МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах

Кафедра «Информационные системы»

**ОТЧЕТ**

о выполнении лабораторной работы №2

по дисциплине “Управление данными”

Выполнил: ст. гр. ИС/б-20-2-о

Белик Г. М.

Проверил: ст. преподаватель каф.

«Информационные системы»

Абрамович А.Ю.

Севастополь

2022

**ЛАБОРАТОРНАЯ РАБОТА №2**

**«Язык SQL. Генераторы. Триггеры.»**

**Цель работы**

Ознакомится с принципом работы генераторов и триггеров, продемонстрировать работу на примерах.

**Задача**

1. Записать запросы, соединяющие две таблицы с помощью JOIN и без него (вариант задания соответствует варианту, полученному в Лабораторной работе №1).

2. Записать запросы, соединяющие более двух таблиц с помощью JOIN и без него.

3. Продемонстрировать следующие возможности SQL:

- использование псевдонимов на примере рекурсивного запроса;

- привести пример запроса с подзапросом;

- использование агрегатных функций в подзапросе;

- подзапросы, возвращающие единственное и множественные значения;

- подзапросы, использующие вычисление;

- использование подзапросов в HAVING.

4. Записать запрос, соединяющий таблицу со своей копией.

5. Привести пример коррелированного запроса, использующего две разные таблицы.

6. Продемонстрировать следующие возможности SQL - работу оператора EXISTS; - работу оператора ALL; - работу оператора ANY.

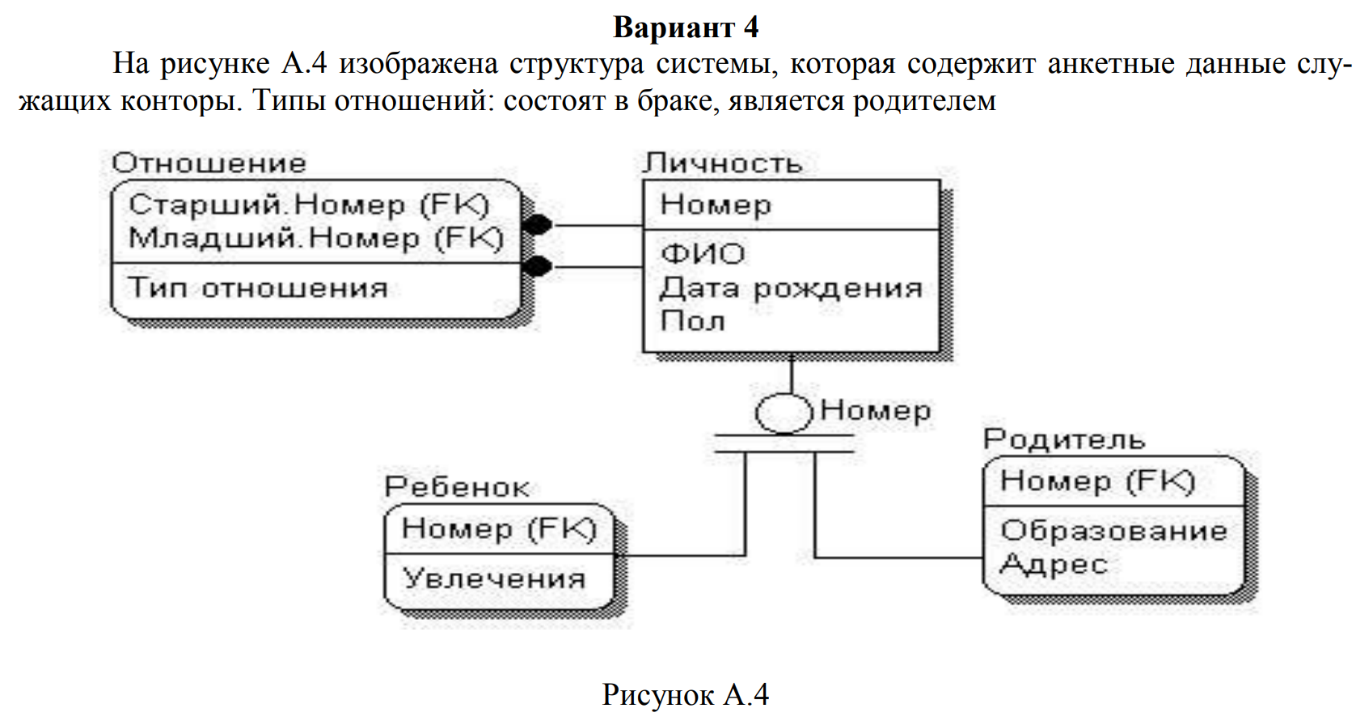
7. В соответствии с вариантом задания создать генератор и триггер (см. приложение А).

8. Изменить значение генератора, в соответствии с хранимыми данными.

9. Ввести данные в таблицу, используя генератор (не менее 5 строк). Просмотреть полученный результат.

10. Внести изменения в указанные таблицы, используя триггеры (не менее 5 строк). Просмотреть полученный результат.

11. Написать отчет.



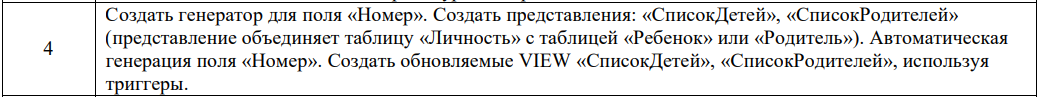


Рисунок 1 – Вариант задания

**Ход работы**

Для начала выполнения работы была создана база данных и создано представление people, соединяющее атрибуты таблиц личностей и родителей без join (Рисунок 2).

