# **Практическая работа №8** 8 Условные запросы. Представления. Настройка ограничений доступа к данным

Объём учебного времени – 2ч

Методические рекомендации

1. Цель работы: Научиться создавать представления в базе данных средствами SQL.
2. Основные теоретические положения:

**– условный оператор языка SQL**

Данный оператор позволяет осуществить проверку условий и возвратить в зависимости от выполнения того или иного условия тот или иной результат.

Оператор CASE имеет 2 формы:

|  |  |
| --- | --- |
| Первая форма: | Вторая форма: |
| CASE WHEN условие\_1 THEN возвращаемое\_значение\_1 … WHEN условие\_N THEN возвращаемое\_значение\_N [ELSE возвращаемое\_значение] END | CASE проверяемое\_значение WHEN сравниваемое\_значение\_1  THEN возвращаемое\_значение\_1 … WHEN сравниваемое\_значение\_N THEN возвращаемое\_значение\_N [ELSE возвращаемое\_значение] END |

В качестве значений здесь могут выступать и выражения.

примере первую форму CASE:

SELECT

ID,Name,Salary,

CASE

WHEN Salary>=3000 THEN 'ЗП >= 3000'

WHEN Salary>=2000 THEN '2000 <= ЗП < 3000'

ELSE 'ЗП < 2000'

END SalaryTypeWithELSE,

CASE

WHEN Salary>=3000 THEN 'ЗП >= 3000'

WHEN Salary>=2000 THEN '2000 <= ЗП < 3000'

END SalaryTypeWithoutELSE

FROM Employees

WHEN-условия проверяются последовательно, сверху-вниз. При достижении первого удовлетворяющего условия дальнейшая проверка прерывается и возвращается значение, указанное после слова THEN, относящегося к данному блоку WHEN.

Если ни одно из WHEN-условий не выполняется, то возвращается значение, указанное после слова ELSE (что в данном случае означает «ИНАЧЕ ВЕРНИ …»).

Если ELSE-блок не указан и не выполняется ни одно WHEN-условие, то возвращается NULL.

И в первой, и во второй форме ELSE-блок идет в самом конце конструкции CASE, т.е. после всех WHEN-условий.

Разберем на примере вторую форму CASE:

Допустим, на новый год решили премировать всех сотрудников и попросили вычислить сумму бонусов по следующей схеме:

Сотрудникам ИТ-отдела выдать по 15% от ЗП;

Сотрудникам Бухгалтерии по 10% от ЗП;

Всем остальным по 5% от ЗП.

Используем для данной задачи запрос с выражением CASE:

SELECT

ID,Name,Salary,DepartmentID,

-- для наглядности выведем процент в виде строки

CASE DepartmentID -- проверяемое значение

WHEN 2 THEN '10%' -- 10% от ЗП выдать Бухгалтерам

WHEN 3 THEN '15%' -- 15% от ЗП выдать ИТ-шникам

ELSE '5%' -- всем остальным по 5%

END NewYearBonusPercent,

-- построим выражение с использованием CASE, чтобы увидеть сумму бонуса

Salary/100\*

CASE DepartmentID

WHEN 2 THEN 10 -- 10% от ЗП выдать Бухгалтерам

WHEN 3 THEN 15 -- 15% от ЗП выдать ИТ-шникам

ELSE 5 -- всем остальным по 5%

END BonusAmount

FROM Employees

Здесь делается последовательная проверка значения DepartmentID с WHEN-значениями. При достижении первого равенства DepartmentID с WHEN-значением, проверка прерывается и возвращается значение, указанное после слова THEN, относящегося к данному блоку WHEN.

Соответственно, значение блока ELSE возвращается в случае, если DepartmentID не совпал ни с одним WHEN-значением.

Если блок ELSE отсутствует, то в случае несовпадения DepartmentID ни с одним WHEN-значением будет возвращено NULL.

