Курсовая работа по дисциплине «Технологии баз данных»

Вариант 1 «Мебель»

Выполнил студент группы 3332 Ларин Анатолий

28 марта 2007 г.

Содержание

1 Постановка задачи **2**

2 Первичная структура базы данных **2**

3 Проектирование базы данных **4**

3.1 Проектирование модели предметной области с использованием UML 4

3.1.1 Продукт 4

3.1.2 Заказ 4

3.1.3 Заказчик 5

3.1.4 Физическое лицо 6

3.1.5 Юридическое лицо 6

3.1.6 Отгрузка 6

3.1.7 Транспортное средство 6

3.1.8 Сотрудник 7

3.1.9 UML диаграмма 7

3.2 Проектирование модели базы данных с использованием ERwin 9

3.2.1 Домены 9

3.2.2 Сущности 9

3.2.3 Отношения 10

3.2.4 ER диаграмма 10

4 Создание базы данных **11**

4.1 Тригеры 11

4.1.1 Суррогатные ключи 11

4.1.2 Запрет удаления невыполненной отгрузки 11

4.1.3 Подсчет стоимости заказа 12

4.2 Хранимые процедуры 13

4.2.1 Увольнение сотрудника 13

4.2.2 Архивация выполненных заказов 13

4.2.3 Популярность продуктов 14

4.3 «Представления» 15

4.3.1 Активность менеджеров 15

4.4 Тестовые данные 15

5 Демонстрационное приложение **17**

5.1 Назначение 17

5.2 Структура программы 17

5.3 Изображение главного окна 18

1 Постановка задачи

База данных предназначена для информационной поддержки деятельности менеджера по продажам на мебельном производстве. В базе данных хранятся сведения о продукции, заказах, заказчиках, а также график отгрузки заказов.

Каждый заказ может включать несколько видов продукции (например, два кресла и один диван). Кроме того, для заказа учитывается следующая информация: ответственный менеджер, дата приема, номер заказа, сумма заказа, сумма со скидкой, дата планируемого завершения, дата фактического завершения, дата отгрузки и данные о заказчике.

Сохраняемая информация о заказчиках включает наименование заказчика (Ф.И.О. физического лица или название юридического), вид заказчика (физическое лицо/юридическое лицо), адрес, телефон, факс, e-mail. Если заказчик является юридическим лицом и осуществляет оплату по безналичному расчету, то указывается: банк, счет, БИК, ИНН, ОКОНХ, ОКПО. Указывается также информация о городе и районе (только по Санкт-Петербургу).

Для каждого заказа учитывается пункт доставки: зона расположения, адрес, этаж, пометка о наличии лифта, номер и код парадной.

В базе данных содержится и основная информация о продаваемой продукции: наименование, производственная цена, цена для продажи.

Наконец, календарь отгрузок содержит информацию об отгрузке с указанием транспортного средства, экспедитора, ответственного за отгрузку, планируемой даты отгрузки, времени, статусе результата (произведена/не произведена).

2 Первичная структура базы данных

На рисунке 1 показана первоначальная структура базы данных, полученная методом обратного проектирования в ERwin.

Отметим некоторые недостатки существующей базы данных:

- В заказе может быть не больше четырех позиций. Если позиций меньше, в клиентском приложении все равно придется проверять все четыре поля.
- Продукт идентифицируется по названию, это не позволяет ввести в базу продукцию с одинаковыми названиями, но разными артикулами (напр. разный цвет, партия). Так же это ведет к увеличению размеров индекса и замедлению выполнения операций с базой данных.
- Контроль за правильностью данных (электропочта, номера телефонов, адрес, стоимость) должен осуществляться приложением.
- Менеджер, экспедитор и транспортные средства представлены обычными строками, что не позволяет хранить дополнительные данные о них и ведет к аномалиям.

Из вышеперечисленных недостатков, можно сделать вывод, что база нуждается в перепроектировании.

