КР 3 ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БД PostgreSQL Server

Курсовая работа выполняется для базы данных, самостоятельно созданной студентом по описанию предметной области.

**Цель**: создаем БД по модели, разработанной в ЛР 2, на СУБД PostgreSQL Server

**Ожидаемый результат**: Физическая реализация БД

**Порядок выполнения работы**:

1. в СУБД PostgreSQL Server (см. [документация к postgresql](https://postgrespro.ru/docs/postgresql/15/index)) создаем все необходимые объекты реляционной БД: таблицы, связи между ними,
2. реализуем пользовательские ограничения предметной области: проверочные ограничения, индексы, триггеры
3. реализуем элементы программирования: функции и хранимые процедуры.
4. оформляем отчет о работе (см. Приложение 1)

Методические указания к выполнению работы

## 3. Разработка объектов БД на стороне СУБД PostgreSQL Server

Проектирование физической структуры существенно зависит от выбранной СУБД. Исходными данными для проектирования физической модели являются: *логическая модель, характеристики выбранной СУБД, требования к реактивности системы (время получения пользователем ответа на запрос).*

В этом разделе приводится описание всех объектов БД: таблицы, связи, триггеры, ограничения, хранимые процедуры и пользовательские функции. Создание этих объектов можно выполнить с помощью соответствующих SQL запросов, либо использовать интегрированные среды разработки datagrip.

Выделим комплекс мер по обеспечению целостности данных, предусмотренный СУБД: обязательные данные, ограничения для доменов полей (целостность приложения), целостность объекта, ссылочная целостность.

Для некоторых столбцов требуется наличие в каждой строке таблицы конкретного и допустимого значения, отличного от *NULL.* Для задания ограничения подобного типа стандарт *SQL* предусматривает использование спецификации *NOT NULL*.

Столбец в таблице имеет собственный домен – набор допустимых значений. Стандарт *SQL* предусматривает различные механизмы определения доменов, например: *CHECK* – позволяет задать перечень требуемых ограничений, *DEFAULT* –позволяет задать значение по умолчанию. В литературе по базам данных механизмы определения доменов часто называют целостностью приложения.

Первичный ключ для каждой строки таблицы должен быть уникальным и иметь непустое значение в каждой строке, что поддерживается с помощью фразы *PRIMARY KEY NOT NULL.* Для потенциальных ключей используется ключевое слово *UNIQUE.* Потенциальный ключ может иметь пустое значение, поэтому фраза *NOT NULL* не обязательна и вставляется по необходимости.

Для обеспечения ссылочной целостности стандарт *SQL* предусматривает механизм определения внешних ключей с помощью предложения *FOREIGN KEY*, а фраза *REFERENCES* определяет имя родительской таблицы. При использовании этого предложения система отклонит выполнение таких *DML-*операций, как *INSERT* или *UPDATE*, с помощью которых будет предпринята попытка создать в дочерней таблице значение внешнего ключа, не соответствующее одному из уже существующих значений первичного или потенциального ключа родительской таблицы. Если пользователь предпринимает попытку удалить или обновить строку в родительской таблице, на которую ссылается одна или более строк дочерней таблицы, язык *SQL* предоставляет возможности, представленные в таблице 3.

Таблица 3 – Спецификация связей

|  |  |
| --- | --- |
| Свойство связи  «Спецификация INSERT,  UPDATE, DELETE» | Действие, выполняемое СУБД при изменении записи в родительской таблице |
| RESTRICT (Нет действия) | Не разрешать удаление (обновление) кортежа в родительской таблице, если имеется хотя бы один кортеж в дочернем отношении, ссылающийся на удаляемый кортеж родительского отношения. |
| CASCADE (Каскадно) | При удалении (обновлении) кортежа в родительской таблице, будут удалены (обновлены) каскадно кортежи в дочернем отношении, ссылающиеся на удаляемый кортеж родительского отношения. |
| SET NULL (Присвоить NULL) | При удалении (обновлении) кортежа в родительской таблице, во всех кортежах в дочернем отношении, ссылающихся на удаляемый кортеж родительского отношения, будет изменено значение внешнего ключа на ***NULL***-значение. |
| SET DEFAULT (Присвоить значение по умолчанию) | При выполнении удаления (обновления) кортежа в родительской таблице, во всех кортежах дочернего отношения, ссылающихся на удаляемый кортеж, изменить значения внешних ключей на некоторое значение, принятое по умолчанию |

Спецификацию CASCADE можно смело ставить на обновление (UPDATE), но каскадное удаление связанных записей может «тихо» удалить большую часть базы. Поэтому следует решить какую спецификацию связи использовать при удалении. Обычно каскадное удаление CASCADE допустимо, если в связи участвует таблица, соответствующая слабой сущности. Во всех остальных случаях ставьте RESTRICT или SET NULL.

