нетология

ГРУППИРОВКИ, ВЫБОРКИ ИЗ НЕСКОЛЬКИХ ТАБЛИЦ

ОЛЕГ БУЛЫГИН

ОЛЕГ БУЛЫГИН

IT-аудитор в ПАО “Сбербанк”

obulygin91@ya.ru

fb.me/obulygin91

Чему вы научились на прошлой лекции?

Обратите внимание на порядок операторов в запросе, он важен!

```
SELECT [DISTINCT | ALL] table_columns
FROM table [JOIN]
[WHERE condition]
[GROUP BY table_columns]
[HAVING condition]
[ORDER BY table_columns [ASC | DESC]]
[LIMIT number]
```

План занятия

1. Агрегирующие функции
2. Вложенные запросы
3. Оператор GROUP BY для группировки данных
4. Оператор HAVING для фильтрации сгруппированных данных
5. Alias
6. Объединение таблиц при помощи JOIN
7. Индексы

Агрегирующие функции

Агрегирующие функции

Агрегирующие функции выполняют вычисления над значениями в столбце.

Существует 5 таких функций:

- SUM() – вычисляет сумму значений;
- MIN() – вычисляет минимальное значение;
- MAX() – вычисляет максимальное значение;
- AVG() – вычисляет среднее значение;
- COUNT() — вычисляет количество строк в столбце.

Вложенные запросы

Вложенные запросы

Вложенный запрос – запрос, который используется внутри другого запроса. Каждый вложенный запрос так же может содержать один или несколько вложенных запросов. Количество вложенных запросов неограничено.

Вложенные подзапросы обрабатываются «снизу вверх». Сначала обрабатывается вложенный запрос самого нижнего уровня. Далее значения, полученные по результату его выполнения, передаются и используются при исполнении подзапроса более высокого уровня и т.д.

Например:

```
SELECT title, length  FROM film  -- данный запрос будет выполнен вторым
    WHERE length >= (
        SELECT AVG(length) FROM film); -- сначала будет выполнен этот запрос
```

Оператор GROUP BY

GROUP BY

GROUP BY – оператор, с помощью которого можно задать агрегацию по нужным столбцам. Применяется в связке с агрегирующими функциями.

```
SELECT rating, COUNT(title) FROM film
    GROUP BY rating;
```

```
SELECT rating, AVG(length) FROM film
   WHERE release_year = 2006
   GROUP BY rating;
```

> Группировочные столбцы должны обязательно присутствовать в SELECT.

Оператор HAVING

HAVING

HAVING – оператор, позволяющий отфильтровать данные **после** группировки (является аналогом WHERE).

```
SELECT last_name FROM actor
    GROUP BY last_name
    HAVING COUNT(last_name) = 1;
```

> Не путайте: WHERE фильтрует данные до группировки, а HAVING после.

Alias

ALIAS

Alias (псевдонимы) используются для *временного* переименования таблиц и столбцов. Это необязательный функционал, но может быть очень полезным и удобным, когда имена длинные и сложные или они повторяются много раз в рамках запроса (пригодится при объединении!). По этой причине псевдонимы делают максимально краткими.

```
SELECT column_name [AS] column_alies
    FROM table_name [AS] table_alies
```

Оператор AS является необязательным.

> При использовании псевдонимов важно знать и помнить о техническом порядке исполнения запросов. WHERE и HAVING исполняются **до** SELECT, поэтому в них псевдонимы использовать не получится.

Объединение таблиц при помощи JOIN

Объединение таблиц

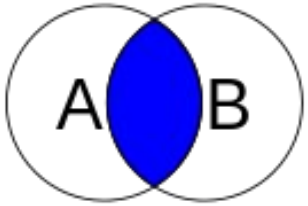
JOIN – оператор, предназначенный для объединения таблиц по определенному столбцу или связке столбцов (как правило, по первичному ключу).