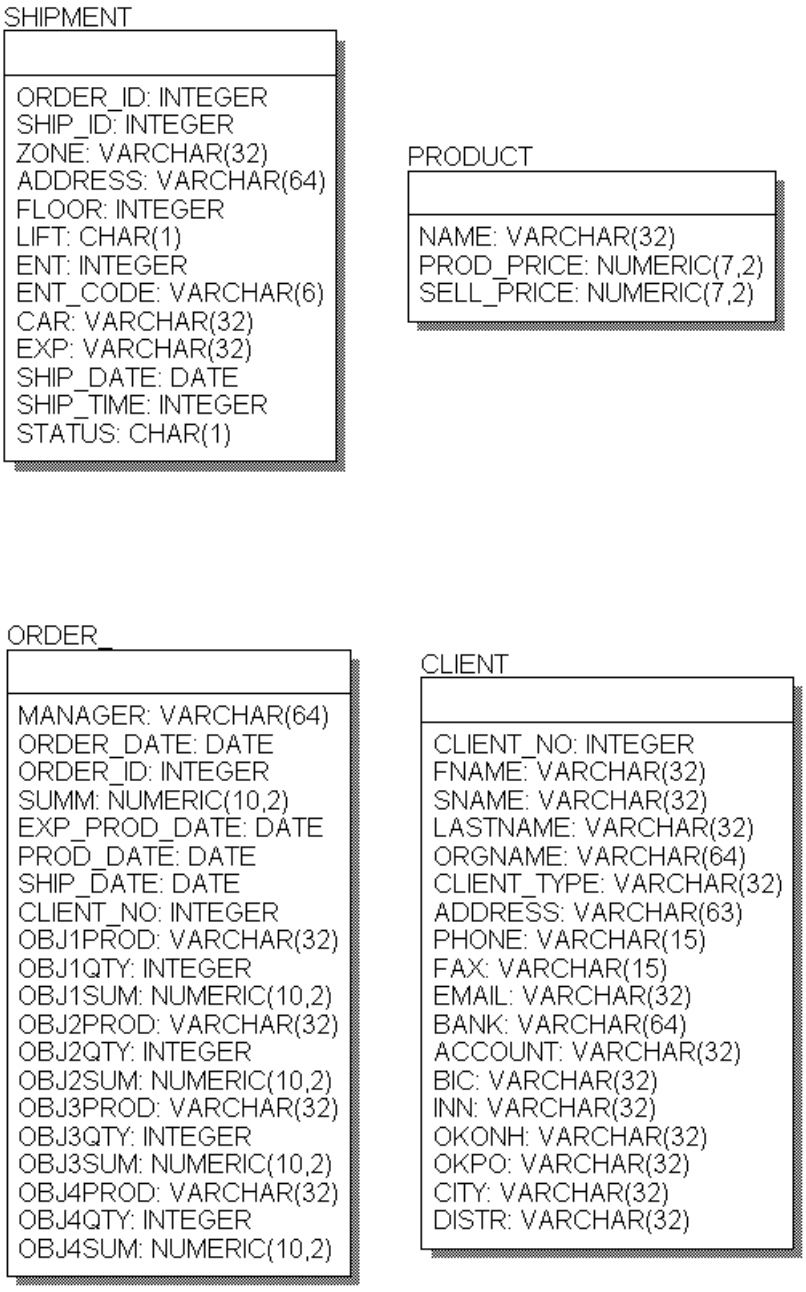

Рис. 1. ER диаграмма. Первичная структура базы данных.

3 Проектирование базы данных

В данном разделе рассматривается основная часть курсовой работы — проектирование базы данных.

Постепенно будут описаны все этапы проектирования, начиная с модели предметной области и заканчивая конкретной реализацией на СУДБ Firebird.

3.1 Проектирование модели предметной области с использованием UML

В рассматриваемой предметной области мы можем выделить следующие объекты:

- продукт;
- заказ;
- заказчик;
- отгрузка.

Рассмотрим каждый объект в отдельности.

3.1.1 Продукт

Объект продукт обладает следующими атрибутами:

- название;
- себестоимость;
- цена для продажи.

Можно обратить внимание на два атрибута себестоимости и цены. Для них не очень подходят целочисленные типы данных (цены не могут быть отрицательными, нулевыми, для работы с копейками придется делить цены в клиентском приложении на 100), как и не подходят дробные типы (отрицательные и нулевые цены, избыточная точность, неравномерность разрядной сетки). Предполагая, что предприятие использует рублевые цены и не торгует эксклюзивной мебелью мы можем ввести дополнительный тип данных Money. Он должен иметь два дробных разряда и недопускать чисел меньших или равных нулю и больших миллиона.

Объект продукт ассоциативно связан с заказом, причем не все продукты могут состоять в заказах.

3.1.2 Заказ

Атрибуты заказа:

- менеджер;
- стоимость;

- стоимость со скидкой;
- дата приема;
- дата планируемого завершения;
- дата фактического завершения.

Для стоимостей применим описанный нами выше тип данный Money.

Учитывая, что мы будем иметь дело с экспедиторами и водителями, стоит убрать атрибут менеджер и ввести новый объект сотрудник. Между заказом и сотрудником будет ассоциативная связь, причем у заказа может быть только один менеджер.

Объект заказ ассоциативно связан с объектом продукт, причем в заказе может быть один или несколько продуктов; с заказчиком, причем у заказа может быть всего один заказчик; с отгрузкой, причем у заказа отгрузка может отсутствовать (клиент забирает мебель сам).

3.1.3 Заказчик

Атрибуты заказчика:

- имя физического лица или ответственного сотрудника юридического лица;
- название юридического лица;
- вид заказчика (физическое/юридическое);
- адрес;
- телефон;
- факс;
- e-mail;
- данные о безналичном расчете.

Для телефона и факса введем новый тип данных телефонный номер, который может быть представлен в двух вариантах с кодом или прямой.

Для e-mail введем новый тип данных состоящий из имени пользователя и домена.

Для адреса также введем соответствующий тип данных.

Из объекта заказчик можно выделить два объекта юридическое и физическое лицо, потомков объекта заказчик, тогда в родительском объекте останутся следующие атрибуты:

- адрес;
- телефон;
- факс;
- e-mail.

Заказчик ассоциативно связан с объектом заказ, причем у заказчика на данным момент может не быть заказов.

3.1.4 Физическое лицо

Атрибуты:

- имя;
- отчество;
- фамилия.

3.1.5 Юридическое лицо

У объекта юридическое лицо остается только один атрибут — название.

Данные о безналичном расчете стоит выделить в отдельную сущность, принимая во внимание тот факт, что у фирмы может быть несколько счетов или вовсе их не быть (оплата наличными).

