```
clinic=# CREATE VIEW PtientImyaFamiliya AS SELECT first_name, last_name FROM patients;
CREATE VIEW
clinic=# SELECT * FROM PtientImyaFamiliya;
 first_name | last_name
------------+-----------
 Vasiliy    | Petrov
 Maria      | Pavlovich
 Ilya       | Haritonov
(3 ёЄЁюъш)


clinic=# CREATE VIEW OrderedByAge AS SELECT age AS patients FROM patients ORDER BY age;
CREATE VIEW
clinic=# SELECT * FROM OrderedByAge;
 patients
----------
       16
       23
       56
(3 ёЄЁюъш)


clinic=#
clinic=# CREATE VIEW OrderedByAge AS SELECT name AS patients FROM patients ORDER BY age;
ОШИБКА:  столбец "name" не существует
СТРОКА 1: CREATE VIEW OrderedByAge AS SELECT name AS patients FROM pat...
                                             ^
ПОДСКАЗКА:  Возможно, предполагалась ссылка на столбец "patients.age".
clinic=# CREATE VIEW OrderedByAge AS SELECT first_name AS patients FROM patients ORDER BY age;
ОШИБКА:  отношение "orderedbyage" уже существует
clinic=# CREATE VIEW OrderedByAge1 AS SELECT first_name AS patients FROM patients ORDER BY age;
CREATE VIEW
clinic=# SELECT * FROM OrderedByAge1;
 patients
----------
 Ilya
 Maria
 Vasiliy
(3 ёЄЁюъш)


clinic=# CREATE VIEW Patient1 AS SELECT first_name, last_name FROM patients WHERE age = 'WA';
ОШИБКА:  неверный синтаксис для типа numeric: "WA"
СТРОКА 1: ...SELECT first_name, last_name FROM patients WHERE age = 'WA';
                                                                    ^
clinic=# CREATE VIEW Patient1 AS SELECT first_name, age FROM patients WHERE last_name = 'WA';
CREATE VIEW
clinic=# SELECT * FROM Patient1;
 first_name | age
------------+-----
(0 ёЄЁюъ)
```

```
clinic=# CREATE OR REPLACE FUNCTION patient_insert (
clinic(# newname IN char,
clinic(# newareacode IN char,
clinic(# newphone IN char,
clinic(# doctorsprofession IN char)
clinic-# RETURNS int AS $patient_insert$ DECLARE doctorcursor CURSOR FOR SELECT profession FROM doctors WHERE profession = doctorsprofession;
clinic$# rowcount int;
clinic$# BEGIN
clinic$# SELECT Count(*) INTO rowcount FROM CUSTOMER WHERE first_name = newname AND last_name = newlastname AND profession = newprofession;
clinic$# IF rowcount > 0 THEN RAISE EXCEPTION 'There is client in DB', END IF;
clinic$# CREATE OR REPLACE FUNCTION sales_price_check() RETURNS trigger AS $sales_price_check$
clinic$# BEGIN
clinic$# IF NEW.SalesPrice < 0.9 * OLD.AskingPrice THEN NEW.SalesPrice = OLD.AskingPrice;
clinic$# NEW.AskingPrice = OLD.AskingPrice;
clinic$# END IF;
clinic$# RETURN NEW;
clinic$# END;
clinic$# $sales_price_check$ LANGUAGE plpgsql;
clinic$# CREATE TRIGGER ROW EXECUTE PROCEDURE sales_price_check();
clinic$# CREATE VIEW ArtistWorkNet AS SELECT W.WorkID, Name, Title, Copy, AcquisitionPrice, SalesPrice, (SalesPrice - AcquisitionPrice) AS NetPrice
 FROM TRANSACTION T JOIN WORK W ON T.WorkID = W.WorkID JOIN ARTIST A ON W.ArtistID = A.ArtistID;
clinic$# CREATE OR REPLACE FUNCTION set_asking_price() RETURNS trigger AS $set_asking_price$ DECLARE avgNetPrice numeric(8,2);
clinic$# newPrice numeric(8,2);
clinic$# rowcount integer;
clinic$# BEGIN SELECT Count(*) INTO rowcount
clinic$# FROM TRANSACTION WHERE WorkID = NEW.WorkID;
clinic$# IF rowcount = 0 THEN NEW.AskingPrice = 2 * (NEW.AcquisitionPrice);
clinic$# ELSE SELECT AVG(NetPrice) INTO avgNetPrice FROM ArtistWorkNet AW WHERE AW.WorkID = NEW.WorkID GROUP BY AW.WorkID;
clinic$# newPrice = avgNetPrice + NEW.AcquisitionPrice;
clinic$# IF newPrice > 2 * (NEW.AcquisitionPrice) THEN NEW.AskingPrice = newPrice;
clinic$# ELSE NEW.AskingPrice = 2 * (NEW.AcquisitionPrice);
clinic$# END IF;
clinic$# END IF;
clinic$# RETURN NEW;
clinic$# END;
clinic$# $set_asking_price$ LANGUAGE plpgsql;
clinic$# CREATE TRIGGER set_asking_price BEFORE INSERT ON TRANSACTION FOR EACH ROW EXECUTE PROCEDURE set_asking_price();
clinic$# CREATE OR REPLACE FUNCTION customer_interests_update() RETURNS trigger AS $customer_interests_update$ BEGIN UPDATE CUSTOMER C1 SET Name =
NEW.Customer WHERE C1.Name = OLD.Customer
clinic$# AND NOT EXISTS (
clinic$# SELECT * FROM CUSTOMER C2 WHERE C2.Name = C1.Name AND C2.CustomerID <> C1.CustomerID );
clinic$# RETURN NEW;
clinic$# END;
clinic$# $customer_interests_update$ LANGUAGE plpgsql;
clinic$# CREATE TRIGGER customer_interests_update INSTEAD OF UPDATE ON CustomerInterests FOR EACH ROW EXECUTE PROCEDURE customer_interests_update()
;
clinic$# CREATE OR REPLACE FUNCTION enforce_trans_child() RETURNS trigger AS $enforce_trans_child$ DECLARE rowcount int;
clinic$# NewID int;
clinic$# BEGIN NewID = NEW.WorkID;
clinic$# SELECT Count(*) INTO rowcount FROM TRANSACTION WHERE WorkID = NewID AND CustomerID IS NULL;
clinic$# IF rowcount = 0 THEN INSERT INTO TRANSACTION (TransactionID, DateAcquired, WorkID) VALUES (nextval('seq_transaction'), NOW(), NewID);
clinic$# END IF;
clinic$# RETURN NEW;
clinic$# END;
clinic$# $enforce_trans_child$ LANGUAGE plpgsql;
clinic$# CREATE TRIGGER enforce_trans_child AFTER INSERT ON WORK FOR EACH ROW EXECUTE PROCEDURE enforce_trans_child();
```

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-циклы - это специальный тип курсоров, который позволяет перебирать значения в таблице в цикле.

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