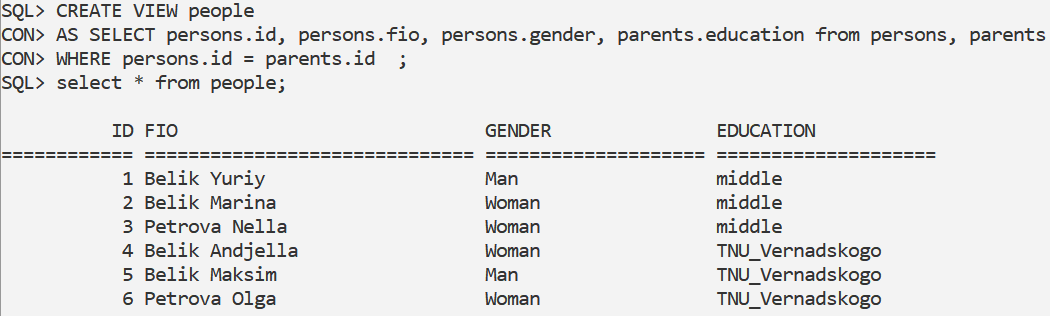
****

Рисунок 2 – Представление people

Затем было создано аналогичное представление с помощью JOIN(Рисунок 3).

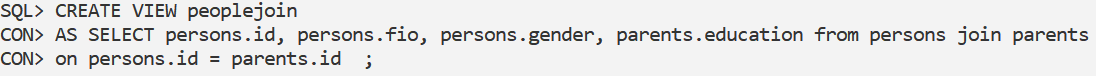


Рисунок 3 – Представление peoplejoin

После этого было создано представление people3, соединяющее 3 таблицы:

личности, родителей и детей(Рисунок 4).

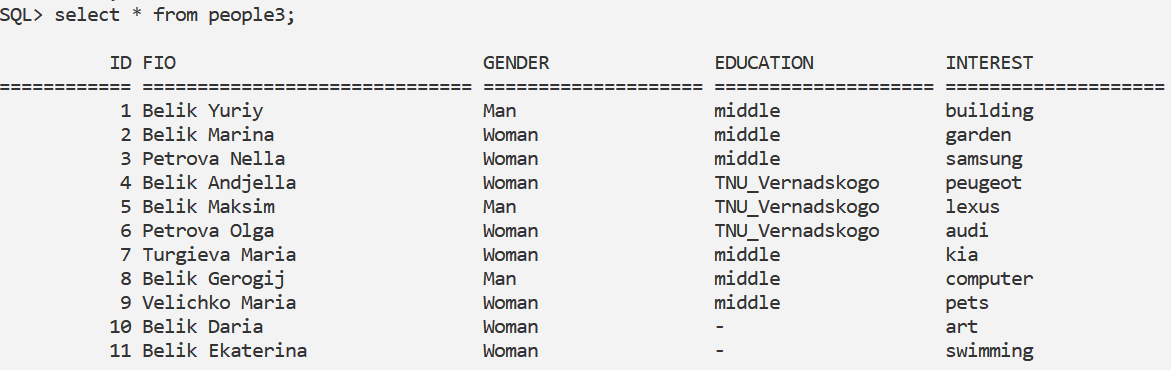


Рисунок 4 – Представление people3

Далее были использованы псевдонимы на примере рекурсивного запроса пар личностей, у которых одинаковый пол (Рисунок 5).

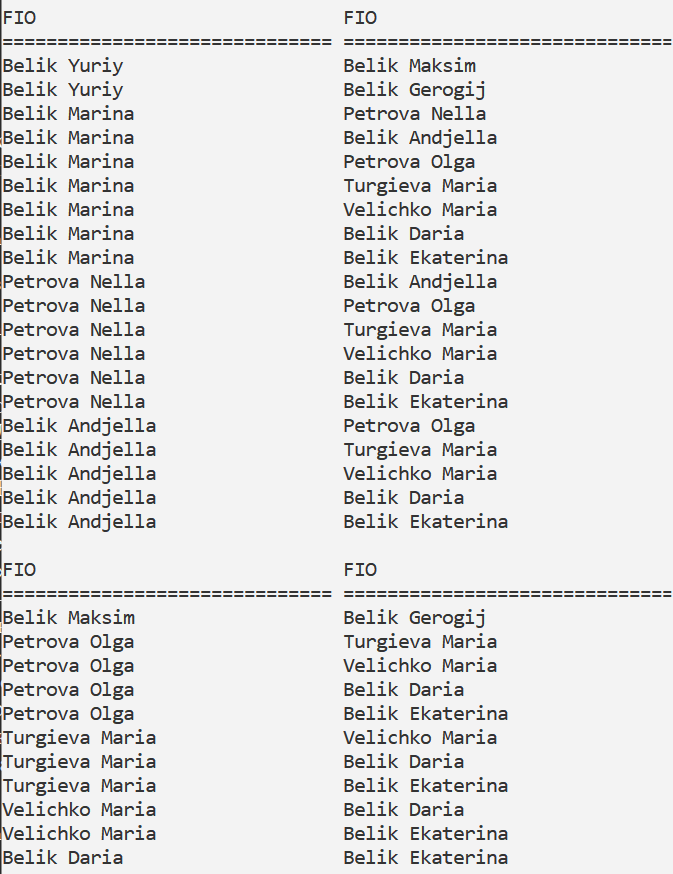


Рисунок 5 – Рекурсивный запрос с использованием псевдонимов.

