МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МАГНИТОГОРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМ. Г. И. НОСОВА»
(ФГБОУ ВО «МГТУ ИМ. Г.И. НОСОВА»)

Кафедра вычислительной техники и программирования

КУРСОВАЯ РАБОТА

по дисциплине «Проектирование баз данных и программирование на языках SQL и PL/SQL»

на тему: «Проект розничная торговля»

Исполнитель: Варламов М.Н., студент 3 курса, группа АВб-19-1

Руководитель: Белявский А.Б., доцент кафедры ВТиП

Работа допущена к защите «____» _____2022 г. _____

Работа защищена «____» _____ 2022 г. с оценкой _____

Магнитогорск, 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МАГНИТОГОРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМ. Г.И. НОСОВА»
(ФГБОУ ВО «МГТУ ИМ. Г.И. НОСОВА»)

Кафедра вычислительной техники и программирования

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Тема: «Проект розничная торговля»

Обучающемуся Варламову Максиму Николаевичу

Исходные данные: Необходимо спроектировать базу данных розничная торговля, информация которой будет использоваться для анализа продаж в магазине..

Срок сдачи: «_____» _____ 2022 г.

Руководитель: _____ /Белявский А.Б./

Задание получил: _____ /Варламов М.Н./

Магнитогорск, 2022

Содержание

ВВЕДЕНИЕ

База данных представляет собой определенным образом структурированную совокупность данных, совместно хранящихся и обрабатывающихся в соответствии с некоторыми правилами. Как правило, база данных моделирует некоторую предметную область или ее фрагмент.

Программа, производящая манипуляции с информацией в базе данных, называется СУБД (система управления базами данных). Она может осуществлять выборки по различным критериям и выводить запрашиваемую информацию в том виде, который удобен пользователю. Основными составляющими информационных систем, построенных на основе баз данных, являются файлы БД, СУБД и программное обеспечение (клиентские приложения), позволяющие пользователю манипулировать информацией и совершать необходимые для решения его задач действия.

Структурирование информации производится по характерным признакам, физическим и техническим параметрам абстрактных объектов, которые хранятся в данной базе. Информация в базе данных может быть представлена как текст, растровое или векторное изображение, таблица или объектно-ориентированная модель. Структурирование информации позволяет производить ее анализ и обработку: делать пользовательские запросы, выборки, сортировки, производить математические и логические операции.

Для различных задач используются различные типы баз данных. Типы баз данных, называемых также моделями БД или семействами БД, представляют собой шаблоны и структуры, используемые для организации данных в системе управления базами данных (СУБД). Выбор типа повлияет на то, какие операции может выполнять приложение, как будут представлены данные, на функции СУБД для разработки и рантайма.

1 РЕЛЯЦИОННАЯ БАЗА ДАННЫХ

Реляционная база данных – это набор данных с предопределенными связями между ними. Эти данные организованы в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных. В каждом столбце таблицы хранится определенный тип данных, в каждой ячейке – значение атрибута. Каждая строка таблицы представляет собой набор связанных значений, относящихся к одному объекту или сущности. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с помощью внешних ключей.

1.1 Постановка задания

Магазин розничной торговли продает персональные компьютеры, средства связи и периферийное оборудование: принтеры, накопители CD-RW и др. Необходимо спроектировать базу данных РОЗНИЧНАЯ ТОРГОВЛЯ, информация которой будет использоваться для анализа продаж в магазине.

В БД должна храниться информация:

- О товарах: код товара, наименование товара, дата поступления в магазин, количество товара, цена закупки (руб.);
- О поставщиках товаров: код поставщика, наименование поставщика, адрес, телефон, к кому обращаться;
- О продажах товаров в магазине: код продажи, код товара, дата продажи, количество проданного товара (шт.), цена розничная (руб.).

При проектировании БД необходимо учитывать следующее:

- Поставщик поставляет несколько товаров. Товар поступает на склад магазина от нескольких поставщиков;
- Товар имеет несколько продаж. Продажа относится к одному товару.

Кроме того следует учесть:

- Поставщик не обязательно поставляет товар (может временно не работать). Каждый товар обязательно поставляется;
- Если товар на складе кончился. Делается заказ к поставщику
- Товар на складе ограничен;
- Товар не обязательно продается. Каждая продажа обязательно связана с товаром.

1.2 Реализация реляционной БД

Реализация реляционной базы данных производилась с помощью инструмента “Oracle SQL data modeler”.

На рисунке 1 изображена логическая схема базы данных.