Вторую форму CASE несложно представить при помощи первой формы:

SELECT

ID,Name,Salary,DepartmentID,

CASE

WHEN DepartmentID=2 THEN '10%' -- 10% от ЗП выдать Бухгалтерам

WHEN DepartmentID=3 THEN '15%' -- 15% от ЗП выдать ИТ-шникам

ELSE '5%' -- всем остальным по 5%

END NewYearBonusPercent,

-- построим выражение с использованием CASE, чтобы увидеть сумму бонуса

Salary/100\*

CASE

WHEN DepartmentID=2 THEN 10 -- 10% от ЗП выдать Бухгалтерам

WHEN DepartmentID=3 THEN 15 -- 15% от ЗП выдать ИТ-шникам

ELSE 5 -- всем остальным по 5%

END BonusAmount

FROM Employees

Так что, вторая форма – это всего лишь упрощенная запись для тех случаев, когда нам нужно сделать сравнение на равенство, одного и того же проверяемого значения с каждым WHEN-значением/выражением.

**Представления**

Основными структурными единицами в реляционных базах данных являются таблицы. Однако язык SQL представляет ещё один способ организации данных – представления.

**Представление** - это виртуальная таблица, которая сама по себе не существует, но для пользователя выглядит таким образом, как будто она существует. Представление не поддерживаются его собственными физическими хранимыми данными. Вместо этого в каталоге таблиц хранится определение, оговаривающее, из каких столбцов и строк других таблиц оно должно быть сформировано при реализации SQL-предложения на получение данных из представления или на модификацию таких данных.

**Представления** – это запрос на выборку, которому присваивается уникальное имя и который можно сохранять или удалять из базы данных. Содержимое представлений выбирается из таблиц или существующих представлений.

Когда СУБД MySQL встречает в запросе ссылку на представление, она отыскивает его определение, сохранённое в базе данных. Затем происходит преобразование пользовательского запроса с участием представления в эквивалентный запрос с исходными таблицами. После этого выполняется запрос. Таким образом клиент может работать с представлениями так, как будто это независимые таблицы.

Клиент, обращаясь к представлению, будет видеть только столбцы результирующей таблицы, при этом не имеет значения сколько столбцов в исходной таблице и является ли запрос, лежащий в основе представления одно- или многотабличным. Кроме этого, клиенту можно запретить обращаться к исходным таблицам, но снабдить привилегиями обращения к представлениям.

**Преимущества использования представлений:**

* Безопасность – каждому пользователю можно разрешить доступ к небольшому числу представлений, содержащих только ту информацию, которую ему позволено знать;
* Простота запросов – с помощью представления можно извлечь данные из нескольких таблиц и представить их как одну таблицу, заменяя запрос ко многим таблицам в однотабличный запрос к представлению;
* Простота структуры – представления позволяют создать для каждого пользователя собственную структуру базы данных, отображая только те данные, которые ему нужны;
* Защита от изменений – в связи с оптимизацией скорости, таблицы и их структура могут быть изменены или переименовываться. Представления позволяют создавать виртуальные таблицы со старыми именами и структурой, помогая избегать модификации внешней прикладной программой;
* Обновление данных – возможность производить обновления данных в представлениях и будут обновляться данные в исходных таблицах. Однако это возможно только с простыми представлениями, сложные представления доступны только для выборки.

Синтаксис предложения CREATE VIEW имеет вид

**CREATE VIEW имя\_представления**

**[(столбец[,столбец] ...)]**

**AS подзапрос**

**[WITH CHECK OPTION];**

где подзапрос, следующий за AS и являющийся определением данного представления, не исполняется, а просто сохраняется в каталоге;

необязательная фраза "WITH CHECK OPTION" (с проверкой) указывает, что для операций INSERT и UPDATE над этим представлением должна осуществляться проверка, обеспечивающая удовлетворение WHERE фразы подзапроса;

список имён столбцов должен быть обязательно определён лишь в тех случаях, когда:

а) хотя бы один из столбцов подзапроса не имеет имени (создаётся с помощью выражения, SQL-функции или константы);

б) два или более столбцов подзапроса имеют одно и то же имя;

если же список отсутствует, то представление наследует имена столбцов из подзапроса.

Например, создадим представление Мясные\_блюда

**CREATE VIEW Мясные\_блюда**

**AS SELECT БЛ, Блюдо, В, Выход**

**FROM Блюда**

**WHERE Основа = 'Мясо';**

которое может рассматриваться пользователем как новая таблица в базе данных.