После этого был совершен запрос с подзапросом: выбор атрибутов из таблицы отношений, где id равен строке с фио ‘Velichko Maria’ (Рисунок 6).

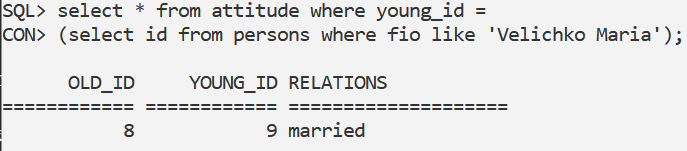


Рисунок 6 – Запрос с подзапросом.

Затем была применена агрегатная функция в подзапросе (Рисунок 7).

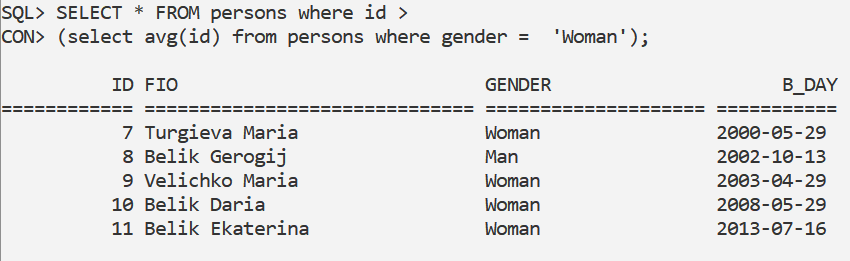


Рисунок 7 – Агрегатная функция в подзапросе.

Далее использовали подзапрос, возвращающий единственное значение (Рисунок 8).

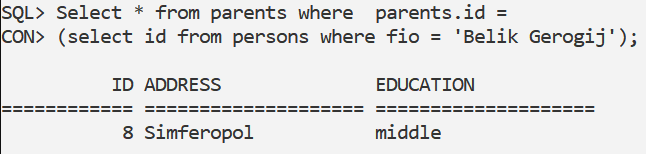


Рисунок 8 – Подзапрос, возвращающий единственное значение.

После этого создали подзапрос, возвращающий множество значений (Рисунок 9).

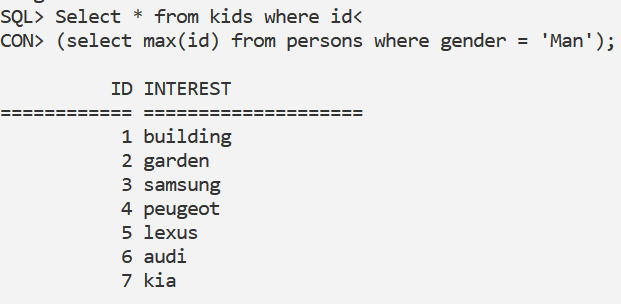


Рисунок 9 – подзапрос, возвращающий множество значений.

Затем продемонстрировали подзапрос, использующий вычисление. Выбрали все id из таблицы детей, у которых id меньше или равен максимальному id среди родителей, умноженный на 1.0 (Рисунок 10).

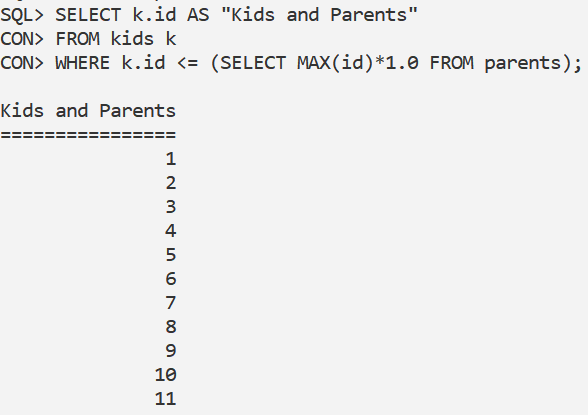


Рисунок 10 – Подзапрос с вычислением.

Далее использовали подзапрос в HAVING. Выбраны все id личностей, у которых номер меньше или равен среднему номеру личностей. (Рисунок 11).