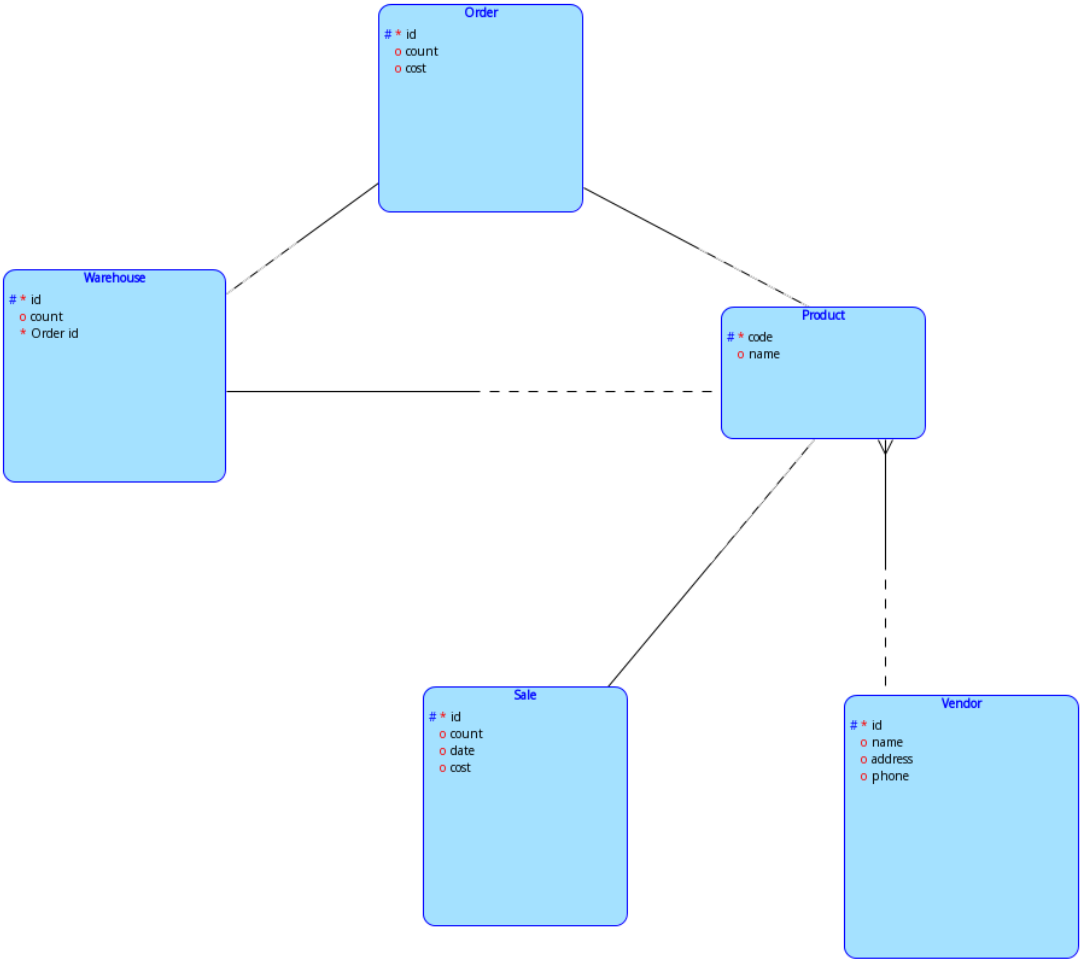


Рисунок 1 - Логическая схема реляционной базы данных.

На рисунке 2 изображена реляционная схема разработанной базы данных.