JOIN записывается после FROM и до WHERE. Через оператор ON необходимо явно указывать столбцы, по которым происходит объединение. В общем виде структура запросы с JOIN выглядит так:

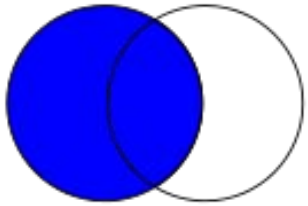
```
SELECT tables_columns FROM table_1
JOIN table_2 ON table_1.column = table_2.column;
```

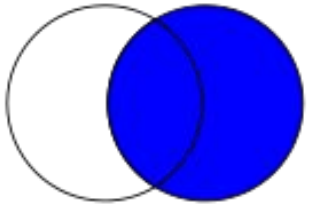
Типы JOIN

```
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key
```




```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
```

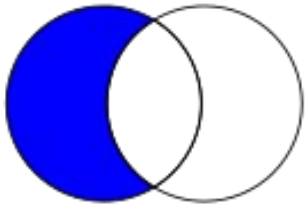


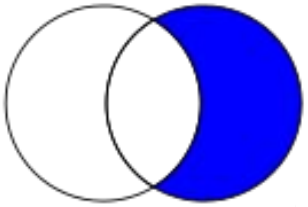


```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
```

SQL JOINS

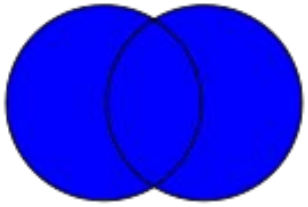
```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL
```