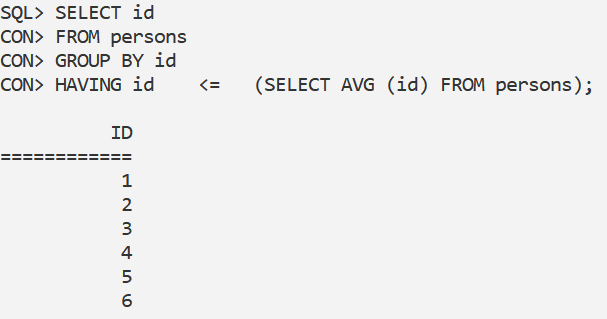
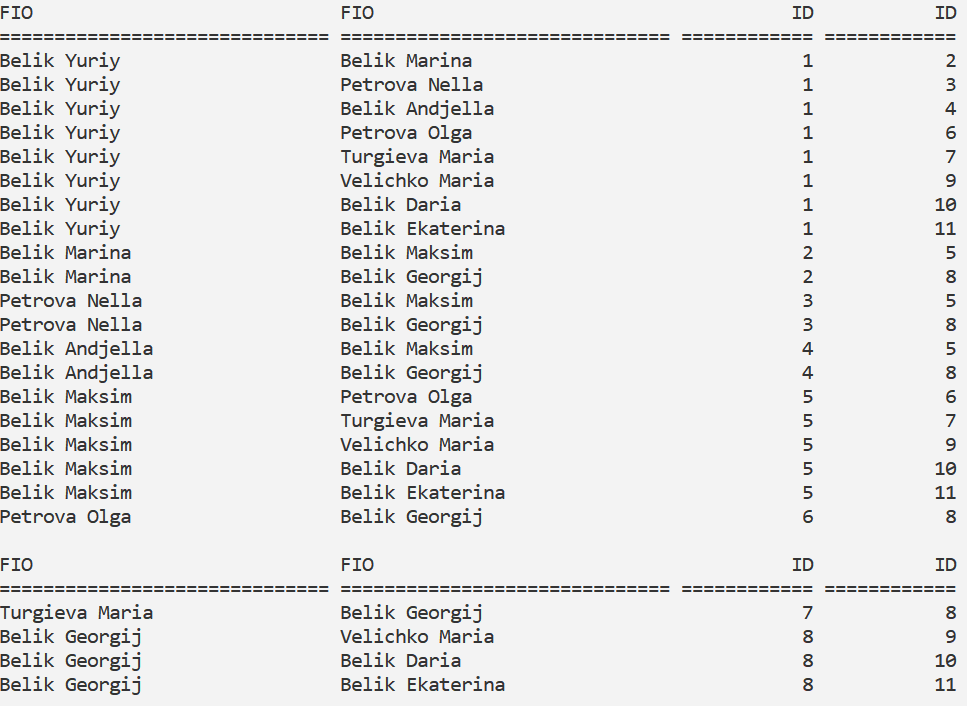


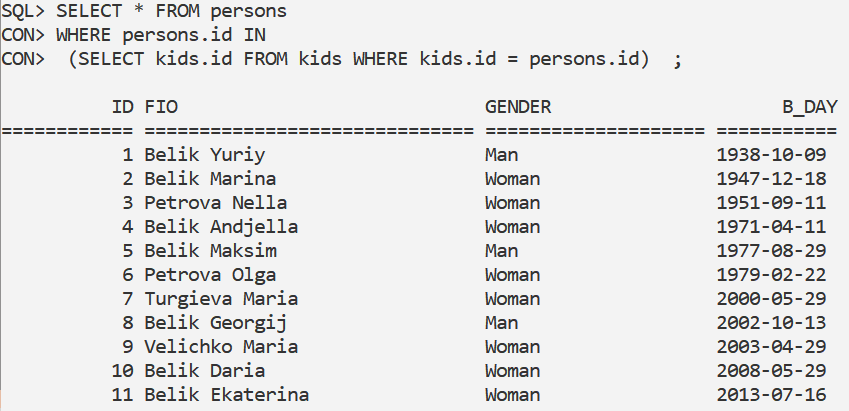
Рисунок 11 – Подзапрос в HAVING.

Далее был записан запрос, соединяющий таблицу со своей копией. Были выбраны пары личностей, у которых разный пол (Рисунок 12).

 Рисунок 12 – Запрос, соединяющий таблицу со своей копией.

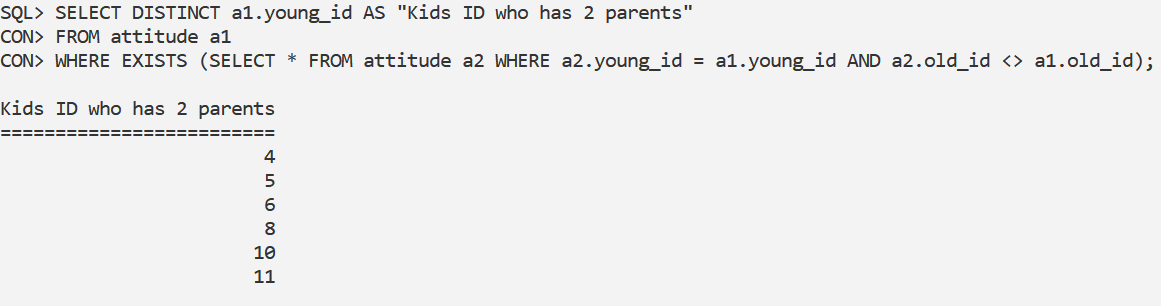
Затем был приведен пример коррелированного запроса, использующего две разные

таблицы (Рисунок 13).

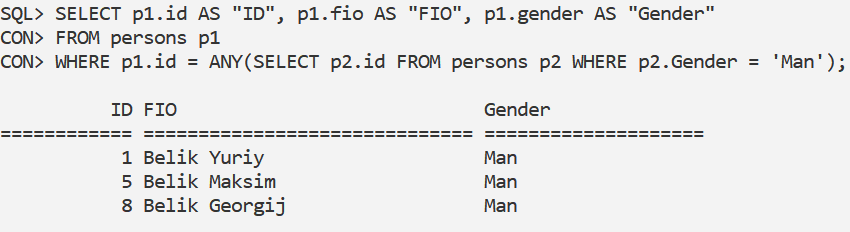
 Рисунок 13 – Коррелированный запрос, использующий две разные таблиц.

