**Оператор EXISTS**

Оператор **EXISTS** рассматривает результат подзапроса не как числовое значение (в отличие от операций сравнения) и не как множество конкретных значений (в отличие от конструкции **IN**), а как логическое значение. Это значение считается истинным, если подзапрос возвращает хотя бы одну запись, и ложным, если возвращаемая выборка пуста. Таким образом, оператор **EXISTS** возвращает либо значение **TRUE**, либо значение **FALSE**.

Например, показать всех тех авторов, книги которых присутствуют сейчас в БД:

|  |
| --- |
| **SELECT name**  **FROM Authors a**  **WHERE EXISTS (**  **SELECT \***  **FROM Books b**  **WHERE a.id = b.id\_author)** |

Количество полей в выборке, возвращаемой подзапросом, не играет роли.

**Операторы ANY и SOME**

Говорим, что это два совершенно эквивалентных оператора. Почему же их тогда два, если они обозначают одно и то же? Это не связано с самим языком SQL, а связано с английским языком, в котором **ANY** используется в отрицательных и вопросительных, а **SOME** в утвердительных предложениях. Таким образом, две формы одного и того же оператора введены для удобства англоязычных разработчиков тем, чтобы сделать SQL как можно более высокоуровневым.

Операторы **ANY** и **SOME** позволяют использовать все логические операции сравнения с подзапросами, возвращающими несколько значений. Т.е. они позволяют проверить, например, что некоторое значение больше любого значения, возвращаемого подзапросом.

Например, показать всех студентов, которым полагается стипендия:

|  |
| --- |
| **SELECT name**  **FROM Students**  **WHERE Id = ANY (**  **SELECT id\_student**  **FROM Grants)** |

Точно также в этой ситуации можно использовать оператор **SOME**:

|  |
| --- |
| **SELECT name**  **FROM Students**  **WHERE Id = SOME (**  **SELECT id\_student**  **FROM Grants)** |

Это даст абсолютно тот же эффект.

**Оператор ALL**

Поведение оператора **ALL** похоже на **ANY** и **SOME**, но имеет существенное отличие: здесь требуется соответствие не хотя бы одной записи, возвращаемой подзапросом, а всех записей.

Например, показать всех авторов, у которых есть книги, толщиной больше чем любая книга издательства «BHV»:

|  |
| --- |
| **SELECT a.name**  **FROM** **Authors a, Books b**  **WHERE**  **a.id = b.id\_author AND**  **b.pages > ALL (**  **SELECT b.pages**  **FROM Books b, Press p**  **WHERE b.id\_press = p.id AND p.name = 'BHV')** |

**Оператор UNION**

Бывают ситуации, когда необходимо объединить в единую выборку результаты работы двух разных запросов. Говорим, что язык SQL дает такую возможность при условии, что типы полей, возвращаемых обоими запросами, будут одинаковы. Для этого используется оператор **UNION**.

Общий вид оператора **UNION**:

|  |
| --- |
| **SELECT ...**  **UNION**  **SELECT ...** |

Например, если у нас есть таблицы «Студенты» и «Преподаватели», то получить общий список фамилий и тех, и других можно следующим образом:

|  |
| --- |
| **SELECT name**  **FROM Students**  **UNION**  **SELECT name**  **FROM Teachers** |

Существуют определённые ограничения, накладываемые на оператор **UNION**. Во-первых, оба запроса должны возвращать одинаковое количество полей. А во-вторых, типы этих полей должны быть одинаковыми. Эти ограничения вполне логичны, поскольку иначе не понятно как СУБД, обрабатывающая выборки, должна будет соединять в единую выборку совершенно разные поля.

Объединяемые запросы могут быть сколь угодно сложны, главное – чтобы выполнялись обозначенные выше ограничения. Например, можно воспользоваться агрегатными функциями:

|  |
| --- |
| **SELECT** **'Студенты' AS 'Группа', COUNT(\*) AS 'Количество'**  **FROM Students**  **UNION**  **SELECT 'Преподаватели' AS 'Группа', COUNT(\*) AS 'Количество'**  **FROM Teachers** |

Может возникнуть ситуация, когда и в первой объединяемой выборке, и во второй объединяемой выборке могут попадаться одинаковые записи. Оператор **UNION** выкидывает повторяющиеся записи, действуя по принципу конструкции **DISTINCT**.

Если же программисту необходимо, чтобы результирующая выборка содержала все повторяющиеся записи, то вместо оператора **UNION** следует использовать оператор **UNION ALL**.

|  |
| --- |
| **SELECT name**  **FROM Students**  **UNION ALL**  **SELECT name**  **FROM Teachers** |

Сортировать выборки в каждом из операторов выборки нельзя, но можно сделать с результирующей выборкой в целом:

|  |
| --- |
| **SELECT name**  **FROM Students**  **UNION ALL**  **SELECT name**  **FROM Teachers**  **ORDER BY 1** |

**Объединения**

На самом деле, объединения мы уже использовали при работе с несколькими таблицами:

|  |
| --- |
| **SELECT a.name, b.name**  **FROM Authors a, Books b**  **WHERE a.id = b.id\_author** |

Т.е. используемые нами многотабличные запросы как раз и осуществляют объединение нескольких таблиц.

Такие объединения называются **внутренними объединениями**. Стандарт языка SQL содержит альтернативный способ записи внутренних объединений: вместо конструкции **WHERE** используется конструкция **INNER JOIN**.

В этом случае приведенный выше запрос будет выглядеть так:

|  |
| --- |
| **SELECT a.name, b.name**  **FROM Authors a INNER JOIN Books b**  **ON a.id = b.id\_author** |

Результат, разумеется, будет тем же.

Однако кроме внутренних объединений, стандарт языка SQL позволяет реализовывать и т.н. **внешние объединения**.

При внутреннем объединении обе объединяемые таблицы симметричны, но при внешнем объединении нет – одна из таблиц будет ведущей, а другая ведомой. Суть в том, что даже если в ведомой таблице не будет ни одной записи, связанной с некоторой записью из ведущей таблицы, то все равно эта запись попадает в результирующую выборку.

Например, необходимо показать всех авторов и имеющиеся в наличии книги, ими написанные. При этом допустим, что в настоящий момент книг автора Иванова И.И. в наличии нет, но в таблице «Авторы» он все же присутствует. Если использовать приведенный выше пример с внутренним объединением, то Иванов И.И. в результирующей выборке присутствовать не будет. Если же воспользоваться внешним объединением, то хотя бы одной записью он представлен в результирующей выборке будет.

Реализуется внешнее объединение при помощи конструкции **OUTER JOIN**. Ведущая таблица внешнего объединения при этом задается либо ключевым словом **LEFT**, либо ключевым словом **RIGHT**.

Тогда пример будет выглядеть так:

|  |
| --- |
| **SELECT a.name, b.name**  **FROM Authors a OUTER JOIN Books b**  **ON a.id = b.id\_author** |

А результирующая выборка будет иметь такой вид:

|  |  |
| --- | --- |
| **Автор** | **Название** |
| Пушкин А.С. | Капитанская дочка |
| Пушкин А.С. | Евгений Онегин |
| Толстой Л.Н. | Война и мир |
| ... | ... |
| Иванов И.И. | (NULL) |
| ... | ... |