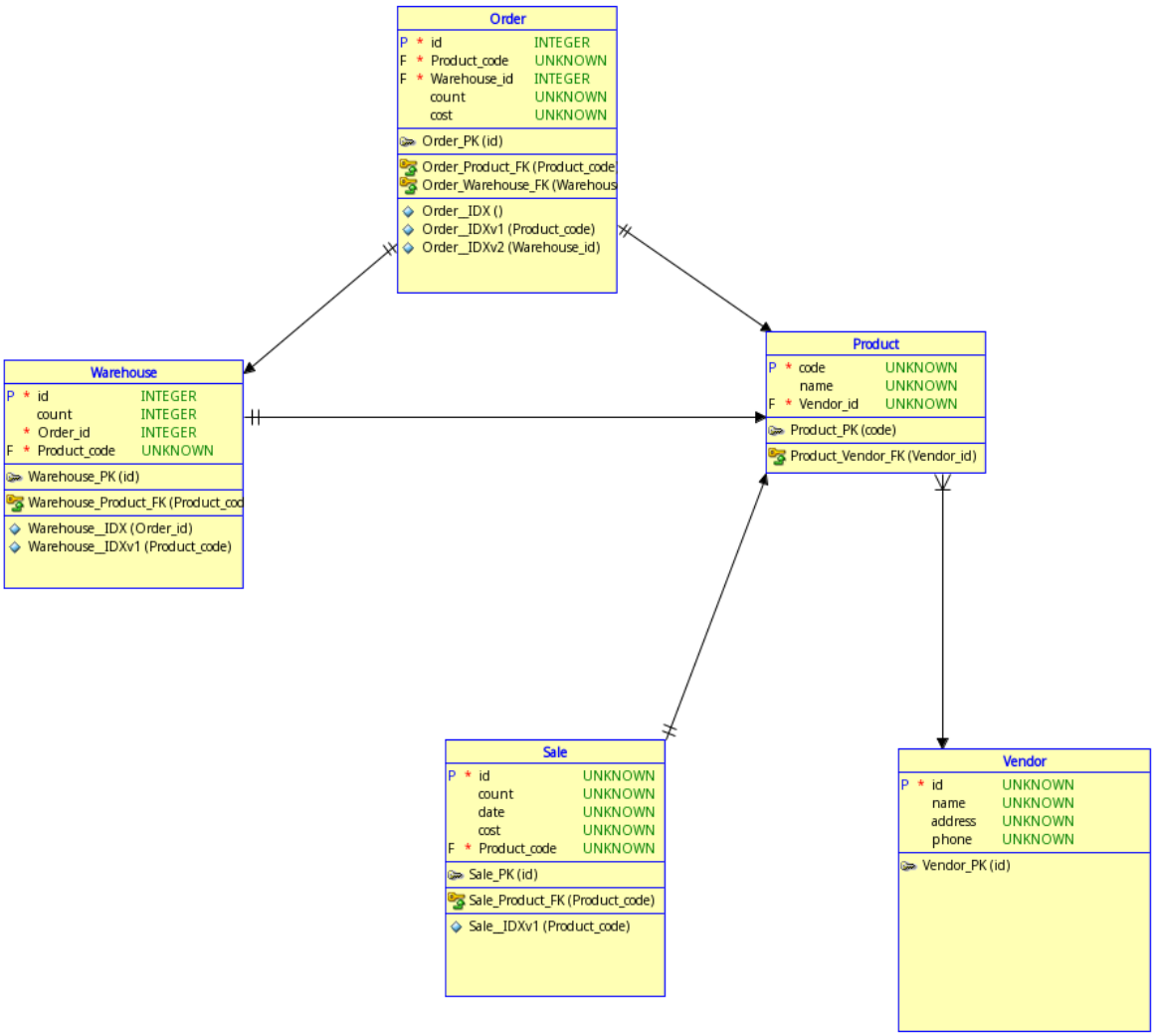


Рисунок 2 - Реляционная схема разработанной базы данных.

Для создания основной модели, указанной в постановке были созданы таблицы:

- Product;
- Vendor;
- Sale.

В каждой таблице присутствуют поля, соответствующие постановке.

Для выполнения условий постановки были созданы дополнительные таблицы:

- Order;
- Warehouse.

Данные таблицы выполняют требования заказа товара и его хранения на складе.

Также, некоторым таблицам, были добавлены следующие поля:

- Product.vendorId - для определения товаров, которые может поставлять поставщик;
- Sale.productId - для определения нескольких продаж одного товара и связи продажи с товаром;
- Warehouse.count - для определения количества товара на складе;
- Order.productId - для заказа определенного товара.

Исходный код создания таблиц приведен в листинге ниже.

```
CREATE TABLE "Order" (
    id           INTEGER NOT NULL,
    product_code  INTEGER NOT NULL,
    warehouse_id  INTEGER NOT NULL,
    count        INTEGER,
    cost         INTEGER
);
CREATE UNIQUE INDEX order__idx ON
    "Order" (
      id
    ASC );

ALTER TABLE "Order" ADD CONSTRAINT order_pk PRIMARY KEY ( id
);

CREATE TABLE product (
    code    INTEGER NOT NULL,
    name    VARCHAR2(4000),
    vendor_id  INTEGER NOT NULL
);
```

```
    ALTER TABLE product ADD CONSTRAINT product_pk PRIMARY KEY (
code );

    CREATE TABLE sale (
        id          INTEGER NOT NULL,
        count       INTEGER,
        "date"      DATE,
        cost        INTEGER,
        product_code  INTEGER NOT NULL
    );

    CREATE UNIQUE INDEX sale__idxv1 ON
        sale (
          product_code
        ASC );

    ALTER TABLE sale ADD CONSTRAINT sale_pk PRIMARY KEY ( id );

    CREATE TABLE vendor (
        id     INTEGER NOT NULL,
        name   VARCHAR2(4000),
        address  VARCHAR2(4000),
        phone  VARCHAR2(4000)
    );

    ALTER TABLE vendor ADD CONSTRAINT vendor_pk PRIMARY KEY ( id
);

    CREATE TABLE warehouse (
        id          INTEGER NOT NULL,
        count       INTEGER,
        order_id    INTEGER NOT NULL,
        product_code  INTEGER NOT NULL
```

```
);

CREATE UNIQUE INDEX warehouse__idx ON
    warehouse (
      order_id
    ASC );

CREATE UNIQUE INDEX warehouse__idxv1 ON
    warehouse (
      product_code
    ASC );

ALTER TABLE warehouse ADD CONSTRAINT warehouse_pk PRIMARY
KEY ( id );

ALTER TABLE "Order"
    ADD CONSTRAINT order_product_fk FOREIGN KEY (
product_code )
      REFERENCES product ( code );

ALTER TABLE "Order"
    ADD CONSTRAINT order_warehouse_fk FOREIGN KEY (
warehouse_id )
      REFERENCES warehouse ( id );

ALTER TABLE product
    ADD CONSTRAINT product_vendor_fk FOREIGN KEY ( vendor_id
)
      REFERENCES vendor ( id );

ALTER TABLE sale
    ADD CONSTRAINT sale_product_fk FOREIGN KEY ( product_code
)
```

```
        REFERENCES product ( code );

    ALTER TABLE warehouse
        ADD CONSTRAINT warehouse_product_fk FOREIGN KEY ( product_code )
        REFERENCES product ( code );

    -- Oracle SQL Developer Data Modeler Summary Report:
    --
    -- CREATE TABLE                             5
    -- CREATE INDEX                             4
    -- ALTER TABLE                              10
    --
    -- ERRORS                                   0
    -- WARNINGS                                 0
```