После этого были продемонстрированы возможности SQL:

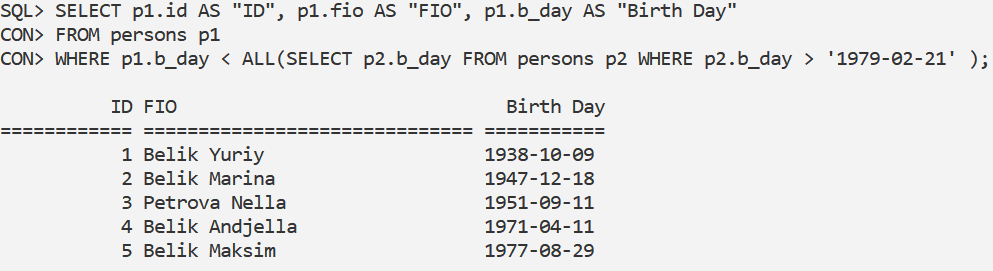
Работа оператора EXISTS (Рисунок 14).

 Рисунок 14 – Запрос с использованием оператора EXISTS.

Работа оператора ANY (Рисунок 15).

 Рисунок 15 – Запрос с использованием оператора ANY.

Работа оператора ALL (Рисунок 16).

 Рисунок 16 – Запрос с использованием оператора ALL.

Затем был создан генератор для поля “ Номер” таблицы личностей “persons” (Рисунок 17).

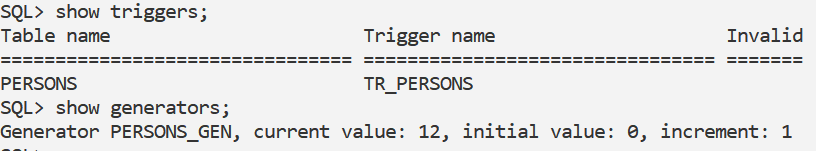
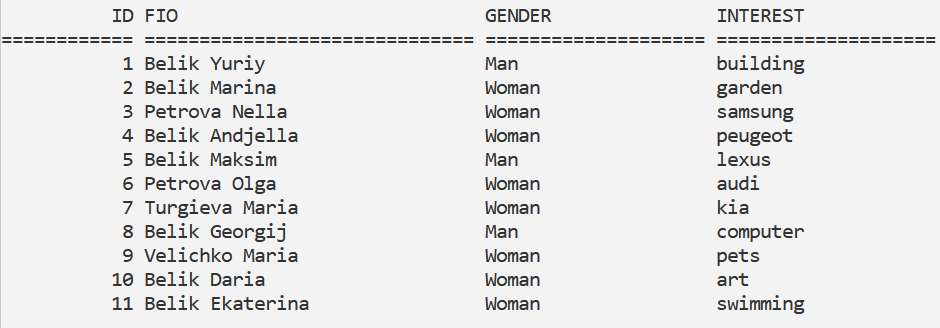
****

Рисунок 17 – Генератор для поля “Номер”.

Далее было создано представление “СписокДетей”, объединяющее таблицу «Личность» с таблицей «Ребенок» (Рисунок 18).

**** Рисунок 18 – Представление “СписокДетей”.

Далее было создано представление “СписокРодителей”, объединяющее таблицу «Личность» с таблицей «Родитель» (Рисунок 19).

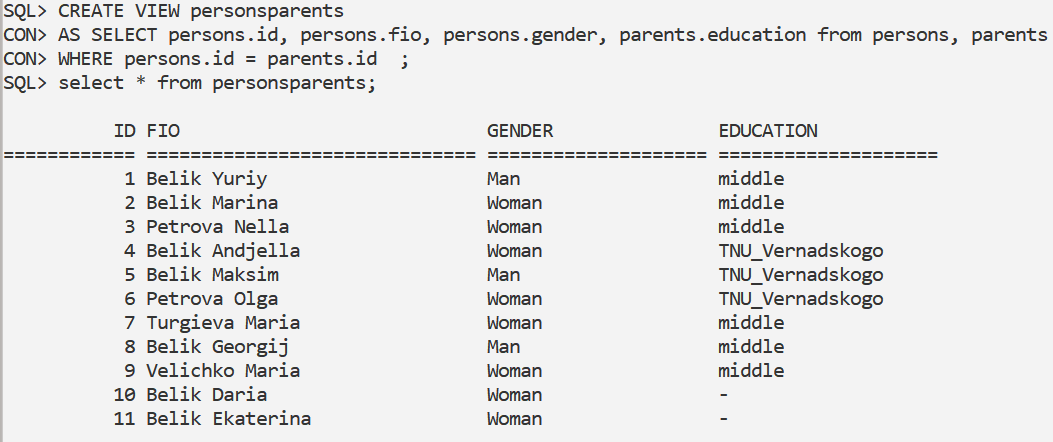
****

Рисунок 19 – Представление “СписокДетей”.

После этого было изменено значение генератора и продемонстрирована работа его работа для таблицы persons (Рисунок 20).

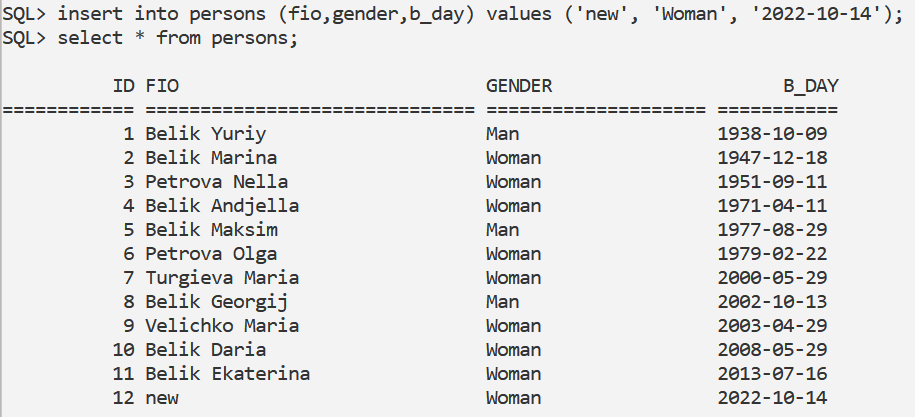
****

Рисунок 20 – Работа генератора persons\_gen.

