**ЛЕКЦИЯ 4. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ ЯЗЫКА SQL**

**Описание команд язык SQL**

Формирование запросов на языке SQL должно осуществляться в строгом соответствии с форматом (правилом формирования) команды.

При описании структуры и правил использования некоторого языка программирования употребляется определенная система обозначений. Для описания форматов команд языка SQL используем так называемую форму Бэкуса-Наура (БНФ), в которой приняты следующиеобозначения*:*

* < > (угловые скобки) – то, что в них указано, определяет пользователь;
* [ ] (квадратные скобки) – выделяют те части команды, которые могут отсутствовать;
* { } (фигурные скобки) – объединяют последовательность элементов в логическую группу;
* ... (многоточие) – указывает на допустимость повторения элемента или группы элементов один или несколько раз;
* ⏐ (вертикальная черта) – означает альтернативный выбор;
* ( ) (круглые скобки) – заключают аргументы команды;
* (пробелы) – используются для разделения элементов команды.

К категории команд ***Управление транзакциями***относятся команды, позволяющие обеспечивать целостность данных в базе данных. Для обеспечения целостности данных в SQL используются средства обработки транзакций.

***Транзакция*** – это совокупность операций манипулирования данными в системе баз данных, которая переводит базу данных из одного целостного состояния в другое.

***SQL-транзакция*** – последовательность команд SQL, обладающая свойством атомарности (неделимости) относительно восстановления состояния базы данных. Иначе говоря, это несколько последовательных команд SQL, которые рассматриваются как единое целое*.*

Вязыке SQL обработка транзакций реализована с помощью двух команд: COMMIT и ROLLBACK. Они управляют изменениями, выполненными группой команд. Команда **COMMIT** сообщает об успешном окончании транзакции. Она информирует СУБД о том, что транзакция завершена, все ее команды выполнены успешно и противоречия в БД не возникли. Команда **ROLLBACK** сообщает о неуспешном окончании транзакции. Она информирует СУБД о том, что пользователь не хочет завершать транзакцию, и СУБД должна отменить все изменения, внесенные в БД в результате выполнения транзакции. В этом случае СУБД возвращает БД в состояние, в котором она находилась до выполнения транзакции.

Команды COMMIT и ROLLBACK используются в основном в программном режиме, хотя возможно их использование и в интерактивном режиме.

К категории ***Управление доступом*** относятся команды для осуществления административных функций, присваивающих или отменяющих право (привилегию) использовать таблицы в БД определенным образом. Каждый пользователь БД имеет определенные права по отношению к объектам БД.

***Права***  – это те действия с объектом, которые может выполнять пользователь. Права могут меняться с течением времени: старые могут отменяться, новые – добавляться.

Стандартом языка SQL предусмотрены следующие права:

* SELECT – право читать таблицу;
* INSERT – право добавлять данные в таблицу;
* UPDATE – право изменять данные таблицы;
* DELETE – право удалять данные из таблицы;
* REFERENCES – право определять первичный ключ.

Пользователь, создавший таблицу, является ее владельцем. Как владелец, пользователь имеет все права на таблицу и может назначить права для работы с ней другим пользователям. Кроме владельца, права может назначать администратор БД.

 Таблица 4.1 - Команды управления доступом

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Команда** | **Назначение** | | **Краткий формат** | |
| **GRANT** | Предоставляет пользователю право доступа | | GRANT ALL [PRIVILEGES]⏐{SELECT ⏐INSERT ⏐DELETE ⏐UPDATE [<список полей>]⏐REFERENCES [<список полей>]},...ON <имя таблицы> TO PUBLIC ⏐<список пользователей> [WITH GRANT OPTION] | |
| Например**,** по команде  *GRANT SELECT, UPDATE ON Сведения TO ГИЛЬ, БРЕЛЬ*  пользователям *ГИЛЬ и БРЕЛЬ* назначаются права чтения и обновления (редактирования) таблицы Сведения | | | | |
| **REVOKE** | | Отменяет право доступа | | REVOKE ALL [PRIVILEGES]⏐{SELECT ⏐INSERT ⏐DELETE ⏐UPDATE [<список полей>]⏐REFERENCES [<список полей>]},...ON <имя таблицы>TO PUBLIC ⏐<список пользователей> [WITH GRANT OPTION] |
| Например**,** по команде  *REVOKE UPDATE ON Сведения TO ГИЛЬ*  отменяется право обновления таблицы Сведения пользователю *ГИЛЬ*. | | | | |