3.1.6 Отгрузка

Атрибуты отгрузки:

- планируемая дата;
- планируемое время;
- экспедитор;
- транспортное средство;
- статус результата.

Учитывая ,рассматриваемый выше объект сотрудник, убираем атрибут экспедитор и вводим ассоциативную связь, учитывая что у отгрузки может быть всего один экспедитор.

Вместо атрибута транспортное средство введем объект транспортное средство, для хранения дополнительной информации и устранения аномалий.

3.1.7 Транспортное средство

Атрибуты транспортного средства:

- тип;
- регистрационный номер;
- водитель.

Атрибут водитель заменим на ассоциативную связь с объектом сотрудник, причем у транспортного средства может быть только один водитель.

3.1.8 Сотрудник

Атрибуты объекта сотрудник:

- имя;
- фамилия;
- дата рождения;
- номер пасспорта;
- должность;
- телефон.

Объект сотрудник имеет ассоциативные связи с заказом (роль менеджер), с отгрузкой (роль экспедитор) и с транспортным средством (роль водитель).

Для атрибута должность стоило бы ввести отдельный объект, но в рамках курсовой работы мы этого делать не будем.

3.1.9 UML диаграмма

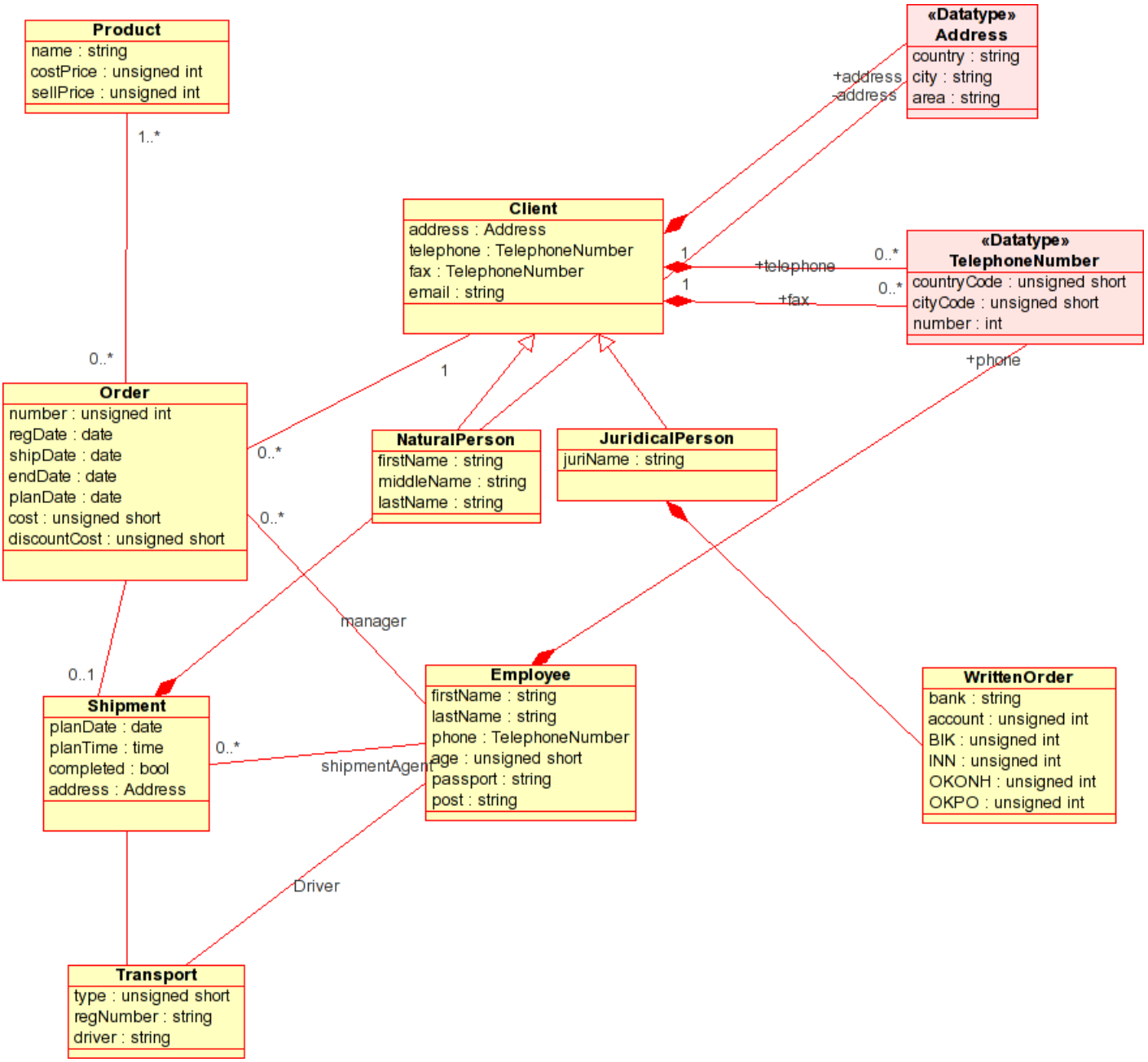

Рис. 2. UML диаграмма. Модель предметной области.