2 ТЕМПОРАЛЬНАЯ БАЗА ДАННЫХ

Темпоральные базы данных – это базы данных, хранящие темпоральные данные. Такие БД и содержащиеся в них данные могут рассматриваться как темпоральные только в том случае, если известно правило интерпретации временных меток и интервалов для конкретной системы управления базами данных (СУБД). Чтобы определить, является ли рассматриваемая СУБД темпоральной в полном смысле этого слова, необходимо понять, можно ли отдельно выделить и специальным образом интерпретировать данные атрибута "время".

Время транзакции записывает период времени, в течение которого запись в базе данных принимается как правильная. Это позволяет выполнять запросы, которые показывают состояние базы данных в данный момент времени. Периоды времени транзакций могут возникать только в прошлом или до текущего времени. В таблице времени транзакций записи никогда не удаляются. Только новые записи могут быть вставлены, а существующие обновлены, установив время окончания транзакции, чтобы показать, что они больше не актуальны.

2.1 Постановка задачи

Схема базы данных и постановка задачи схожа с постановкой для реляционной базы данных, но имеет следующие дополнения:

- Продажа осуществляется только при наличии товара
- Регистрация изменения состояния товара на складе (кто и когда изменил)
- В продаже не может быть “мифического числа товаров” (нельзя продать больше, чем есть на складе)

2.2 Реализация темпоральной БД

На рисунке 3 представлена логическая схема темпоральной БД.

