

# SQL PRACTICE

## КРАТКОЕ СОДЕРЖАНИЕ И БАЗОВЫЙ СИНТАКСИС:

### СОЗДАНИЕ ТАБЛИЦ. ОБНОВЛЕНИЕ, УДАЛЕНИЕ, КОРРЕКТИРОВКА ДАННЫХ В ТАБЛИЦАХ:

Команда	Синтаксис	Описание	Пример
<b>CREATE TABLE</b>	<b>CREATE TABLE</b> table_name ( col1 datatype optional keyword, col2 datatype optional keyword, col3 datatype optional keyword,..., coln datatype optional keyword)	<b>CREATE TABLE</b> инструкция заключается в создании таблицы. Каждый столбец в таблице задается своим именем, типом данных и необязательным ключевым словом, которое может быть <b>PRIMARY KEY</b> , <b>NOT NULL</b> и т.д.,	<b>CREATE TABLE</b> employee ( employee_id char(2) <b>PRIMARY KEY</b> , first_name varchar(30) <b>NOT NULL</b> , mobile int);
<b>INSERT</b>	<b>INSERT INTO</b> table_name (column1,column2,column3...) <b>VALUES</b> (value1,value2,value3...);	<b>INSERT INTO</b> используется для вставки новых строк в таблицу. Строковые данные, символьные данные, даты при заполнении столбцов берутся в кавычки: 'F' Количественные типы данных (int, float и т.п) в кавычки не берутся. Так же можно копировать содержимое той или иной таблицу, если она имеет идентичные характеристики, но это позднее.	<b>INSERT INTO</b> placeofinterest (name,type,city,country,airport) <b>VALUES</b> ('Niagara Waterfalls','Nature','Toronto','Canada','Pearson');
<b>UPDATE</b>	<b>UPDATE</b> table_name <b>SET</b> [[column1]=[VALUES]] <b>WHERE</b> [condition];	<b>UPDATE</b> используется для обновления строк в таблице.	<b>UPDATE</b> placeofinterest <b>SET</b> name = 'Niagara Falls' <b>WHERE</b> name = "Niagara Waterfalls";
<b>DELETE</b>	<b>DELETE FROM</b> table_name <b>WHERE</b> [condition];	<b>DELETE</b> используется для удаления из таблицы строк, указанных в операторе <b>WHERE</b> . Без указания <b>WHERE</b> оператор удалит все строки из таблицы	<b>DELETE FROM</b> placeofinterest <b>WHERE</b> city <b>IN</b> ('Rome','Vienna');
<b>ALTER TABLE - ADD COLUMN</b>	<b>ALTER TABLE</b> table_name <b>ADD COLUMN</b> column_name_1 datatype, <b>ADD COLUMN</b> ..., <b>ADD COLUMN</b> column_name_n datatype;	<b>ALTER TABLE - ADD COLUMN</b> используется для добавления столбцов в таблицу.	<b>ALTER TABLE</b> employee <b>ADD COLUMN</b> income bigint;

<b>ALTER TABLE - ALTER COLUMN</b>	<b>ALTER TABLE</b> table_name <b>ALTER COLUMN</b> column_name_1 <b>SET DATA TYPE</b> datatype; (modification_type)	<b>ALTER TABLE ALTER COLUMN</b> используется для <b>изменения типа данных</b> столбцов.	<b>ALTER TABLE</b> employee <b>ALTER COLUMN</b> mobile <b>SET DATA TYPE</b> CHAR(20);
<b>ALTER TABLE - DROP COLUMN</b>	<b>ALTER TABLE</b> table_name <b>DROP COLUMN</b> column_name_1 ;	<b>ALTER TABLE DROP COLUMN</b> используется для удаления столбцов из таблицы.	<b>ALTER TABLE</b> employee <b>DROP COLUMN</b> mobile;
<b>ALTER TABLE - RENAME COLUMN</b>	<b>ALTER TABLE</b> table_name <b>RENAME COLUMN</b> current_column_name <b>TO</b> new_column_name;	<b>ALTER TABLE RENAME COLUMN</b> используется для переименования столбцов в таблице.	<b>ALTER TABLE</b> employee <b>RENAME COLUMN</b> first_name <b>TO</b> name;
<b>TRUNCATE TABLE</b>	<b>TRUNCATE TABLE</b> table_name <b>IMMEDIATE;</b>	<b>TRUNCATE TABLE</b> используется для удаления всех строк в таблице. <b>IMMEDIATE</b> указывает, что оператор должен быть обработан немедленно и что его нельзя отменить.	<b>TRUNCATE TABLE</b> employee <b>IMMEDIATE;</b>
<b>DROP TABLE</b>	<b>DROP TABLE</b> table_name ;	<b>DROP TABLE</b> - удаляет таблицу полностью	<b>DROP TABLE</b> employee ;