## Разграничение прав доступа пользователей к БД

Так как база данных является общим ресурсом, к которому обращаются многие приложения и разные категории пользователей, то важной задачей ее проектирования является разработка средств защиты данных от несанкционированного доступа.

*Защищать данные следует средствами СУБД*, т. е. на сервере БД, не рассчитывая на «разумность» программ, взаимодействующих с БД.

В качестве базовой модели доступа в СУБД PostgreSQL используется двух уровневая ролевая модель. *Субъектами* доступа (участниками) могут быть учетные записи серверного уровня и пользователи баз данных.

1. Учетные записи и *серверные роли* наделяются привилегиями серверного уровня. Каждая из серверных ролей дает возможность выполнять то или иное действие на сервере: устанавливать соединение с сервером, просматривать список имеющихся баз данных, создавать новые базы данных, изменять свойства существующих баз данных или удалять их, управлять настройками сервера (в т.ч. настройками безопасности) и т.п. Чтобы создать серверную учетную запись нужно иметь права администратора ОС или администратора PostgreSQL [1]. Для выполнения курсовой работы на сервере созданы следующие учетные записи с логинами DIRECTOR, OPERATOR, WORKER, INSPECTOR, пароли совпадают с логином, но прописными символами. Эти учетные записи используются для обращения к серверу ***групповых пользователей*** приложения.

Кроме того создана еще одна учетная запись (точка входа на сервер) с именем ГОСТЬ, без пароля. Как показано ниже, эта точка входа может использоваться для авторизации пользователя БД.

1. *Пользователи и роли баз данных* наделяются привилегиями доступа к объектам тех баз данных, в которых они зарегистрированы: привилегиями чтения/записи информации в таблицах, запуска хранимых процедур, изменения свойств объектов базы данных, управления безопасностью на уровне базы данных и т.п. Для их создания достаточно иметь права владельца базы данных. Вы, как создатель базы данных, этими правами обладаете.

4.6.1 Создание пользователей БД и предоставление им доступа

Допустим, что есть пользователь сервера *Mary.* Пока у Mary нет доступа к объектам вашей БД. Последовательность действий:

1. Создание роли БД

CREATE ROLE role\_name;

1. Предоставление прав доступа роли. Инструкции GRANT/REVOKE предоставляют/запрещают доступ к таблицам, представлениям, хранимым процедурам.

USE [TestData];

GO

-- разрешаем SELECT таблицы EmployeeData роли role\_name

GRANT SELECT ON EmployeeData TO role\_name;

GO

-- аналогично для всех объектов

1. Регистрация Mary в качестве пользователя вашей БД testData:

USE [TestData];

GO

CREATE USER [Mary] FOR LOGIN [LOGIN\_Mary];

GO

1. Присоединение Mary к роли

ALTER ROLE role\_name ADD MEMBER [Mary];

GO

1. Создадим хранимую процедуру getPersonalData и дадим Mary право на ее выполнение

USE [TestData];

GO

CREATE PROCEDURE dbo.getPersonalData @User\_ID uniqueidentifier

AS

SET NOCOUNT ON;

SELECT \*

FROM EmployeeData

WHERE User\_ID = @User\_ID

GO

1. разрешаем EXECUTE хранимой процедуры getPersonalData пользователю Mary

GRANT EXECUTE ON getPersonalData TO [Mary];

GO

В данном сценарии Mary имеет доступ только к персональным данным пользователя User\_ID в таблице EmployeeData посредством хранимой процедуры getPersonalData.

К современным информационным системам часто предъявляется требование разграничения доступа пользователей не только к объектам БД, но и к отдельным записям, с применением авторизации пользователя. Рассмотренный механизм ролей не позволяет сделать это средствами СУБД.

Рассмотрим механизм, позволяющий решить эту задачу средствами СУБД PostgreSQL Server.

