Лекция №8 20.10.23

SQL (продолжение)

Сортировка строк результатов

В разное время для одного и того же запроса порядок строк результата может оказаться различным. Если требуется получить строки в определённом порядке, используют `ORDER BY`, в котором можно задать имя столбцов, по значениям которого должен быть отсортирован результат.

```
ORDER BY <имя столбцов> [ASC, DESC]
```

По умолчанию сортировка производится по возрастанию. **ASC** - возрастание, **DESC** - убывание. Если в разделе `ORDER BY` указать несколько столбцов, то сортировка будет производиться по значениям первого из них для одинаковых значений 1 и значений 2 и т.д.

Неизвестные значения типа `NULL` при сортировке обрабатываются по-разному в разных СУБД.

Группировка и объединение таблиц

Наиболее часто применяются следующие простые групповые функции:

1. `AVG` - для вычисления среднего
2. `MIN`, `MAX` - для определения минимального или максимального значения
3. `SUM` - для вычисления суммы значений
4. `COUNT` - вычисляет количество строк, в которых в указанном столбце присутствует любое известное значение
5. `COUNT(*)` - подсчёт общего количества строк

Пример:

Таблица employs:

emp_id	last_name	salary	commission	dept_id	date
101	K	17000		90	21-09-89
102	D	17000		90	13-01-83
145	R	14000	0.4	80	01-10-96
146	P	13500	0.3	80	05-01-97

Запрос:

```
SELECT COUNT(DISTINCT salary), MIN(last_name), MAX(date)
FROM employs
WHERE emp_id IN (101, 102, 145, 146);
```

Результат:

COUNT(...)	MIN(...)	MAX(...)
3	D	05-01-97

Если среди значений, переданных групповой функции, есть неизвестный типа `NULL`, то практически все функции не будут их замечать.

В отличие от других функций, `COUNT(*)` выбивается из общего рада. Она просто считает общее количество строк, не обращая внимания на конкретные столбцы и значения в них. В отличие от `COUNT(*)`, `COUNT` с параметром в виде столбца посчитает только те строки, где значение не 0.

```
SELECT COUNT(commission) as cnt -- то `cnt` = 2
SELECT COUNT(*) as cnt -- то `cnt` = 4
```

Бывают ситуации, когда не достаточно получить только одно значение. Для этого применяют `GROUP BY`.

Запрос:

```
SELECT dept_id, SUM(salary)
FROM employs
GROUP BY dept_id
```

Результат:

dept_id	SUM(...)
90	34000
80	27500

Все столбцы из раздела `SELECT`, которые не участвуют в вычислении групповых функций, должны быть указаны в разделе `GROUP BY`. С помощью условий в разделе `HAVING` можно произвести фильтрацию.

```
HAVING SUM(salary) > 30000
```

В разделе `GROUP BY` может быть указано несколько столбцов. В этой ситуации группировка будет производиться слева направо.

Выборка данных из нескольких таблиц

Перед тем как осуществлять выборку, все источники данных должны быть соединены при помощи оператора `JOIN`.

```
FROM <имя левой таблицы> <вид соединения> JOIN <имя правой таблицы> ON <условие соединения>
```

Виды `JOIN`:

1. `INNER JOIN` - внутреннее соединение (попадают исключительно те строки, которые соответствуют условию соединения) (пересечение двух кругов)
2. `OUTER JOIN` - внешнее соединение (может содержать дополнительные строки из одной или двух таблиц)
    - `LEFT JOIN` - левое внешнее соединение (попадают все строки из таблицы, которая записана слева) (только левый круг)
    - `RIGHT JOIN` - правое внешнее соединение (попадают все строки из таблицы, которая записана справа) (только правый круг)
    - `FULL JOIN` - все строки из обеих таблиц (оба круга)
3. `CROSS JOIN` - декартово произведение (каждая строка одной таблицы объединяется с каждой строкой другой таблицы, поэтому условие не задаётся)

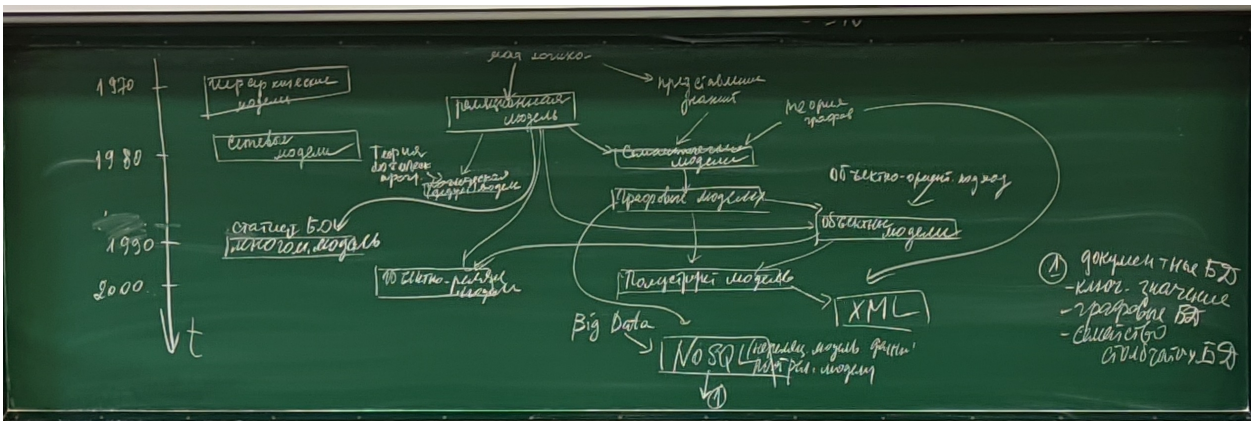
Если мы хотим получить левостороннее исключающее объединение, то мы должны прописать дополнительное условие (левый круг без правого круга).

```
SELECT …
FROM table_A LEFT JOIN table_B ON table_A id = table_B.id
WHERE table_B.id is NULL
```

Помимо соединения `JOIN`, существует еще оператор, но он не стандартизирован и считается устаревшим. Он предполагает, что все данные будут перечислены в `FROM` через запятую, а условие соединения выносится в раздел `WHERE`.

Модели данных

Каждая модель баз данных основана на определённых критических принципах и служит основанием для разработки других связанных моделей.



Теоретика графовых моделей

К теоретике графовых моделей данных относят иерархическую модель данных и сетевую модель данных.

Иерархическая модель является наиболее простой из всех датологических моделей. Появление иерархической модели связано с тем, что в реальном мире очень многие связи соответствуют иерархии, когда один объект выступает главой, а с ним может быть связано несколько объектов.

Основные компоненты в иерархической модели:

- База данных
- Сегмент
- Поле

Поле данных - минимальное неделимое единица данных, доступных пользователям с помощью СУБД.

Сегмент - запись. Делится на тип сегмента, тип записи и экземпляр сегмента или записи.

Тип сегмента - поименованная совокупность типов элементов данных.

Экземпляр сегмента - образуется из конкретных значений полей.

Каждый тип сегмента образует набор однородных записей. Для возможности различия отдельных записей каждый тип сегмента должен иметь ключ или набор ключевых атрибутов.

Ключ - набор элементов данных, однозначно индицирующих экземпляр сегмента.

Иерархическая модель данных

В иерархической модели сегменты объединяются в ориентированный древовидный граф. Направленные рёбра отражают иерархические связи между сегментами.

```
Сегмент типа A
    Сегмент типа B
```

```
        Сегмент типа C
```