## ИЗВЛЕЧЕНИЕ ДАННЫХ:

Команда	Синтаксис	Описание	Пример
<b>SELECT</b>	<b>SELECT</b> column1, column2, ... <b>FROM</b> table_name;	<b>SELECT</b> используется для извлечения данных из БД.	<b>SELECT</b> city <b>FROM</b> placeofinterest;
<b>WHERE</b>	<b>SELECT</b> column1, column2, ... <b>FROM</b> table_name <b>WHERE</b> condition;	<b>WHERE</b> используется для извлечения только тех записи, которые соответствуют заданному <b>условию</b> .	<b>SELECT * FROM</b> placeofinterest <b>WHERE</b> city == 'Rome';
<b>COUNT</b>	<b>SELECT COUNT * FROM</b> table_name;	<b>COUNT</b> - это функция, которая принимает имя столбца в качестве аргумента и подсчитывает количество строк, если столбец не является NULL.	<b>SELECT COUNT(country) FROM</b> placeofinterest <b>WHERE</b> country='Canada';
<b>DISTINCT</b>	<b>SELECT DISTINCT</b> column_name <b>FROM</b> table_name;	<b>DISTINCT</b> используется для указания того, что оператор является запросом, который возвращает уникальные значения в указанных столбцах.	<b>SELECT DISTINCT country FROM</b> placeofinterest <b>WHERE</b> type='historical';

<b>LIMIT</b>	<b>SELECT * FROM</b> table_name <b>LIMIT</b> number;	<b>LIMIT</b> это предложение для указания максимального количества строк, которые должен содержать набор результатов.	<b>SELECT * FROM</b> placeofinterest <b>WHERE</b> airport = "pearson" <b>LIMIT 5</b> ;
<b>LIKE</b>	<b>SELECT</b> column1, column2, ... <b>FROM</b> table_name <b>WHERE</b> column <b>LIKE</b> pattern;	<b>LIKE</b> используется в предложении <b>WHERE</b> для поиска заданного шаблона в столбце. В сочетании с оператором <b>LIKE</b> часто используются два знака подстановки: <ul style="list-style-type: none"><li>• знак процента (%) – заменяет содержимое начала (конца или середины) строки</li><li>• знак подчеркивания (_) – заменяет 1 символ</li></ul>	<b>SELECT</b> f_name , l_name <b>FROM</b> employees <b>WHERE</b> address <b>LIKE</b> '%Elgin,IL%';
<b>AND</b>	boolean_expression1 <b>AND</b> boolean_expression2	<b>Оператор AND</b> – это логический оператор, который объединяет два логических выражения или предиката. Требует истинности обоих логических выражений	<b>SELECT</b> title, rating, total_pages <b>FROM</b> books <b>WHERE</b> rating >= 4 <b>AND</b> rating <= 5 <b>ORDER BY</b> title;
<b>OR</b>	boolean_expression1 <b>OR</b> boolean_expression2	<b>Оператор OR</b> – это логический оператор, который объединяет два логических выражения или предиката. Требует истинности хотя бы одного из логических выражений	<b>SELECT</b> title, total_pages <b>FROM</b> books <b>WHERE</b> total_pages = 500 <b>OR</b> total_pages = 1000 <b>ORDER BY</b> total_pages;
<b>IN</b>	expression <b>IN</b> (item1, item2, item3, ...)	<b>Оператор IN</b> – который ищет совпадение в списке значений, указанном в нем. Может быть одно или более истинных значений (совпадений значений столбца со значениями в списке)	<b>SELECT</b> title, total_pages <b>FROM</b> books <b>WHERE</b> total_pages <b>IN</b> (500, 1000) <b>ORDER BY</b> total_pages;