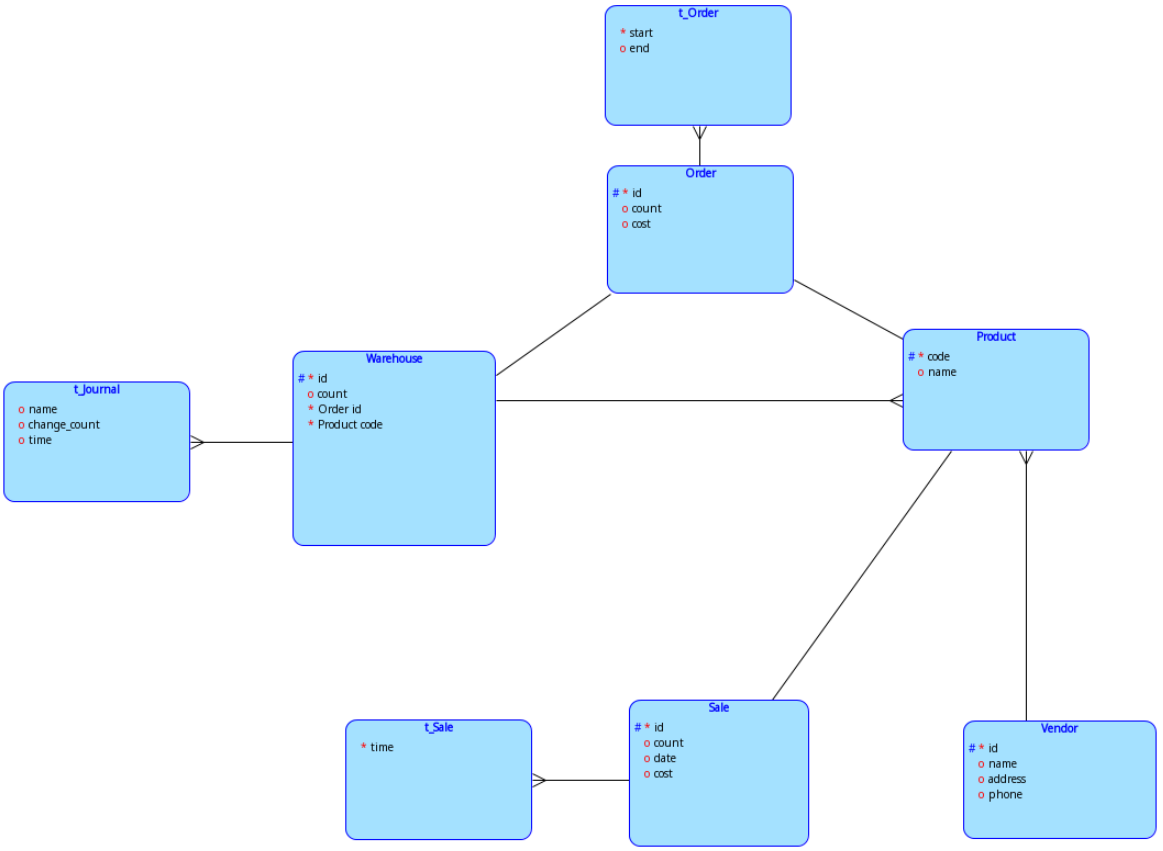


Рисунок 3 - Логическая схема темпоральной БД.

На рисунке 4 представлена реляционная схема темпоральной базы данных.

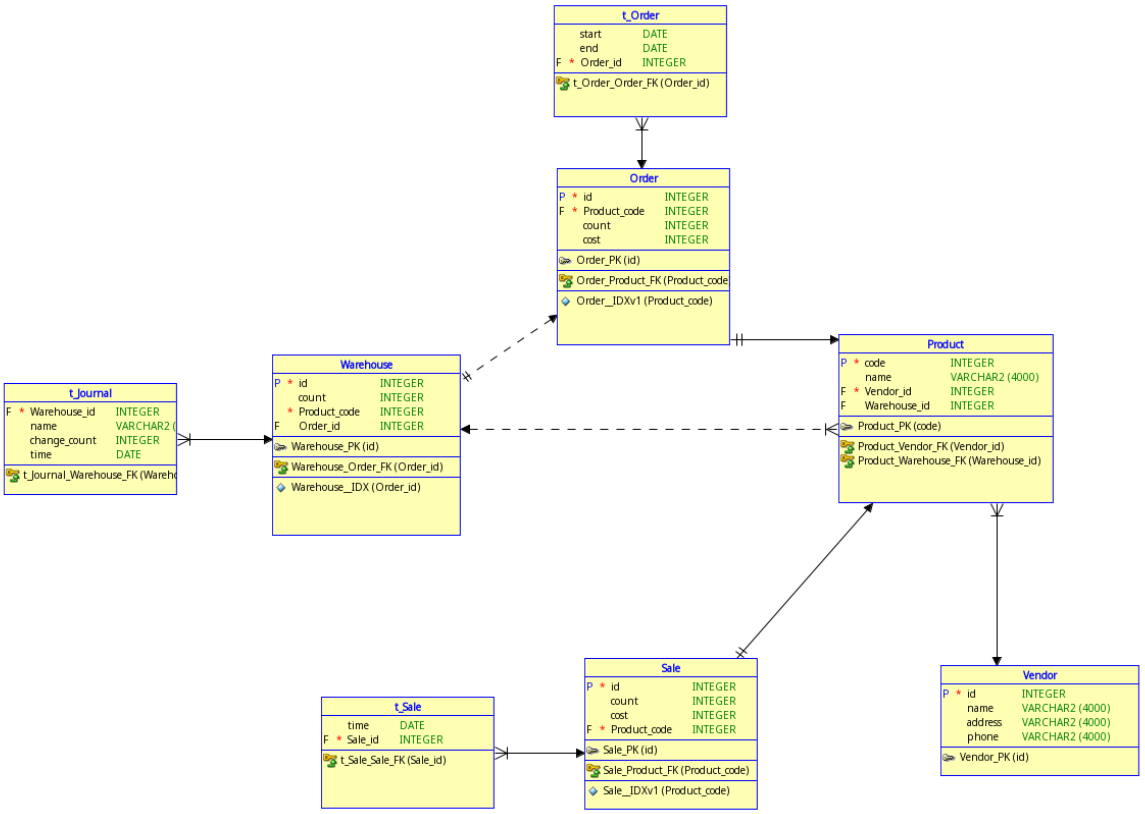


Рисунок 4 - Логическая схема темпоральной базы данных.

