButton23: TToolButton;
ToolButton24: TToolButton;
DBGrid1: TDBGrid;
StatusBar1: TStatusBar;
Bevel1: TBevel;
ImageList1: TImageList;
ToolButton25: TToolButton;
N36: TMenuItem;
N37: TMenuItem;
N38: TMenuItem;
N39: TMenuItem;
N40: TMenuItem;
N41: TMenuItem;
N42: TMenuItem;
N43: TMenuItem;
N44: TMenuItem;
N45: TMenuItem;
N46: TMenuItem;
N47: TMenuItem;
N48: TMenuItem;
N49: TMenuItem;
N50: TMenuItem;
N51: TMenuItem;
N52: TMenuItem;
N53: TMenuItem;
N54: TMenuItem;
N55: TMenuItem;
N56: TMenuItem;
ActionManager1: TActionManager;

```
Tochka: TAction;
Kurs: TAction;
Add: TAction;
VnutrMove: TAction;
Prodaj: TAction;
DataSource1: TDataSource;
cbMagazyn: TComboBox;
Postach: TAction;
Pocupets: TAction;
groups: TAction;
Zavod: TAction;
IBDatabase1: TIBDatabase;
IBTransaction1: TIBTransaction;
cbQuery: TIBQuery;
IBQuery1: TIBQuery;
IBQuery1NAME1: TIBStringField;
IBQuery1NAME2: TIBStringField;
IBQuery1NAME3: TIBStringField;
IBQuery1NUMOF: TIntegerField;
IBQuery1PR: TIntegerField;
IBQuery1M: TIBStringField;
IBQuery1PRA: TIntegerField;
IBQuery1MA: TIBStringField;
IBQuery1PRB: TIntegerField;
IBQuery1MB: TIBStringField;
IBQuery1PRC: TIntegerField;
IBQuery1MC: TIBStringField;
IBQuery1DEN: TDateField;
IBQuery1NOTE: TIBStringField;
IBQuery1suma: TFloatField;
IBQuery1IDTOV: TIntegerField;
IBQuery1CODE: TIntegerField;
MoveAll: TAction;
procedure ToolButton24Click(Sender: TObject);
procedure FormShow(Sender: TObject);
```

```
    procedure TochkaExecute(Sender: TObject);
    procedure KursExecute(Sender: TObject);
    procedure AddExecute(Sender: TObject);
    procedure VnutrMoveExecute(Sender: TObject);
    procedure ProdajExecute(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure N11Click(Sender: TObject);
    procedure N9Click(Sender: TObject);
    procedure Lists;
    procedure PostachExecute(Sender: TObject);
    procedure PocupetsExecute(Sender: TObject);
    procedure groupsExecute(Sender: TObject);
    procedure ZavodExecute(Sender: TObject);
    procedure cbMagazynChange(Sender: TObject);
    procedure OpenIBQuery;
    procedure IBQuery1CalcFields(DataSet: TDataSet);
    procedure DBGrid1DblClick(Sender: TObject);
    procedure MoveAllExecute(Sender: TObject);
    procedure N15Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  BaseF: TBaseF;
  List1, List2, List3, List4, List5: TStringList;
  Lcod1, Lcod2, Lcod3, Lcod4, Lcod5: TStringList;

implementation

uses AboutU, TableU, KursU, AddU, TovarU, DollarU, MoveU;

{$R *.dfm}
```

```
procedure TBaseF.Lists;
var i: integer;
begin
List1 := TStringList.Create; Lcod1 := TStringList.Create; //Підрозділи
List2 := TStringList.Create; Lcod2 := TStringList.Create; //Групи товарів
List3 := TStringList.Create; Lcod3 := TStringList.Create; //Виробники
List4 := TStringList.Create; Lcod4 := TStringList.Create; //Покупці
List5 := TStringList.Create; Lcod5 := TStringList.Create; //Постачальники

cbQuery.SQL.Clear;
cbQuery.SQL.Add('SELECT name, code FROM MAGAZYN');
cbQuery.Open; cbQuery.Last; cbQuery.First;
for i:=1 to cbQuery.RecordCount do   begin
  List1.Add(cbQuery.FieldByName('name').AsString);
  Lcod1.Add(cbQuery.FieldByName('code').AsString);
  cbQuery.Next; end;
cbQuery.Close;

cbQuery.SQL.Clear;
cbQuery.SQL.Add('SELECT * FROM TOVGRUP');
cbQuery.Open; cbQuery.Last; cbQuery.First;
for i:=1 to cbQuery.RecordCount do   begin
  List2.Add(cbQuery.FieldByName('name').AsString);
  Lcod2.Add(cbQuery.FieldByName('code').AsString);
  cbQuery.Next; end;
cbQuery.Close;

cbQuery.SQL.Clear;
cbQuery.SQL.Add('SELECT * FROM ZAVOD');
cbQuery.Open; cbQuery.Last; cbQuery.First;
for i:=1 to cbQuery.RecordCount do   begin
  List3.Add(cbQuery.FieldByName('name').AsString);
  Lcod3.Add(cbQuery.FieldByName('code').AsString);
  cbQuery.Next; end;
```

```
cbQuery.Close;

cbQuery.SQL.Clear;