<b>BETWEEN</b>	<pre><b>SELECT</b> column_name(s) <b>FROM</b> table_name <b>WHERE</b> column_name <b>BETWEEN</b> value1 <b>AND</b> value2;</pre>	<p>Оператор <b>BETWEEN</b> выбирает значения в заданном диапазоне. Значениями могут быть числа, текст или даты. Оператор <b>BETWEEN</b> является <b>инклюзивным</b>: начальное и конечное значения включаются.</p>	<pre><b>SELECT * FROM</b> employees <b>WHERE</b> salary <b>BETWEEN</b> 40000 <b>AND</b> 80000;</pre>
<b>ORDER BY</b>	<pre><b>SELECT</b> column1, column2, ... <b>FROM</b> table_name <b>ORDER BY</b> column1, column2, ... <b>ASC DESC</b>;</pre>	<p><b>ORDER BY</b> ключевое слово используется для сортировки набора результатов по возрастанию или убыванию. По умолчанию используется сортировка по возрастанию. Дефолтное значение оператора - <b>ASC</b>.</p>	<pre><b>SELECT</b> f_name, l_name, dep_id <b>FROM</b> employees <b>ORDER BY</b> dep_id <b>DESC</b>, l_name;</pre>
<b>GROUP BY</b>	<pre><b>SELECT</b> column_name(s) <b>FROM</b> table_name <b>WHERE</b> condition <b>GROUP BY</b> column_name(s) <b>ORDER BY</b> column_name(s);</pre>	<p><b>GROUP BY</b> предложение используется с оператором SELECT для упорядочивания идентичных данных в группы.</p>	<pre><b>SELECT</b> dep_id, <b>COUNT(*) FROM</b> employees <b>GROUP BY</b> dep_id;</pre>
<b>ПСЕВДОНИМЫ (ALIASES)</b>			
<b>INNER JOIN</b>	<p><b>Для двух таблиц:</b></p> <pre><b>SELECT</b> select_list (columns) <b>FROM</b> Table1_name <b>INNER JOIN</b> Table2_name <b>ON</b> join_condition;</pre> <p><b>Для более чем 2 таблиц:</b></p> <pre><b>SELECT</b> select_list (columns) <b>FROM</b> Table1_name <b>INNER JOIN</b> Table2_name <b>ON</b> join_condition2 <b>INNER JOIN</b> Table3_name <b>ON</b> join_condition3...;</pre>	<p>Предложение <b>INNER JOIN</b> объединяет каждую строку из первой таблицы с каждой строкой из второй таблицы и так далее, оставляя только те строки, в которых условие соединения оценивается как <b>истинное</b>. Требует четкого указания принадлежности столбца той или иной таблице (смотрите ПСЕВДОНИМЫ)</p>	<pre><b>SELECT</b> b.title, a.first_name, a.last_name <b>FROM</b> books b <b>INNER JOIN</b> book_authors ba <b>ON</b> ba.book_id = b.book_id <b>INNER JOIN</b> authors a <b>ON</b> a.author_id = ba.author_id <b>ORDER BY</b> b.title;</pre>

