ЛАБОРАТОРНАЯ РАБОТА № 2
Язык SQL. Генераторы. Функции. Триггеры

1.1 Цель работы

Выработать у обучающихся практические навыки по работе с реляционными базами данных, ознакомить с принципом работы генераторов, функций и триггеров.

1.2 Основные положения

1.2.1 Генераторы

Генератор – объект базы данных, служащий для генерации последовательностей целых чисел. Во многих таблицах используются искусственные первичные ключи. Например, в таблице «Должность» два поля – «Номер должности» (первичный ключ) и «Наименование должности». В данном случае первичный ключ – искусственный, он не является характеристикой сущности «Должность», а был введен специально (искусственно) для того, чтобы не сносить текстовое поле в дочерние таблицы. В случае применения искусственного ключа по натуральному ключу «Наименование должности» должен быть создан альтернативный ключ.

Каждый раз, занося новую строку в таблицу «Должность», в качестве номера первичного ключа надо использовать следующее целое число. Это можно сделать двумя способами:

1. Сначала запросом получить максимальный номер должности в таблице, затем прибавить к нему единицу и затем занести новую строку.
2. Воспользоваться генератором, который подставит следующее значение автоматически.

Первый подход хоть и неудобен, но работает в однопользовательских базах. В многопользовательских базах нет гарантии, что два клиента одновременно не сгенерируют одинаковый номер. Этого недостатка лишен генератор – каждый клиент получит разные номера.

Создание генератора. Для создания генератора служит команда CREATE SEQUENCE. После создания, генератор имеет значение 1. **Синтаксис оператора:**

СУБД Firebird 4.0:

```
CREATE {SEQUENCE | GENERATOR} seq_name
[START WITH start_value] [INCREMENT [BY] increment]
```

Здесь ***seq_name*** – имя генератора. Имя генератора, как правило, содержит имя поля, для которого он предназначен. Например, генератор для поля TaskID может называться TaskID_GEN. ***start_value*** – начальное значение последовательности (генератора); ***increment*** – шаг приращения (по умолчанию равно 1).

Оператор CREATE SEQUENCE создаёт новую последовательность. Слова SEQUENCE и GENERATOR являются синонимами. Вы можете использовать любое из них, но рекомендуется использовать SEQUENCE.

СУБД PostgreSQL:

```
CREATE SEQUENCE [ IF NOT EXISTS ] имя
    [ INCREMENT [ BY ] шаг ]
        [ START [ WITH ] начало ]
```

Здесь ***имя*** – имя генератора. Имя генератора, как правило, содержит имя поля, для которого он предназначен. Например, генератор для поля TaskID может называться TaskID_GEN. ***шаг*** – необязательное предложение INCREMENT BY шаг определяет, какое число будет добавляться к текущему значению последовательности для получения нового значения. С положительным шагом последовательность будет возрастающей, а с отрицательным – убывающей. Значение по умолчанию: 1; ***начало*** – необязательное предложение START WITH начало позволяет запустить последовательность с любого значения.

Для того чтобы установить генератор в другое значение, необходимо использовать команду ALTER SEQUENCE:

```
ALTER SEQUENCE name RESTART WITH value;
```

Здесь name - имя генератора, value - новое значение генератора (число в диапазоне от -264 до 264-1).

В СУБД Firebird 4.0 также есть команда RECREATE SEQUENCE, которая позволяет создать или пересоздать последовательность (генератор).

Использование генераторов. Генератор представляет собой переменную для хранения некоторого текущего значения. После создания генератора место под переменную отведено, генератору присвоено начальное значение. Для того чтобы получить следующее значение генератора, необходимо применить функцию:

СУБД Firebird 4.0:

```
GEN_ID(name, step);
```

Здесь name – имя генератора, step – инкремент генератора.

Функция GEN_ID может быть использована для ввода значений в таблицу, в триггере или хранимой процедуре. Пример использования функции:

```
INSERT INTO Subject (SubjectID, SubjectName)
       VALUES (GEN_ID(SubjectID_GEN,1),»OBD»);
```

СУБД PostgreSQL:

nextval («name») – получение следующего значения генератора.

setval («name»,n) – присваиваетчисло n текущему значению заданной последовательности. Следующий вызов nextval() возвращает значение n+приращение, где приращение – изменение текущего значения последовательности при каждой итерации.

```
INSERT INTO distributors
         VALUES (nextval(«serial»), «nothing»);
```

Генератор возвращает 64-битовое значение. Столбец, в котором сохраняется значение генератора, должен быть соответствующего типа (DECIMAL или NUMERIC). В процедуре переменная для сохранения значения генератора должна быть типа ISC_INT64.

Удаление генератора. Для того, чтобы удалить генератор необходимо использовать команду DROP SEQUENCE.

```
DROP SEQUENCE EMP_NO_GEN;
```

1.2.2 Функции

Функция является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой функции могут обращаться хранимые процедуры, хранимые функции (в том числе и сама к себе), триггеры и клиентские программы. В отличие от хранимых процедур хранимые функции всегда возвращают одно скалярное значение. Для возврата значения из хранимой функции используется оператор RETURN, который немедленно прекращает выполнение функции.

Оператор **CREATE FUNCTION** создаёт новую хранимую функцию. Имя хранимой функции должно быть уникальным среди имён всех хранимых функций и внешних функций.

СУБД Firebird 4.0:

```
CREATE FUNCTION funcname [(<inparam> [, <inparam> ...])]
  RETURNS <type> [COLLATE collation][DETERMINISTIC]
  AS
BEGIN
END;
```

Здесь **create function funcname** – создает функцию, с заданным именем и параметрами; **returns <type>** – тип данных, который возвращает функция; блок кода **[begin – end]** – содержит всю логику функции; **begin** – указывает на начало запросов; **end** – указывает конец функции. Необязательное предложение **DETERMINISTIC** указывает, что функция детерминированная.

СУБД PostgreSQL:

```
CREATE[or REPLACE]FUNCTION function_name(param_list)
   RETURNS return_type
     LANGUAGE plpgsql
  AS
$$
DECLARE
BEGIN
END;
$$
```

Здесь **create [or replace] function имя_функции** – создает или заменяет функцию, если она существует, с заданным именем и параметрами; **returns return_type** – тип данных, который возвращает функция; **язык plpgsql** – указывает на процедурное расширени; внутри знака **$** является телом функции; **declare** – показывает, как объявляются или инициализируются переменные; блок кода **[begin – end]** – содержит всю логику функции; **begin** – указывает на начало запросов; **end** – указывает конец функции.

ALTER FUNCTION изменяет определение функции. Выполнить ALTER FUNCTION может только владелец соответствующей функции. Чтобы сменить схему функции, необходимо также иметь право CREATE в новой схеме.

DROP FUNCTION удаляет определение существующей функции. Пользователь, выполняющий эту команду, должен быть владельцем функции.

СУБД Firebird 4.0:

```
DROP FUNCTION funcname
```

Если от хранимой функции существуют зависимости, то при попытке удаления такой функции будет выдана соответствующая ошибка.

Здесь **funcname** – имя хранимой функции.

СУБД PostgreSQL:

```
      DROP FUNCTION [ IF EXISTS ] имя [ ( [ [ режим_аргумента ] [
имя_аргумента ] тип_аргумента [, ...] ] ) ] [, ...]
          [ CASCADE | RESTRICT ]
```

Помимо имени функции требуется указать типы её аргументов, так как в базе данных могут существовать несколько функций с одним именем, но с разными списками аргументов.

Здесь **имя** – имя существующей функции (возможно, дополненное схемой). Если список аргументов не указан, имя функции должно быть уникальным в её схеме; **режим_аргумента** – режим аргумента: IN, OUT, INOUT или VARIADIC. По умолчанию подразумевается IN. Заметьте, что DROP FUNCTION не учитывает аргументы OUT, так как для идентификации функции нужны только типы входных аргументов. Поэтому достаточно перечислить только аргументы IN, INOUT и VARIADIC; **тип_аргумента** – тип данных аргументов функции (возможно, дополненный именем схемы), если таковые имеются; **CASCADE** – автоматически удалять объекты, зависящие от данной функции (например, операторы или триггеры), и, в свою

очередь, все зависящие от них объекты; **RESTRICT** – отказать в удалении функции, если от неё зависят какие-либо объекты (это поведение по умолчанию).

1.2.3 Триггеры

Триггер – процедура, автоматически вызываемая при операциях с таблицей. Под операциями понимаются операторы INSERT, UPDATE, DELETE. Каждый триггер может быть вызван до соответствующей операции или после нее. Преимущества использования триггеров:

- автоматическое отслеживание целостности данных не только на уровне связи между таблицами, но и любым произвольным образом;
- облегчение написания приложений БД и их поддержки.

1.2.3.1 СУБД Firebird 4.0

Синтаксис оператора CREATE TRIGGER. Оператор состоит из заголовка триггера и тела триггера. Заголовок содержит:

- имя триггера, уникальное по БД;
- имя таблицы, с которой ассоциируется триггер;
- указание на момент, когда триггер должен вызываться.

Тело триггера состоит из опционального списка локальных переменных и операторов.

Синтаксис оператора:

```
CREATE TRIGGER name FOR { table | view}
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} {DELETE | INSERT | UPDATE}
  [POSITION number]
  AS < trigger_body>
< trigger_body> = [<variable_declaration_list>] < block>
< variable_declaration_list> = DECLARE VARIABLE variable datatype;
[DECLARE VARIABLE variable datatype; .]
< block> =
BEGIN
< compound_statement> [<compound_statement> .]
END
< compound_statement> = {<block> | statement;}
```

Используемые обозначения в синтаксисе оператора приведены в таблице 1.1.

Заголовок триггера. Все, что идет до части **AS** оператора **CREATE TRIGGER** составляет заголовок триггера. Заголовок триггера должен определять имя триггера и имя ассоциированной с триггером таблицы или пользовательского представления. Таблица или пользовательское представление должны существовать на момент выполнения оператора **CREATE TRIGGER**. Оставшаяся часть оператора определяет, когда и как вызывается триггер:

- статус триггера, ACTIVE or INACTIVE. Если триггер активный, он вызывается при наступлении события триггера. Если триггер неактивный, он не вызывается.
- время вызова триггера: BEFORE (до) или AFTER (после) некоторого действия.
- операция, с которой связан триггер: INSERT, UPDATE, или DELETE. Может быть указана только одна операция. Если один и тот же триггер должен выполнятся на несколько операций, то необходимо создать несколько триггеров с одинаковым телом, различающихся именем и операцией.
- (опционально) номер триггера по порядку среди всех триггеров, ассоциированных с данной операцией на данной таблице - POSITION. Номер позиции может быть любым числом от 0 до 32767. Значение по умолчанию - 0. Триггеры с меньшими номерами вызываются раньше.

Тело триггера. Все, что следует за частью **AS** оператора **CREATE TRIGGER** составляет тело триггера. Тело триггера состоит из опционального списка локальных переменных, за которым идет блок операторов. Блок состоит из набора операторов на языке хранимых процедур и триггеров, заключенных в операторные скобки.

Таблица 1.1 – Используемые обозначения

name	Имя триггера. В имени обычно упоминают имя таблицы и момент запуска триггера. Например, WORKER_BI_T1 - триггер #1 таблицы WORKER, вызываемый перед добавлением (before insert).
table	Имя таблицы или пользовательского представления, с которым ассоциируется триггер.
ACTIVE|INACTIVE	Указывает «активность» триггера. Не активные триггеры игнорируются. По умолчанию триггер активен.
BEFORE|AFTER	Указывает момент запуска триггера до операции (BEFORE) или после операции (AFTER).
DLETE|INSERT|UPDATE	Указывает операцию, с которой связан триггер.
POSITION number	Указывает порядок срабатывания триггера. На одно и то же действие могут быть назначены несколько триггеров. В таком случае они должны различаться позициями. Триггеры вызываются в порядке увеличения позиции. Позиции триггеров не обязательно должны идти строго друг за другом (1, 2, 3) допустимы произвольные позиции (5, 25, 37). Триггеры с одинаковыми позициями вызываются по алфавиту, в соответствии с именами.
DECLARE VARIABLE var <datatype>	Описывает локальную переменную триггера. С помощью DECLARE VARIABLE можно описать только одну переменную, за оператором должна следовать точка с запятой. Var - имя локальной переменной, должно быть уникально для триггера. <datatype> - тип данных переменной
statement	Любой одиночный оператор на языке процедур и триггеров Interbase. Любой оператор за исключением BEGIN END должен заканчиваться точкой с запятой.
terminator	Разделитель, определенный оператором SET TERM. Разделитель служит для указания конца тела триггера. Используется только в isql.

Контекстные переменные NEW и OLD. Контекстная переменная OLD содержит текущие или старые значения элементов строки, которая обновляется или удаляется. OLD не используется для операций добавления (INSERT). Контекстная переменная NEW содержит новые значения столбцов строки, которые будут присвоены после операции добавления или обновления. NEW не имеет смысла для операции удаления. Контекстные переменные, как правило, используются для сравнения нового и старого значения столбца до его изменения и после.