К категории *Встроенный SQL* относятся команды, предназначенные для реализации обращения к БД из прикладных программ, написанных на определенном языке программирования. Для пересылки данных из БД в прикладную программу используются временная таблица, называемая *SQL-курсором*.

Таблица 4.2 – Команды встроенного SQL

|  |  |
| --- | --- |
| **Команда** | **Назначение** |
| **DECLARE** | Определяет набор записей, в который будут возвращены результаты запроса |
| **OPEN** | Открывает определенный набор записей |
| **FETCH** | Извлекает строку из таблицы результатов запроса |
| **CLOSE** | Закрывает набор записей запроса |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 4.3 – Средства управления транзакциями | | | | |
| **Оператор** | **Смысл** | | | **Действие** |
| **COMMIT** | Завершить транзакцию | | | Завершить комплексную взаимосвязанную обработку информации, объединенную в транзакцию |
| **ROLLBACK** | Откатить транзакцию | | | Отменить изменения, проведенные в ходе выполнения транзакции |
| **SAVEPOINT** | Сохранить промежуточную точку выполнения транзакции | | | Сохранить промежуточное состояние БД, пометить его для того, чтобы можно было в дальнейшем к нему вернуться |
| Таблица 4.4 – Средства администрирования данных | | | | |
| Оператор | | Смысл | Действие | |
| **ALTERDATABASE** | | Изменить БД | Изменить набор основных объектов в базе данных, ограничений, касающихся всей базы данных | |
| **ALTERDBAREA** | | Изменить область хранения БД | Изменить ранее созданную область хранения | |
| **ALTERPASSWORD** | | Изменить пароль | Изменить пароль для всей базы данных | |
| **CREATEDATABASE** | | Создать БД | Создать новую базу данных, определив основные параметры для нее | |
| **CREATEDBAREA** | | Создать область хранения | Создать новую область хранения и сделать ее доступной для размещения данных | |
| **DROPDATABASE** | | Удалить БД | Удалить существующую базу данных (только в том случае, когда вы имеете право выполнить это действие) | |
| **DROP DBAREA** | | Удалить область хранения БД | Удалить существующую область хранения (если в ней на настоящий момент не располагаются активные данные) | |

**Вложенные запросы**

С помощью *SQL* можно вкладывать запросы внутрь друг друга. Обычно внутренний *запрос* генерирует *значение*, которое проверяется в предикате внешнего *запроса (в предложении* WHERE или HAVING ), определяющего, верно оно или нет. Совместно с подзапросом можно использовать *предикат* EXISTS, который возвращает истину, если *вывод* подзапроса не пуст.

В сочетании с другими возможностями оператора выбора, такими как группировка, *подзапрос* представляет собой мощное средство для достижения нужного-результата. В части FROM оператора SELECT допустимо применять синонимы к именам таблицы, если при формировании запроса нам требуется более чем один экземпляр некоторого отношения. Синонимы задаются с использованием ключевого слова AS, которое может быть вообще опущено.

Поэтому часть FROM может выглядеть следующим образом:

Листинг 1

*FROM R1 AS A, R1 AS B*

*или*

*FROM R1 A, R1 B;*

оба выражения эквивалентны и рассматриваются как применения оператора SELECT к двум экземплярам таблицы R1.

Например, покажем, как выглядят на *SQL* некоторые запросы к *БД* "Сессия":

Список тех, кто сдал все положенные экзамены.