<b>LEFT JOIN</b>	<p><b>Для двух таблиц:</b></p> <p><b>SELECT</b> select_list (columns)</p> <p><b>FROM</b> Table1_name</p> <p><b>LEFT JOIN</b> Table2_name <b>ON</b> join_condition;</p> <p><b>Для более чем 2 таблиц:</b></p> <p><b>SELECT</b> select_list (columns)</p> <p><b>FROM</b> Table1_name</p> <p><b>LEFT JOIN</b> Table2_name <b>ON</b> join_condition2</p> <p><b>LEFT JOIN</b> Table3_name <b>ON</b> join_condition3...;</p>	<p><b>LEFT JOIN</b> – это один из вариантов объединения двух и более таблиц для запроса данных из объединённого множества таблиц.</p> <p>Предложение <b>LEFT JOIN</b> выбирает данные, начиная с <b>левой таблицы</b> (<b>Table1_name</b>). Он сравнивает каждую строку в левой таблице с каждой строкой в правой таблице и возвращает <b>все строки из левой таблицы</b> (<b>Table1_name</b>) и соответствующие строки или значения NULL из правой таблицы (<b>Table2_name</b>).</p> <p>Так же, как и другие операторы объединения требует четкого указания принадлежности столбца той или иной таблице (смотрите ПСЕВДООНИМЫ)</p> <p><b>ЛЕВАЯ ТАБЛИЦА</b>, это таблица, стоящая <b>СЛЕВА</b> от оператора.</p>	<p><b>SELECT</b> b.title, p.name</p> <p><b>FROM</b> books b</p> <p><b>LEFT JOIN</b> publishers p <b>ON</b> p.publisher_id = b.publisher_id</p> <p><b>ORDER BY</b> b.title;</p>
<b>RIGHT JOIN</b>	<p><b>Для двух таблиц:</b></p> <p><b>SELECT</b> select_list (columns)</p> <p><b>FROM</b> Table1_name</p> <p><b>RIGHT JOIN</b> Table2_name <b>ON</b> join_condition;</p> <p><b>Для более чем 2 таблиц:</b></p> <p><b>SELECT</b> select_list (columns)</p> <p><b>FROM</b> Table1_name</p> <p><b>RIGHT JOIN</b> Table2_name <b>ON</b> join_condition2</p> <p><b>RIGHT JOIN</b> Table3_name <b>ON</b> join_condition3...;</p>	<p>Предложение <b>RIGHT JOIN</b> является перевернутой версией предложения <b>LEFT JOIN</b>. Предложение <b>RIGHT JOIN</b> позволяет запрашивать данные из двух или более таблиц.</p> <p>Другими словами, <b>RIGHT JOIN</b> возвращает все строки из правой таблицы (<b>Table2_name</b>) и соответствующие строки или NULL значения из левой таблицы (<b>Table1_name</b>).</p> <p><b>ПРАВАЯ ТАБЛИЦА</b>, это таблица, стоящая <b>СПРАВА</b> от оператора.</p>	<p><b>SELECT</b> b.title, p.name</p> <p><b>FROM</b> books b</p> <p><b>RIGHT JOIN</b> publishers p <b>ON</b> p.publisher_id = b.publisher_id</p> <p><b>ORDER BY</b> b.title <b>NULLS FIRST</b>;</p>
<b>FULL JOIN</b>	<p><b>Для двух таблиц:</b></p>	<p><b>FULL JOIN</b> возвращает результирующий набор, включающий</p>	<p><b>SELECT</b> b.title,</p>

	<pre><b>SELECT</b>     select_list (columns) <b>FROM</b>     Table1_name <b>FULL OUTER JOIN</b> Table2_name <b>ON</b> join_condition;</pre> <p><b>Для более чем 2 таблиц:</b></p> <pre><b>SELECT</b>     select_list (columns) <b>FROM</b>     Table1_name <b>FULL JOIN</b> Table2_name <b>ON</b> join_condition2 <b>FULL JOIN</b> Table3_name <b>ON</b> join_condition3...;</pre>	<p>строки как из левой, так и из правой таблиц. Если для строки из левой таблицы не существует совпадающих строк, столбцы правой таблицы заполняются NULL. Точно так же, когда для строки из правой таблицы не существует совпадающих строк, столбцы левой таблицы будут заполнены NULL значениями.</p> <p>Ключевое <b>OUTER</b> слово является необязательным и его можно опускать.</p>	<pre>p.name <b>AS</b> publisher <b>FROM</b>     books b <b>FULL OUTER JOIN</b> publishers p <b>ON</b> p.publisher_id = <b>b.publisher_id</b> <b>ORDER BY</b>     b.title <b>NULLS FIRST</b>,     publisher;</pre>
<b>CROSS JOIN</b>	<pre><b>SELECT</b>     select_list (columns) <b>FROM</b>     Table1_name <b>CROSS JOIN</b> Table2_name</pre> <p>Или стоит просто забыть условие соединения:</p> <pre><b>SELECT</b>     select_list (columns) <b>FROM</b> Table1_name, Table2_name;</pre>	<p><b>Объединяет CROSS JOIN</b> каждую строку первой таблицы (Table1_name) с каждой строкой второй таблицы (Table2_name). Он возвращает набор результатов, включающий комбинацию каждой строки в обеих таблицах. Если в объединенных таблицах есть строки <i>n</i> и <i>m</i>, CROSS JOIN будут возвращены <i>n*m</i> строки.</p> <p>Такой оператор соединения стоит использовать с особой осторожностью из-за большого потребления ресурсов. Однако, если вы забудете <i>join_condition</i>, любое соединение превратится в CROSS JOIN</p>	<pre><b>SELECT</b>     c1,     c2 <b>FROM</b>     t1 <b>CROSS JOIN</b> t2;</pre>
<b>INTERSECT</b>	<pre><b>SELECT</b> column1, column2, ... <b>FROM</b> table_name <b>INTERSECT</b> <b>SELECT</b> column1, column2, ... <b>FROM</b> table2_name;</pre>	<p><b>INTERSECT</b> используется для поиска совпадающих данных в указанных столбцах 2 таблиц. Фактически это объединение двух и более <b>SELECT</b> запросов в один. Тип данных столбцов (или выражений) в списке выбора</p>	<pre><b>SELECT</b>     name <b>FROM</b>     customers <b>INTERSECT</b> <b>SELECT</b></pre>