Исходя из данных рисунков - можно заметить несколько новых таблиц:

- t_Journal
- t_Sale
- t_Order

Данные таблицы представляют собой сущности, определяющие временные интервалы и метки. Данная информация может быть использована для логирования, регулирования и статистики.

Для реализации автоматического заполнения таблиц и выполнения дополнительных требований в постановке задачи были созданы триггеры, представленные в листинге ниже.

```
CREATE OR REPLACE TRIGGER Trg1
    BEFORE INSERT ON Sale REFERENCING
    NEW AS new
IF (:new.count > (SELECT count FROM Warehouse WHERE
Product_code = :new.Product_code))
INSERT INTO t_Sale (Order_id) VALUES (:new.id) ;
```

```
/

CREATE OR REPLACE TRIGGER Trg2
    BEFORE INSERT ON "Order"
    FOR EACH ROW
INSERT INTO t_Order (Order_id) VALUES (:new.id) ;
/

CREATE OR REPLACE TRIGGER Trg3
    BEFORE INSERT OR UPDATE ON Warehouse
    FOR EACH ROW
INSERT INTO t_journal (Warehouse_id, name, change_count,
time) VALUES (:new.id,  USER, :new.count - nvl(:old.count, 0),
CURRENT_DATE) ;
/
```

Данные триггеры имеют следующее назначение:

- Tgr1 - Выполняет проверочную функцию при создании заказа. Это нужно, чтобы пользователь бд не смог продать больше количества товара на складе.
- Tgr2 - Выполняет автоматизирующую функцию - задает транзакционное время продаже товара.
- Tgr3 - Выполняет логирующую функцию - сохраняет изменения количества товара на складе, записывает пользователя, который это кол-во изменил, а также транзакционное время.

Полный листинг скрипта создания базы данных представлен ниже.

```
CREATE TABLE "Order" (
   id           INTEGER NOT NULL,
   product_code  INTEGER NOT NULL,
   count        INTEGER,
   cost         INTEGER
)
```

```
LOGGING;

CREATE UNIQUE INDEX order__idxv1 ON
   "Order" (
     product_code
   ASC )
     LOGGING;

ALTER TABLE "Order" ADD CONSTRAINT order_pk PRIMARY KEY (
id );

CREATE TABLE product (
   code         INTEGER NOT NULL,
   name         VARCHAR2(4000),
   vendor_id    INTEGER NOT NULL,
   warehouse_id  INTEGER
)
LOGGING;

ALTER TABLE product ADD CONSTRAINT product_pk PRIMARY KEY (
code );

CREATE TABLE sale (
   id           INTEGER NOT NULL,
   count        INTEGER,
   cost         INTEGER,
   product_code  INTEGER NOT NULL
)
LOGGING;

CREATE UNIQUE INDEX sale__idxv1 ON
   sale (
     product_code
```

```
   ASC )
     LOGGING;

ALTER TABLE sale ADD CONSTRAINT sale_pk PRIMARY KEY ( id );

CREATE TABLE t_journal (
   warehouse_id  INTEGER NOT NULL,
   name         VARCHAR2(4000),
   change_count  INTEGER,
   time         DATE
)
LOGGING;

CREATE TABLE t_order (
   "start"   DATE,
   end    DATE,
   order_id  INTEGER NOT NULL
)
LOGGING;

CREATE TABLE t_sale (
   time   DATE,
   sale_id  INTEGER NOT NULL
)
LOGGING;

CREATE TABLE vendor (
   id     INTEGER NOT NULL,
   name   VARCHAR2(4000),
   address  VARCHAR2(4000),
   phone  VARCHAR2(4000)
)
LOGGING;
```

```
ALTER TABLE vendor ADD CONSTRAINT vendor_pk PRIMARY KEY (
id );

CREATE TABLE warehouse (
   id           INTEGER NOT NULL,
   count        INTEGER,
   product_code  INTEGER NOT NULL,
   order_id     INTEGER
)
LOGGING;

CREATE UNIQUE INDEX warehouse__idx ON
   warehouse (
     order_id
   ASC )
     LOGGING;

ALTER TABLE warehouse ADD CONSTRAINT warehouse_pk PRIMARY
KEY ( id );

ALTER TABLE "Order"
   ADD CONSTRAINT order_product_fk FOREIGN KEY (
product_code )
     REFERENCES product ( code )
   NOT DEFERRABLE;

ALTER TABLE product
   ADD CONSTRAINT product_vendor_fk FOREIGN KEY ( vendor_id
)
     REFERENCES vendor ( id )
   NOT DEFERRABLE;
```

```
ALTER TABLE product
   ADD CONSTRAINT product_warehouse_fk FOREIGN KEY ( warehouse_id )
     REFERENCES warehouse ( id )
   NOT DEFERRABLE;

ALTER TABLE sale
   ADD CONSTRAINT sale_product_fk FOREIGN KEY ( product_code )
     REFERENCES product ( code )
   NOT DEFERRABLE;

ALTER TABLE t_journal
   ADD CONSTRAINT t_journal_warehouse_fk FOREIGN KEY ( warehouse_id )
     REFERENCES warehouse ( id )
   NOT DEFERRABLE;

ALTER TABLE t_order
   ADD CONSTRAINT t_order_order_fk FOREIGN KEY ( order_id )
     REFERENCES "Order" ( id )
   NOT DEFERRABLE;

ALTER TABLE t_sale
   ADD CONSTRAINT t_sale_sale_fk FOREIGN KEY ( sale_id )
     REFERENCES sale ( id )
   NOT DEFERRABLE;

ALTER TABLE warehouse
   ADD CONSTRAINT warehouse_order_fk FOREIGN KEY ( order_id )
     REFERENCES "Order" ( id )
   NOT DEFERRABLE;
```

```
CREATE OR REPLACE TRIGGER Trg2
   BEFORE INSERT ON Sale REFERENCING
   NEW AS new
IF (:new.count > (SELECT count FROM Warehouse WHERE
Product_code = :new.Product_code))
INSERT INTO t_Sale (Order_id) VALUES (:new.id) ;
/

CREATE OR REPLACE TRIGGER Trg4
   BEFORE INSERT ON "Order"
   FOR EACH ROW
INSERT INTO t_Order (Order_id) VALUES (:new.id) ;
/

CREATE OR REPLACE TRIGGER Trg5
   BEFORE INSERT OR UPDATE ON Warehouse
   FOR EACH ROW
INSERT INTO t_journal (Warehouse_id, name, change_count,
time) VALUES (:new.id,  USER, :new.count - nvl(:old.count,
0), CURRENT_DATE) ;
/

-- Oracle SQL Developer Data Modeler Summary Report:
--
-- CREATE TABLE                                8
-- CREATE INDEX                                3
-- ALTER TABLE                                 13
-- CREATE TRIGGER                              3
--
-- ERRORS                                      0
-- WARNINGS                                    0
```

