Базы Данных. Защита лабораторной работы 3.

В контексте баз данных нормализация относится к процессу организации данных в таблицах с целью

- устранения избыточности
- обеспечения целостности
- эффективности работы с данными.

Нормализация данных помогает устранить проблемы, такие как

- дублирование информации
- аномалии вставки
- обновления и удаления данных и противоречивость данных.

Она основана на принципах, известных как нормальные формы. В PostgreSQL используются следующие нормальные формы:

- Первая нормальная форма (1NF): Таблицы должны быть "плоскими" и не содержать повторяющихся групп данных. Каждый столбец должен содержать только одно значение, а каждая ячейка должна содержать только одно значение.

- Вторая нормальная форма (2NF): Таблицы должны быть в 1NF, и каждый неключевой столбец должен зависеть от всего первичного ключа. Если в таблице есть составной первичный ключ, то неключевые столбцы должны зависеть от всех столбцов составного ключа.

- Третья нормальная форма (3NF): Таблицы должны быть в 2NF, и каждый неключевой столбец должен зависеть только от первичного ключа. Неключевые столбцы, которые зависят от других неключевых столбцов, должны быть вынесены в отдельные таблицы.

Функциональные зависимости. Виды.

Функциональная зависимость (Functional Dependency) в базах данных определяет связь между атрибутами (столбцами) в таблице. Она указывает на то, что значение одного или нескольких атрибутов определяет значение другого атрибута.

В PostgreSQL функциональная зависимость записывается в виде A -> B, где A и B представляют атрибуты (столбцы) таблицы. Это означает, что значение атрибута A определяет значение атрибута B. Виды функциональных зависимостей в PostgreSQL:

- Полная функциональная зависимость (Full Functional Dependency): говорит о том, что ни один атрибут нельзя удалить из зависимости без потери связи. Если A -> B, то ни один другой атрибут не функционально зависит от A.

- Частичная функциональная зависимость (Partial Functional Dependency): Говорит о том, что некоторые атрибуты можно удалить из зависимости без потери связи. Если A -> B, но существует другой атрибут C, который также функционально зависит от A.

- Транзитивная функциональная зависимость (Transitive Functional Dependency): говорит о том, что существует цепочка функциональных зависимостей. Если A -> B и B -> C, то можно сделать вывод, что A -> C.

Функциональные зависимости являются важным инструментом при нормализации данных и проектировании баз данных. Они помогают определить правильную структуру таблиц и связей между ними, чтобы избежать избыточности данных и обеспечить целостность информации.

Денормализация

Денормализация в базах данных – это процесс добавления избыточности в структуру данных с целью улучшения производительности или удовлетворения определенных запросов. Денормализация позволяет сохранять повторяющуюся или связанную информацию в различных таблицах для улучшения производительности запросов и упрощения сложных запросов.

Преимущества денормализации:

- Улучшение производительности запросов: Денормализация может сократить количество операций объединения (join) при выполнении запросов, что приводит к ускорению выполнения запросов и улучшению производительности системы.
- Упрощение сложных запросов: При наличии денормализованных данных сложные запросы могут быть более простыми и легкими в написании и понимании.
- Сокращение числа соединений между таблицами: Денормализация позволяет избежать лишних операций соединения таблиц, что может быть выгодно в случае большого объема данных или при выполнении запросов с большим количеством связей.
- Повышение удобства использования: Денормализация может улучшить удобство использования системы за счет предоставления данных в более удобном и понятном формате для конечных пользователей.

Однако денормализация также имеет некоторые недостатки, среди которых:

- Избыточность данных: Денормализация приводит к дублированию данных в различных таблицах, что может потребовать дополнительного пространства для хранения данных.
- Усложнение поддержки целостности данных: поскольку данные дублируются, необходимо уделять больше внимания обновлению и поддержке целостности данных для избежания противоречий или несогласованности в системе.

Денормализация следует использовать с осторожностью и осознанно, основываясь на конкретных требованиях системы и ее производительности. Она может быть полезной в случаях, когда требуются высокая производительность чтения данных или упрощение сложных запросов

Триггеры

Триггеры (triggers) в PostgreSQL – это специальные хранимые процедуры, которые выполняются автоматически при определенных событиях, происходящих в базе данных, таких как вставка (INSERT), обновление (UPDATE) или удаление (DELETE) записей в таблице.
Триггеры позволяют автоматизировать определенные действия и реагировать на изменения данных в базе.