Синтаксис контекстных переменных:

```
NEW.column
OLD.column
```

где column - имя столбца.

Контекстные переменные могут использоваться так же, как и простые локальные переменные.

Новое значение для столбца может быть получено только до операции. В триггере, который вызывается после операции INSERT, присвоение переменной NEW не имеет смысла. При проведении операции сначала вызываются триггеры BEFORE, они «видят» старое значение строки, затем производится действие, после чего вызываются триггеры AFTER, которые «видят» новое значение строки. Триггеры, описанные как BEFORE и обращающиеся к NEW не имеют смысла.

Например, **рассмотрим триггер**:

```
SET TERM !! ;
CREATE TRIGGER SAVE_SALARY_CHANGE FOR EMPLOYEE
 AFTER UPDATE AS
 BEGIN
    IF (old.salary <> new.salary) THEN
          INSERT INTO SALARY_HISTORY
(EMP_NO, CHANGE_DATE, UPDATER_ID, OLD_SALARY, PERCENT_CHANGE)
   VALUES (old.emp_no, «now», USER, old.salary,
     (new.salary - old.salary) * 100 / old.salary);
  END!!
SET TERM ; !!
```

Тригер вызывается после обновления таблицы EMPLOYEE, и сравнивает старую и новую сумму продаж. Если сумма изменилась, триггер производит запись в таблицу SALARY_HISTORY (история продаж).

Модификация триггеров. Для модификации триггеров используется оператор ALTER TRIGGER. С помощью ALTER TRIGGER можно менять тело триггера, заголовок триггера, а также триггер целиком.

Для того чтобы изменить триггер, автоматически создаваемый СУБД для проверки условия CHECK, следует использовать оператор ALTER TABLE.

Синтаксис ALTER TRIGGER:

```
ALTER TRIGGER name
  [ACTIVE | INACTIVE]
  [{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
  [POSITION number]
AS <trigger_body>
```

Синтаксис ALTER TRIGGER похож на CREATE TRIGGER за исключением:

- CREATE заменено на ALTER;
- пропущена часть FOR <имя таблицы>, так как нельзя менять таблицу, с которой ассоциирован триггер;
- оператор должен содержать только те параметры, которые должны быть изменены, ниже приводятся исключения.

Модификация заголовка триггера. При модификации заголовка триггера ALTER TRIGGER требует, чтобы была изменена хотя бы одна позиция после имени триггера. Все характеристики триггера, пропущенные в ALTER TRIGGER сохраняют свои старые значения.

Следующий оператор делает триггер не активным.

```
ALTER TRIGGER SAVE_SALARY_CHANGE INACTIVE;
```

Если меняется момент вызова триггера (BEFORE, AFTER), то обязательно надо указать операцию (UPDATE, INSERT, DELETE).

Следующий оператор делает триггер активным, и меняет момент его запуска на BEFORE UPDATE:

```
ALTER TRIGGER SAVE_SALARY_CHANGE
ACTIVE
BEFORE UPDATE;
```

Модификация тела триггера. Заменить можно только тело триггера целиком. При этом новое описание полностью заменяет ранее существующее. При изменении тела триггера оператор может не содержать заголовочной информации за исключением имени триггера.

Последовательность действий для модификации тела триггера:

- извлечь описание триггера;
- заменить CREATE на ALTER, удалить всю заголовочную информацию после имени триггера до ключевого слова AS;
- модифицировать тело триггера, запустить оператор.

Удаление триггера. Триггер можно удалить командой DROP TRIGGER. Синтаксис оператора:

```
DROP TRIGGER <имя триггера>
```

Триггеры и транзакции. Триггеры работают в контексте транзакции вызвавшего их приложения. В случае если транзакция приложения будет отменена, будут отменены и все действия триггера.

Триггеры и сообщения о событиях. Триггеры могут быть использованы для того, чтобы уведомить приложение о том, что произошло некоторое событие. Для примера

рассмотрим триггер POST_NEW_ORDER, который генерирует событие с именем «NEW_ORDER» в случае если добавляется запись в таблицу SALES.

```
SET TERM !! ;
CREATE TRIGGER POST_NEW_ORDER FOR SALES
  AFTER INSERT AS
  BEGIN
      POST_EVENT «NEW_ORDER»;
  END !!
SET TERM;
```

В общем случае триггер может использовать переменную в качестве имени события:

```
POST_EVENT :EVENT_NAME;
```

1.2.3.2 СУБД PostgreSQL

Синтаксис оператора CREATE TRIGGER. Оператор состоит из заголовка триггера и триггерной функции. Заголовок содержит:

- имя триггера, уникальное по БД;
- указание на момент, когда триггер должен вызываться;
- имя таблицы, с которой ассоциируется триггер;
- указание вызывается ли триггер для каждой строки или для конкретной операции.

Триггерная функция создаётся командой **CREATE FUNCTION**, при этом у функции не должно быть аргументов, а типом возвращаемого значения должен быть trigger (для триггеров, срабатывающих при изменениях данных) или event_trigger (для триггеров, срабатывающих при событиях в базе).

```
CREATE [ OR REPLACE ] [ CONSTRAINT ] TRIGGER name
{ BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
   [ NOT DEFERRABLE | [ DEFERRABLE ]
[ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [   REFERENCING   {   {   OLD   |   NEW   }   TABLE   [   AS   ]
transition_relation_name }
[... ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )
```

CREATE TRIGGER создаёт новый триггер, а **CREATE OR REPLACE TRIGGER** создаёт новый триггер или заменяет существующий. Триггер будет связан с указанной таблицей, представлением или сторонней таблицей и будет выполнять заданную функцию function_name при определённых операциях с этой таблицей.

Триггер активизируется при попытке изменения данных в таблице, для которой определен. SQL выполняет эту процедуру при операциях добавления, обновления и удаления **(INSERT, UPDATE, DELETE)** в данной таблице.

Триггер может выполняться в трех фазах изменения данных: **до** (before) какого-то события, **после** (after) него или **вместо операции** (instead of).

Триггер с пометкой **FOR EACH ROW** вызывается один раз для каждой строки, изменяемой в процессе операции. Триггер с пометкой **FOR EACH STATEMENT** вызывается только один раз для конкретной операции, вне зависимости от того, как много строк она изменила (при выполнении операции, изменяющей ноль строк, всё равно будут вызваны все триггеры FOR EACH STATEMENT).

Триггеры, срабатывающие в режиме INSTEAD OF, должны быть помечены FOR EACH ROW и могут быть определены только для представлений. Триггеры BEFORE и AFTER для представлений должны быть помечены FOR EACH STATEMENT.

В определении триггера можно указать логическое условие WHEN, которое определит, вызывать триггер или нет. В триггерах на уровне строк условия WHEN могут проверять старые и/или новые значения столбцов в строке. Триггеры на уровне оператора так же могут содержать условие WHEN, хотя для них это не имеет смысла, так как в этом условии нельзя ссылаться на какие-либо значения в таблице.

Например, **рассмотрим триггер**:

```
CREATE TRIGGER user_insert_trigger
AFTER INSERT
ON users
FOR EACH ROW
EXECUTE PROCEDURE user_insert_trigger_fnc();
```

Для работы триггера, необходимо перед созданием самого триггера создать триггерную функцию, которая будет вносить новую запись в таблицу (добавление информации о новом сотруднике в таблицу «Audit»).

```
CREATE OR REPLACE FUNCTION user_insert_trigger_fnc()
      RETURNS trigger AS
      $$
      BEGIN
      INSERT INTO Audit (UsersId, Name, AuditUserName, UserAdditionTime)
      VALUES(NEW.id,NEW.name,current_user,current_date);
      RETURN NEW;
      END;
      $$
      LANGUAGE «plpgsql»;
```

Триггер будет добавлять в таблицу информацию о новом сотруднике, если эти данные появились в другой таблице.

Модификация триггера.

Чтобы изменить свойства триггера, необходимо использовать **CREATE OR REPLACE TRIGGER**, указав имя существующей триггерной функции и связанную таблицу. Остальные свойства можно изменять так, как нужно для выполнения вашей задачи.

Также можно переименовать триггер. Для этого используется запрос ALTER TRIGGER:

```
ALTER TRIGGER name ON table_name RENAME TO new_name;
```

Удаление триггера.

Для удаления триггера необходимо использовать DROP TRIGGER. Синтаксис:

```
DROP TRIGGER [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ]
```

Здесь **IF EXISTS** – указание на то, что не надо выдавать ошибку, если такого триггера нет; **CASCADE** – автоматически удалять все объекты, которые зависят от триггера, объекты, которые зависят от этих объектов; **RESTRICT** – не удалять триггер, если от него зависят другие объекты. Это значение по умолчанию.

1.2.4 Использование триггеров для обновления пользовательских представлений

Пользовательские представления, основанные на соединении нескольких таблиц, как правило, не обновляемы, и могут служить только для чтения. Тем не менее, можно написать триггеры для операций добавления, обновления и удаления таким образом, что они будут правильно модифицировать базовые таблицы, из которых «собран» VIEW. Таким образом, можно редактировать не редактируемые пользовательские представления.

Следующие операторы создают две таблицы, создают пользовательское представление и три триггера для модификации представления:

```
CREATE TABLE Table1 (
  ColA INTEGER NOT NULL,
  ColB VARCHAR(20),
  CONSTRAINT pk_table PRIMARY KEY(ColA)
);

CREATE TABLE Table2 (
  ColA INTEGER NOT NULL,
  ColC VARCHAR(20),
  CONSTRAINT fk_table2 FOREIGN KEY (ColA)
                  REFERENCES Table1(ColA)
);

CREATE VIEW TableView AS
  SELECT Table1.ColA, Table1.ColB, Table2.ColC
      FROM Table1, Table2
    WHERE Table1.ColA = Table2.ColA;
```

Создание триггера, который позволит производить удаление данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Delete FOR TableView BEFORE DELETE AS
  BEGIN
      DELETE FROM Table1
        WHERE ColA = OLD.ColA;
      DELETE FROM Table2
        WHERE ColA = OLD.ColA;
  END;
```

Создание триггера, который позволит производить обновление данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Update FOR TableView
BEFORE UPDATE AS
  BEGIN
      UPDATE Table1
        SET ColB = NEW.ColB
        WHERE ColA = OLD.ColA;
      UPDATE Table2
        SET ColC = NEW.ColC
        WHERE ColA = OLD.ColA;
  END;
```

Создание триггера, который позволит производить выполнять ввод данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Insert FOR TableView BEFORE INSERT AS
  BEGIN
      INSERT INTO Table1 values (NEW.ColA,NEW.ColB);
      INSERT INTO Table2 values (NEW.ColA,NEW.ColC);
  END;
```

1.2.5 Использование исключений в триггерах

Исключение – это именованное сообщение об ошибке, которое может быть вызвано из триггера или хранимой процедуры. Исключения создаются с помощью оператора CREATE EXCEPTION, модифицируется с помощью ALTER EXCEPTION, и удаляются с помощью DROP EXCEPTION.

Исключения - это объекты базы данных, они хранятся в системной таблице и могут использоваться любой процедурой или триггером.

Если исключение вызывается в хранимой процедуре или триггере, исключение возвращает строковое сообщение вызывающей программе и завершает выполнение блока операторов, если исключение не обрабатывается с помощью оператора WHEN ниже в тексте процедуры или триггера.

Например, если количество неудачных попыток зайти в систему (каждая попытка записывается в таблицу) превышает 5, можно выводить предупреждающее сообщение.

Генерация исключения. Для генерации исключения в триггере нужно воспользоваться следующим оператором: **EXCEPTION** name; здесь name - это имя существующего исключения.

Пример создания исключения:

```
CREATE EXCEPTION RAISE_TOO_HIGH «New salary exceeds old
            by more than 50%. Cannot update record.»;
```

Пример вызова исключения из триггера:

```
SET TERM !! ;
CREATE TRIGGER SAVE_SALARY_CHANGE
FOR EMPLOYEE
  AFTER UPDATE AS
  DECLARE VARIABLE PCNT_RAISE;
BEGIN
  PCNT_RAISE=(NEW.SALARY-OLD.SALARY)*100/OLD.SALARY;
      IF (OLD.SALARY <> NEW.SALARY)
      THEN IF (PCNT_RAISE > 50)
            THEN EXCEPTION RAISE_TOO_HIGH;
      ELSE BEGIN
            INSERT INTO SALARY_HISTORY (EMP_NO, HANGE_DATE,
                UPDATER_ID, OLD_SALARY, PERCENT_CHANGE)
            VALUES (OLD.EMP_NO, «NOW», USER, OLD.SALARY, PCNT_RAISE);
              END
   END !!
SET TERM ; !!
```

1.3 Порядок выполнения лабораторной работы

1. В соответствии с вариантом задания (приложение А) создать **генератор для каждого первичного ключа** (исключение, если первичный ключ является символьным полем).
2. Согласно варианту, создать функции.
3. Согласно варианту, написать триггеры.
4. Написать отчет.

1.4 Содержание отчета

Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.

Отчет должен содержать исходные данные, этапы работы по созданию и изменению генератора, по созданию функции и триггеров; ввод данных в таблицу, с использованием генератора и триггера, выборку информации из таблицы на каждом шаге работы.

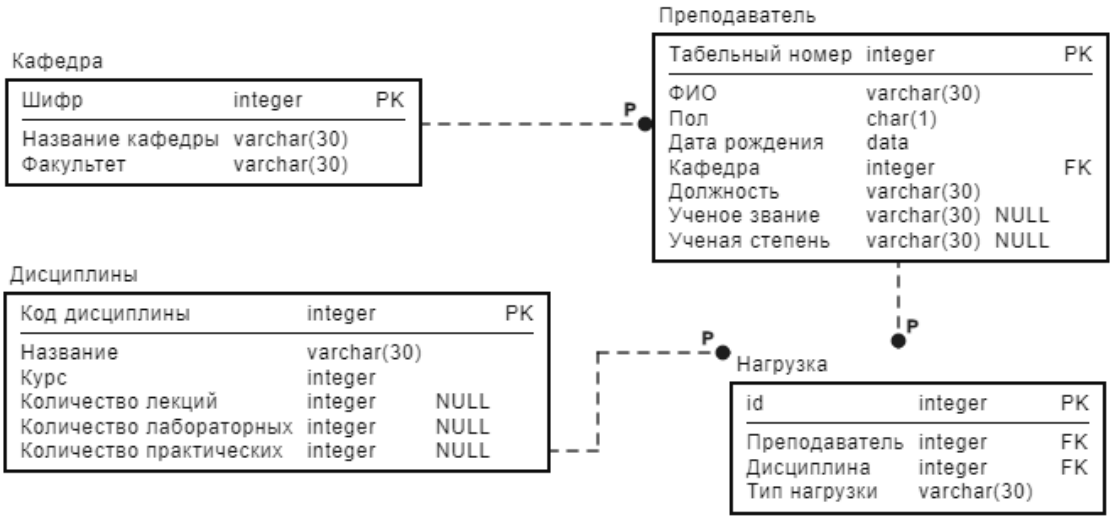
1.5 Контрольные вопросы

1. Назначение генераторов.
2. Как сгенерировать следующее значение генератора?
3. Как переустановить значение генератора?
4. Как удалить генератор?
5. Как создать функцию?
6. Что такое детерминированные функции?
7. В чем отличие хранимой функции от хранимой процедуры?
8. Организация многопользовательского режима доступа к данным.
9. Что такое «триггер»?
10. Из каких частей состоит триггер?

11. Какая информация содержится в заголовочной части триггера?
12. Как сделать триггер временно неактивным? Как удалить триггер?
13. Для чего используются триггерные функции?
14. Для чего используются триггеры?
15. Назначение переменных new и old.
16. Различие между триггером уровня строк и триггером уровня оператора?

ПРИЛОЖЕНИЕ А
Варианты заданий

Вариант 1



Пол – значения – «м» и «ж», по умолчанию – «ж»
Должность – ассистент, преподаватель, доцент, профессор
Ученая степень – кандидат или доктор наук
Ученое звание – доцент или профессор

Рисунок А.1 – Вариант 1. БД отдела кадров института

Создание функций.

1. Функция, определяющая по дате рождения, является ли человек юбиляром в текущем месяце, и выдающая для юбиляра возраст (юбилейную дату, например, «45»), а в противном случае –NULL. Параметр – дата рождения.

2. Функция, вычисляющая премии для сотрудников, празднующих юбилей в текущем месяце. Входные параметры – базовый размер премии. Размер премии для сотрудника рассчитывать по следующей формуле: за каждые 100 часов лекций нагрузки – один базовый размер премии для доктора наук, 0.8 для доцента и 0.5 для всех остальных. Результат работы функции добавляется в таблицу «Премии» (поля ФИО сотрудника, дату юбилея, возраст и размер премии), которая в начале работы очищается от старых данных.

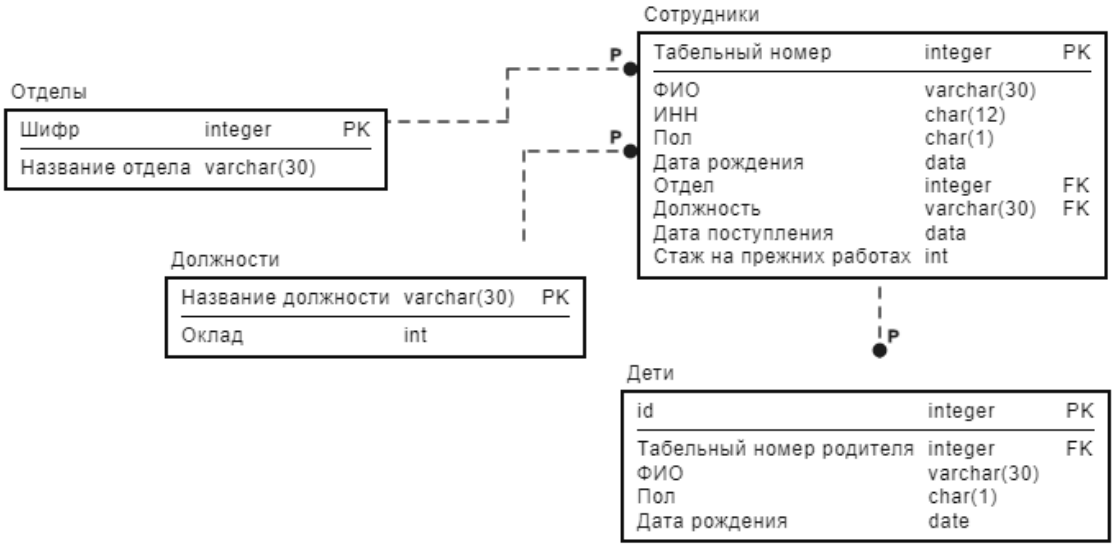
Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Преподаватели», для которых могут быть определены домены (в частности, значения полей «Пол», «Должность», «Ученая степень», «Ученое звание», возраст не может быть меньше 20 лет).

2. Регистрация изменений, вносимых в таблицу «Преподаватели» (дублирование старой записи в специальной таблице с указанием даты изменения и пользователя, который их проводит).

Вариант 2



Пол – значения – «м» и «ж», по умолчанию – «ж»

Рисунок А.2 – Вариант 2. БД бухгалтерии

Создание функций.

1. Функция, определяющая стаж работы человека. Параметры: стаж на прежних местах работы и дата поступления на работу. Возвращает число полных лет стажа на текущую дату.

2. Функция начисления премий сотрудникам. Входной параметр – базовая ставка. Размер премии вычисляется как произведение базовой ставки на оклад и на коэффициент, который зависит от стажа работы на данном предприятии:

- 0.1, если стаж от 1 до 5 лет;
- 0.2, если стаж от 5 до 10 лет;
- 0.3, если стаж от 10 до 20 лет;
- 0.5, если стаж от 20 до 30 лет;
- 1, если стаж свыше 30 лет.

Результат работы функции добавляется в таблицу «Премии» (поля ФИО сотрудника, стаж, размер премии), которая в начале работы очищается от старых данных. Людям, проработавшим менее 1 года, премия не выплачивается (в таблицу они не добавляются).

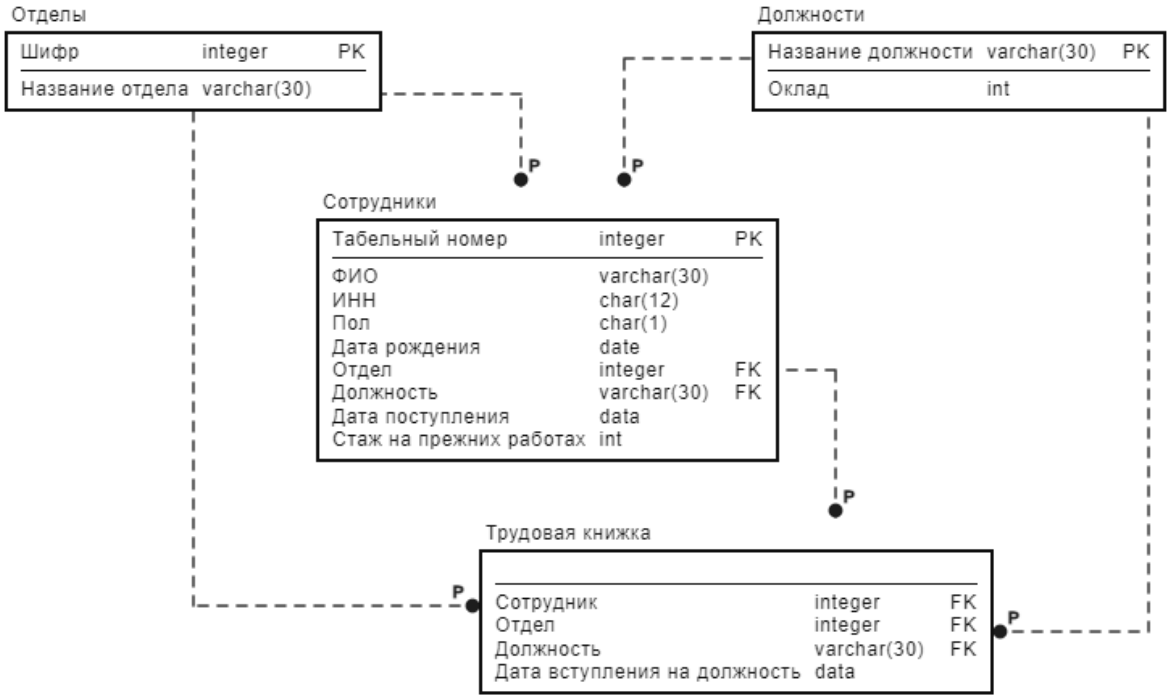
Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Сотрудники», для которых могут быть определены домены (в т.ч., ИНН может содержать только цифры, (возраст сотрудника)-(стаж на прежних работах)-(стаж работы на данном предприятии) не может быть меньше 16 лет, а дата поступления на работу должна быть не больше текущей даты). Если при вводе данных дата поступления не указана, устанавливать текущую дату.

2. Установка значения поля «пол», если оно не установлено. Правила: если отчество оканчивается на «-НА», то пол женский, если на «-ИЧ», то мужской. В противном случае триггер должен генерировать ошибку.

Вариант 3



Пол – значения – «м» и «ж», по умолчанию – «ж»

Рисунок А.3 – Вариант 3. БД бухгалтерии

Создание функций.

1. Функция, определяющая, является ли человек на текущую дату пенсионером. Параметры: пол и дата рождения. Возвращает строку «пенсионер» или пустую строку.

2. Функция начисления премий сотрудникам. Входной параметр – базовая ставка. Размер премии вычисляется как произведение базовой ставки на оклад и на коэффициент, который зависит от стажа работы на данном предприятии:

- 0.1, если стаж от 1 до 5 лет;
- 0.2, если стаж от 5 до 10 лет;
- 0.3, если стаж от 10 до 20 лет;
- 0.5, если стаж от 20 до 30 лет;
- 1, если стаж свыше 30 лет.

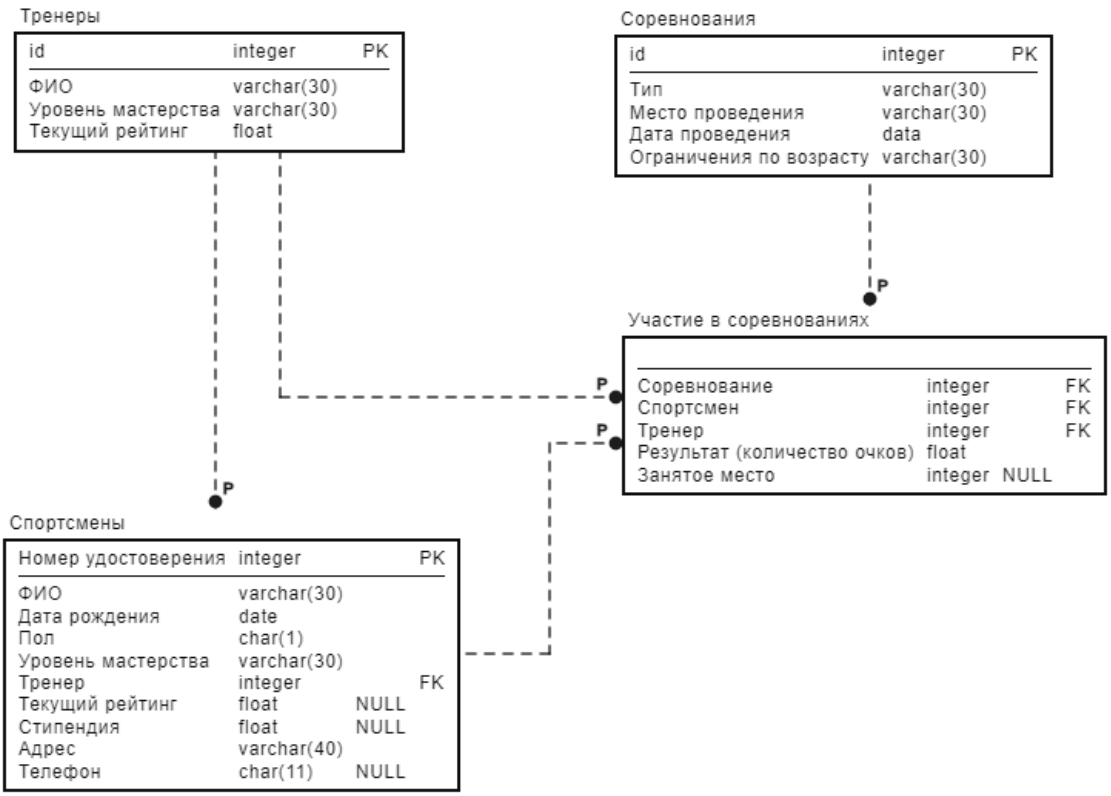
Результат работы функции добавляется в таблицу «Премии» (поля ФИО сотрудника, стаж, размер премии), которая в начале работы очищается от старых данных. Стаж рассчитывается с учетом трудовой книжки. Людям, проработавшим менее 1 года, премия не выплачивается (и в таблицу они не добавляются). Пенсионерам выплачивается половина премии. Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Сотрудники», для которых могут быть определены домены (в т.ч., ИНН может содержать только цифры, а дата поступления на работу должна быть не больше текущей даты). Если при вводе данных дата поступления не указана, устанавливать текущую дату.

2. Формирование таблицы «Трудовая книжка»: при изменении занимаемой должности в таблице «Сотрудники», предыдущая должность добавляется в таблицу «Трудовая книжка» с указанием даты начала работы в этой должности (старое значение даты вступления в должность из таблицы «Сотрудники»).

Вариант 4



Пол – значения – «м» и «ж», по умолчанию – «ж»
Уровень мастерства – 1 разряд, 2 разряд, КМС, МС и т.д

Рисунок А.4 – Вариант 4. БД спортивного клуба

Создание функций.

1. Функция, определяющая по дате рождения и текущей дате, к какой категории относится спортсмен:

Д-1	Дети I	9 лет и моложе
Д-2	Дети II	10-11 лет
Ю-1	Юниоры I	12-13 лет
Ю-2	Юниоры II	14-15 лет
М	Молодежь	16-18 лет
ВЗ	Взрослые	19 лет и старше
С	Сеньоры	35 лет и старше

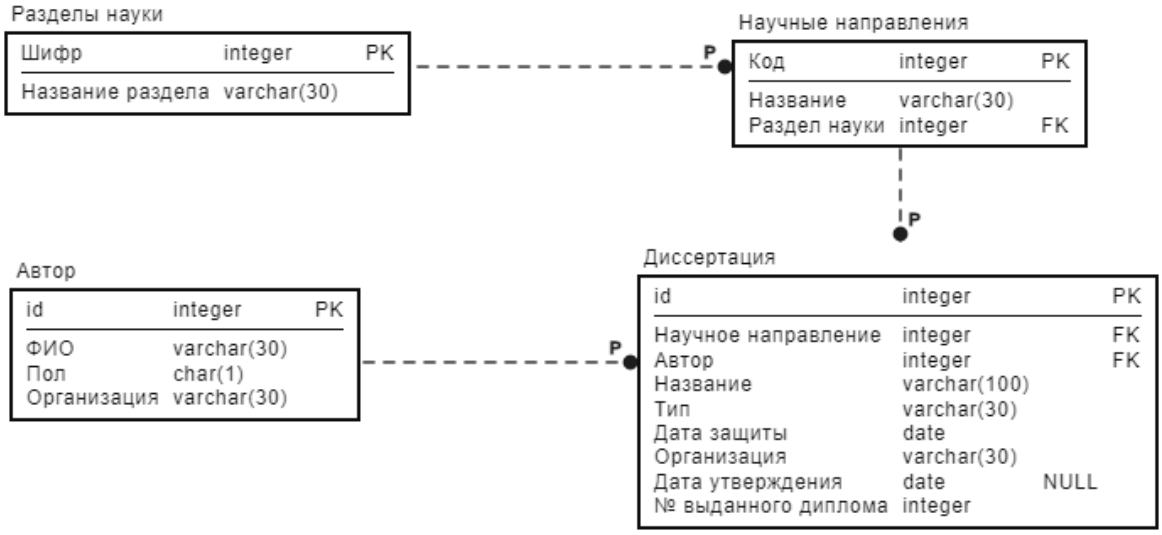
2. Функция, проверяющая соответствие места и завоеванных очков на соревновании. Параметр: идентификатор соревнования. Если два спортсмена получили одинаковое количество очков на одном соревновании, давать обоим одно и то же место.

Создание триггеров.

1. Увеличение рейтинга спортсмена при добавлении сведений о его участии в соревнованиях: за 1-е место – плюс 20 баллов к рейтингу, за 2-е место – плюс 15, за 3-е место – плюс 10, за простое участие – плюс 2 балла.

2. Проверка значений всех полей отношения «Спортсмены», для которых могут быть определены домены, например: номер домашнего телефона не должен совпадать с номером мобильного телефона; текущий рейтинг и стипендия не могут быть отрицательными; спортсмен не может быть моложе пяти лет.

Вариант 5



Пол – значения – «м» и «ж», по умолчанию – «ж»

Рисунок А.5 – Вариант 5. БД диссертаций

Создание функций.

1. Функция, выдающая полное название ученой степени по параметрам «Тип» и «Раздел науки». Например, для типа «докторская» и раздела «Технические науки» функция должна вернуть «доктор технических наук».

2. Функция, выдающая для автора его ученую степень. Параметр – идентификатор автора. Если автор защитил кандидатскую и докторскую диссертации по одному разделу науки, то он является доктором наук. Если разделы разные, то ученые степени перечисляются через запятую (например, «кандидат экономических наук, доктор технических наук»). Сначала – кандидат, потом доктор. Докторских степеней может быть несколько, кандидатская – одна. Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Диссертации», для которых могут быть определены домены (дата защиты и дата утверждения не могут быть больше текущей даты; дата защиты меньше даты утверждения, если последняя определена; тип диссертации – кандидатская или докторская).

2. Триггер, переносящий в архив (в специальную таблицу) предыдущие значения данных об авторах при их изменении: могут поменяться любые сведения, кроме даты рождения. При попытке изменить дату рождения триггер выдает ошибку.

Вариант 6

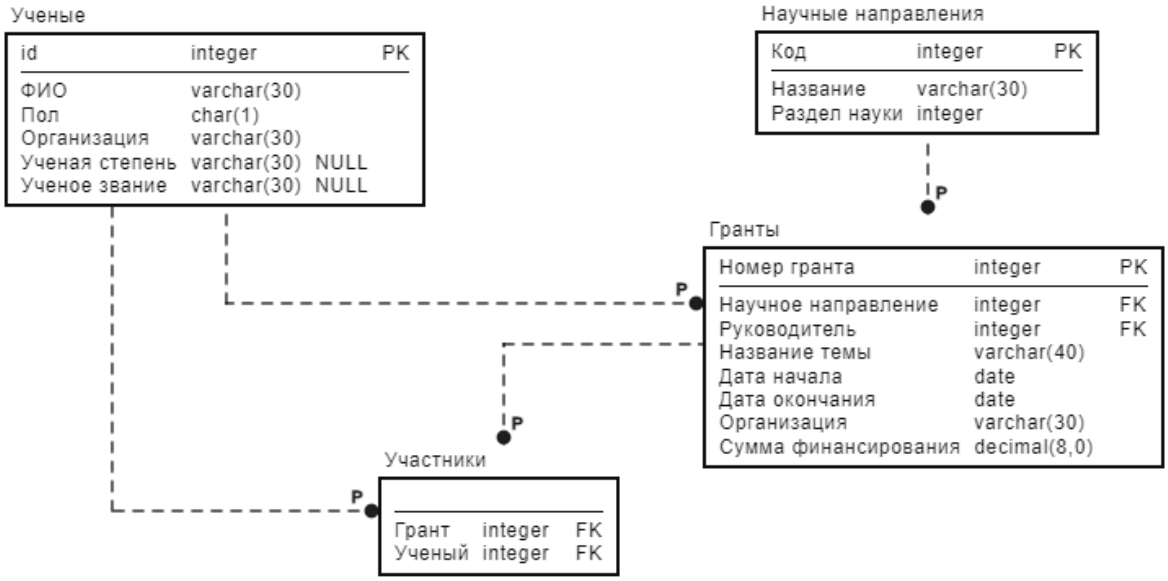


Рисунок А.6 – Вариант 6. БД научного фонда

Создание функций.

1. Функция, принимающая в качестве параметра даты начала и завершения гранта и возвращающая строку «не начался», если текущая дата меньше даты начала гранта, строку «закончился», если текущая дата больше даты завершения, и пустую строку в других случаях.

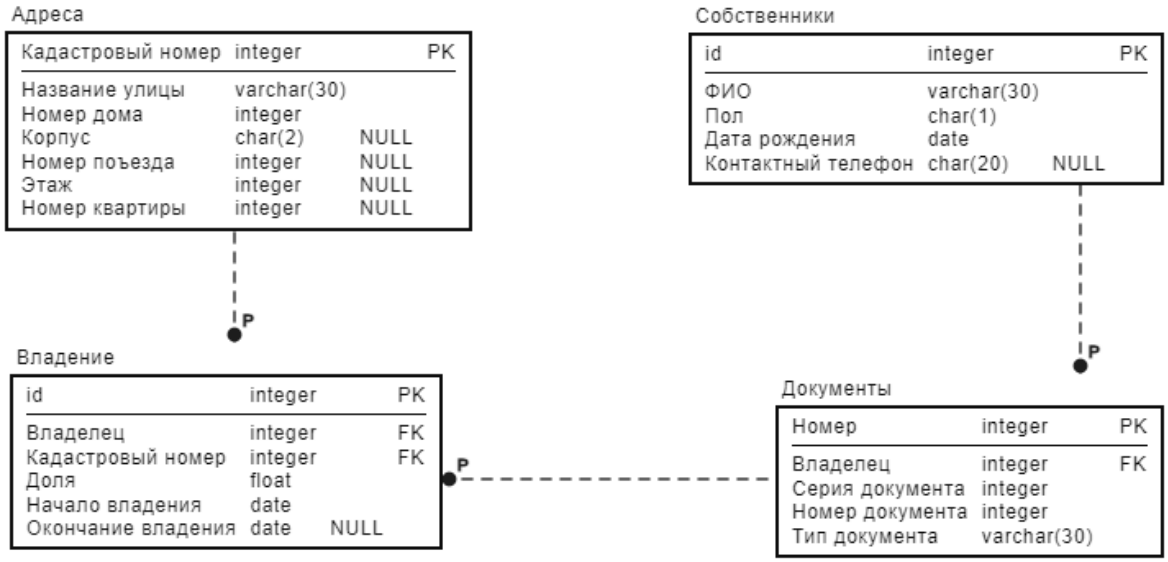
2. Функция, возвращающая для руководителя гранта количество грантов, в которых он участвует параллельно с текущим грантом. Параметры: идентификатор руководителя и номер текущего гранта. Параллельным с текущим грантом считается тот грант, период выполнения которого совпадает хотя бы частично с периодом выполнения текущего гранта. Например, периоды 01.02.2019-30.10.2020 и 01.01.2020-30.12.2022 пересекаются. Функция должна возвращать 2 числа: количество грантов, в которых человек является руководителем (кроме текущего гранта), и количество грантов, в которых он является участником. Если человек не является руководителем текущего гранта, функция должна выдавать ошибку.

Создание триггеров.

1. Проверка значений всех полей отношения «Гранты», для которых могут быть определены домены (дата начала меньше даты завершения; сумма финансирования не меньше 100000; первые две цифры идентификатора гранта равны двум последним цифрам первого года выполнения гранта, например, 15-01-00678 для даты «01.01.2015»).

2. Триггер, переносящий в архив (в специальную таблицу) предыдущие значения данных об авторах при их изменении: могут поменяться любые сведения, кроме даты рождения. При попытке изменить дату рождения триггер выдает ошибку.

Вариант 7



Пол – значения – «м» и «ж», по умолчанию – «ж»
