База Данных

Лекция 10

Запросы

Прошлая лекция

- Псевдоним AS
- Конкатенация ||
- DISTINCT
- IS NULL и IS NOT NULL
- Условия выборки:
  - <,<=,>, >=
  - BETWEEN & NOT BETWEEN
- LIKE с постановочными символами % и _
- CAST и ::

Агрегатные функции

Агрегатные функции воздействуют на значения столбца, чтобы получить единое результирующее значение с помощью различных математических операций.

SQL предоставляет следующие агрегатные функции, которые отображаются в инструкции SELECT:

- Min() Возвращает минимальное значение для столбца
- Max() Возвращает максимальное значение для столбца
- Avg() Возвращает среднее значение для столбца
- Sum() Возвращает суммарное значение столбца таблицы базы данных
- Count() Возвращает количество записей таблицы базы данных

Все эти функции оперируют со значениями в единственном столбце таблицы и возвращают единственное значение. Функции COUNT, MIN и MAX применимы как к числовым, так и к нечисловым полям, тогда как функции SUM и AVG могут использоваться только в случае числовых полей.

Агрегатные функции

- Пример: Выберите минимальный, максимальный и средний балл в таблице Студенты.

```
SELECT min(gpa), max(gpa), avg(gpa)
FROM Students;
```

Агрегатные функции

- Выборка count(*) или count(выражение) возвращает количество кортежей, удовлетворяющих условию выбора.
- По умолчанию функция COUNT() возвращает только ненулевые значения. Однако если необходимо подсчитать все записи, даже записи с ошибками или нулевыми значениями, рекомендуется использовать символ звездочки * или поле первичного ключа. Символ звездочки * обозначает «вернуть все записи». Поэтому, используя ее с агрегатной функцией COUNT(), мы получим количество всех записей в таблице

- Пример: Возвращает количество всех студентов.
- SELECT count(*)

  FROM Students;

Агрегатные функции

- Пример: Получаем количество студентов с group_id = 1. Столбец должен называться NumOfStud.

```
SELECT count(*) AS NumOfStud FROM Students
WHERE group_id=1;
```

Пример с count()

Students table

stud_id	fname	group_id
1	student1	2
2	student2	2
3	student3	

Count (*)

count
3

Count (group_id)

count
2

Группирование результатов (конструкция GROUP BY)

- Конструкция GROUP BY в SQL используется для группировки строк, которые имеют одинаковые значения в указанных столбцах. Это позволяет выполнять агрегатные функции на каждой группе отдельно, такие как суммирование, подсчет, нахождение максимального и минимального значения и т.д.
- Конструкция GROUP BY используется для указания этих группировок.
- При использовании в операторе SELECT конструкции GROUP BY каждый элемент списка в списке выборки SELECT должен иметь единственное значение для всей группы.

Группирование результатов (конструкция GROUP BY)

Запрос, в котором присутствует конструкция GROUP BY, называется *группирующим запросом*, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная итоговая строка. Столбцы, перечисленные в конструкции GROUP BY, называются *группируемыми столбцами*.

Все имена столбцов, приведенные в списке выборки SELECT, должны присутствовать и в конструкции GROUP BY, за исключением случаев, когда имя столбца используется только в агрегирующей функции.

Использование конструкции GROUP BY

- Пример: Выберите group_id, в которых учатся студенты, и количество студентов, которые учатся в этих группах.

```
SELECT group_id, count(*) FROM Students
GROUP BY group_id;
```

- Примечание: Группируемый атрибут (group_id) должен быть частью выборки Select.

Группирование результатов (конструкция GROUP BY)

Students table

stud_id	fname	group_id
1	student1	1
2	student2	1
3	student3	2

```
SELECT count(*)
FROM Students;
```

count
3

Группирование результатов (конструкция GROUP BY). Пример

Students table

stud_id	fname	group_id
1	student1	1
2	student2	1
3	student3	2

```
SELECT group_id, count(*)
FROM Students
GROUP BY group_id;
```

group_id	count
1	2
2	1

Ограничения на выполнение группирования (конструкция HAVING)

Условие HAVING позволяет фильтровать результат группировки, сделанной с помощью команды GROUP BY.

Условие HAVING фильтрует агрегированные данные.

Если вы попытаетесь использовать HAVING без условия GROUP BY, то получите сообщение об ошибке.

Имена столбцов, применяемые в конструкции HAVING, обязательно присутствовали в списке элементов GROUP BY или применялись в агрегирующих функциях.