Затем ввели 5 строк в таблицу, используя генератор (Рисунок 21).

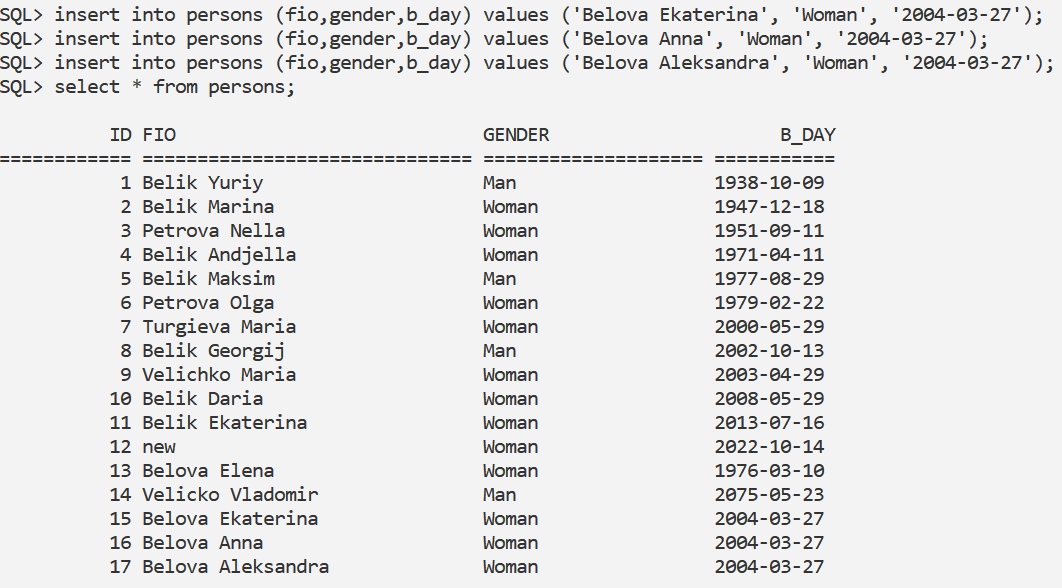
****

Рисунок 21 – Работа генератора persons\_gen.

**Выводы**

В ходе выполнения данной лабораторной работы было совершено ознакомление с принципом работы генераторов и триггеров, и продемонстрирована их работа на примерах.

**ПРИЛОЖЕНИЕ А**

**Тексты запросов**

/\* Создание базы данных \*/

CREATE database 'C:\BELIK2.fdb'

user 'SYSDBA' password 'masterkey'

page\_size = 4096

default character set win1251;

/\*----- Создание базовых таблиц -----\*/

/\* Создание таблицы личностей \*/

CREATE TABLE persons(

id INTEGER NOT NULL PRIMARY KEY,

fio varchar(30) NOT NULL,

gender varchar(20) NOT NULL,

b\_day DATE

);

/\* Создание таблицы детей \*/

CREATE TABLE kids(

id INTEGER NOT NULL PRIMARY KEY,

interest varchar(20) NOT NULL,

FOREIGN KEY (id) REFERENCES persons ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Создание таблицы родитель \*/

CREATE TABLE parents(

id INTEGER NOT NULL PRIMARY KEY,

address VARCHAR(20) default 'Russia',

education VARCHAR(20) default 'middle',

FOREIGN KEY (id) REFERENCES persons ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Создание таблицы отношение \*/

CREATE TABLE attitude(

old\_id INTEGER NOT NULL,

young\_id INTEGER NOT NULL,

relations VARCHAR(20) CHECK (relations = 'married' or relations = 'kids'),

PRIMARY KEY (old\_id, young\_id),

FOREIGN KEY (old\_id) REFERENCES persons ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (young\_id) REFERENCES persons ON DELETE CASCADE ON UPDATE CASCADE

);

/\*----- Заполнение таблиц -----\*/

/\* Ввод значений личностей \*/

insert into persons values(1, 'Belik Yuriy', 'Man', '1938-10-09');

insert into persons values(2, 'Belik Marina', 'Woman', '1947-12-18');

insert into persons values(3, 'Petrova Nella', 'Woman', '1951-09-11');

insert into persons values(4, 'Belik Andjella', 'Woman', '1971-04-11');

insert into persons values(5, 'Belik Maksim', 'Man', '1977-08-29');

insert into persons values(6, 'Petrova Olga', 'Woman', '1979-02-22');

insert into persons values(7, 'Turgieva Maria', 'Woman', '2000-05-29');

insert into persons values(8, 'Belik Georgij', 'Man', '2002-10-13');

insert into persons values(9, 'Velichko Maria', 'Woman', '2003-04-29');

insert into persons values(10, 'Belik Daria', 'Woman', '2008-05-29');

insert into persons values(11, 'Belik Ekaterina', 'Woman', '2013-07-16');

/\* Ввод значений родителей \*/

INSERT INTO parents (id, address) VALUES (1, 'Norilsk' );