Доля – часть квартиры, которой он владеет (например, 1, 1/3, 0.25 и т.д.)

Рисунок А.7 – Вариант 7. БД собственников квартир

Создание функций.

1. Функция, принимающая в качестве параметров две даты и возвращающая строку «несовершеннолетний», если между этими датами прошло менее 18-и лет и одного дня (человек становится совершеннолетним на следующий день после того, как ему исполняется 18 лет). Если вторая дата не определена, считать до текущей даты.

2. Функция, выводящая по номеру здания в специальную таблицу список квартир в этом доме, у которых в настоящее время более 10 собственников, или квартир, у которых в числе собственников есть несовершеннолетние. Параметры – номер здания (ПК) и режим работы: 0 – функция должна перед началом работы очищать таблицу, 1 – новые данные просто добавляются в таблицу. Таблица имеет следующие поля: номер здания, начало владения, ФИО собственника, доля, которой он владеет. Если таких квартир нет, выдавать сообщение об этом.

Создание триггеров.

1. Триггер, проверяющий правильность формирования полей «Серия документа» и «Номер документа». Маска серии:

- для свидетельства о рождении или смерти – X…X-КК, где X…X – латинское число, КК – две русские буквы.
- для российского паспорта – 4 цифры.

Маска номера: шесть цифр (номер хранится в виде строки, т.к. в нем могут быть ведущие нули).

2. Проверка значений всех полей отношения «Владение», для которых могут быть определены домены (даты не больше текущей; дата окончания владения больше даты начала или не определена; номер квартиры содержит только цифры или цифры и одну букву в конце). Если при вводе данных дата начала владения не указана, устанавливать текущую дату.

Вариант 8

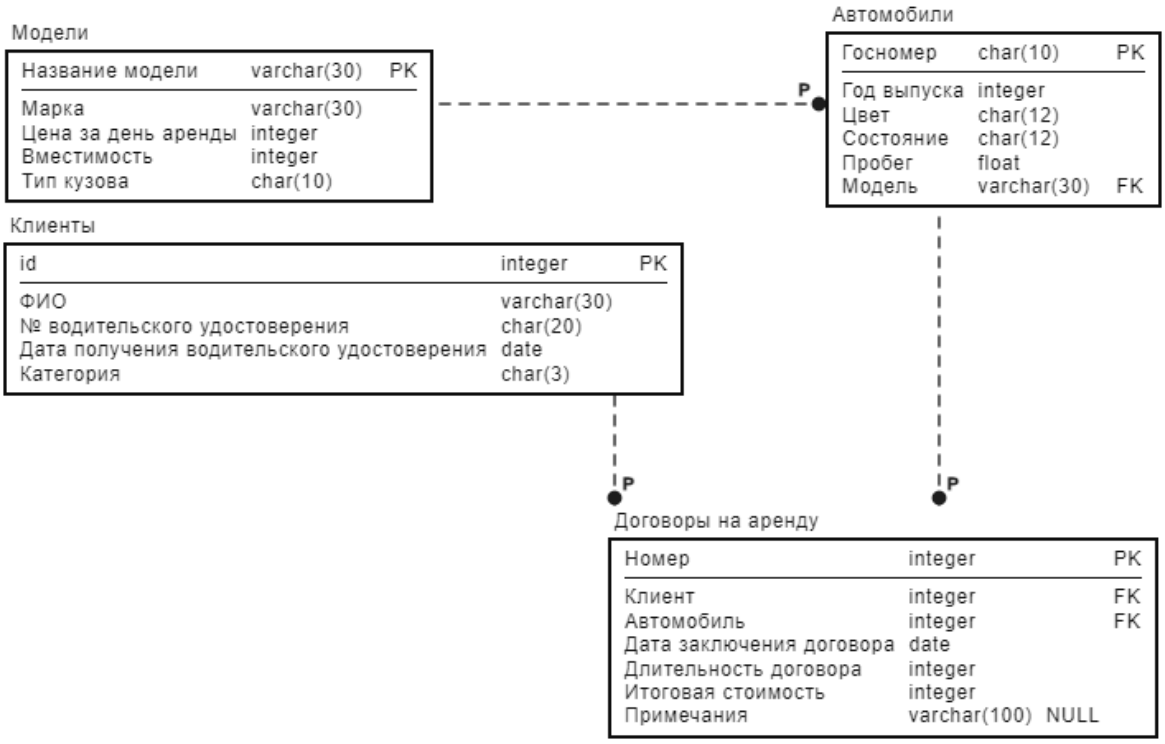


Рисунок А.8 – Вариант 8. БД пункта проката автомобилей

Создание функций.

1. Функция, принимающая два параметра – дату и продолжительность аренды. Функция должна возвращать предположительную дату возврата автомобиля.

2. Функция, добавляющая в специальную таблицу «Отчет» сведения о каждом автомобиле определенной марки на текущую дату. Параметр – марка автомобиля. Функция должна перед началом работы очищать таблицу. Для автомобилей, находящихся в прокате, функция должна вносить номер, год выпуска, цвет, пробег, ФИО арендатора, начало и предполагаемую дату возврата автомобиля. Для автомобилей, не находящихся в прокате, функция должна вносить номер, год выпуска, цвет, пробег, дату последнего возврата из аренды и продолжительность аренды за последний год (в днях). Использовать ранее созданную функцию.

Создание триггеров.

1. Триггер, устанавливающий стоимость проката после установки даты возврата автомобиля как произведение стоимости аренды на ее реальную продолжительность.

2. Проверка значений всех полей отношения «Договоры на аренду», для которых могут быть определены домены: дата начала договора не меньше текущей, дата возврата автомобиля не определена, стоимость договора равна продолжительности, умноженной на стоимость аренды; продолжительность больше 0.

Вариант 9

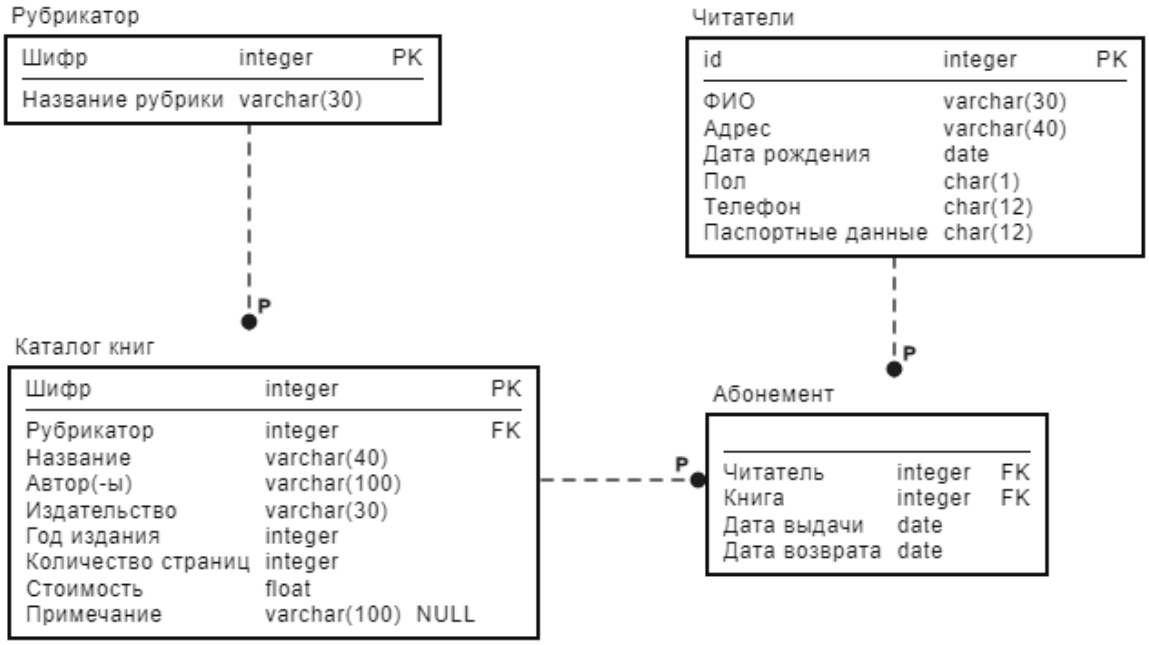


Рисунок А.9 – Вариант 9. БД библиотеки (книги)

Создание функций.

1. Функция, возвращающая пустую строку или строку «старое издание» для учебников, выпущенных 20 и более лет назад, для справочников, выпущенных 10 и более лет назад и для остальных книг, если они выпущены более 30 лет назад. Параметры: год издания и тип издания (значение поля «примечание»).

2. Функция добавления в специальную таблицу «Отчет» списка книг по указанной рубрике. Таблица должна содержать следующие поля: название рубрики, авторы, название книги, год выпуска, место издания, примечание (старое издание). Параметры – название рубрики и режим: 0 – функция должна перед началом работы очищать таблицу, 1 – новые данные просто добавляются в таблицу. Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Каталог книг», для которых могут быть определены домены (в т.ч., год издания не может быть больше текущего года и меньше 1950; количество страниц – от 10 до 2000). Если год издания не указан, устанавливать текущий год. Если значение поля примечание – «учебник», поле «Авторы» не может быть пустым.

2. При изменении данных о рубриках – копирование старых значений в специальную таблицу. Сохранять в этой таблице дату изменения и имя пользователя, который произвел изменения

Вариант 10

Страницы – диапазон значений, например: 35, 56-62, 98-111

Рисунок А.10 – Вариант 10. БД библиотеки (журналы)

Создание функций.

1. Функция, возвращающая количество страниц на основании значения поля «Страницы». Если значение поля «x-y», то значение вычисляется как (y–x+1). Если значение имеет вид «x», возвращается 1. Если обнаружена ошибка (y<x или вообще невозможно выделить числа), то возвращается –1.

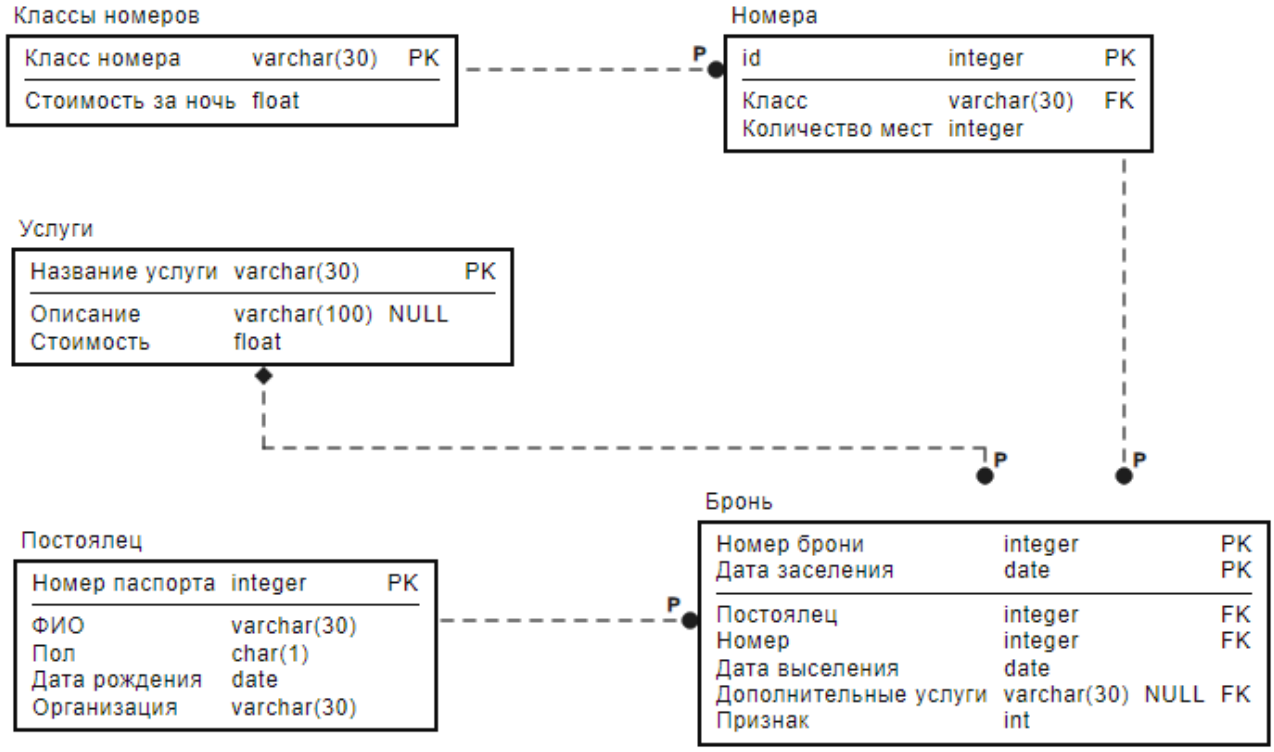
2. Функция добавления в специальную таблицу «Отчет» списка книг указанного автора. Таблица должна содержать следующие поля: идентификатор автора, ФИО автора, название публикации, название журнала, год, номер выпуска, количество страниц, соавторы. Параметры – идентификатор автора и режим: 0 – функция должна перед началом работы очищать таблицу, 1 – новые данные просто добавляются в таблицу. Использовать ранее созданную функцию. Если у автора не было статей, выдавать сообщение об этом.

Создание триггеров.