1. Создадим таблицу Users, которая содержит пользователей системы, в том числе их личные User\_Login и User\_Password. Поле первичного ключа User\_ID этой таблицы лучше сделать uniqueidentifier. У каждого пользователя есть также ссылка role\_ID на таблицу Roles – назначенная ему роль в БД.
2. Таблица Roles содержит имена Serv\_Login и пароли Serv\_Password пользователей сервера, сопоставленные с их ролями в приложении. Таблица Roles должна быть недоступна пользователям за исключением администратора БД.
3. Для процедуры авторизации пользователя создается хранимая процедура Check\_Password, принимающая на вход логин и пароль пользователя и возвращающая в случае успешной проверки User\_ID, логин и пароль сервера.

create procedure Check\_PSW @login varchar(50), @pass varchar(50)

AS

BEGIN

SELECT User\_ID, Serv\_Login, Serv\_Pass

FORM Users u INNER JOIN Roles r ON u.Role\_ID=r.Role\_ID

where u.User\_Login=@login and u.User\_Password=@pass;

if @@ROWCOUNT<=0 RAISERROR('Неверен логин или пароль,16,1);

END;

Здесь 16 – степень серьезности ошибки, пользователь могу устанавливать этот уровень от 1-18, эти ошибки по существу являются информационными сообщениями и не влияют на работу сервера. Права на выполнение этой процедуры могут быть назначены любому пользователю. Рекомендуется использовать пользователя ГОСТЬ, так как можно зайти без пароля и этот пользователь не видит ни одной таблицы.

1. Права доступа к таблицам SELECT, INSERT, UPDATE, DELETE и т.п. запрещены всем пользователям БД. Работа с данными осуществляется только через хранимые процедуры. Для всех запросов к данным создаются хранимые процедуры (ХП), предоставляющие своеобразный «интерфейс» для взаимодействия с базой. Процедуры для выборки/изменения персональных данных обязательно требуют идентификатор пользователя User\_ID.

Преимущества и недостатки доступа к БД на основе ролей и хранимых процедур приведены в таблице 4.

Таблица 4 – Свойства методов разграничения прав пользователей

|  |  |
| --- | --- |
| На основе ролей | На основе хранимых процедур |
| Доступ к таблицам целиком | Доступ к отдельным записям в таблицах |
| Пользователь может писать собственные запросы к БД | Пользователь ограничен созданным «интерфейсом» взаимодействия с БД |
| Пользователь должен владеть тонкостями языка SQL и знать структуру БД | Пользователь может не знать структуру БД, знания языка SQL минимальны |
| При изменении схемы БД в приложении придется изменять запросы | При изменении БД администратор подправляет «интерфейс», приложение не затрагивается. |
| Добавление нового функционала не обременительно | Для добавления новых функций требуется обращение к администратору БД |
| Требуется тщательно планировать средства защиты целостности БД: триггеры, ограничения | Часть задач защиты можно возложить на хранимые процедуры |
| При выполнении запросов работает оптимизатор | Оптимизатор ограничен информацией на момент создания ХП |

Как видите, оба метода имеют свои преимущества и недостатки, так что решать вам. Можно использовать гибрид, например, работа с таблицами, содержащими персональные данные, осуществляется с помощью хранимых процедур, в остальных случаях применяются права роли.

Контрольные вопросы

1. Что такое независимость, безопасность, целостность, защита данных?
2. Когда нужно применять индексирование полей БД?
3. Для чего нужен триггер?
4. Какие типы триггеров PostgreSQL вы знаете?
5. Как обеспечиваются целостность и независимость данных?
6. Какими средствами можно задать права пользователей БД?
7. Каковы недостатки реляционных БД?

ПРИЛОЖЕНИЕ 1 ПРИМЕР ОТЧЕТА

4 ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

В этом разделе подробно описаны используемые хранимые процедуры, табличные и скалярные функции, триггеры, а также права, предоставленные различным группам пользователей.

4.1 Права пользователей

Защита базы данных от несанкционированного доступа осуществляется с помощью создания ролей Ученика, Преподавателя и Руководителя, которые имеют определенные права доступа к таблицам, отдельным столбцам таблиц базы, скалярным и табличным функциям и хранимым процедурам.

Рассмотрим каждую роль и предоставленные ей права подробнее.