cbQuery.SQL.Add('SELECT * FROM POCUPETS');
cbQuery.Open; cbQuery.Last; cbQuery.First;
for i:=1 to cbQuery.RecordCount do   begin
  List4.Add(cbQuery.FieldByName('name').AsString);
  Lcod4.Add(cbQuery.FieldByName('code').AsString);
  cbQuery.Next; end;
cbQuery.Close;

cbQuery.SQL.Clear;
cbQuery.SQL.Add('SELECT * FROM POSTACH');
cbQuery.Open; cbQuery.Last; cbQuery.First;
for i:=1 to cbQuery.RecordCount do   begin
  List5.Add(cbQuery.FieldByName('name').AsString);
  Lcod5.Add(cbQuery.FieldByName('code').AsString);
  cbQuery.Next; end;
cbQuery.Close;
end;

procedure TBaseF.OpenIBQuery;
begin
IBQuery1.ParamByName('cb').AsInteger := StrToInt(Lcod1.Strings[cbMagazyn.ItemIndex]);
IBQuery1.Open;
end;

procedure TBaseF.ToolButton24Click(Sender: TObject);
begin
Close;
end;

procedure TBaseF.FormShow(Sender: TObject);
var s: string;
begin
```

```
AboutF.ShowModal;
if not IBDatabase1.Connected then exit;

Lists;
CBmagazyn.Items := List1;    cbMagazyn.ItemIndex := 0;
OpenIBQuery;

DollarF.IBQuery1.SQL.Clear;
DollarF.IBQuery1.SQL.Add('select * from kurs where den = '+#39+DateToStr(Date)+#39);
DollarF.IBQuery1.Open;
DollarF.Label2.Caption := DateToStr(Date);
if DollarF.IBQuery1.IsEmpty then DollarF.ShowModal;
if DollarF.Tag = 1 then begin
   s := DateToStr(Date);
   DollarF.IBQuery1.Close;
   DollarF.IBQuery1.SQL.Clear;
   DollarF.IBQuery1.SQL.Add('insert into kurs (den, dollar) values ('+#39+s+#39+',
'+#39+st+#39+')');
   DollarF.IBQuery1.ExecSQL;
   end;
end;

procedure TBaseF.TochkaExecute(Sender: TObject);
begin
TableF.Caption := 'Підрозділи';
TableF.ShowModal;
end;

procedure TBaseF.KursExecute(Sender: TObject);
begin
KursF.ShowModal;
end;

procedure TBaseF.AddExecute(Sender: TObject);
begin
```

```
AddF.Tag := 1; //Прихід товару
AddF.Show;
end;

procedure TBaseF.VnutrMoveExecute(Sender: TObject);
begin
AddF.Tag := 2; //Внутрішнє переміщення товару
AddF.Show;
end;

procedure TBaseF.ProdajExecute(Sender: TObject);
begin
AddF.Tag := 3; //Продаж товару
AddF.Show;
end;

procedure TBaseF.FormClose(Sender: TObject; var Action: TCloseAction);
begin
IBDatabase1.Close;
List1.Free; Lcod1.Free;
List2.Free; Lcod2.Free;
List3.Free; Lcod3.Free;
List4.Free; Lcod4.Free;
List5.Free; Lcod5.Free;
end;

procedure TBaseF.N11Click(Sender: TObject);
begin
Close;
end;

procedure TBaseF.N9Click(Sender: TObject);
begin
AboutF.Edit1.Visible := false;
AboutF.Edit2.Visible := false;
```

```
AboutF.ShowModal;
AboutF.Edit1.Visible := true;
AboutF.Edit2.Visible := true;
end;

procedure TBaseF.PostachExecute(Sender: TObject);
begin
TableF.Caption := 'Постачальники';
TableF.ShowModal;
end;

procedure TBaseF.PocupetsExecute(Sender: TObject);
begin
TableF.Caption := 'Покупці';
TableF.ShowModal;
end;

procedure TBaseF.groupsExecute(Sender: TObject);
begin
TableF.Caption := 'Групи товарів';
TableF.ShowModal;
end;

procedure TBaseF.ZavodExecute(Sender: TObject);
begin
TableF.Caption := 'Виробники';
TableF.ShowModal;
end;

procedure TBaseF.cbMagazynChange(Sender: TObject);
begin
IBQuery1.Close;
IBQuery1.ParamByName('cb').AsInteger := StrToInt(Lcod1.Strings[cbMagazyn.ItemIndex]);
OpenIBQuery;
end;
```

```
procedure TBaseF.IBQuery1CalcFields(DataSet: TDataSet);
begin
IBQuery1suma.AsFloat := IBQuery1numof.AsInteger * IBQuery1pr.AsFloat;
end;

procedure TBaseF.DBGrid1DblClick(Sender: TObject);
begin
if AddF.Visible and ((AddF.Tag = 2) or (AddF.Tag = 3)) then  AddF.TovarForMove;
end;

procedure TBaseF.MoveAllExecute(Sender: TObject);
begin
MoveF.ShowModal;
end;

procedure TBaseF.N15Click(Sender: TObject);
begin
if IBQuery1.SQL.Strings[IBQuery1.SQL.Count - 2] = ' order by' then
  begin
  IBQuery1.SQL.Delete(IBQuery1.SQL.Count - 2);
  IBQuery1.SQL.Delete(IBQuery1.SQL.Count - 1);
  end;
IBQuery1.Close;
IBQuery1.SQL.Add(' order by');
IBQuery1.SQL.Add(' zavod.name');
IBQuery1.Open;
end;
```

end