3.2 Проектирование модели базы данных с использованием ERwin

Используя результаты предыдущего пункта смоделируем базу данных в ERwin.

3.2.1 Домены

Для всех введенных нами типов данных создадим домены:

Адрес Определим домен как строку переменной длины и возложим выполнение проверок корректности на клиентские приложения.

Стоимость Для стомости идеально подходит тип DECIMAL с 7 целыми и двумя дробными разрядами. Ограничение для домена: `VALUE BETWEEN 0 AND 1000000 OR VALUE IS NULL`.

Электронная почта Определим домен как строку переменной длины. Ограничением для домена будет сравнение с маской `_%@_%._%`.

Телефон Определим домен как строку переменной длины. Телефон может быть городским (маска `_%`) так и междунородным или мобильным (маска `+_%(_%)_%`).

Так как в FireBird отсутствует булевый тип данных придется определить домен BOOL типа SMALLINT с ограничением `VALUE IN (0, 1)`

3.2.2 Сущности

Определим сущности, составляющие нашу диаграмму.

Сущности практически соответствуют объектам модели предметной области, с небольшими модификациями возникающими из-за «ограниченности» реляционного подхода.

Объекты заказчик, физическое лицо, юридическое лицо, объединим в одну сущность заказчик, добавив булевый атрибут определяющий принадлежность к юридическим лицам.

Также необходимо добавить сущностью PRODUCTS_IN_ORDER, возникающую при разбиении отношения (ассоциативной связи) многие-ко-многим между заказом и продуктом. Эта сущность будет определять продуктовый состав заказа.

Атрибуты всех сущностей дополним суррогатными первичными ключами.

Окончательный набор сущностей:

- PRODUCT
- PRODUCTS_IN_ORDER
- ORDER
- CLIENT
- WRITTENORDER
- SHIPMENT
- TRANSPORT
- EMPLOYEE

3.2.3 Отношения

Отношения ER-модели строятся из связей модели предметной области.

Обобщающие связи не переносятся, из-за объединения потомков с родителем в одну сущность.

Связи имеющие с обеих сторон множественную кратность разбиваются при помощи выделения новой сущности, к которой от связанных ранее сущностей строятся отношения один-ко-многим.

Ассоциативные связи не обладающие множественной кратностью с обеих сторон переносятся в модель базы данных.