		подзапросов должен быть одинаковым или, по крайней мере, совместимым.	name <b>FROM</b> contacts;
<b>SUB-QUERIS</b>	<b>SELECT</b> select_list (columns), ( <b>SELECT</b> condition ) <b>FROM</b> Table_name <b>ORDER BY</b> Column_name;	Подзапрос — это вложенный оператор <b>SQL</b> , который содержит <b>SELECT</b> оператор внутри предложения <b>WHERE</b> или <b>HAVING</b> . Подзапрос позволяет формировать условие поиска на основе данных другой таблицы, так же применяется для формирования условий на основе вложенных математических расчетов и так далее	<b>SELECT</b> title, total_pages, <b>(SELECT</b> <b>ROUND(AVG(total_pages),0)</b> <b>FROM</b> books ) <b>as</b> avg_pages <b>FROM</b> books <b>ORDER BY</b> title;
<b>HAVING</b>	<b>SELECT</b> select_list <b>FROM</b> table_name <b>GROUP BY</b> column1, column2, ... <b>HAVING</b> search_condition;	<b>HAVING</b> , аналогичный оператору <b>WHERE</b> . Он оценивает каждую группу результатов выборки и включает только те результаты, которые соответствуют условию. В отличии от оператора WHERE работает ТОЛЬКО с <b>GROUP BY</b>	<b>SELECT</b> p.name publisher, <b>COUNT(*)</b> book_count <b>FROM</b> books b <b>INNER JOIN</b> publishers p <b>ON</b> p.publisher_id = b.publisher_id <b>GROUP BY</b> p.name <b>HAVING</b> <b>COUNT(*) &gt; 30</b> <b>ORDER BY</b> book_count;

## СОЗДАНИЕ ТАБЛИЦ:

Команда	Синтаксис	Описание	Пример
	<b>Упрощенный синтаксис:</b> <b>CREATE TABLE</b> table_name ( col1 <b>datatype</b> optional keyword, col2 <b>datatype</b> optional keyword, col3 <b>datatype</b> optional keyword, ..., coln <b>datatype</b> optional keyword)	<b>CREATE TABLE</b> инструкция заключается в создании таблицы. Каждый столбец в таблице задается своим именем, типом данных и необязательным ключевым словом, которое может быть <b>PRIMARY KEY</b> , <b>NOT NULL</b> и т.д.,	<b>CREATE TABLE</b> employee ( employee_id <b>CHAR(2) PRIMARY KEY</b> , first_name <b>VARCHAR(30) NOT NULL</b> , mobile int); <b>или</b>