HAVING: пример

- Вывести group_id и количество студентов в каждой группе.

```
SELECT group_id, count(*)  FROM Students
GROUP BY group_id;
```

- Теперь выведите group_id, в которых более чем 20 студентов.

```
SELECT group_id, count(*)
FROM Students
GROUP BY group_id
HAVING count(*) >20;
```

WHERE vs HAVING

Конструкции WHERE и HAVING в SQL используются для фильтрации данных, но они имеют различные области применения:

WHERE:

1. Применяется для фильтрации строк перед тем, как они будут сгруппированы с помощью GROUP BY.
2. Может использоваться без GROUP BY.
3. Не может использоваться с агрегатными функциями.

Пример:

```
SELECT * FROM Студенты
WHERE Год_обучения = 2;
```

Помните, что агрегирующие функции не могут использоваться в конструкции WHERE.

WHERE vs HAVING

HAVING:

1. Применяется для фильтрации групп после того, как они были сформированы с помощью GROUP BY.
2. Используется только в сочетании с GROUP BY.
3. Может использоваться с агрегатными функциями.

Пример:

```
SELECT Группа, COUNT(*) FROM Студенты
 GROUP BY Группа
 HAVING COUNT(*) > 20;
```

Важно: Если вам нужно отфильтровать строки перед группировкой, используйте WHERE. Если необходимо отфильтровать группы по результатам агрегатных функций, используйте HAVING.

HAVING: Пример с join

```
SELECT g.name as group_name,  count(*) as
number_of_students
FROM Students s, Groups g
WHERE s.group_id=g.group_id
GROUP BY g.name
HAVING count(*) > 20;
```

group_name	number_of_students
CSSE-131	21
CSSE-132	24
…	…

Сортировка результатов (конструкция ORDER BY)

- Конструкция ORDER BY позволяет упорядочить выбранные записи в порядке возрастания в алфавитном порядке (ASC) или убывания (DESC) значений любого столбца или комбинации столбцов.
- Если направление сортировки не указано, по умолчанию используется порядок возрастания (ASC).

- **Пример**: Вывести имя и фамилию каждого учащегося в порядке возрастания их фамилий
- SELECT fname, lname

  FROM Students

  ORDER BY lname ASC;

Сортировка результатов: пример

- Порядок результатов в запросе может быть смешанным: один столбец может быть отсортирован в порядке возрастания, в то время как другой столбец может быть отсортирован в порядке убывания.
- Для предыдущего запроса отсортируйте результаты в порядке возрастания фамилий и в порядке убывания имен:

```
SELECT fname, lname  FROM Students
ORDER BY lname ASC, fname DESC;
```

- **ПРИМЕЧАНИЕ** Если условие ORDER BY отсутствует, каждый запрос будет возвращать данные в том порядке, в котором они были изначально сохранены в таблице.

Пример сортировки с join

```
SELECT g.name as group_name, count(*)  as
number_of_students
FROM Students s, Groups g  WHERE
s.group_id=g.group_id
GROUP BY g.name  HAVING
count(*) > 20  ORDER BY g.name
ASC;
```

group_name	number_of_students
CSSE-131	21
CSSE-132	24
…	…

Получение ограниченного числа записей с помощью условия LIMIT и OFFSET

- Конструкции LIMIT и OFFSET в SQL используются для управления количеством строк, возвращаемых запросом. Эти конструкции особенно полезны при работе с большими объемами данных и при реализации постраничного вывода результатов.
- Конструкция **LIMIT** ограничивает количество строк, возвращаемых запросом.

  Пример: Вывод первых 10 студентов

  ```
  SELECT * FROM students
  LIMIT 10;
  ```
- Конструкция **OFFSET** используется для пропуска определенного количества строк перед возвращением результатов.

  Пример: Вывод студентов с 11 по 20

  ```
  SELECT * FROM students LIMIT 10 OFFSET 10;
  ```

  Этот запрос пропускает первые 10 строк и возвращает следующие 10 строк из таблицы students, тем самым выводя студентов с 11 по 20.