Пример создания представления с вычисляемыми полями:

**CREATE VIEW GOODS (name, rubl, doll, evro)  
 as SELECT name, price, price/29.0, price/32.1 From PRODS;**

Уничтожение ненужных представлений выполняется с помощью предложения DROP VIEW (уничтожить представление), имеющего следующий формат:

**DROP VIEW представление;**

Для изменения данных в представлениях используется команда, аналогичная изменению данных в таблицах базы данных.

**Модифицирование представлений**

Представление может теперь изменяться командами модификации DML, но модификация не будет воздействовать на само представление. Команды будут на самом деле перенаправлены к базовой таблице:

UPDATE Salesown

SET city = 'Palo Alto'

WHERE snum = 1004;

**Оператор Update.** Позволяет изменить строки таблицы. Синтаксис:

**UPDATE** <имя таблицы> **SET** <имя столбца>=<выражение>

**WHERE** <условие>

Все строки, которые удовлетворяют условию в таблице (представлении) будут изменены: содержимое указанных столбцов будут заменены на значение выражения (в нём могут участвовать старые значения тех же строк). Можно указать несколько "присваиваний", перечислив их через запятую.

**WHERE** вместе с условием можно опустить. Кроме того, можно ограничить количество изменяемых строк с помощью указания слова **LIMIT** в конце запроса, после которого указывается максимальное число изменяемых строк.

Так же, как и в **DELETE** можно указать пониженный приоритет выполнения запроса **LOW\_PRIORITY**.

**Оператор Delete**

Запрос для удаления строки из таблицы (представления): имеет синтаксис **DELETE FROM** <имя таблицы> **WHERE** <условие>

Такой запрос удалит из таблицы все строки, удовлетворяющие условию. Можно указать, сколько максимально строк удалять, указав это количество в конце, после слова **LIMIT**.

Между **DELETE** и **FROM** можно указать слово **LOW\_PRIORITY**, которое показывает, что операция удаления имеет меньший приоритет, чем операции чтения из таблицы.

Слово **WHERE** вместе с условием можно опустить, в этом случае будут удалены все строки таблицы (если не указан **LIMIT**).

**Именование столбцов**

Иногда для удобства требуется снабжать столбцы новыми именами:

* Когда некоторые столбцы являются выводимыми, и поэтому не имеющими имен.
* Когда два или более столбцов в объединении, имеют те же имена что в их базовой таблице.

Имена, которые могут стать именами полей, даются в круглых скобках ( ), после имени создаваемого представления.

**Групповые представления**

Групповые представления - это представления, которые содержат предложение GROUP BY, или которые основывается на других групповых представлениях. Групповые представления могут стать превосходным способом обрабатывать полученную информацию непрерывно. Предположим, что каждый день вы должны следить за порядком номеров заказчиков, номерами продавцов, принимающих порядки, номерами порядков, средним от порядков, и общей суммой приобретений в порядках.

Чем конструировать каждый раз сложный запрос, вы можете просто создать следующее представление:

CREATE VIEW Totalforday

AS SELECT odate, COUNT (DISTINCT cnum), COUNT

(DISTINCT snum), COUNT (onum), AVG

(amt), SUM (amt)

FROM Orders

GROUP BY odate;

Теперь вы сможете увидеть всю эту информацию с помощью простого запроса:

SELECT \*

FROM Totalforday;

**Представления и подзапросы**

Представления могут также использовать и подзапросы, включая соотнесенные подзапросы. Предположим ваша компания предусматривает премию для тех продавцов которые имеют заказчика с самым высоким порядком для любой указанной даты. Вы можете проследить эту информацию с помощью представления:

CREATE VIEW Elitesalesforce

AS SELECT b.odate, a.snum, a.sname,

FROM Salespeople a, Orders b

WHERE a.snum = b.snum

AND b.amt =

(SELECT MAX (amt)

FROM Orders c

WHERE c.odate = b.odate);

Если, с другой стороны, премия будет назначаться только продавцу, который имел самый высокий порядок за последние десять лет, вам необходимо будет проследить их в другом представлении основанном на первом:

CREATE VIEW Bonus

AS SELECT DISTINCT snum, sname

FROM Elitesalesforce a

WHERE 10 < =

(SELECT COUNT (\*)