3.2.4 ER диаграмма

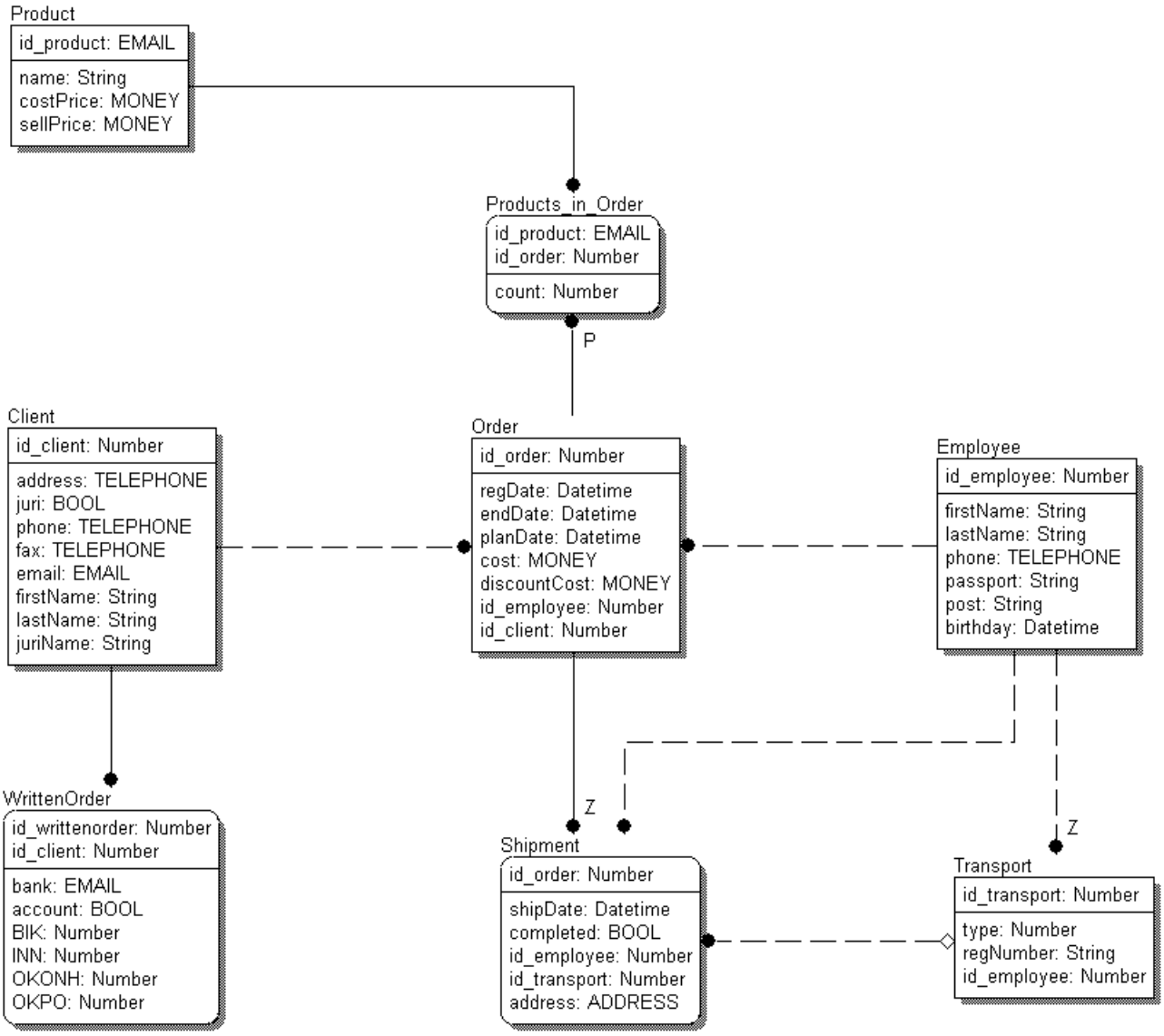

Рис. 3. ER диаграмма. Модель базы данных.

4 Создание базы данных

В этом разделе описана реализация предложенной выше модели для СУБД firebird.

Начальный SQL скрипт мы получим экспортом физической модели из ERwin. Этот скрипт необходимо дополнить тригерами на инкрементные поля, тригерами и хранимыми процедурами реализующими бизнес логику, также необходимо добавить View для вероятно часто используемых запросов.

4.1 Тригеры

4.1.1 Суррогатные ключи

Для каждого введенного нами суррогатного первичного ключа требуется ввести тригеры и генераторы. Генерирующие новое значение ключа увеличением значения генератора на единицу. Оставлена возможность ручной установки ключа без изменения генератора, для передачи каких-либо специфичных значений, или взаимодействия с IBExpert (свойство genfield).

Тригеры добавляются по образцу:

```
CREATE GENERATOR <имя генератора>; //Создание генератора
SET GENERATOR <имя генератора> TO 1;//Инициализация генератора

/*Тригер использующий генератор, при пустом значении ID*/
CREATE TRIGGER <имя тригера> for <имя таблицы>
active before insert position 0
as
begin
  if ((new.<имя поля> is null) or (new.<имя поля> = 0)) then
  begin
    new.<имя поля> = gen_id(<имя генератора>, 1);
  end
end;
```

4.1.2 Запрет удаления невыполненной отгрузки

Этот тригер помогает поддерживать целостность базы данных, предотвращая случайное (или злоумышленное) удаление невыполненной отгрузки. Предусмотрен вариант удаления случайно созданной отгрузки, если поле транспорт пусто, тригер позволит удалить отгрузку.

В добавление к тригеру создается исключение, вызываемое при нарушении условий.

```
CREATE EXCEPTION NO_READY_SHIPMENT_DELETE 'Ошибка удаения. Отгрузка еще не совершена';
CREATE TRIGGER tdel_Shipment FOR Shipment BEFORE DELETE AS
BEGIN
    if((OLD.completed<>1) AND (OLD.id_transport IS NOT NULL))
        THEN EXCEPTION NO_READY_SHIPMENT_DELETE;
```

```
END;
```

4.1.3 Подсчет стоимости заказа

Приведенный ниже скрипт создает набор тригеров обеспечивающих подсчет стоимости заказа.

Предполагая, что фирма будет иметь дело с огромными заказами, стоимость не подсчитывается суммированием стоимости позиций при каждой изменении базы. Создается три тригера на добавление, изменение или удаление из таблицы PRODUCTS_IN_ORDER. Тригер на добавление суммирует стоимость добавленной позиции со стоимостью заказа, тригер на удаление вычитает стоимость удаленной позиции, а тригер на изменение вычитает старую стоимость и добавляет новую (срабатывая только на смену продукта или количества, но не заказа).

```
CREATE TRIGGER tins_Products_in_Order FOR Products_in_Order
ACTIVE
AFTER INSERT AS
DECLARE VARIABLE addCost DECIMAL(2);
BEGIN
    SELECT (costPrice*NEW.pcount) FROM Product
        WHERE Product.id_product = NEW.id_product INTO addCost;
    UPDATE Orders SET ncost = (ncost + :addCost);
END;

CREATE TRIGGER tdel_Products_in_Order  FOR Products_in_Order
ACTIVE
AFTER DELETE AS
DECLARE VARIABLE subCost DECIMAL(2);
BEGIN
    SELECT (costPrice*OLD.pcount) FROM Product
        WHERE Product.id_product = OLD.id_product INTO subCost;
    UPDATE Orders SET ncost = (ncost - :subCost);
END;

CREATE TRIGGER tupd_Products_in_Order FOR Products_in_Order
ACTIVE
AFTER UPDATE AS
DECLARE VARIABLE addCost DECIMAL(2);
DECLARE VARIABLE subCost DECIMAL(2);
BEGIN
    IF((OLD.pcount<>NEW.pcount) OR (OLD.id_product<>NEW.id_product) ) THEN
    BEGIN
        SELECT (costPrice*OLD.pcount) FROM Product
```

```
            WHERE Product.id_product = OLD.id_product INTO subCost;
        SELECT (costPrice*NEW.pcount) FROM Product
            WHERE Product.id_product = NEW.id_product INTO addCost;
        UPDATE Orders SET ncost = (ncost - :subCost + :addCost);
    END
END;
```

4.2 Хранимые процедуры

4.2.1 Увольнение сотрудника

Процедура `TRUMP_EMPLOYEE` призвана помочь директору при увольнении сотрудника.