Листинг 2

*SELECT ФИО*

*FROM R1 as a*

*WHERE Оценка > 2*

*GROUP BY ФИО*

*HAVING COUNT(\*) = (SELECT COUNT(\*)*

*FROM R2,R3*

*WHERE R2.Группа=R3.Группа AND ФИО=a.ФИО)*

Здесь во встроенном запросе определяется общее число экзаменов, которые должен сдавать каждый студент, обучающийся в группе, в которой учится данный студент, и это число сравнивается с числом экзаменов, которые сдал данный студент.

Список тех, кто должен был сдавать экзамен по БД, но пока еще не сдавал.

Листинг 3

*SELECT ФИО*

*FROM R2 a, R3*

*WHERE R2.Группа=R3.Группа AND Дисциплина = "БД" AND NOT EXISTS (SELECT ФИО*

*FROM R1*

*WHERE ФИО=a.ФИО AND Дисциплина = "БД")*

Предикат EXISTS ( *SubQuery*) истинен, когда подзапрос *SubQuery* не пуст, то есть содержит хотя бы один кортеж, в противном случае предикат EXISTS ложен.

Предикат NOT EXISTS обратно — истинен только тогда, когда подзапрос *SubQuery* пуст.

Обратите внимание, каким образом NOT EXISTS с вложенным запросом позволяет обойтись без *операции разности* отношений. Например, формулировка запроса со словом "все" может быть выполнена как бы с двойным отрицанием.

*Ключевое слово* ANY, поставленное в любом предикате сравнения, означает, что *предикат* будет истинен, если хотя бы для одного значения из подзапроса *предикат сравнения* истинен. *Ключевое слово* ALL требует, чтобы *предикат сравнения* был бы истинен при сравнении со всеми строками подзапроса.

Например, найдем студентов, которые сдали все экзамены на оценку не ниже чем "хорошо".

Листинг 4

*R1 = (ФИО, Дисциплина, Оценка);*

*R2 = (ФИО, Группа);*

*R3 = (Группы, Дисциплина )*

*R4 = (ФИО, Дисциплина, Номерлабраб, Оценка);*

*Select R1.Фио From R1 Where 4 < = All (Select R1.Оценка*

*From R1 as R11*

*Where R1.Фио = R11.Фио)*

Рассмотрим еще один пример:

Выбрать студентов, у которых оценка *по* экзамену не меньше, чем хотя бы одна оценка *по* сданным им лабораторным работам *по* данной дисциплины:

Листинг 5

*Select R1.ФИО*

*From R1*

*Where R1.Оценка >= ANY (Select R4.Оценка*

*From R4*

*Where R1.Дисциплина = R4. Дисциплина AND R1.ФИО = R4.ФИО)*

**Объединения UNION**

Объединением двух множеств называется множество всех элементов, принадлежащих какому-либо одному или обоим исходным множествам. Результатом **объединения** будет множество, состоящее из всех строк, входящих в какое-либо одно или в оба первоначальных отношения. Однако, если этот результат сам по себе должен быть другим отношением, а не просто разнородной смесью строк, то два исходных отношения должны быть совместимыми по объединению, т. е. строки в обоих отношениях должны быть одной и той же формы. Что касается **SQL**, то две таблицы совместимы по объединению (и к ним может быть применен оператор объединения **UNION**) тогда и только тогда, когда:

* они имеют одинаковое число полей (например, m);
* для всех i (i = 1, 2, …, m) i-е поле первой таблицы и i-е поле второй таблицы имеют в точности одинаковый тип данных.

В качестве примера выберем коды товаров, которые имеют стоимость, превышающую 1000, либо покупаются покупателем с кодом 23 (либо то и другое):

Листинг 6

*SELECT stock FROM goods*

*WHERE unitprice > 1000*

*UNION*

*SELECT stock FROM ordsale*

*WHERE customerno = 23;*

Из результата выборки, использующей оператор UNION, всегда исключаются избыточные дубликаты. Поэтому, хотя в рассматриваемом примере товар выбирается обеими из двух составляющих предложений SELECT, в результирующей таблице каждый товар присутствует только один раз.

Оператором UNION можно соединить любое число конструкций SELECT.  
Оператор ORDER BY в запросе с использованием оператора UNION может входить только в последнее предложение SELECT. При указании критерия упорядочивания используйте номера полей результирующей таблицы.