FROM Elitesalestorce b

WHERE a.snum = b.snum);

Извлечение из этой таблицы продавца, который будет получать премию - выполняется простым вопросом:

SELECT \*

FROM Bonus;

**Что не могут делать представления**

Имеются большое количество типов представлений, которые являются доступными только для чтения. Это означает, что их можно запрашивать, но они не могут подвергаться действиям команд модификации. Имеются также некоторые виды запросов, которые не допустимы в определениях представлений. Одиночное представление должно основываться на одиночном запросе; ОБЪЕДИНЕНИЕ (UNION) и ОБЪЕДИНЕНИЕ ВСЕГО (UNION ALL) не разрешаются. УПОРЯДОЧЕНИЕ ПО (ORDER BY) никогда не используется в определении представлений. Вывод запроса формирует содержание представления, которое напоминает базовую таблицу и является - по определению - неупорядоченным.

**Удаление представлений**

Синтаксис удаления представления из базы данных подобен синтаксису удаления базовых таблиц: **DROP VIEW view\_name**

Содержание представления не является созданным и сохраняется в течении определённой команды. Базовая таблица, из которой представление выводится, не эффективна, когда представление удалено. Помните, вы должны являться владельцем представления, чтобы иметь возможность удалить его.

1. Практические указания:
2. **Изменение структуры данных и обновление записей в базе данных**
   1. Создайте запрос на добавление в таблицу **Books** поля **About** типа данных **text**, для хранения данных краткого содержания книги. Напишите запрос **Update** для добавления данных в поле.
   2. В таблицу **Users** добавьте поле **kategoria**, типа данных **varchar(15)** по умолчанию – студент.
   3. Создайте запрос для добавления поля **Zadol** в таблицу **Bilet** типа **real**, а также поле **Kol\_dney** со значением по умолчанию, равным 14 дням – для хранения количества дней, на которые выдается книга.
3. **Создание представлений для выборки данных**
   1. Создайте следующее представление:

Create view Book AS select nazv, cena from books;

Создайте запрос на выборку всех записей вашего представления.

* 1. Создайте представление **Razdel**, содержащее поля: название раздел, название книги, год издания между 2007 и 2017. Создайте запрос на выборку всех записей созданного представления.
  2. Создайте представление **ST**, которое выбирает данные из таблицы **Books**: **Nazv**, **Cena**, Nalog=Cena\*0.18; создайте запрос на выборку всех записей созданного представления.
  3. Создайте представление **Knigi\_avtor**, которое бы показывало фамилию +имя\_автора (как поле **Avtor**) и название книги, как поле Book и поле цена книги - Cena.
  4. Создайте представление **Kol\_book**, на основании представления **Knigi\_avtor**, для нахождения общего количества книг по каждому автору.
  5. Создайте представление **St\_book**, на основании представления **Knigi\_avtor**, для нахождения минимальной, максимальной, средней цены книг по каждому автору.