INSERT INTO parents (id, address) VALUES (2, 'Belgorod' );

INSERT INTO parents (id, address) VALUES (3, 'Tambov' );

INSERT INTO parents VALUES (4, 'Norilsk','TNU\_Vernadskogo');

INSERT INTO parents VALUES (5, 'Norilsk','TNU\_Vernadskogo');

INSERT INTO parents VALUES (6, 'Simferopol','TNU\_Vernadskogo');

INSERT INTO parents VALUES (7, 'Vladikavkaz', 'middle');

INSERT INTO parents VALUES (8, 'Simferopol', 'middle');

INSERT INTO parents VALUES (9, 'Evpatoria','middle');

INSERT INTO parents VALUES (10, 'Simferopol','-');

INSERT INTO parents VALUES (11, 'Simferopol','-');

/\* Ввод значений детей \*/

INSERT INTO kids VALUES (1, 'building');

INSERT INTO kids VALUES (2, 'garden');

INSERT INTO kids VALUES (3, 'samsung');

INSERT INTO kids VALUES (4, 'peugeot');

INSERT INTO kids VALUES (5, 'lexus');

INSERT INTO kids VALUES (6, 'audi');

INSERT INTO kids VALUES (7, 'kia');

INSERT INTO kids VALUES (8, 'computer');

INSERT INTO kids VALUES (9, 'pets');

INSERT INTO kids VALUES (10, 'art');

INSERT INTO kids VALUES (11, 'swimming');

/\*Ввод значений отношение \*/

INSERT INTO attitude values(1,2, 'married');

INSERT INTO attitude values(1,4, 'kids');

INSERT INTO attitude values(1,5, 'kids');

INSERT INTO attitude values(2,4, 'kids');

INSERT INTO attitude values(2,5, 'kids');

INSERT INTO attitude values(3,6, 'kids');

INSERT INTO attitude values(5,6, 'married');

INSERT INTO attitude values(4,7, 'kids');

INSERT INTO attitude values(5,8, 'kids');

INSERT INTO attitude values(5,10, 'kids');

INSERT INTO attitude values(5,11, 'kids');

INSERT INTO attitude values(6,8, 'kids');

INSERT INTO attitude values(6,10, 'kids');

INSERT INTO attitude values(6,11, 'kids');

INSERT INTO attitude values(8,9, 'married');

CREATE VIEW people

AS SELECT persons.id, persons.fio, persons.gender, parents.education from persons, parents

WHERE persons.id = parents.id ;

CREATE VIEW peoplejoin

AS SELECT persons.id, persons.fio, persons.gender, parents.education from persons join parents

on persons.id = parents.id ;

CREATE VIEW people3

AS SELECT persons.id, persons.fio, persons.gender, parents.education, kids.interest from persons, parents, kids

WHERE persons.id = parents.id and persons.id = kids.id;

Select A.fio , B.fio from persons A, persons B

where A.gender = B.gender and A.id<B.id;

select \* from attitude where young\_id =

(select id from persons where fio like 'Velichko Maria');

SELECT \* FROM persons where id >

(select avg(id) from persons where gender = 'Woman');

Select \* from parents where parents.id =

(select id from persons where fio = 'Belik Georgij');

Select \* from kids where id<

(select max(id) from persons where gender = 'Man');

-- Подзапрос с вычислениями

SELECT k.id AS "Kids and Parents"

FROM kids k

WHERE k.id <= (SELECT MAX(id)\*1.0 FROM parents);

-- Подзапрос с использованием HAVING

SELECT id

FROM persons

GROUP BY id

HAVING id <= (SELECT AVG (id) FROM persons);

Select persons.fio, copy.fio , persons.id, copy.id

From persons , persons copy

**Where persons.Gender != copy.Gender**

**And persons.id < copy.id;**

SELECT \* FROM persons

WHERE persons.id IN

 (SELECT kids.id FROM kids WHERE kids.id = persons.id)  ;

Запрос в IN (...) сам по себе выполнить нельзя, ибо он зависит от таблицы persons. Такой запрос называют коррелированным.

/\* Демонстрация оператора EXISTS \*/

SELECT DISTINCT a1.young\_id AS "Kids ID who has 2 parents"

FROM attitude a1

WHERE EXISTS (SELECT \* FROM attitude a2 WHERE a2.young\_id = a1.young\_id AND a2.old\_id <> a1.old\_id);

/\* Демонстрация оператора ANY \*/

SELECT p1.id AS "ID", p1.fio AS "FIO", p1.gender AS "Gender"

FROM persons p1

WHERE p1.id = ANY(SELECT p2.id FROM persons p2 WHERE p2.Gender = 'Man');

/\* Демонстрация оператора ALL \*/

SELECT p1.id AS "ID", p1.fio AS "FIO", p1.b\_day AS "Birth Day"

FROM persons p1

WHERE p1.b\_day < ALL(SELECT p2.b\_day FROM persons p2 WHERE p2.b\_day > '1979-02-21' );

CREATE GENERATOR persons\_gen;

CREATE VIEW personskids

AS SELECT persons.id, persons.fio, persons.gender, kids.interest from persons, kids

WHERE persons.id = kids.id ;

CREATE VIEW personsparents

AS SELECT persons.id, persons.fio, persons.gender, parents.education from persons, parents

WHERE persons.id = parents.id ;

SET GENERATOR persons\_gen TO 11;

/\*Создаем триггер\*/

SET TERM ^;