1. Установка правильного формата поля «Страницы»: оно должно содержать значение типа «x-y», причем x должен быть меньше y. Если формат поля «c.x-y», то «c.» убирается. Если x равен y, то значение преобразуется к виду «x». Если формат нарушен (нельзя выделить одно или два числа), то триггер выдает сообщение об ошибке.

2. Фиксация всех изменений (по UPDATE и DELETE) отношения «Рубрикатор» в отдельной таблице (с указанием пользователя, который произвел изменения, и даты внесения изменений).

Вариант 11



Пол – значения – «м» и «ж», по умолчанию – «ж»
Признак – значения 0 – «не прибыл», 1 – «прибыл», 2 – «выбыл»

Рисунок А.11 – Вариант 11. БД гостиницы

Создание функций.

1. Функция, возвращающая количество дней, прожитых постояльцем в гостинице на основании двух дат (день приезда – день отъезда считать, как один день). Если второй параметр не передается, считать до текущей даты.

2. Функция подготовки счёта клиенту за предоставленные услуги. Функция помещает данные счета в таблицу «Счет» с полями: ФИО клиента, дата оказания услуги, количество, стоимость. Учитываются предоставленные услуги и проживание. Параметр – номер паспорта клиента. Если в один день клиенту было оказано несколько одинаковых услуг, выводить их одной строкой, указывая количество. Функция предварительно должна очищать таблицу Счет. Последняя добавляемая строка: ФИО, дата вселения, дата выезда, общая стоимость. Использовать ранее созданную функцию.

Создание триггеров.

1. Копирование в архив (специальную таблицу) всех изменений таблицы «Услуги». Указывать в архиве дату внесения изменений и имя пользователя, который вносил изменения.

2. Проверка значений всех полей отношения «Постояльцы», для которых могут быть определены домены: дата вселения не меньше текущей; признак 0, 1 или 2; дата выселения не определена или больше даты вселения. Если значение поля «дата вселения» не определено, устанавливает текущую дату. Если признак равен 2, то дата выбытия должна быть определена, иначе триггер выдает ошибку.

Вариант 12

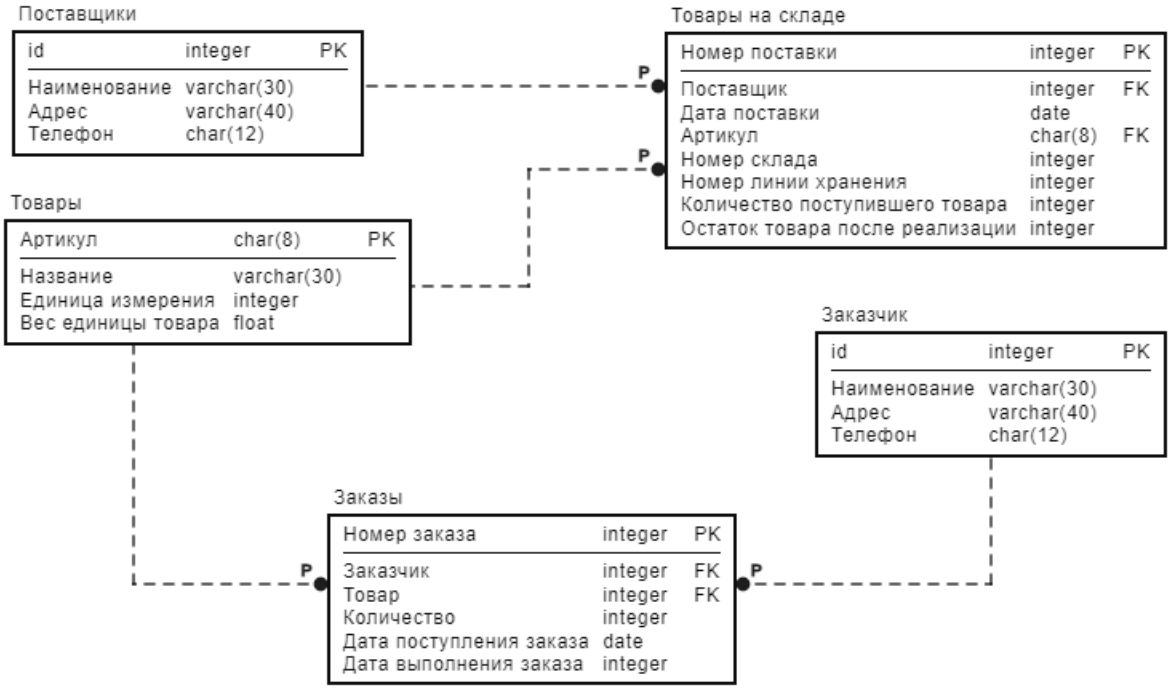


Рисунок А.12 – Вариант 12. БД складского предприятия

Создание функций.

1. Функция, возвращающая количество дней, прошедших между двумя датами. Если первая дата больше второй, функция возвращает -1.

2. Функция, создающая отчет о движении товаров за период путем добавления данных в специальную таблицу «Отчет». Таблица должна содержать следующие поля: название товара, единица измерения, количество поступившего товара, количество реализованного товара. Параметры: начало и окончание периода (даты). Если окончание периода не определено, считать его равным текущей дате. Если начало периода больше окончания, процедура должна выдавать соответствующее сообщение. Использовать ранее созданную функцию. Для каждого товара должна выводиться одна строка с суммарным количеством поставленного и реализованного товара.

Создание триггеров.

1. Проверка значений всех полей отношения «Товары на складе», для которых могут быть определены домены (например, дата поставки не может быть больше текущей, остаток товара не больше количества поступившего товара, не меньше 0; количество поступившего товара больше 0). Если при вводе данных дата поставки не указана, устанавливать текущую дату.

2. Перенос в архив (специальную таблицу) сведений о поставках товаров, которые уже реализованы (при установке значения «Остаток товара» равным 0).

Вариант 13

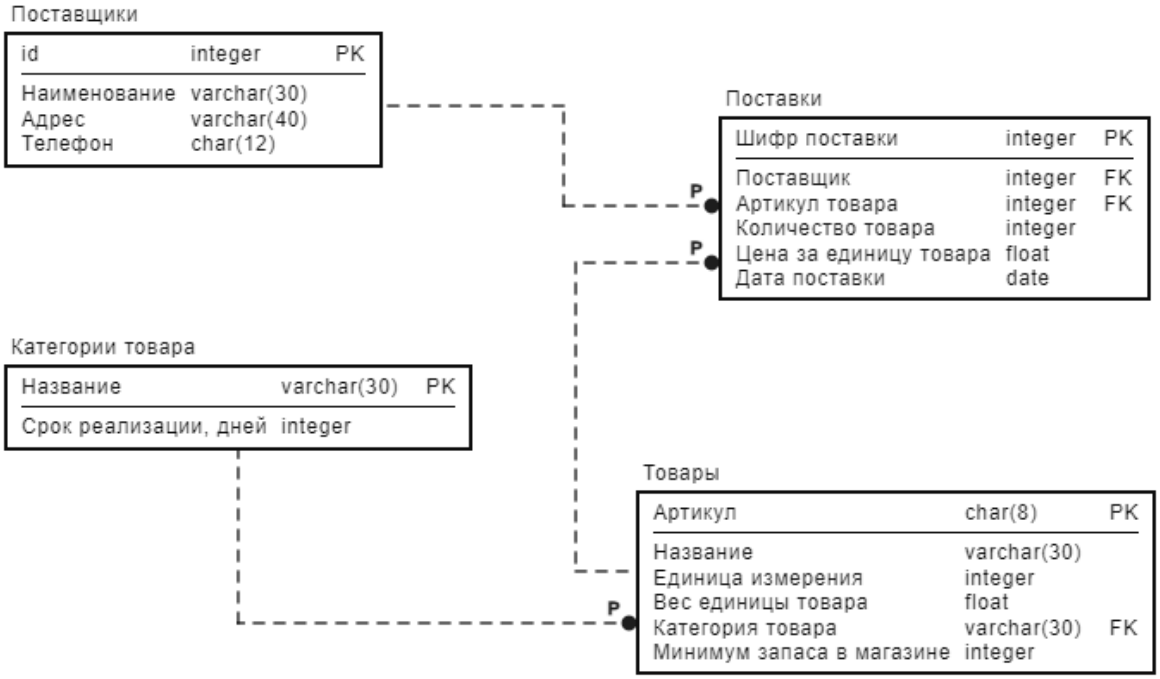


Рисунок А.13 – Вариант 13. БД магазина

Создание функций.

1. Функция, возвращающая новую цену товара, если до завершения срока реализации осталось менее 3-х часов для скоропортящегося товара и менее одного дня для всех остальных. Цена уменьшается на 20%, но не более чем на 50 рублей. Параметры: цена товара, дата поставки и срок реализации.

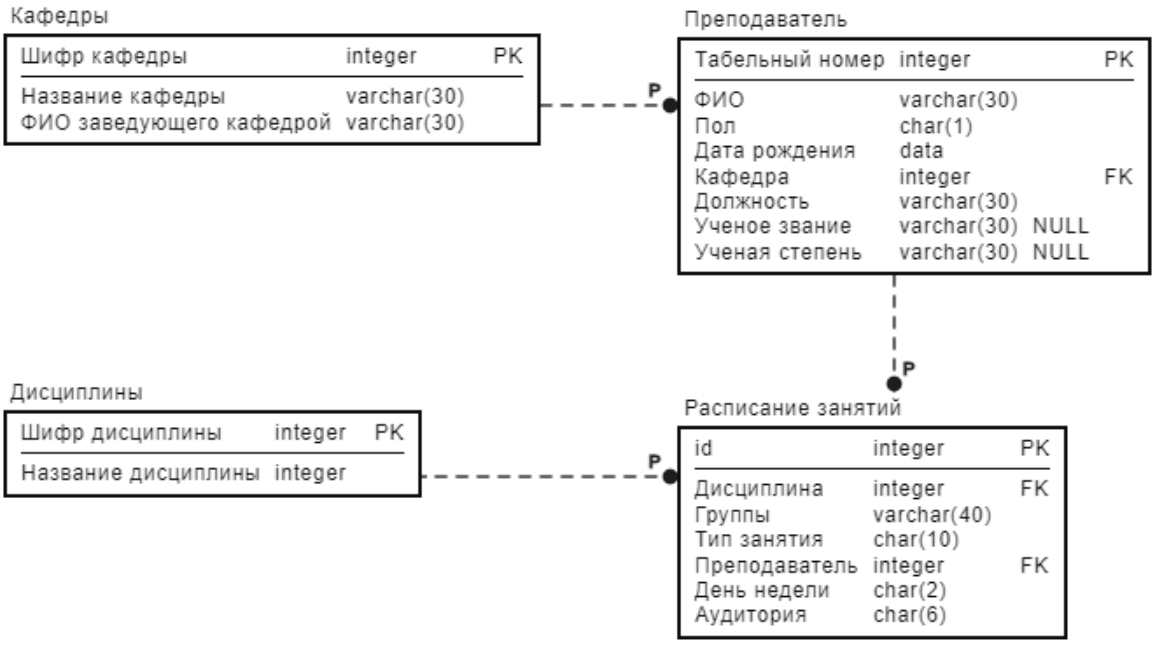
2. Функция, создающая отчет об изменении цен на товары путем добавления данных в специальную таблицу «Изменение цен». Таблица должна содержать следующие поля: текущая дата, категория, артикул товара, название товара, единица измерения, остаток, старая цена, новая цена. Параметр: режим работы: 0 – функция должна перед началом работы очищать таблицу, 1 – новые данные просто добавляются в таблицу. Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Поставки», для которых могут быть определены домены: количество товара и цена единицы поставки больше 0, дата поставки не может быть больше текущей, остаток товара не может превышать количество поставленного товара и быть меньше 0. Если при вводе данных дата поставки не указана, устанавливать текущую дату.

2. При изменении данных о товарах – копирование старых значений в специальную таблицу.

Вариант 14



День недели – пн, вт, ср, чт, пт, сб

Рисунок А.14 – Вариант 14. БД деканата

Создание функций.

1. Функция, возвращающая по параметрам «время» и «тип занятия» время окончания занятия, например:
   - для лекции или семинара в «09.00» вернет «10.20», т.е. занятие длится час двадцать;
   - для лабораторной работы в «09.00» вернет «10.30», т.е. полтора часа.
2. Функция, создающая отчет о некорректных данных в расписании для преподавателей кафедры путем добавления данных в специальную таблицу «Отчет». Таблица должна содержать следующие поля: ФИО преподавателя, день недели, количество занятий, начало первого занятия, начало следующего занятия. Расписание считается некорректным, если у одного преподавателя более 4-х занятий в день или если время разных занятий у одного преподавателя совпадает или пересекается. Первое и следующее занятия – это те занятия, которые пересекаются по времени (каждая пара – в отдельной строке). Если таких занятий нет, то эти поля пустые; если занятий не больше 4-х – соответствующее поле пустое. Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Расписание занятий», для которых могут быть определены домены. Проверить тип занятий, день недели, аудиторию, занятия не могут начинаться раньше 9.00 и заканчиваться позднее, чем в 21.00.