Оператор **UNION ALL** позволяет, в отличие от UNION, разрешить выборку повторяющихся значений.

Листинг 7

*SELECT name*

*FROM employees\_Rus*

*UNION ALL*

*SELECT name*

*FROM employees\_Usa*;

**Объединения таблиц**

Стандарт SQL делит **объединение таблиц на три вида**: внутреннее объединение таблиц (**INNER JOIN**), внешнее объединение таблиц (**LEFT OUTER JOIN, RIGHT JOIN, FULL JOIN**) и перекрестное объединение таблиц (**CROSS JOIN**). Принцип работы любого объединения схож, но результаты будут всегда или почти всегда отличаться.

Принцип работы запросов на объединения таблиц в SQL и реляционных базах данных заключается в том, что внутри одного SQL запроса SELECT выполняется два или более подзапроса (в зависимости от того, сколько мы хотим объединить таблиц), подзапросы разделяются между собой ключевым словом JOIN. У этого JOIN есть ограничение **ON**, которое называют предикатом объединения. **Предикат объединения** – это всегда какое-то условие, с помощью которого СУБД определяет, какие строки из двух таблиц ей нужно объединять. С тем, как объединять строки, SQL разбирается специальным модификаторам: **INNER, LEFT OUTER или просто LEFT и CROSS**.

Стандарт SQL выделяет несколько модификаторов JOIN:

* INNER JOIN – **внутреннее объединение** таблиц.
* LEFT JOIN или LEFT OUTER JOIN – **левое внешнее объединение** таблиц.
* RIGHT JOIN или RIGHT OUTER JOIN – **правое внешнее объединение** таблиц.
* FULL JOIN – **полное объединение** таблиц.
* CROSS JOIN – **перекрестное объединение** таблиц.

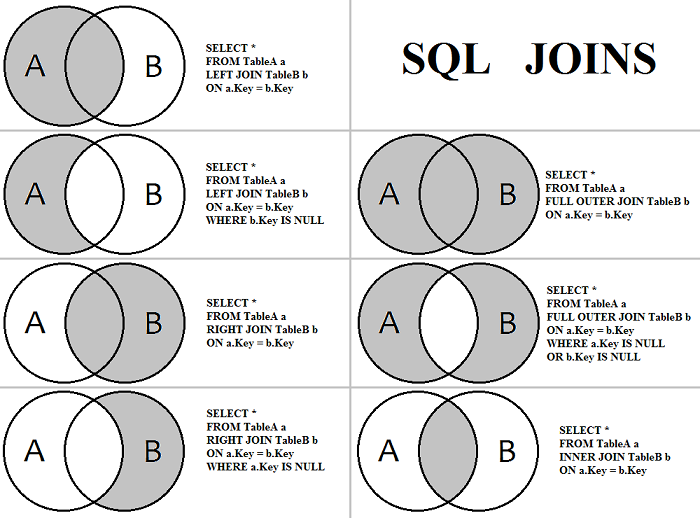


Рисунок 4.1 – Виды объединений таблиц

Рассмотрим примеры выполнения внешних объединений.

Если студент не сдавал данного экзамена, то вместо оценки у него будет стоять неопределенное *значение*.

Для этого выполним последовательно естественное внутреннее *объединение* таблиц R2 и R3 *по* атрибуту Группа, а полученное *отношение* соединим левым внешним естественным объединением с таблицей R1, используя столбцы ФИО и Дисциплина. При этом в стандарте разрешено использовать скобочную структуру, так как результат объединения может быть одним из аргументов в части FROM оператора SELECT.