В PostgreSQL существуют несколько видов триггеров:

- После триггер (AFTER trigger): Триггер выполняется после выполнения операции (INSERT, UPDATE, DELETE) и может использоваться для выполнения дополнительных действий после изменения данных. Вот пример создания после триггера, который записывает информацию об изменениях в отдельную таблицу:

```
sql
CREATE TRIGGER log_changes AFTER INSERT OR UPDATE OR DELETE
ON my_table
FOR EACH ROW
EXECUTE FUNCTION log_changes_function();
```

- Перед триггер (BEFORE trigger): Триггер выполняется перед выполнением операции и может использоваться для проверки и изменения данных перед их сохранением. Вот пример создания перед триггера, который проверяет условие перед выполнением операции:

```
sql
CREATE TRIGGER check_condition BEFORE INSERT OR UPDATE ON
my_table
```

```
FOR EACH ROW
WHEN (NEW.column1 IS NULL)
EXECUTE FUNCTION check_condition_function();
```

- Вместо триггер (INSTEAD OF trigger): Триггер используется для замены стандартного выполнения операции. Он позволяет вам определить собственную логику обработки операции. Этот тип триггера часто используется с представлениями (views) в PostgreSQL.

```
sql
CREATE TRIGGER instead_trigger
INSTEAD OF INSERT ON my_view
FOR EACH ROW
EXECUTE FUNCTION instead_trigger_function();
```

Приведенные примеры демонстрируют базовый синтаксис создания триггеров в PostgreSQL. Функции, указанные в триггерах, являются пользовательскими функциями, которые вы должны определить для выполнения необходимых действий. Триггеры предоставляют мощный инструмент для автоматизации и кастомизации операций в базе данных, позволяя вам реагировать на изменения данных или схемы.

Табличные и строковые триггеры

В PostgreSQL существуют два основных типа триггеров: строковые и табличные. Эти типы триггеров различаются в том, на каком уровне они работают и какую информацию они могут использовать.

- Строковые триггеры (Row-level triggers): Строковые триггеры выполняются для каждой обрабатываемой строки данных в результате операции. Они могут получать доступ и манипулировать данными, привязанными к каждой отдельной строке. Строковые триггеры предоставляют доступ к двум специальным таблицам: NEW и OLD.

NEW представляет новые значения данных, которые будут вставлены, обновлены или удалены. Вы можете получить доступ к этим значениям и выполнить некоторые действия на основе них.

OLD представляет текущие значения данных до вставки, обновления или удаления. Это позволяет вам сравнивать старые значения с новыми и принимать решения на основе этих данных.

Пример создания строки триггера:

```
sql
CREATE TRIGGER my_trigger
AFTER INSERT OR UPDATE OR DELETE ON my_table
FOR EACH ROW
EXECUTE FUNCTION my_trigger_function();
```

Внутри функции my_trigger_function() вы можете использовать NEW и OLD для доступа к данным и выполнения необходимых операций на уровне каждой строки данных.

- Табличные триггеры (Statement-level triggers): Табличные триггеры выполняются один раз для каждой операции, а не для каждой строки данных. Они предоставляют общую информацию о выполняемой операции, но не имеют непосредственного доступа к конкретным строкам данных. Табличные триггеры предоставляют доступ к специальной таблице TG_OP, которая указывает тип операции, и к переменной TG_TABLE_NAME, содержащей имя таблицы, к которой применяется триггер.

Пример создания табличного триггера:

```
sql
CREATE TRIGGER my_trigger
AFTER INSERT OR UPDATE OR DELETE ON my_table
EXECUTE FUNCTION my_trigger_function();
```

Внутри функции my_trigger_function() вы можете использовать TG_OP для определения типа операции (INSERT, UPDATE, DELETE) и TG_TABLE_NAME для получения имени таблицы, к которой применяется триггер. Вы можете выполнять действия, которые зависят от этих общих данных.