Introduction to Databases and SQL

ЛЕКЦИЯ 6

Темы занятия

Подзапросы

Группировка при выборке

Агрегатные функции

Фильтрация групп

Представления (Views)

Исходные таблицы

Persons

	Column Name	Data Type	Allow Nulls
🔑	ID	int	☐
	FirstName	varchar(50)	☐
	LastName	varchar(50)	☑
	Department	char(2)	☐
			☐

Departments

	Column Name	Data Type	Allow Nulls
🔑	ID	char(2)	☐
	Name	varchar(50)	☐
			☐

	ID	FirstName	LastName	Department
1	1	Anna	Klimenok	QA
2	2	Olga	Chekan	QA
3	3	Olga	Naumik	QA
4	4	Alexey	NULL	TC
5	5	Oleg	NULL	TC
6	6	Sergey	Pavlov	DV

	ID	Name
1	QA	Quality assurance
2	TC	Training Center

Подзапросы

Подзапрос (subquery) – инструкция выборки, которая содержится внутри другой инструкции выборки.

Обычно подзапросы применяют в `WHERE`-части.

*) подзапрос может содержаться в инструкциях вставки, обновления или удаления данных – но об этом позже.

Что может возвращать подзапрос?

A. *Произвольная выборка* (несколько строк и колонок). Операция `EXISTS` (`NOT EXISTS`) может проверить, что выборка не пустая (или пустая).
B. *Набор однотипных значений*. Удобно использовать в сочетании с операций `IN` (`NOT IN`).
C. *Одно значение* (или ничего). Это значение можно использовать в операциях сравнения.

Подзапросы в WHERE – пример 1

Выведем сотрудников с зарплатой < 1400. Подзапрос возвращает из `Salaries` набор чисел (`PersonID`). Мы используем этот набор как основу для операции `IN`:

```
SELECT FirstName, LastName FROM Persons
WHERE ID IN (SELECT PersonID FROM Salaries
                WHERE Salary < 1400)
```

*) эту задачу можно решить, используя `JOIN`.

Подзапросы в WHERE – пример 2

```
SELECT FirstName, LastName FROM Persons
WHERE Department=(SELECT ID FROM Departments
                  WHERE Name='Quality assurance')
```

Важно: если подзапрос вернёт более одного значения, то будет ошибка при выполнении запроса. Если вернёт ровно одно значение – всё ОК. Если ничего не вернёт (то же самое, что `NULL`) – тоже всё ОК, ибо любая операция сравнения с `NULL` возможна, но всегда равна `FALSE`.

Связанный подзапрос

Связанный подзапрос (correlated subquery) использует в работе информацию внешнего запроса.

Например, проверяет своё условие `WHERE`, используя колонки таблицы из внешнего запроса.

Связанный подзапрос – пример

Выведем людей из таблицы `Persons`, для которых нет соответствующей информации в таблице `Departments`:

```
SELECT CONCAT(P.FirstName, ' ', P.LastName)
FROM Persons AS P
WHERE NOT EXISTS (SELECT * FROM Departments
                  WHERE ID = P.Department)
```

	(No column name)
1	Sergey Pavlov

Подзапрос после FROM

Подзапрос можно использовать не только в `WHERE`-части, но и после `FROM`. В этом случае получается *выборка из результатов подзапроса*. Синтаксис T-SQL требует в этом случае дать подзапросу псевдоним.

```
SELECT T.ID FROM (SELECT * FROM Persons) AS T
```

Группировка при выборке

Группировка при выборке – разбивка строк набора данных на **непересекающиеся группы**.

В одну группу входят все строки с одинаковым значением **указанного поля** (или **комбинации полей**; или **выражения**, построенного с использованием полей).

После группировки выборка работает не со множеством строк, а со множеством групп!

Группировка – простейший пример

Разобьём строки таблицы `Persons` на группы по значениям поля `Department` и выведем эти значения:

```
SELECT Department FROM Persons
GROUP BY Department
```

	Department
1	DV
2	QA
3	TC

Группировка – пример 2

Разобьём строки `Persons` на группы по значениям в полях `FirstName` и `Department`:

```
-- комбинация (Olga, QA) встречается два раза
SELECT FirstName, Department FROM Persons
GROUP BY FirstName, Department
```

	FirstName	Department
1	Alexey	TC
2	Anna	QA
3	Oleg	TC
4	Olga	QA
5	Sergey	DV

Группировка – пример 3

Создадим группы в зависимости от длины `FirstName`:

```
-- встроенная функция LEN() - длина строки
-- (без учёта концевых пробелов)
SELECT LEN(FirstName) FROM Persons
GROUP BY LEN(FirstName)
```

	(No column name)
1	4
2	6

Группировка – выбираемые колонки

Случай 1 – *группировка по колонкам, но без выражений*. В этом случае после `SELECT` можно упоминать колонки группировки и записывать выражения с ними.

Случай 2 – *группировка с выражением*. После `SELECT` применяем это же выражение, но не отдельные колонки, входящие в него.

Группировка – выбираемые колонки

```
SELECT A, B FROM T GROUP BY A, B        -- OK

SELECT A FROM T GROUP BY A, B           -- OK

SELECT A + B FROM T GROUP BY A, B       -- OK

SELECT A + B FROM T GROUP BY A + B      -- OK

SELECT A, C FROM T GROUP BY A, B        -- NOT OK!

SELECT A, B FROM T GROUP BY A + B       -- NOT OK!
```

Группировка – выбираемые колонки

А как быть с колонками, по которым не проводилась группировка? Их тоже можно упоминать, но только в составе агрегатных функций.