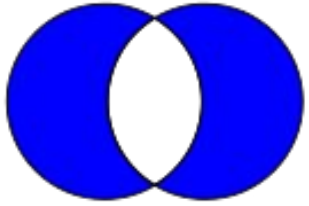




```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
```

```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
```





```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

Основные виды JOIN

[INNER] JOIN – в выборку попадут только те данные, по которым выполняются условия соединения;

LEFT [OUTER] JOIN – в выборку попадут все данные из таблицы **A** и только те данные из таблицы **B**, по которым выполнится условие соединения. Недостающие данные вместо строк таблицы B будут представлены **NULL**.

RIGHT [OUTER] JOIN – в выборку попадут все данные из таблицы **B** и только те данные из таблицы **A**, по которым выполнится условие соединения. Недостающие данные вместо строк таблицы **A** будут представлены **NULL**.

FULL [OUTER] JOIN – в выборку попадут все строки таблицы **A** и таблицы **B**. Если для строк таблицы **A** и таблицы **B** выполняются условия соединения, то они объединяются в одну строку. Если данных в какой-то таблице не имеется, то на соответствующие места вставляются **NULL**.

CROSS JOIN – объединение каждой строки таблицы **A** с каждой строкой таблицы **B.**

[INNER] JOIN

[INNER] JOIN – в выборку попадут только те данные, по которым выполняются условия соединения.

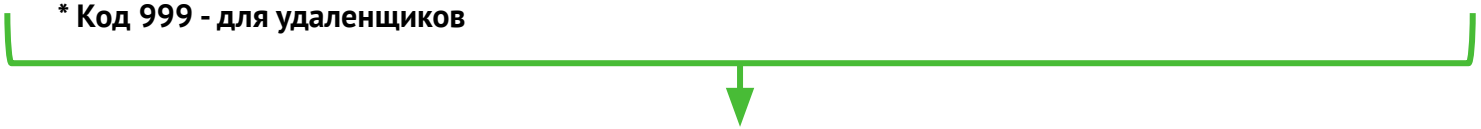
Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

*** Код 999 - для удаленщиков**

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22



Получаем только тех сотрудников, которые сидят в кабинетах и только те кабинеты, в которых сидят сотрудники.

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126

LEFT JOIN

LEFT JOIN – в выборку попадут все данные из левой таблицы и только те данные из правой, по которым выполняется условие соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

*** Код 999 - для удаленщиков**

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем всех сотрудников и только те кабинеты, в которых кто-то сидит.

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
4	Mary	NULL	NULL

RIGHT JOIN

RIGHT JOIN – в выборку попадут все данные из правой таблицы и только те данные из левой, по которым выполняется условие соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

*** Код 999 - для удаленщиков**

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем все кабинеты и только тех сотрудников, которые сидят в них (не удаленщиков).

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
NULL	NULL	4	Кабинет 22

LEFT JOIN с фильтрацией по полю

LEFT JOIN с фильтрацией по полю – в выборку попадут только те данные из левой таблицы, по которым не выполняется условия соединения.

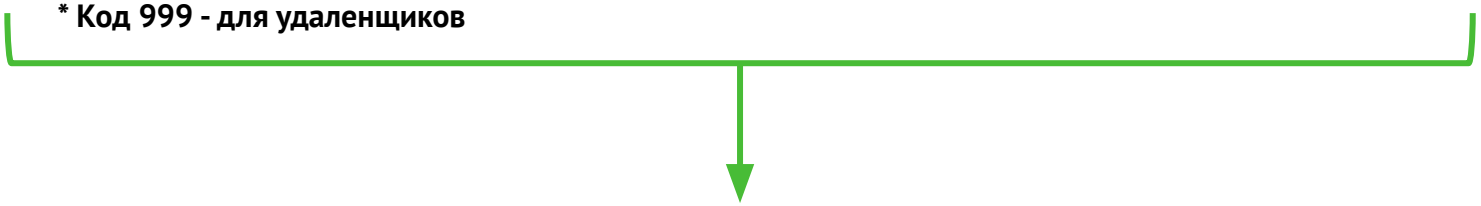
Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

*** Код 999 - для удаленщиков**



Получаем только удаленщиков.

employees.id	name	offices.id	office_name
4	Mary	NULL	NULL

RIGHT JOIN с фильтрацией по полю

RIGHT JOIN с фильтрацией по полю – в выборку попадут только те данные из правой таблицы, по которым не выполняется условия соединения.

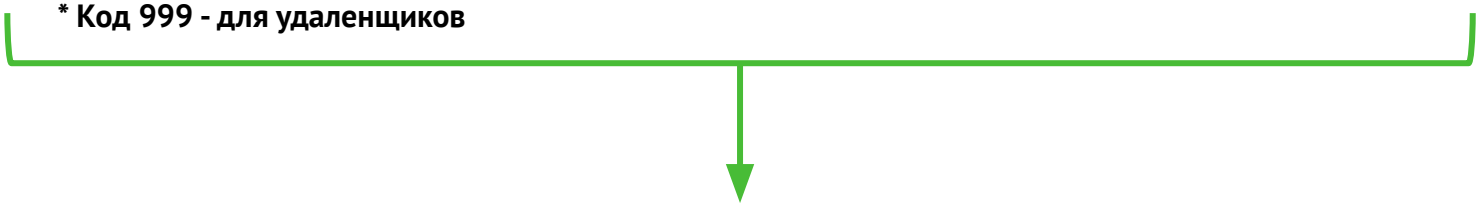
Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

*** Код 999 - для удаленщиков**

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22



Получаем только пустые кабинеты.

employees.id	name	offices.id	office_name
NULL	NULL	4	Кабинет 22

FULL OUTER JOIN

FULL OUTER JOIN – выборку попадут все строки исходных таблиц. Если по данным не выполняется условие соединения, то на соответствующие места вставляются **NULL**.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

*** Код 999 - для удаленщиков**

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем абсолютно все кабинеты и абсолютно всех сотрудников.

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
4	Mary	NULL	NULL
NULL	NULL	4	Кабинет 22

{Практика}

Для закрепления материала напишем ряд запросов к БД, которые будут содержать в себе одновременно:

- JOIN с одной/несколькими таблицами;
- WHERE;
- группировки;
- сортировки.

Индексы

Индексы

Индексы – это специальные объекты баз данных, которые нужны для ускорения получения данных из них.

Для добавления индекса используется команда CREATE INDEX.

```
CREATE INDEX index_name
ON table_name (colums_names);
```

Удалить индекс можно так:

```
DROP INDEX index_name;
```

Когда использовать индексы

Индексы ускоряют запросы на получение данных (SELECT), но замедляют на изменение (UPDATE/INSERT). Выбор столбца/столбцов для создания индексов зависит от того, какие столбец/столбцы чаще используются для фильтрации в WHERE.

Индексы использовать не рекомендуется, когда:

- таблицы в БД небольшие;
- данные в таблицах/столбцах часто меняются;
- в столбце много NULL-значений.

Дополнительный материал об индексах:
https://habr.com/ru/post/247373/

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задаём в чате Slack!
- Задачи можно сдавать по частям.
- Зачёт по домашней работе проставляется после того, как приняты **все задачи**.

нетология

Задавайте вопросы и напишите отзыв о лекции!

ОЛЕГ БУЛЫГИН

obulygin91@ya.ru

fb.me/obulygin91