CREATE TRIGGER Tr\_persons FOR persons

ACTIVE BEFORE INSERT

AS

BEGIN

IF (NEW.ID IS NULL) THEN NEW.ID = GEN\_ID(persons\_gen, 1);

END^

SET TERM ;^ /\* Завершаем транзакцию: \*/

COMMIT;

insert into persons (fio,gender,b\_day) values ('new', 'Woman', '2022-10-14');

select \* from persons;

insert into persons (fio,gender,b\_day) values ('Belova Elena', 'Woman', '1976-03-10');

insert into persons (fio,gender,b\_day) values ('Velicko Vladimir', 'Man', '1975-05-23');

insert into persons (fio,gender,b\_day) values ('Belova Ekaterina', 'Woman', '2004-03-27');

insert into persons (fio,gender,b\_day) values ('Belova Anna', 'Woman', '2004-03-27');

insert into persons (fio,gender,b\_day) values ('Belova Aleksandra', 'Woman', '2004-03-27');

**Контрольные вопросы**

1. Назначение генераторов?

Генератор – объект базы данных, служащий для генерации уникальных последовательностей целых чисел для заполнения искусственных ключей.

1. Как сгенерировать следующее значение генератора?

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 GEN\_ID (здесь name – имя генератора, step – инкремент генератора):  
GEN\_ID( name, step);  
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');  
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.

1. Как переустановить значение генератора?

Для того чтобы установить генератор в другое значение, необходимо использовать команду SET GENERATOR:  
SET GENERATOR name TO value;  
Здесь name - имя генератора, value - новое значение генератора (число в диапазоне от - 264 до 264- 1). После вызова оператора SET GENERATOR следующее значение, которое вернет функция GEN\_ID() будет равно value + инкремент, указанный в функции. Перед вызовом функции необходимо удостовериться, что после того, как генератор будет переустановлен, он будет возвращать уникальные значения (если он используется для генерации первичного ключа).

1. Как удалить генератор?

Не существует оператора DROP GENERATOR. Генератор можно удалить, удалив его из системной таблицы:  
DELETE FROM RDB$GENERATORS  
WHERE RDB$GENERATORS\_NAME = 'SubjectID\_GEN';

5. Как повысить надежность данных?

Для повышения надежности данных, в системе управления базами данных используются различные средства обеспечения целостности данных. Среди них можно выделить использование ограничений данных на уникальность, соответствие условию, проверку наличия значения, каскадирование сводных таблиц при изменении данных, а также использование таких конструкций, как генераторы и триггеры.

6. Организация многопользовательского режима доступа к данным?

В многопользовательских базах данных два клиента случайно могут одновременно сгенерировать одинаковый номер при попытке вручную создать уникальное значение для атрибута. Этого недостатка лишен генератор – с его использованием каждый клиент получит разные значения.

7. Что такое «триггер»?

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.

8. Из каких частей состоит триггер?

Оператор состоит из заголовка триггера (до AS) и тела триггера (после AS). 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 блока операторов. Блок состоит из набора операторов на языке хранимых процедур и триггеров, заключенных в операторные скобки.  
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;}

9. Какая информация содержится в заголовочной части 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.

10. Как сделать триггер временно неактивным? Как удалить триггер?

Следующий оператор делает триггер не активным.  
ALTER TRIGGER <имя триггера> INACTIVE;  
Следующий оператор делает удаляет триггер.  
DROP TRIGGER <имя триггера>;

11. Для чего используются триггеры?

Триггеры могут быть использованы для того, чтобы уведомить приложение о том, что произошло некоторое событие.  
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. С помощью триггеров и исключений можно выводить сообщения об ошибке.

12.Назовите элементы языка хранимых процедур и триггеров.

BEGIN...END – определяет блок операторов (операторные скобки).  
variable = expression – оператор присваивания.  
/\* comment\_text \*/ – многострочный комментарий.  
EXCEPTION exception\_name – вызывает исключительную ситуацию с именем exception\_name, если она не обрабатывается оператором WHEN.  
EXECUTE PROCEDURE  
proc\_name [var [, var ...]]  
[RETURNING\_VALUES var [, var ...]] – вызывает хранимую процедуру с именем proc\_name, с указанными входными и выходными параметрами.  
FOR select\_statement DO compound\_statement – повторяет выполнение блока кода следующего за DO для каждой строки, возвращенной оператором select\_statement.  
select\_statement – обычный запрос на выборку, за исключением того, что он обязательно должен содержать часть INTO, и данная часть должна идти на последнем месте.  
compound\_statement – или одиночный оператор языка, или блок операторов заключенных в операторные скобки.  
IF (condition) THEN compound\_statement [ELSE compound\_statement] – условный оператор condition - условие, выражение булевской трехзвенной логики (TRUE, FALSE, UNKNOWN).  
NEW.column – контекстная переменная, содержащая новое значение столбца с именем. column при выполнении операций INSERT или UPDATE.  
OLD.column – контекстная переменная, содержащая старое значение столбца с именем column при выполнении операций INSERT или UPDATE.  
POST\_EVENT event\_name – отсылает сообщение с именем event\_name.  
WHILE (condition) DO compound\_statement – цикл с предусловием.  
WHEN {error [, error .]|ANY} DO compound\_statement – оператор обработки исключительных ситуаций и ошибок. В случае если имеет место одна из ошибок, перечисленных в операторе WHEN, вызывается compound\_statement. Оператор WHEN должен быть последним оператором перед END тела триггера.