4.1.1 Ученик

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица | Права пользователя | | | |
| INSERT | SELECT | UPDATE | DELETE |
| Puples | Запрещено | Разрешено через ХП | Запрещено | Запрещено |
| Users | Запрещено | Разрешено через ХП | Запрещено | Запрещено |
| Subjects | Запрещено | Разрешено | Запрещено | Запрещено |
| Roles | Запрещено | Запрет на отдельные столбцы: LoginServer,  PasswordServer | Запрещено | Запрещено |
| Quarters | Запрещено | Разрешено | Запрещено | Запрещено |
| Plans | Запрещено | Разрешено | Запрещено | Запрещено |
| Marks | Запрещено | Разрешено через ХП | Запрещено | Запрещено |
| Classes | Запрещено | Разрешено | Запрещено | Запрещено |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Права | Хранимые процедуры, функции | | | | | | | | | | | |
| DeleteUser | DeleteSubject | DeletePlan | DeleteClass | MarksTableReport | MarksQuarterForOnePupil | MarksQuarterTable | NameSubjectToID | NameRoleToID | NameClassToID | MarkYear | MarkQuarter |
| EXECUTE | - | - | - | - | - | + | - | + | + | + | - | + |

4.2.2 Преподаватель

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица | | Права пользователя | | | | | | | | | | | | | | |
| INSERT | | | SELECT | | | | | UPDATE | | | | DELETE | | |
| Puples | | Запрещено | | | Запрет на столбцы: Login, Password | | | | | Запрещено | | | | Запрещено | | |
| Users | | Запрещено | | | Разрешено через ХП | | | | | Запрещено | | | | Запрещено | | |
| Subjects | | Запрещено | | | Разрешено | | | | | Запрещено | | | | Запрещено | | |
| Roles | | Запрещено | | | Запрет на отдельные столбцы: LoginServer  PasswordServer | | | | | Запрещено | | | | Запрещено | | |
| Quarters | | Запрещено | | | Разрешено | | | | | Запрещено | | | | Запрещено | | |
| Plans | | Запрещено | | | Разрешено | | | | | Запрещено | | | | Запрещено | | |
| Marks | | Разрешено | | | Разрешено через ХП | | | | | Разрешено | | | | Разрешено | | |
| Classes | | Запрещено | | | Разрешено | | | | | Запрещено | | | | Запрещено | | |
| Права | Хранимые процедуры, функции | | | | | | | | | | | | | | | |
| DeleteUser | | DeleteSubject | DeletePlan | | DeleteClass | MarksTableReport | MarksQuarterForOnePupil | MarksQuarterTable | | NameSubjectToID | NameRoleToID | NameClassToID | | MarkYear | MarkQuarter |
| EXECUTE | - | | - | - | | - | - | - | + | | + | + | + | | - | + |

4.3.3 Руководство

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица | Права пользователя | | | |
| INSERT | SELECT | UPDATE | DELETE |
| Puples | Разрешено | Разрешено через ХП | Разрешено | Разрешено |
| Users | Разрешено | Разрешено через ХП | Разрешено | Разрешено |
| Subjects | Разрешено | Разрешено | Разрешено | Разрешено |
| Roles | Запрещено | Запрет на отдельные столбцы: LoginServer, PasswordServer | Запрещено | Запрещено |
| Quarters | Запрещено | Разрешено | Разрешено | Запрещено |
| Plans | Разрешено | Разрешено | Разрешено | Разрешено |
| Marks | Разрешено | Разрешено | Разрешено | Разрешено |
| Classes | Разрешено | Разрешено | Разрешено | Разрешено |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Права | Хранимые процедуры, функции | | | | | | | | | | | |
| DeleteUser | DeleteSubject | DeletePlan | DeleteClass | MarksTableReport | MarksQuarterForOnePupil | MarksQuarterTable | NameSubjectToID | NameRoleToID | NameClassToID | MarkYear | MarkQuarter |
| EXECUTE | + | + | + | + | + | - | + | + | + | + | + | + |

4.2 Хранимые процедуры

Рассмотрим хранимые процедуры: DeleteUser, DeleteClass, отвечающие за корректное удаление из базы данных пользователей, классов соответственно.

