Отчет по практическому заданию 4. Применение SQL.

```
hotel_reason=# CREATE VIEW EmployeerView AS SELECT firstname AS Name FROM employees ORDER BY fi
rstname;
CREATE VIEW
```

Изображение 1: создание представления с упорядочиванием на основе столбцов из employees

```
hotel_reason=# SELECT * FROM EmployeerView;
    name
------------
 Александра
 Андрей
 Ольга
 Елена
 Иван
(5 ёЄЁюъ)


hotel_reason=#
```

Изображение 2: вывод

```
hotel_reason=# CREATE VIEW CustomerData AS SELECT firstname, position, phone FROM empl
oyees;
CREATE VIEW
hotel_reason=# SELECT * FROM CustomerData;
 firstname  |   position    |    phone
------------+---------------+-------------
 Елена      | Менеджер      | 89991234567
 Иван       | Администратор | 89997654321
 Александра | Горничная     | 89991234568
 Андрей     | Охранник      | 89991234569
 Ольга      | Консьерж      | 89991234570
```

Изображение 3: создание представления на основе таблицы employees.

```
hotel_reason=# CREATE VIEW BasicCustomerDates AS SELECT firstname, position, phone
FROM employees WHERE phone = '89997654321';
CREATE VIEW
hotel_reason=# SELECT * FROM BasicCustomerDates;
 firstname |   position    |    phone
-----------+---------------+-------------
 Иван      | Администратор | 89997654321
```

Изображение 4: создание представления на основе таблицы employees с условием.

```
hotel_reason=# CREATE VIEW CustoerInterests AS SELECT firstname AS Name, position F
ROM employeers E JOIN CUSTOMER_EMPLOYEER_INT CI ON E.id = CI.id;
```

Изображение 5: использование представления для скрытия сложного синтаксиса

```
hotel_reason=# CREATE OR REPLACE FUNCTION sales_price_check() RETURNS TRIGGER AS $s
ales_price_check$ BEGIN IF NEW.SalesPrice < 0.9 * OLD.AskingPrice THEN NEW.SalesPri
ce := OLD.AskingPrice; NEW.AskingPrice := OLD.AskingPrice; END IF; RETURN NEW; END;
 $sales_price_check$ LANGUAGE plpgsql; CREATE TRIGGER sales_price_check BEFORE UPDA
TE ON employees FOR EACH ROW EXECUTE FUNCTION sales_price_check();
CREATE FUNCTION
CREATE TRIGGER
```

Изображение 6: использование триггеров для проверки допустимости вводимых данных

```
hotel_reason=# CREATE VIEW EmployeeWorkNet AS SELECT W.WorkID, E.first_name || ' '
|| E.last_name AS Name, W.Title, W.Copy, W.AcquisitionPrice, W.SalesPrice, (W.Sales
Price - W.AcquisitionPrice) AS NetPrice FROM TRANSACTION T JOIN WORK W ON T.WorkID
= W.WorkID JOIN employees E ON W.ArtistID = E.id; CREATE OR REPLACE FUNCTION set_as
king_price() RETURNS TRIGGER AS $set_asking_price$ DECLARE avgNetPrice NUMERIC(8,2)
; newPrice NUMERIC(8,2); rowCount INTEGER; BEGIN SELECT COUNT(*) INTO rowCount FROM
 TRANSACTION WHERE WorkID = NEW.WorkID; IF rowCount = 0 THEN NEW.AskingPrice = 2 *
NEW.AcquisitionPrice; ELSE SELECT AVG(NetPrice) INTO avgNetPrice FROM EmployeeWorkN
et EW WHERE EW.WorkID = NEW.WorkID GROUP BY EW.WorkID; newPrice = avgNetPrice + NEW
.AcquisitionPrice; IF newPrice > 2 * NEW.AcquisitionPrice THEN NEW.AskingPrice = ne
wPrice; ELSE NEW.AskingPrice = 2 * NEW.AcquisitionPrice; END IF; END IF; RETURN NEW
; END; $set_asking_price$ LANGUAGE plpgsql; CREATE TRIGGER set_asking_price BEFORE
INSERT ON TRANSACTION FOR EACH ROW EXECUTE FUNCTION set_asking_price();
```

Изображение 7: использование триггеров для присвоения значений по умолчанию

```
hotel_reason=# CREATE OR REPLACE FUNCTION employee_interests_update() RETURNS TRIGG
ER AS $employee_interests_update$ BEGIN UPDATE employees E1 SET first_name = NEW.fi
rst_name WHERE E1.first_name = OLD.first_name AND NOT EXISTS ( SELECT * FROM employ
ees E2 WHERE E2.first_name = E1.first_name AND E2.id <> E1.id ); RETURN NEW; END; $
employee_interests_update$ LANGUAGE plpgsql; CREATE TRIGGER employee_interests_upda
te INSTEAD OF UPDATE ON EmployeeInterests FOR EACH ROW EXECUTE PROCEDURE employee_i
nterests_update();
CREATE FUNCTION
```

Изображение 8: триггер обновляющий представление

```
hotel_reason=# CREATE RULE update_employee_interests AS ON UPDATE TO EmployeeIntere
sts DO INSTEAD UPDATE employees E1 SET first_name = NEW.first_name WHERE E1.first_n
ame = OLD.first_name AND NOT EXISTS ( SELECT * FROM employees E2 WHERE E2.first_nam
e = E1.first_name AND E2.id <> E1.id );
```

Изображение 9: правила

Контрольные вопросы:

1. Среда выполнения PL/SQL - это среда выполнения программ на языке PL/SQL, которая обеспечивает связь между базой данных и программистом, позволяя выполнять операции с данными из базы данных.

2. Неименованный блок PL/SQL - это блок кода на языке PL/SQL, который не имеет имени и может использоваться для выполнения простых задач.

3. Курсоры в PL/SQL - это объекты, которые позволяют программисту перебирать результаты запросов к базе данных и выполнять над ними операции.

4. Атрибуты курсора - это свойства курсора, которые позволяют программисту получить информацию о курсоре, такую как его имя, тип, статус и пр.

5. Неявные курсоры - это курсоры, которые создаются автоматически при выполнении операций в SQL.

6. Курсоры и циклы - это механизмы, используемые в языках программирования и базах данных для обработки наборов данных.

7. Курсоры с параметрами - это курсоры, которые могут принимать параметры и использоваться для выполнения динамических запросов.

8. Курсоры с обновлением - это курсоры, которые могут использоваться для обновления данных в таблице.

9. Процедуры и функции PL/SQL - это блоки кода на языке PL/SQL, которые могут быть вызваны из других блоков кода и выполнять определенные задачи.

10. Строковые и операторные триггеры - это типы триггеров, которые могут быть использованы для автоматического выполнения определенных операций при изменении данных в таблице.

11. Псевдозаписи триггера - это специальные записи, которые доступны внутри триггеров и позволяют программисту получить информацию о том, какие данные были изменены.

12. Триггерные предикаты - это условия, которые выполняются перед выполнением триггера и позволяют определить, должен ли триггер выполниться или нет