<b>CREATE TABLE</b>	<b>Общий синтаксис</b> <b>CREATE TABLE</b> <code>[schema_name.]table_name (</code> <code>    column_1 data_type NOT NULL,</code> <code>    column_2 data_type DEFAULT</code> <code>        VALUE,</code> <code>    column_3 DATA_TYPE</code> <code>    CHECK(expression),</code> <code>    ...,</code> <code>    table_constraints</code> <code>);</code>	<ul style="list-style-type: none"> <li>Можно указать схему ([schema_name.]), к которой принадлежит таблица.</li> <li>Список столбцов таблицы.</li> <li>Столбец связан с определенным типом данных и может иметь ограничение (<b>CHECK(expression)</b>), такое как not null и, например, primary key, foreign key и <b>check ограничения</b> (об этом позднее).</li> </ul>	<b>CREATE TABLE</b> stores( <code>    store_id INT GENERATED BY</code> <code>    DEFAULT AS IDENTITY NOT NULL,</code> <code>    store_name VARCHAR(150) NOT</code> <code>    NULL,</code> <code>    address_line_1 VARCHAR(255) NOT</code> <code>    NULL,</code> <code>    address_line_2 VARCHAR(100),</code> <code>    city_id INT NOT NULL,</code> <code>    state_id INT NOT NULL,</code> <code>    zip_code VARCHAR(6),</code> <code>    PRIMARY KEY (store_id);</code>							
<b>РЕЗУЛЬТАТ – ПУСТАЯ ТАБЛИЦА, СОДЕРЖАНИЕ ОПРЕДЕЛЕННЫЕ СТОЛБЦЫ С КАКИМИ УКАЗАННЫМИ ХАРАКТЕРИСТИКАМИ:</b>										
<table border="1" style="width: 100%; text-align: center;"> <thead> <tr> <th>STORE_ID</th><th>STORE_NAME</th><th>ADDRESS_LINE_1</th><th>ADDRESS_LINE_2</th><th>CITY_ID</th><th>STATE_ID</th><th>ZIP_CODE</th></tr> </thead> </table>				STORE_ID	STORE_NAME	ADDRESS_LINE_1	ADDRESS_LINE_2	CITY_ID	STATE_ID	ZIP_CODE
STORE_ID	STORE_NAME	ADDRESS_LINE_1	ADDRESS_LINE_2	CITY_ID	STATE_ID	ZIP_CODE				
<p>У вас сейчас нет никаких данных</p>										
<b>PRIMARY KEY</b>										
Команда	Синтаксис	Описание	Пример							
	<b>CREATE TABLE</b> contacts( <code>    contact_id INT NOT NULL</code> <code>    GENERATED ALWAYS AS IDENTITY,</code> <code>    first_name VARCHAR(100) NOT NULL,</code> <code>    last_name VARCHAR(100) NOT NULL,</code> <code>    PRIMARY KEY(contact_id)</code> <code>);</code> <b>CREATE TABLE</b> phones( <code>    phone_id INT NOT NULL GENERATED</code> <code>    ALWAYS AS IDENTITY,</code> <code>    phone_no VARCHAR(20) NOT NULL,</code> <code>    phone_type VARCHAR(10) NOT NULL,</code> <code>    contact_id INT NOT NULL,</code> <code>    PRIMARY KEY(phone_id)</code> <code>);</code>	<b>PRIMARY KEY</b> означает, что <b>столбец</b> будет хранить уникальные значения, которые уникально идентифицируют <b>каждую строку конкретной таблицы</b> . Предложение <b>GENERATED BY DEFAULT AS IDENTITY</b> помечает <b>store_id</b> столбец как столбец идентификаторов, поэтому, когда вы вставляете новую строку в <b>stores</b> таблицу, Db2 автоматически генерирует последовательное целое число для <b>store_id</b> столбца. Ограничение <b>NOT NULL</b> гарантирует, что <b>store_id</b> не будет принимать никаких значений NULL.	<b>CREATE TABLE</b> favorite_books( <code>    member_id INT NOT NULL,</code> <code>    book_id INT NOT NULL,</code>  <b>FOREIGN KEY</b> (book_id) <code>    REFERENCES books(book_id)</code> <code>    ON UPDATE RESTRICT</code> <code>    ON DELETE CASCADE,</code> <b>FOREIGN KEY</b> (member_id) <code>    REFERENCES</code> <code>    members(member_id)</code> <code>    ON UPDATE RESTRICT</code> <code>    ON DELETE CASCADE</code>							
<b>FOREIGN KEY</b>										