Агрегатные функции

Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение.

Основные агрегатные функции в T-SQL:

```
AVG()      -- среднее значение
MIN()      -- минимум
MAX()      -- максимум
SUM()      -- сумма
COUNT()    -- количество элементов
```

Агрегатные функции – синтаксис

Функция `AVG()` возвращает *среднее арифметическое* набора значений (`NULL` пропускаются). Если набор пуст, функция `AVG()` возвращает `NULL`.

```
AVG(col)            -- вычисления по колонке col
AVG(ALL col)        -- полный аналог AVG(col)
AVG(DISTINCT col) -- выбрасываем дубликаты из col
```

У функций `MIN()`, `MAX()`, `SUM()` синтаксис аналогичный.

Агрегатные функции – синтаксис

Функция `COUNT()` используется в двух форматах.

Первый формат аналогичен другим агрегатным функциям и требует указания колонки. Считается число значений в колонке, которые не `NULL`.

Второй формат – это `COUNT(*)`. Считает общее число строк в таблице или группе, включая повторяющиеся значения и `NULL`.

Использование агрегатных функций

Основной вариант – при группировке. Аргументом функции может быть любая колонка набора. Агрегатная функция вычисляется на значениях из каждой группы:

```
SELECT Department, MAX(ID) AS MaxID FROM Persons
GROUP BY Department
```

	Department	MaxID
1	DV	6
2	QA	3
3	TC	5

Использование агрегатных функций

Агрегатные функции можно применять и без группировки, в обычном `SELECT`. Но тогда запрещено выбирать обычные колонки (можно константы):

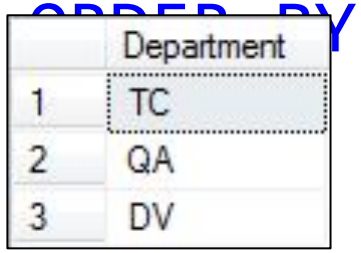
```
SELECT MAX(ID) AS MaxRow,
       COUNT(*) AS AllCount,
       1 AS Const FROM Persons
```

	MaxRow	AllCount	Const
1	6	6	1

Группировка и сортировка

Предложение `GROUP BY` можно использовать совместно с предложением `ORDER BY`. Сортировать можно по колонкам и выражениями, допустимым после `SELECT`:

```
SELECT Department FROM Persons
GROUP BY Department
ORDER BY Department DESC
```



	Department
1	TC
2	QA
3	DV

Группировка и фильтрация

`GROUP BY` можно применить вместе с `WHERE`. Строки вначале фильтруются, затем происходит группировка:

```
SELECT Department FROM Persons
WHERE FirstName <> 'Sergey'
GROUP BY Department
```

	Department
1	QA
2	TC

Фильтрация групп

Как быть, если надо отфильтровать не строки исходного набора, а уже сформированные группы?

В этом случае следует использовать предложение `HAVING` с предикатом, записанное сразу после `GROUP BY`.

Фильтрация групп – пример

Сгруппируем строки по полю `Department` и оставим те группы, где `Department <> 'DV'`:

```
SELECT Department FROM Persons
GROUP BY Department
HAVING Department <> 'DV'
```

	Department
1	QA
2	TC

Фильтрация групп

Сила `HAVING` в том, что его предикат может содержать агрегатные функции, вычисленные по строкам группы.

Пример: выведем группы с количеством строк больше 1:

```
SELECT Department FROM Persons
GROUP BY Department
HAVING COUNT(*) > 1
```

Фильтрация, группировка, сортировка

В инструкции `SELECT` отдельные предложения должны быть записаны в указанном порядке:

```
SELECT . . .
FROM . . .
WHERE . . .
GROUP BY . . .
HAVING . . .
ORDER BY . . .
```

Представления

Представление (view) – это виртуальная таблица, которая представляет собой именованный запрос.

Можно сказать, что представление – это синоним к запросу.

Изменение данных в реальной таблице немедленно отражается в содержимом всех представлений, которые построены с использованием этой таблицы.

Преимущества представлений

1. Дополнительный уровень абстракции – представления скрывают сложность запросов от «внешнего мира».
2. Дополнительная защита данных – пользователю можно дать права на работу с представлением, но не с таблицами, входящими в него.
3. СУБД может оптимизировать запрос в представлении, так как этот запрос зафиксирован в момент создания представления.

Создание представления

Для создания представления используется инструкция `CREATE VIEW`. Указывается имя представления и запрос:

```
CREATE VIEW view_Persons
AS
SELECT P.ID, P.FirstName, P.LastName, D.Name
FROM Persons AS P
     JOIN Departments AS D
     ON P.Department = D.ID
```

Использование представления

После создания представление можно использовать в выборках как обычную таблицу:

```
SELECT * FROM view_Persons
```

	ID	FirstName	LastName	Name
1	1	Anna	Klimenok	Quality assurance
2	2	Olga	Chekan	Quality assurance
3	3	Olga	Naumik	Quality assurance
4	4	Alexey	NULL	Training Center
5	5	Oleg	NULL	Training Center

Изменение и удаление представлений

Для изменения представления используется инструкция `ALTER VIEW`. Так как суть представления – некий запрос, то изменение подразумевает указание нового запроса.

Удаление определённого представления выполняется инструкцией `DROP VIEW`.