Процедура принимает на входе два параметра `t_empl_id` — номер увольняемого сотрудника и `new_empl_id` — номер сотрудника, которому передаются дела уволенного. Скрипт проходит по таблицам TRANSPORT, SHIPMENT, ORDERS заменяя номер увольняемого на номер нового сотрудника, а затем удаляет старого сотрудника из базы данных.

```
CREATE PROCEDURE TRUMP_EMPLOYEE(t_empl_id INTEGER, new_empl_id INTEGER) AS
BEGIN
 UPDATE TRANSPORT SET id_employee = :new_empl_id WHERE id_employee = :t_empl_id;
 UPDATE SHIPMENT SET id_employee = :new_empl_id WHERE id_employee = :t_empl_id;
 UPDATE ORDERS SET id_employee = :new_empl_id WHERE id_employee = :t_empl_id;

 DELETE FROM ORDERS WHERE id_employee = :t_empl_id;
END;
```

4.2.2 Архивация выполненных заказов

Выполненные заказы продолжают хранится в нашей базе (для разрешения претензий, составления статистики), увеличивая время поиска, вставки и обновления. Поэтому было бы разумным хранить важные данные о выполненных заказах в другой таблице.

Нижеследующих код создает архивную таблицу и процедуру `ARCHIVE_ORDERS` архивирующую заказы.

Процедура принимает на входе параметр `before_date` определяющий дату, до которой следует архивировать выполненные заказы. Дальнейший процесс состоит в выборке нужных заказов, посчете требуемых данных, записи в архивную таблицу и наконец удаления из старой.

```
/*Архивная таблица*/
CREATE TABLE ORDERS_ARCHIVE (
       id_order             INTEGER NOT NULL,
       endDate              DATE,
       ncost                MONEY DEFAULT 0,
       id_employee          INTEGER NOT NULL,
```

```
    id_client           INTEGER NOT NULL,
    pcount              INTEGER NOT NULL
);


CREATE PROCEDURE ARCHIVE_ORDERS(before_date DATE) AS
DECLARE VARIABLE id_order INTEGER;
DECLARE VARIABLE endDate DATE;
DECLARE VARIABLE ncost DECIMAL(2);
DECLARE VARIABLE managerName VARCHAR(50);
DECLARE VARIABLE employeeName VARCHAR(50);
DECLARE VARIABLE pcount INTEGER;
BEGIN
    FOR
    SELECT ord.id_order, endDate, ncost,
        emp.firstName||emp.lastName, cl.firstName||cl.lastName
    FROM ORDERS ord LEFT JOIN EMPLOYEE emp ON emp.id_employee = ord.id_employee
    LEFT JOIN CLIENT cl ON cl.id_client = ord.id_client
    WHERE endDate > :before_date
    INTO :id_order, :endDate, :ncost, :managerName, :employeeName
    DO
    BEGIN
        SELECT SUM(pcount) FROM PRODUCTS_IN_ORDER WHERE id_order = :id_order INTO :pcount;
        INSERT INTO ORDERS_ARCHIVE VALUES
            (:id_order, :endDate, :ncost, :managerName, :employeeName, :pcount);

        DELETE FROM PRODUCTS_IN_ORDER WHERE id_order = :id_order;
        DELETE FROM SHIPMENT WHERE id_order = :id_order;
    END
    DELETE FROM ORDERS WHERE endDate>:before_date;
END;
```

4.2.3 Популярность продуктов

Процедура вывода, позволяющая оценить популярность продукции (по заказам) в процентном соотношении. Получение этих данных с помощью View невозможно по причине необходимости подсчета процентов.

```
CREATE PROCEDURE PRODUCTS_POPULARITY
    RETURNS (name VARCHAR(50), popularity DECIMAL(2)) AS
DECLARE VARIABLE allcount INTEGER;
BEGIN
```

```
    SELECT SUM(pcount) FROM PRODUCTS_IN_ORDER into allcount;
    FOR
        SELECT name, SUM(pcount) FROM PRODUCTS_IN_ORDER
        LEFT JOIN PRODUCT ON
            PRODUCT.id_product = PRODUCTS_IN_ORDER.id_product
        GROUP BY name INTO :name, :popularity
    DO BEGIN
        popularity = popularity / allcount;
        SUSPEND;
    END
END;
```

4.3 «Представления»

4.3.1 Активность менеджеров

Данное представление помогает получить поименный список менеджеров и количества заказов, обслуживаемых каждым из них. Это позволяет директору выявлять загруженность и активность своих сотрудников.

```
CREATE VIEW MANAGERS_ORDERS (name , ordersCount) AS
    SELECT firstName ||' '|| lastName, COUNT(id_order)
        FROM ORDERS LEFT JOIN EMPLOYEE
            ON ORDERS.id_employee = EMPLOYEE.id_employee
        GROUP BY firstName, lastName;
;
```