2. Триггер, копирующий в архив (специальную таблицу) все данные об изменениях в таблице «Преподаватели» с указанием даты изменения и пользователя, который эти изменения внес.

Вариант 15

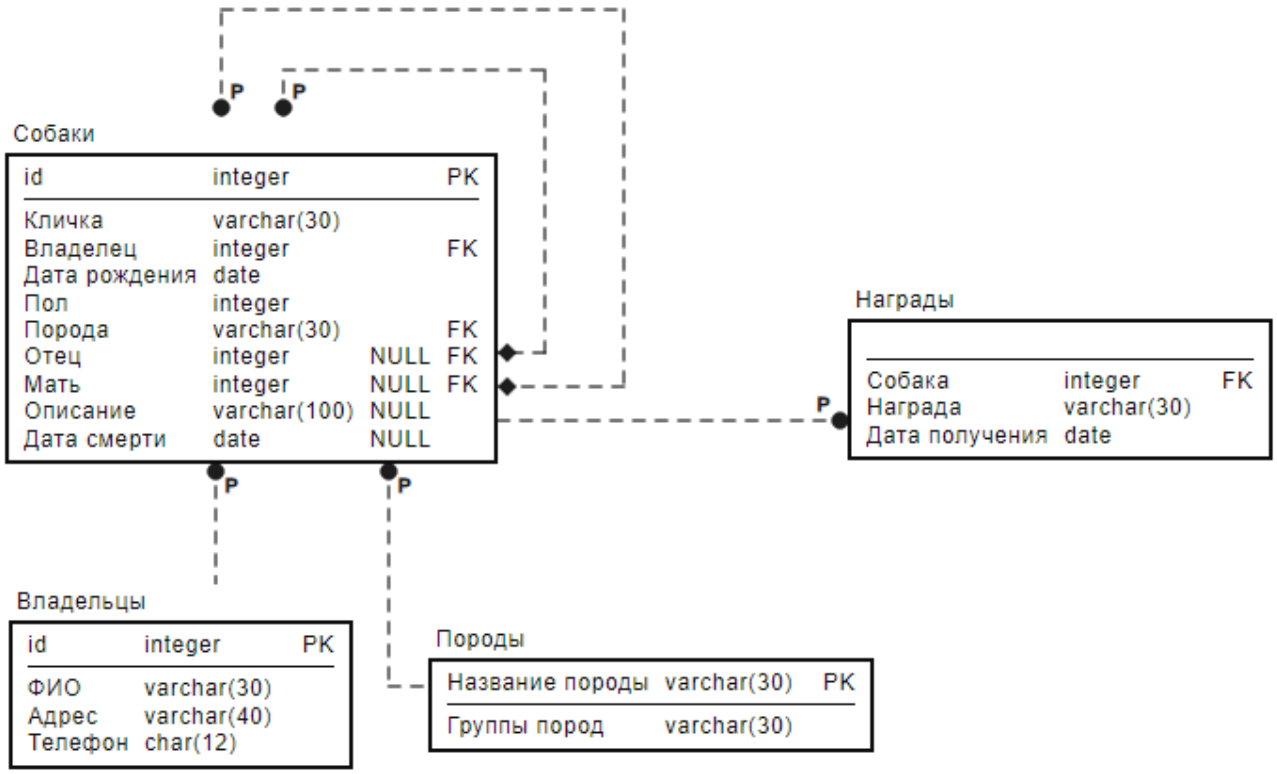


Рисунок А.15 – Вариант 15. БД кинологического клуба

Создание функций.

1. Функция, возвращающая список наград по идентификатору собаки (через запятую). Параметры – идентификатор собаки и период (в годах), за который нужны награды. Если период не указан, считать его равным одному году. Формат: название награды1 (дата), название награды2 (дата),… Награды выводить в порядке получения.

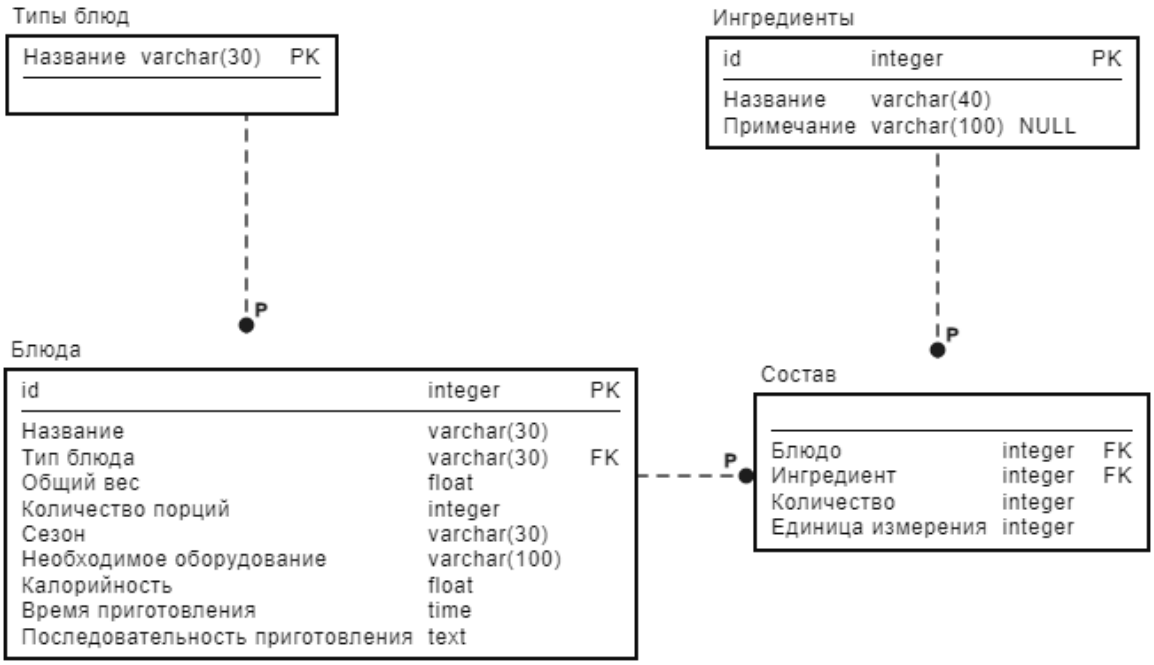
2. Функция, формирующая список собак определенной породы путем добавления данных в специальную таблицу «Список». Таблица должна содержать следующие поля: Название породы, Кличка, Пол, Возраст, Награды. Параметр – название породы. Использовать ранее созданную функцию.

Создание триггеров.

1. Если при вводе данных не указана дата получения награды, устанавливать текущую дату. Если указана дата больше текущей – генерировать ошибку.

2. Проверка значений всех полей отношения «Собаки», для которых могут быть определены домены (дата рождения не больше текущей; пол «м» или «ж»; значение поля «отец» не определено или относится к самцу, поля «мать» – не определено или относится к самке).

Вариант 16



Сезон – список значений (лето, зима, все, весна-лето и т.д.)
Необходимое оборудование – плита, духовка, микроволновая печь и т.д
Калорийность – на 100 г продукта

Рисунок А.16 – Вариант 16. БД рецептов блюд

Создание функций.

1. Функция, возвращающая состав блюда одной строкой. Параметр: идентификатор блюда. Формат результата: название ингредиента (количество единица измерения), … Например: «мука (1 ст.), яйцо (2 шт.), вода (100 мл), соль (3 г)».

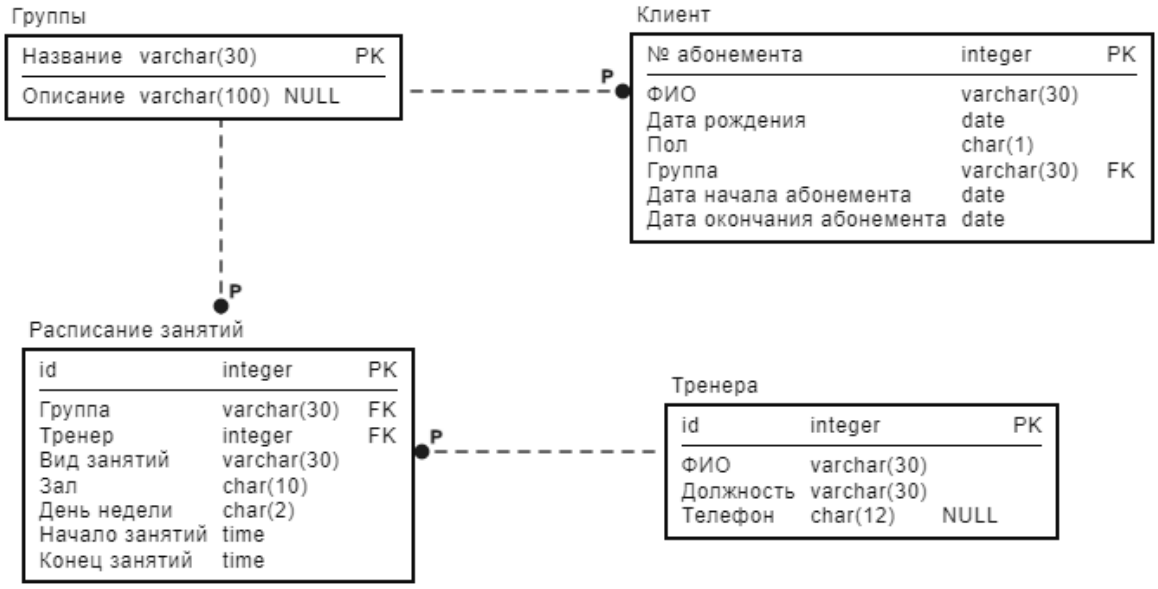
2. Функция подбора блюд по значениям входных параметров. Параметры: тип блюда, ингредиенты (одной строкой, через запятую), сезон, максимальное время приготовления. Если сезон не указан, определять по текущей дате. Если не указано время, не учитывать его при выборе. Блюдо подходит, если в его рецепте есть хотя бы один ингредиент из перечисленных в параметре (и совпадает сезон, тип и время не больше указанного). Результат работы функции добавляется в таблицу «Подбор», которая в начале работы очищается от старых данных. Поля таблицы совпадают с полями таблицы «Блюда» плюс поле «состав». Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Блюда», для которых могут быть определены домены (количество порций больше 0, вес больше 10 (граммов), калорийность больше 0, сезон – список значений). Если при вводе данных не указано количество порций, устанавливать 1.

2. Автоматизация переноса в архив (специальную таблицу) данных о блюде при удалении его из основной таблицы.

Вариант 17



День недели – пн, вт, ср, чт, пт, сб, вс
Пол – значения – «м» и «ж», по умолчанию – «ж»

Рисунок А.17 – Вариант 17. БД фитнес-клуба

Создание функций.

1. Функция, возвращающая по ФИО фамилию с инициалами. Параметр: ФИО (одной строкой). При невозможности преобразования должна возвращать исходное значение.

2. Функция, создающая отчет о некорректных данных в расписании для тренеров путем добавления данных в специальную таблицу «Отчет». Таблица должна содержать следующие поля: ФИО тренера, день недели, количество занятий, начало первого занятия, начало следующего занятия. Расписание считается некорректным, если у одного тренера более 4-х тренировок в день или если время разных занятий у одного тренера совпадает или пересекается. Первое и следующее занятия – это те занятия, которые пересекаются по времени, или перерыв между которыми меньше 20 минут (каждая пара пересекающихся занятий – в отдельной строке). Если таких занятий нет, то эти поля пустые; если занятий не больше 4-х – соответствующее поле пустое. Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка значений всех полей отношения «Клиенты», для которых могут быть определены домены (возраст не менее 14 лет; пол «м» или «ж»; вес не менее 40 кг, рост от 140 см до 220 см, окончание действия абонемента больше начала его действия). Если при вводе данных не указана дата начала действия абонемента, устанавливать текущую дату.

2. Автоматизация переноса в архив (специальную таблицу) всех изменений о клиентах (с указанием пользователя, вносящего изменения, и даты).

Вариант 18

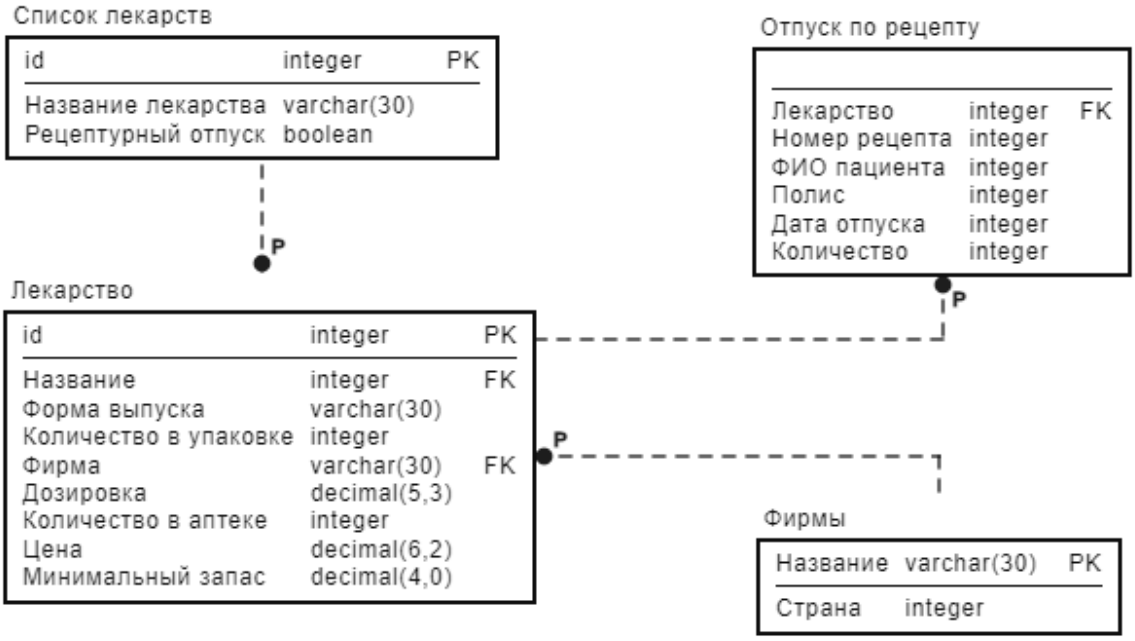


Рисунок А.18 – Вариант 18. БД аптеки

Создание функций.

1. Функция, возвращающая строку «необходимо закупить», если количество упаковок, имеющихся в аптеке, меньше, чем минимальный запас, строку «подходит к концу», если количество упаковок, имеющихся в аптеке, отличается от минимального запаса менее чем на 30%; пустую строку в остальных случаях. Аргументы функции: количество лекарства в аптеке и минимальный запас.

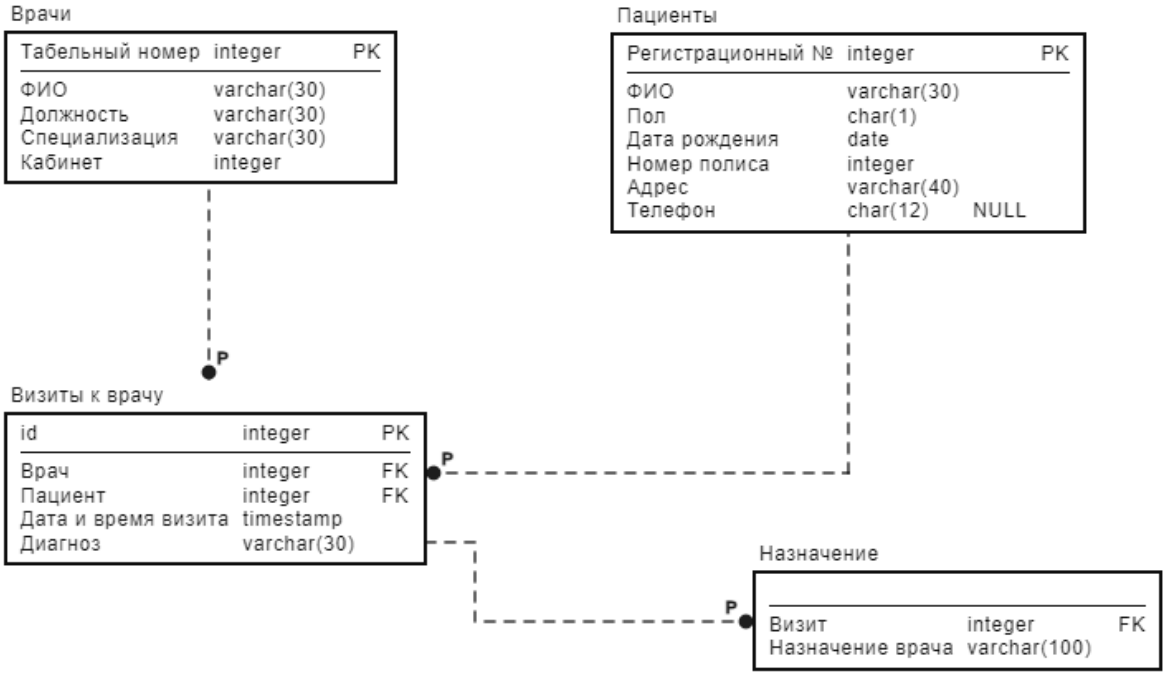