Резюме: Полный синтаксис конструкции SELECT

```
SELECT attribute(s)
FROM table(s)

[WHERE selection condition(s)]
[GROUP BY condition(s)]
[HAVING condition(s)]
[ORDER BY condition(s)]
[LIMIT number]
[OFFSET number];
```

Резюме: Полный синтаксис конструкции SELECT

SELECT: указывает, какие столбцы должны быть включены в результаты запроса.
attribute(s): список столбцов, которые необходимо включить в результаты запроса. Можно использовать * для выбора всех столбцов.
FROM table(s): указывает, из какой таблицы следует выбирать данные.
WHERE selection condition(s): опциональное условие, которое ограничивает строки, включаемые в результаты запроса.
GROUP BY condition(s) опциональный компонент, используемый для группировки результатов запроса по одному или нескольким столбцам.
HAVING condition(s): опциональное условие, применяемое к сгруппированным результатам для дальнейшего ограничения результатов.
ORDER BY condition(s): [ASC|DESC]: опциональный компонент, используемый для сортировки результатов запроса по одному или нескольким столбцам. ASC указывает на сортировку по возрастанию, DESC – по убыванию.
LIMIT number: опциональный компонент, используемый для ограничения количества строк, возвращаемых запросом.
OFFSET number: опциональный компонент, используемый в сочетании с LIMIT для указания смещения начала выборки строк.

Функции даты и времени

Функции даты и времени позволяют управлять данными, хранящимися в различных форматах даты и времени.

Основные функции даты и времени:

CURRENT_DATE - Возвращает текущую дату.

CURRENT_TIME - Возвращает текущее время суток.

CURRENT_TIMESTAMP - Возвращает текущую дату и время.

Пример:

```
SELECT CURRENT_DATE;
```

Функции даты и времени: EXTRACT и date_part

В PostgreSQL функции EXTRACT и date_part используются для извлечения определенных компонентов из даты или времени, таких как год, месяц, день и т.д. Эти функции очень похожи и часто могут использоваться взаимозаменяемо. Однако существуют некоторые различия:

Синтаксис:

1. **EXTRACT** использует ключевое слово и скобки: EXTRACT(field FROM source)

2. **date_part** использует строковый литерал и запятую: date_part (‘*field*’, *source*)

Стандарт SQL:

1. EXTRACT является частью стандарта SQL и поддерживается многими системами управления базами данных.
2. date_part является специфичной для PostgreSQL и не является частью стандарта SQL.

EXTRACT и date_part: примеры

```
-- Использование EXTRACT
SELECT Имя, Фамилия, EXTRACT(YEAR FROM
Дата_рождения) AS Год_рождения
FROM Студенты;

-- Использование date_part
SELECT Имя, Фамилия, date_part('year', Дата_рождения)
AS Год_рождения
 FROM Студенты;
```

Функции даты и времени: EXTRACT

EXTRACT(field FROM source)

Функция EXTRACT получает из значений даты/времени поля, такие как год или час.
Здесь ***источник (source)*** *должен быть выражением значения типа даты.*
поле (field) – это идентификатор или строка, которая выбирает, какое поле извлечь из исходного значения.

Пример:

```
SELECT EXTRACT(year FROM bdate)
FROM Students;
```

Функции даты и времени: date_part

date_part (‘*field*’, *source*)

Здесь **источник (source)** должен быть выражением значения типа даты.
поле (field) – это идентификатор или строка, которая выбирает, какое поле извлекать из исходного значения.

Пример:

```
SELECT date_part('year', bdate) FROM Students;
```

Функции даты и времени

В PostgreSQL, поля, которые можно использовать с функциями EXTRACT и date_part, включают в себя различные компоненты даты и времени. Вот список некоторых из наиболее часто используемых полей:

1. **YEAR**: Год.
2. **MONTH**: Месяц (1-12).
3. **DAY**: День месяца (1-31).
4. **HOUR**: Час (0-23).
5. **MINUTE**: Минута (0-59).
6. **SECOND**: Секунда (0-59 с дробной частью).
7. **DOW (Day of Week)**: День недели (0-6, где 0 - воскресенье, 1 - понедельник и т.д.).
8. **DOY (Day of Year)**: День года (1-366).
9. **QUARTER**: Квартал года (1-4).
10. **WEEK**: Номер недели в году (в зависимости от настроек, может начинаться с воскресенья или понедельника).
11. **CENTURY**: Век.
12. **MILLENNIUM**: Тысячелетие.

Книги

- Коннолли, Томас М. Системы баз данных: практический подход к проектированию, внедрению и управлению / Томас М. Коннолли, Кэролин Э. Бегг.- Соединенные Штаты Америки: Pearson Education
- Гарсия-Молина, Х. Система баз данных: Полная книга / Эктор Гарсия-Молина.- Соединенные Штаты Америки: Пирсон Прентис Холл
- Шарма, Н. Основы баз данных: Книга для сообщества от сообщества / Нирадж Шарма, Ливиу Перниу.- Канада

- www.postgresql.org/docs/manuals/