4.4 Тестовые данные

Для разработки демонстрационного приложения нам потребуются тестовые данные. Данный скрипт производит начальное заполнение базы данных.

```
/* Продукция */
INSERT INTO PRODUCT VALUES (0, 'Стул',10,100);
INSERT INTO PRODUCT VALUES (0, 'Кресло',20,200);
INSERT INTO PRODUCT VALUES (0, 'Мягкое кресло',30,300);
INSERT INTO PRODUCT VALUES (0, 'Табурет',5,50);
INSERT INTO PRODUCT VALUES (0, 'Раскладушка',25, 250);
INSERT INTO PRODUCT VALUES (0, 'Кровать',35,350);
INSERT INTO PRODUCT VALUES (0, 'Кресло-кровать',56,560);
INSERT INTO PRODUCT VALUES (0, 'Диван',89,890);
INSERT INTO PRODUCT VALUES (0, 'Кожаный диван',167,2000);
INSERT INTO PRODUCT VALUES (0, 'Водяная кровать', 60, 1500);
```

```
/*Сотрудники*/
INSERT INTO EMPLOYEE VALUES(0, 'Николай', 'Требушев',
    '+7(904)4444444', '234567', 'Генеральный директор', '23.02.1967');
INSERT INTO EMPLOYEE VALUES(0, 'Петр', 'Власов',
    '+7(904)4444444', '234568', 'Старший менеджер', '23.02.1967');
INSERT INTO EMPLOYEE VALUES(0, 'Станислав', 'Броцкий',
    '+7(904)4444444', '234569', 'Менеджер', '23.02.1967');
INSERT INTO EMPLOYEE VALUES(0, 'Иван', 'Куров',
    '+7(904)4444444', '2345611', 'Менеджер', '23.02.1967');
INSERT INTO EMPLOYEE VALUES(0, 'Виктор', 'Малышев',
    '+7(904)4444444', '2345612', 'Старший техник', '23.02.1967');
INSERT INTO EMPLOYEE VALUES(0, 'Вера', 'Горкова',
    '+7(904)4444444', '2345613', 'Водитель', '23.02.1967');
INSERT INTO EMPLOYEE VALUES(0, 'Слава', 'Роботам',
    '+7(904)4444444', '2345614', 'Водитель', '23.02.1967');

/*Транспорт*/
INSERT INTO TRANSPORT VALUES(0, 0, 'а666дд', 6);
INSERT INTO TRANSPORT VALUES(0, 1, 'м777яс', 7);

/*Клиенты*/
INSERT INTO CLIENT VALUES(0,'Профессора Попова 5', 0,
    '+7(821)4444444', '5555555', 'yuriy@pupkin.info', 'Юрий', 'Киров', NULL);
INSERT INTO CLIENT VALUES(0,'наб. реки Мойки 20', 0,
    '+7(821)4444444', '5555555', 'marya@pupkin.info', 'Марианна', 'Белоусова', NULL);
INSERT INTO CLIENT VALUES(0,'Советовская 34', 0,
    '+7(821)4444444', '5555555', 'kostya@pupkin.info', 'Константин', 'Першин', NULL);

INSERT INTO CLIENT VALUES(0,'Проспект пионеров 122а', 1, '+7(821)4444444',
    '5555555', 'kolebas@rogaikopita.info', 'Григорий', 'Калебас', 'Рога и Копыта');
INSERT INTO CLIENT VALUES(0,'Невский 56', 1, '+7(821)4444444',
    '5555555', 'orders@zlatovlaska.ru', 'Татьяна', 'Крашенникова', 'Златовласка');
INSERT INTO CLIENT VALUES(0,'Проспект просвещения 6', 1, '+7(821)4444444',
    '5555555', NULL, 'Ашот', 'Гигашвили', 'Шаурма сервис');
```

5 Демонстрационное приложение

5.1 Назначение

Реализуемая программа предназначена для управления заказами. Она позволяет добавлять, удалять и модифицировать заказы и их состав.

5.2 Структура программы

Для доступа к базе данных были выбраны компоненты IBExpert, так как в отличие от BDE не нужно устанавливать на компьютер клиента дополнительные библиотеки; компоненты разработаны специально для Inter Base, что гарантирует скорость, стабильность и реализацию большинства возможностей этой СУБД. dbExpress был отброшен сразу, по причине того, что не работает.

Основным компонентом доступа к базе данных является TIBDatabase, настроенный на подключение к СУБД.

Для управления транзакциями (выбрана автоматическая модель управления) используется TIBTransaction.

Для работы пользователя с данными используются компоненты TDBGrid, связанные через TDatasource с компонентами TIBDataset, которые реализуют доступ к базе данных использую заданные пользователем запросы.