4.2.1 DeleteUser

При удалении пользователя (преподавателя) необходимо учитывать, что также из базы данных должны быть связанные записи в таблице TeacherSubject и в планах установлена в Null ссылка на запись в TeacherSubject (см. листинг 4.1). Параметром процедуры является UserID:

-- Листинг 4.1 Хранимая процедура DeleteUser

CREATE PROCEDURE [dbo].[DeleteUser] @ID uniqueidentifier

AS

BEGIN

UPDATE Plans Set TSubjectID=null

where TSubjectID in

Select TSubjectID from Users inner join TeacherSubject on

Users.UserId=TeacherSubject.User)

where UserID=@ID;

DELETE from TeacherSubject

where [TeacherSubject.UserID=@ID](mailto:TeacherSubject.UserID=@ID);

END

4.2.2 CHECK\_PSW

Проверяет логин и пароль пользователя. В случае успеха возвращает логин и пароль сервера для роли и ID пользователя (листинг 4.3). При неудаче генерируется прерывание «Неверен логин или пароль»

-- Листинг 4.3 Хранимая процедура CHECK\_PSW

CREATE PROCEDURE [dbo].[CHECK\_PSW] @user\_ID unigueidentifier,

@login varchar(20), @psw varchar(20)

AS

BEGIN

SELECT u.UserID, r.LoginServer,r.PasswordServer

FROM Roles r INNER JOIN Users u ON r.RoleID=u.RoleID

WHERE u.Login=@login and [u.Password=@psw](mailto:u.Password=@psw)

IF @@rowcout=0 then

SELECT u.PipleID, r.LoginServer,r.PasswordServer

FROM Roles r INNER JOIN Puples u ON r.RoleID=u.RoleID

WHERE u.Login=@login and [u.Password=@psw](mailto:u.Password=@psw);

IF @@rowcout=0 then RAISERROR ('Неверен логин или пароль', 16, 1)

END

4.3 Функции

Реализовано 4 табличных и 5 скалярных функций, основная задача которых – облегчить работу с запросами к БД.

4.3.1 Скалярная функция MarkQuarter

Входные параметры функции – идентификационный номер ученика, предмета и четверти. Возвращает средний балл ученика за четверть по данному предмету (см. листинг 4.3)

-- Листинг 4.3 Средний балл ученика по предмету за четверть

CREATE FUNCTION [dbo].[MarkQuarter]@PupilID uniqueidentifier,

@SubjectID int, @QuarterNumber int

RETURNS numeric(3, 2)

AS

BEGIN

DECLARE @summa float, @i int

SELECT @summa = SUM(Marks.Mark), @i = COUNT(Marks.MarkID)

FROM Marks, Quarters

WHERE Marks.SubjectID = @SubjectID

AND Marks.QuarterNumber = @QuarterNumber

AND Marks.PupilID = @PupilID

AND Quarters.QuarterNumber = @QuarterNumber

AND Marks.Date between Quarters.DateBegin AND

Quarters.DateEnd

RETURN(@summa/@i)

END

4.4 Триггеры

Реализованы триггеры: для таблицы «Classes», для таблицы «Marks», для «Plans», для таблицы «Users».

4.4.1 Триггеры таблицы Classes

Триггер ClassUnique не позволяет создавать в таблице классы с одинаковым наименованием (см. листинг 4.8).

--Листинг 4.8 Классы с одинаковым наименованием

CREATE TRIGGER [dbo].[ClassUnique] ON [dbo].[Classes]

FOR INSERT, UPDATE

AS

BEGIN

IF EXISTS(

SELECT Classes.ClassID

FROM Classes, inserted

WHERE Classes.ClassID != inserted.ClassID

AND Classes.ParallelNumber = inserted.ParallelNumber

AND Classes.ClassLetter = inserted.ClassLetter

)

BEGIN

ROLLBACK TRANSACTION

raiserror('Такой класс уже существует',16,1)

END

END

В данном отчете должны быть приведены все скрипты, необходимые для физической реализации БД: создание таблиц, связей, ограничений, триггеры и т.п. При сдаче ЛР будет оцениваться полнота обеспечения целостности данных в базе.

**Библиографический список**

1. Документация к PostgreSQL 15.4 [Электронный ресурс]. – URL: <https://postgrespro.ru/docs/postgresql/15/index> (дата обращения 14.09.2023)
2. Шёниг, Г. -. PostgreSQL 11. Мастерство разработки / Г. -. Шёниг ; перевод с английского А. А. Слинкина. — Москва : ДМК Пресс, 2020. — 352 с. – <https://e.lanbook.com/book/131714> (дата обращения: 13.09.2023)
3. Наместников, А. М. Базы данных. Практический курс : учебное пособие : в 2 частях / А. М. Наместников. — Ульяновск : УлГТУ, 2017 — Часть 1 : Объектно-реляционные базы данных на примере PostgreSQL 9.5 — 2017. — 113 с . — URL: https://e.lanbook.com/book/165100 (дата обращения: 13.09.2023).