Листинг 8

*SELECT R1.ФИО, R1.Дисциплина, R1.Оценка*

*FROM (R2 NATURAL INNER JOIN R3 ) LEFT JOIN R1 USING ( ФИО, Дисциплина)*

Результат:

|  |  |  |
| --- | --- | --- |
| **ФИО** | **Дисциплина** | **Оценка** |
| Петров Ф. И. | Базы данных | 5 |
| Сидоров К. А. | Базы данных | 4 |
| Миронов А. В. | Базы данных | 2 |
| Степанова К. Е. | Базы данных | 2 |
| Крылова Т. С | Базы данных | 5 |
| Владимиров В. А. | Базы данных | 5 |
| Петров Ф. И. | Теория информации | Null |
| Сидоров К. А. | Теория информации | 4 |
| Миронов А. В. | Теория информации | Null |
| Степанова К. Е. | Теория информации | 2 |
| Крылова Т. С | Теория информации | 5 |
| Владимиров В. А. | Теория информации | Null |
| Петров Ф. И. | Английский язык | 5 |
| Сидоров К. А. | Английский язык | Null |
| Миронов А. В. | Английский язык | Null |
| Степанова К. Е. | Английский язык | Null |
| Крылова Т. С. | Английский язык | Null |
| Владимиров В. А. | Английский язык | 4 |
| Трофимов П. А. | Сети и телекоммуникации | 4 |
| Иванова Е. А. | Сети и телекоммуникации | 5 |
| Уткина Н. В. | Сети и телекоммуникации | 5 |
| Трофимов П. А. | Английский язык | 5 |
| Иванова Е. А. | Английский язык | 3 |
| Уткина Н. В. | Английский язык | Null |

Рассмотрим еще один пример, для этого возьмем *БД* "Библиотека".

*BOOKS(ISBN, TITLE, AUTOR, COAUTOR, YEARIZD, PAGES)*

*READER(NUM\_READER, NAME\_READER, ADRESS,*

*HOOM\_PHONE, WORK\_PHONE, BIRTH\_DAY)*

*EXEMPLARE(INV, ISBN, YES\_NO, NUM\_READER, DATE\_IN, DATE\_OUT)*

Здесь *таблица* BOOKS описывает все книги, присутствующие в библиотеке, она имеет следующие атрибуты:

ISBN — уникальный шифр книги;

TITLE — название книги;

AUTOR — фамилия автора;

COAUTOR — фамилия соавтора;

YEARIZD — год издания;

PAGES — число страниц.

*Таблица* READER хранит сведения обо всех читателях библиотеки, и она содержит следующие атрибуты:

NUM\_READER — уникальный номер читательского билета;

NAME\_READER — фамилию и инициалы читателя;

ADRESS — адрес читателя;

HOOM\_PHONE — номер домашнего телефона;

WORK\_PHONE — номер рабочего телефона;

BIRTH\_DAY — дату рождения читателя.

*Таблица* *EXEMPLARE* содержит сведения о текущем состоянии всех экземпляров всех книг. Она включает в себя следующие столбцы:

INV — уникальный инвентарный номер экземпляра книги;

ISBN — шифр книги, который определяет, какая это книга, и ссылается на сведения из первой таблицы;

YES\_NO — признак наличия или отсутствия в библиотеке данного экземпляра в текущий момент;

NUM\_READER — номер читательского билета, если книга выдана читателю, и Null в противном случае;

DATE\_IN — если книга у читателя, то это дата, когда она выдана читателю;

DATE\_OUT — дата, когда читатель должен вернуть книгу в библиотеку.

Определим перечень книг у каждого читателя; если у читателя нет книг, то номер экземпляра книги равен NULL. Для выполнения этого поиска нам надо использовать левое внешнее *объединение*, то есть мы берем все строки из таблицы READER и соединяем со строками из таблицы *EXEMPLARE*, если во второй таблице нет строки с соответствующим номером читательского билета, то в строке результирующего отношения *атрибут* *EXEMPLARE*.INV будет иметь неопределенное *значение* NULL:

Листинг 9

*SELECT READER.NAME\_READER, EXEMPLARE.INV*

*FROM READER LEFT JOIN EXEMPLARE*

*ON READER.NUM\_READER=EXEMPLARE.NUM\_READER*

При этом для книг, ни один экземпляр которых не находится на руках у читателей, значения номера читательского билета и дат взятия и возврата книги будут неопределенными.