2. Функция, выводящая в специальную таблицу «Отчет» список всех лекарств с пометкой о необходимости закупки или о подходящем к концу запасе лекарства. Параметр – режим: 0 – функция должна перед началом работы очищать таблицу, 1 – новые данные просто добавляются в таблицу. Таблица должна содержать следующие поля: название лекарства, фирма-производитель, остаток, примечание. Использовать ранее созданную функцию, результат которой записывается в поле Примечание.

Создание триггеров.

1. Проверка значений всех полей отношения «Лекарства», для которых могут быть определены домены, например: количество таблеток в упаковке и цена – положительные числа. Для каждого поля выдавать свое сообщение об ошибке.

2. При удалении данных о лекарствах – перенос этих данных в архив (в специальную таблицу) с указанием даты переноса и пользователя, который это сделал.

Вариант 19



Пол – значения – «м» и «ж», по умолчанию – «ж»

Рисунок А.19 – Вариант 19. БД поликлиники

Создание функций.

1. Функция, определяющая по дате рождения и текущей дате, к какой категории относится пациент:

Н	Новорождённый	от 1 до 10 дней
Г	Грудной ребёнок	от 10 дней до 1 года
Д-0	Раннее детство	от 1 до 2 лет
Д-1	I период детства	от 3 до 7 лет
Д-2	II период детства	от 8 до 12 лет
Пд	Подростковый возраст	от 13 до 16 лет
Ю	Юношеский возраст	от 17 до 21 года
Ср	Средний возраст	от 22 до 60 лет
Пж	Пожилой человек	от 61 до 75 лет
Ст	Старческий возраст	от 76 до 90 лет
Д	Долгожитель	старше 90 лет

2. Функция, записывающая в специальную таблицу ошибки в расписании приемов врачей. Параметры – начало о конец периода, за который выводятся данные. Корректные данные удовлетворяют следующим ограничениям: 1) все приемы происходят с 8.00 до 20.00, интервал между приемами – 15 минут; 2) один и тот же врач в одно время принимает только одного пациента (или ни одного, если никто не записался); 3) разница по времени между первым и последним приемом одного врача в один день должна быть меньше 8-ми часов; 4) два разных приема не могут проходить в одно время в одном кабинете. Таблица с ошибками должна содержать следующие поля: ФИО врача, специализация, дата и время приема, описание ошибки.

Создание триггеров.

1. При удалении информации о пациентах, занесение данных о них в архив (специальную таблицу) с указанием даты переноса.

2. При добавлении записи в отношение «Назначения» проверять, не назначен ли пациенту визит к другому врачу в это же время.

Вариант 20

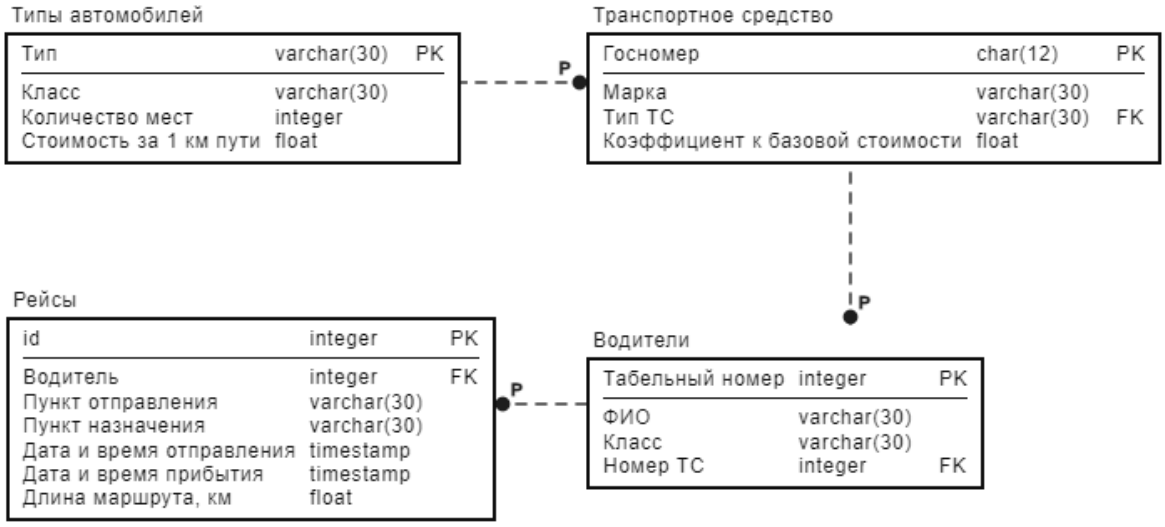


Рисунок А.20 – Вариант 20. БД транспортного предприятия

Создание функций.

1. Функция, определяющая продолжительность транспортного рейса. Параметры: дата и время отправления, дата и время прибытия.

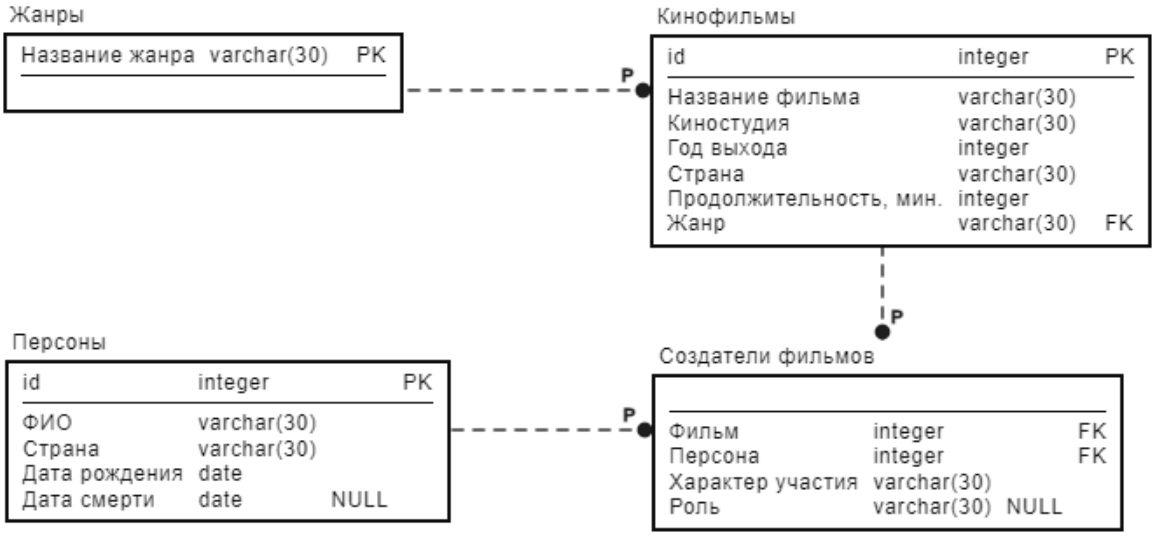
2. Функция, записывающая в специальную таблицу ошибки в расписании рейсов. Параметры – начало о конец периода, за который выводятся данные. Корректные данные удовлетворяют следующим ограничениям: 1) рейс не может длиться более 8-ми часов; 2) интервал между двумя последовательными рейсами – не менее 30 минут, если рейсы длятся менее 4-х часов, не менее 2-х часов, если хотя бы один из рейсов длится более 4-х часов; 3) рейсы одного водителя не могут пересекаться по времени; 4) следующий рейс одного водителя должен начинаться из того же населенного пункта, в который он приехал предыдущим рейсом. Таблица с ошибками должна содержать следующие поля: ФИО водителя, дата/время отправления предыдущего рейса, пункт отправления предыдущего рейса, дата/время отправления следующего рейса, пункт отправления следующего рейса. Если нарушается правило (1), то два последних поля пустые.

Создание триггеров.

1. Проверка значений всех полей отношения «Рейсы», для которых могут быть определены домены, например: время отправления не может быть раньше времени прибытия, длина маршрута не может быть отрицательной или быть равной нулю, пункт отправления не может быть равен пункту прибытия.

2. При изменении данных о водителе – перенос старой информации о нём в специальную таблицу (архив).

Вариант 21



Роль – для актеров

Рисунок А.21 – Вариант 21. БД кинофильмов

В таблице «Кинофильмы» тип поля «Продолжительность» надо изменить на interval.

Создание функций.

1. Функция, возвращающая перечень создателей фильма в виде строки «А.Алов, В.Наумов». Параметры: идентификатор фильма и вид участия (режиссер, актер и т.д.) Для актеров возвращаемая строка примет следующий вид: «Е.Леонов (Сарафанов), Н.Караченцов (Бусыгин)» – в скобках указывается роль.

2. Функция, выводящая в специальную таблицу «Подбор» фильмы по определенному жанру. Параметры: жанр и период выхода, например, («Фантастика», 2018, 2020). Если второй и/или последний параметр не указан, считать их равным текущему году. Перед началом работы функция должна очищать таблицу. Таблица должна содержать следующие поля: Название фильма, Страна, Год выхода, Продолжительность, Режиссеры, Сценаристы, Операторы, Композиторы, Актеры. Использовать ранее созданную функцию.

Создание триггеров.

1. Проверка и изменение данных в таблице «Кинофильмы». Продолжительность фильма больше 10 минут. Если при вводе данных год выхода фильма не указан, устанавливать текущий год. Если указан год больше текущего – генерировать ошибку. Содержимое поля «Страна» изменять следующим образом: РФ, Российская Федерация – «Россия»; Великобритания – «Англия»; Голландия – «Нидерланды»; КНР – «Китай».

2. При удалении данных о фильмах записывать их в архив (в специальную таблицу). Добавлять туда дату внесения данных и имя пользователя, который удалил фильм.