1. **Создание изменяемых представлений** 
   1. Создайте представление **Tek\_z** для выборки записей из таблицы **Bilet**, в которых поле **data\_vozv** меньше текущей даты.
   2. Напишите инструкцию для изменения представления **Tek\_z** – заполнения поля **Zadol** для пользователей, сумма задолженности определяется как количество дней задолженности \*1.5 от стоимости книги.
   3. Напишите инструкцию для изменения представления – заполнения поля **Zadol** для пользователей, которые не сдали книги в указанный срок, сумма задолженности определяется как количество дней задолженности \*1% от стоимости книги.
2. **Создание условных представлений**
   1. Создайте представление **About** на основании запроса, который бы в зависимости от шифра книги в таблице **Books** выдавал бы для какого возраста данная книга, например, Айболит – от 0 до 7 лет и т.д.
   2. Создайте представление, которое в зависимости от наличия задолженности выдачи книги (количество дней между датой возврата и датой выдачи не больше допустимого количества дней выдачи книги), выдавал бы текста «Есть задолженность», иначе «Нет задолженности».
3. **Создание представлений c подзапросами**
   1. Создайте представление **Min\_g**, которое содержит список (название книги, ФИО автора, год\_издания), имеющих минимальный год выпуска.
   2. Создайте представление **Max\_z**, которое содержит список пользователей, которые имели самую высокую задолженность за последние 3 года.
4. **Создание новых пользователей базы данных и задание привилегий доступа к данным**
   1. Создайте пользователя **user\_8901/02** с паролем **P@ssw0rd** с помощью команды **Create user**.
   2. Задайте привилегии доступа *(учтите, что разрешение чтение данных должно быть дано при всех других вариантах разрешений)*
      1. Пользователю **user\_8901/02** дайте разрешение:
      * на чтение представления **book**.
      * изменение данных в таблице **users**.
      * на создание материализованных представлений.
      1. Установите права доступа к вашей базе данных для пользователя **student\_8902/01**:
      * разрешение на добавление и удаление данных в таблице **Knigi\_avtor**.
      * чтение данных представления **Tek\_z** с возможностью передачи привилегии.
      1. Для всех пользователей установите привилегию – разрешение на добавление записей в таблицу **bilet**.
5. **Проверка заданных привилегий доступа**
   1. Проверка привилегий пользователя **user\_8902/01**.
      1. Создайте новое подключение для пользователя **user\_8901/02**.
      2. Проверьте привилегии данного пользователя. Для этого:

* Выполните запрос для выборки всех записей таблицы **book**: select \* from bibl.book;
* Выполните запрос для добавления поля в таблицу **book**. Какой результат запроса?
* Напишите запрос для добавления пользователю **user\_6901/02** привилегиичтение данных представления **Tek\_z.**
  + 1. Проверка привилегий пользователя **user\_8901/02**.
    2. Войдите под именем **user\_7901/02**. Самостоятельно проверьте привилегии данного пользователя. **Выполните запросы, для проверки и запишите в тетрадь.**
    3. Проверьте привилегию, данную на таблицу **bilet**, для этого добавьте по одной записи в таблицу под каждым из пользователей БД. Проверьте, что записи добавлены в таблицу.

1. **Создание материализованных представлений**
   1. Создайте представление BOOKS\_USERS для получения данных о выдачи книг пользователям библиотеки (ФИО пользователя, Книга, Дата выдачи, Дата возврата)
   2. Создайте таблицы логов - журналов материализованного представления для каждой из таблиц, входящих в определение представления BOOKS\_USERS, при создании необходимо перечислить все столбцы, упоминаемые в материализованном представлении, и предусмотреть конструкции *SEQUENCE* и *INCLUDING NEW VALUES*
   3. Добавьте по одной записи в таблицы Book, Users, и две записи в таблицу Bilet. Просмотрите результат выборки созданных материализованных представлений.
2. Содержание отчёта по практической работе
   1. Синтаксис запросов SQL для создания представлений для выборки данных из представления.
   2. Синтаксис запросов SQL для создания представлений, используя функции
   3. Синтаксис запросов SQL для изменения представлений
   4. Выполненные работы в электронном виде представляются на сетевом диске H (X)
3. Контрольные вопросы
   1. Преимущества и недостатки при использовании представлений.
   2. Синтаксис создания, удаления и изменения данных в представлениях.
   3. Можно ли вносить изменение в представления, созданные на основании нескольких таблиц?
   4. Ограничения представлений.
   5. Групповые представления.
   6. Какие представления изменяемые?
   7. Синтаксис запроса на удаление данных.
4. Список рекомендуемой литературы:
   1. Агальцов В.П. Базы данных. Распределённые и удалённые базы данных. Учебник М.: ИД «Форум»: ИНФРА-М, 2011. - 272с.
   2. Дунаев В.В. Базы данных. Язык SQL. СПб.: БХВ-Петербург-2006, 288с.
   3. Дюбуа П. MySQL.: учеб.пособие.- М.-:Вильямс,2007.- 1168с.
   4. Кузнецов М.В., MySQL 5, СПб.: БХВ-Петербург, 2010, 1024с.
   5. Петров В.Н. Избачков Ю.С., Информационные системы. СПб.: Питер, 2006, 656с.
   6. Шелдон Р. MySQL.: учеб.пособие.- М.-:Вильямс,2007.- 880с.