Запросы для управления заказами:

Выборка:

```
SELECT id_order AS oid_order, regDate, planDate,
    endDate, ncost, id_employee, id_client
FROM ORDERS
```

Обновление строки:

```
SELECT id_order AS oid_order, regDate, planDate,
    endDate, ncost, id_employee, id_client
FROM ORDERS WHERE id_order = :oid_order
```

Вставка:

```
INSERT INTO
ORDERS (id_order, ncost, regDate, endDate, planDate, id_employee, id_client)
VALUES (:oid_order,0, CURRENT_DATE, :endDate, :planDate, :id_employee, :id_client)
```

Удаление:

```
DELETE FROM ORDERS WHERE id_order = :oid_order
```

Модификация:

```
UPDATE ORDERS SET planDate = :planDate, endDate = :endDate,
    id_employee = :id_employee, id_client = :id_client
WHERE id_order = :oid_order
```

Запросы для управления составом заказа:

Выборка:

```
SELECT id_order, id_product, pcount FROM PRODUCTS_IN_ORDER
WHERE id_order = :oid_order
```

Обновление строки:

```
SELECT id_order, id_product, pcount FROM PRODUCTS_IN_ORDER
WHERE id_order = :id_order AND id_product = :id_product
```

Вставка:

```
INSERT INTO PRODUCTS_IN_ORDER(id_order, id_product,pcount)
VALUES (:id_order, :id_product, :pcount)
```

Удаление:

```
DELETE FROM PRODUCTS_IN_ORDER
WHERE id_product = :id_product AND id_order=:id_order
```

Модификация:

```
UPDATE PRODUCTS_IN_ORDER
SET id_product = :new_id_product, pcount = :new_pcount
WHERE id_product = :old_id_product AND id_order = :old_id_order
```

Для связи между заказами и их составом использовалась связь master/detail (свойство DataSource) по полю id_order.

Для отображения вместо номеров менеджеров, клиентов, продуктов их имен и названий, а также для заполнения списков выбора по этим полям использовались Lookup fields, которые в свою очередь использовали отдельные TIBDataSet для выборки имен и номеров из соответствующих таблиц.

5.3 Изображение главного окна

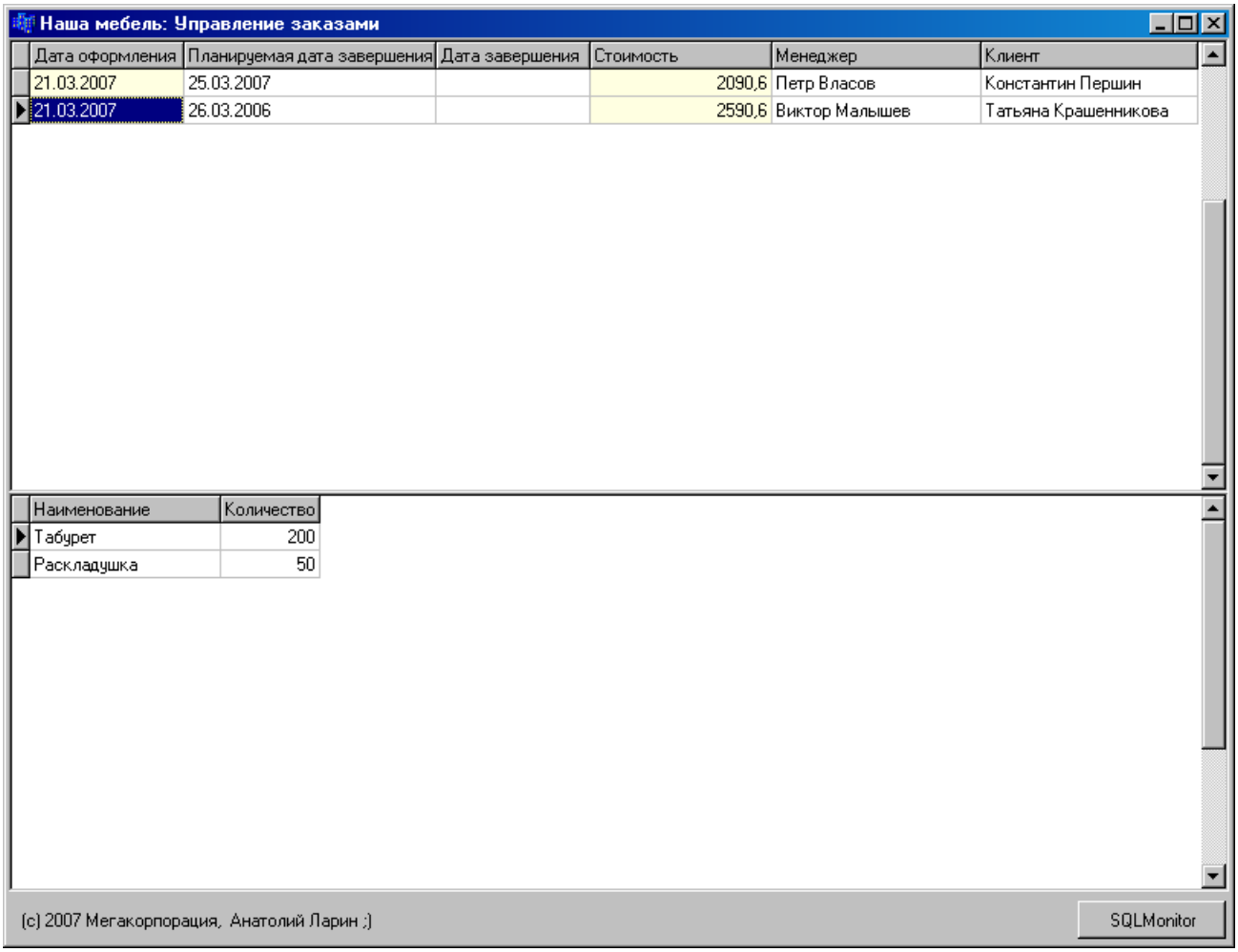

Рис. 4. Изображение главного окна программы.