3 МНОГОМЕРНАЯ БАЗА ДАННЫХ

Многомерной называют базу данных, в которой данные организованы не в виде множества связанных двумерных таблиц, как в реляционных структурах, а в виде упорядоченных многомерных массивов. В таких базах данных многомерность реализуется на физическом уровне, а не эмулируется реляционными структурами типа звезда или снежинка, которые обеспечивают многомерность на логическом уровне.

В многомерных СУБД информация является логически целостной. Это уже не просто наборы строковых и числовых значений, которые в случае реляционной модели нужно получать из различных таблиц, а целостные структуры типа «кому, что и в каком количестве было продано на данный момент времени». Преимуществами многомерных баз данных являются:

- Поиск и извлечение данных производится значительно быстрее, чем в реляционных базах, поскольку многомерная база данных денормализована и содержит заранее вычисленные агрегаты;
- Более простая процедура встраивания функций в многомерную модель данных;
- Стоимость поддержки многомерной базы данных в среднем ниже, чем у реляционной.

В качестве недостатков многомерных баз данных можно выделить:

- Сложность изменения структуры данных;
- Неэффективное использование памяти.

3.1 Постановка задачи

Основной моделью для многомерной базы данных является модель их постановки для темпоральной базы данных.

Помимо этого необходимо организовать многомерность следующим образом: Разработать многомерную базу данных, основанную на факте продаже продукта.

Измерениями являются:

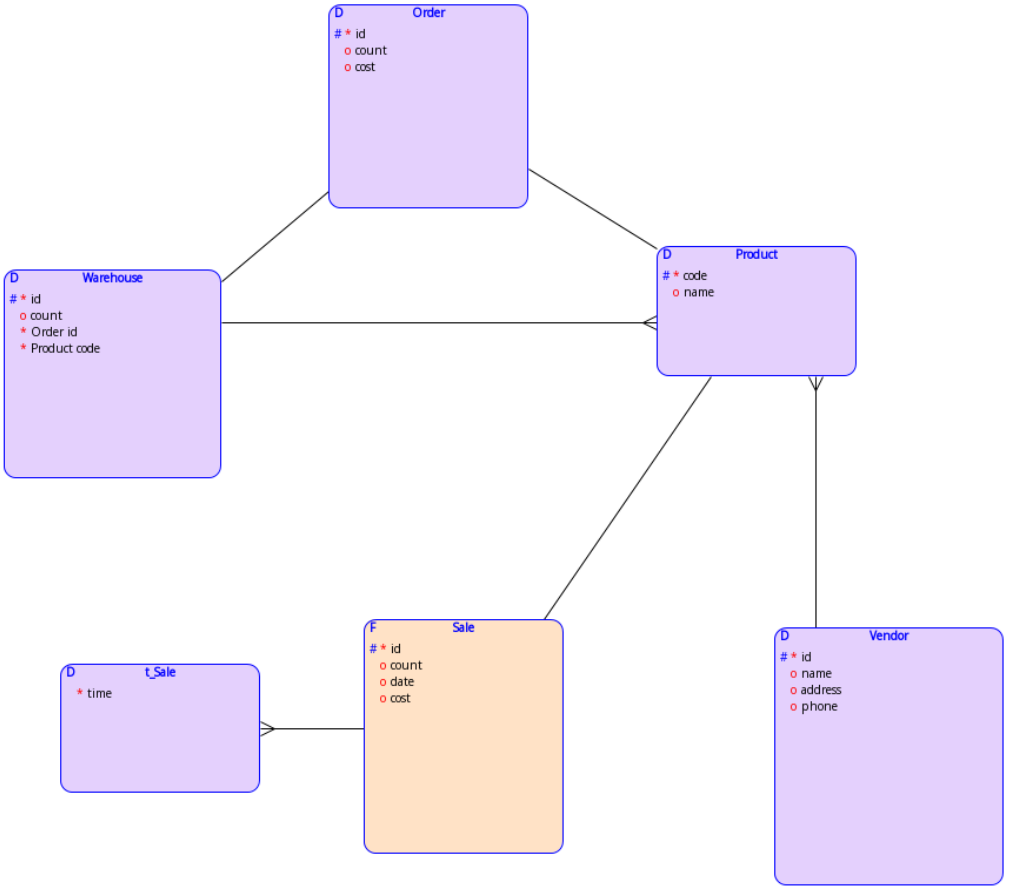
- Время продажи продукта.
- Проданный продукт.

Иерархия проданного продукта:

- Проданный продукт
    - Поставщик
    - Склад
        - Заказ

3.2 Реализация многомерной БД

На рисунке 5 представлена логическая модель многомерной базы



данных.

Рисунок 5 - Логическая модель многомерной базы данных.

На рисунке 6 представлена многомерная модель разработанной базы данных.

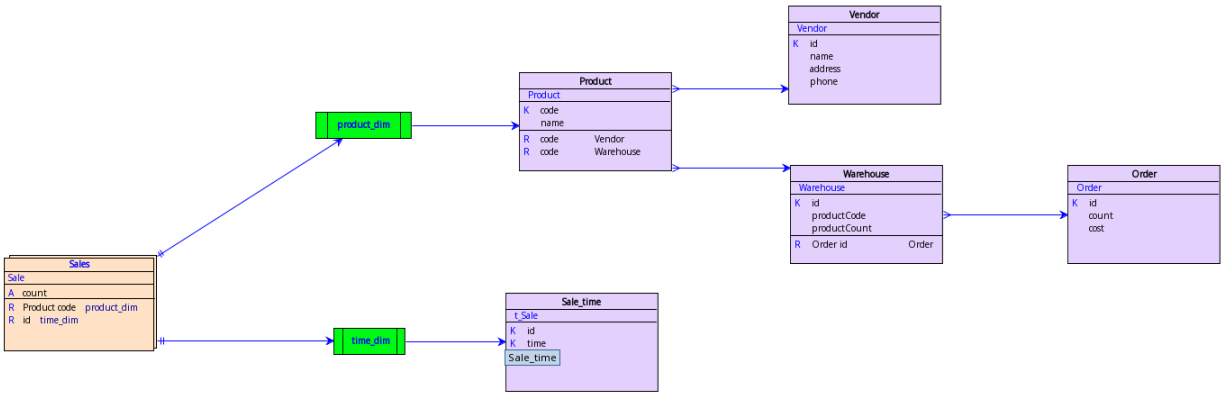


Рисунок 6 - Многомерная модель разработанной базы данных.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы мы научились графически разрабатывать базы данных с помощью инструмента “Oracle SQL data modeler”

Была разработанная модель базы данных “розничная торговля”, информация которой будет использоваться для анализа продаж в магазине.

Данная база данных была разработана в трех видах:

- Реляционная;
- Темпоральная;
- Многомерная.

Для реализации реляционной базы данных были использованы стандартные возможности реляционных БД для контроля логической составляющей базы данных. В их число входят:

- Первичные ключи
- Вторичные ключи
- Связь один к одному
- Связь один ко многим

Для реализации темпоральной базы данных были использованы триггеры, которые выполняют проверочную и автоматизирующую функции во время внесения данных и обновления таблиц. Помимо этого была расширена стандартная модель постановки разработки базы данных. Были добавлены новые таблицы, отвечающие за временные интервалы и метки.

Для реализации многомерной базы данных были использованы стандартные методы разработки многомерных баз данных. В их число входят:

- Пространства
- Иерархии