<pre>[CONSTRAINT constraint_name] <b>FOREIGN KEY</b> (fk1, fk2,...) <b>REFERENCES</b> parent_table(c1,2,..)     ON UPDATE [ NO ACTION   RESTRICT]     ON DELETE [ NO ACTION   RESTRICT   CASCADE   SET NULL];</pre>	<p>Внешний ключ — это столбец или группа столбцов в таблице, которые однозначно идентифицируют строку <b>в другой таблице</b>. Ограничения внешнего ключа определяют внешние ключи. Таблица <b>contacts</b> называется родительской таблицей, на которую ссылается внешний ключ. Таблица <b>phones</b> называется дочерней таблицей (или зависимой таблицей), к которой применяется ограничение внешнего ключа.</p>	<pre><b>ALTER TABLE</b> phones <b>FOREIGN KEY</b> (contact_id) <b>REFERENCES</b> contacts (contact_id)  <b>ON UPDATE NO ACTION</b> <b>ON DELETE CASCADE;</b></pre>
---	---	--

**ON UPDATE, ON DELETE** - ограничения внешнего ключа. это разрешения автоматически взаимодействовать со связанными таблицами при действиях в родительских или дочерних таблицах. То есть, если обновляются значения строк, имеющих свойства внешних ключей, обновляются ли или удаляются значения этих же строк в связанных таблицах.

### ОГРАНИЧЕНИЯ ВНЕШНЕГО КЛЮЧА:

**CONSTRAINT** является необязательным.

- список разделенных запятыми столбцов внешнего ключа, заключенных в круглые скобки в FOREIGN KEY.
- имя родительской таблицы и список столбцов, разделенных запятыми, на которые ссылаются столбцы внешнего ключа.

**ON UPDATE**, свойство, при котором, когда вы обновляете строку в родительской или дочерней таблице, происходит обновление этой же строки (с этим же идентификатором) в связанной таблице. Есть две опции: NO ACTION и RESTRICT

Когда вы обновляете строку **в родительском ключевом столбце родительской таблицы**, БАЗА отклоняет обновление, если в дочерней таблице существует соответствующая строка для обоих параметров RESTRICT и NO ACTION

Когда вы обновляете строку **в столбце внешнего ключа дочерней таблицы**, БАЗА отклоняет RESTRICT опцию обновления для и разрешает обновление для NO ACTION при условии, что новое значение столбца внешнего ключа существует в родительской таблице.

**ON DELETE** определяет, следует ли удалять строки в дочерней таблице, на основе следующих параметров:

- **NO ACTION** или RESTRICT не удаляет ни одной строки в обеих таблицах и выдает ошибку.
- **CASCADE** удаляет строку в родительской таблице и все связанные строки в дочерней таблице.
- **SET NULL** удаляет строку в родительской таблице и обновляет значения в столбцах внешнего ключа в дочерней таблице до NULL, только если эти столбцы не являются столбцами, допускающими значение NULL.

Вы можете использовать ограничение внешнего ключа для определения внешних ключей в операторе CREATE TABLE or ALTER TABLE.

## УПРАВЛЕНИЕ ОБЪЕКТАМИ БАЗЫ ДАННЫХ

### 1. Создаем таблицу с необходимыми параметрами

**CREATE TABLE stores(**

```
store_id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,  
store_name VARCHAR(150) NOT NULL,  
address_line_1 VARCHAR(255) NOT NULL,  
address_line_2 VARCHAR(100),  
city_id INT NOT NULL,  
state_id INT NOT NULL,  
zip_code VARCHAR(6),  
PRIMARY KEY (store_id)
```

**);**

В ЭТОЙ **STORES** ТАБЛИЦЕ:

- Столбец **store\_id** является целочисленным столбцом. Предложение **GENERATED BY DEFAULT AS IDENTITY** помечает **store\_id** столбец как столбец идентификаторов, поэтому, когда вы вставляете новую строку в **stores** таблицу система управления базой данных генерирует последовательное целое число для **store\_id** столбца. Ограничение **NOT NULL** гарантирует, что **store\_id** не будет принимать никаких значений **NULL**.
- Это столбец **store\_name** с изменяющимся символом (**VARCHAR**) с максимальной длиной 150. Он имеет **NOT NULL** ограничение, которое будет обеспечивать ненулевые значения.
- Это **address\_line\_1** также столбец с переменными символами, максимальная длина которого составляет 255 символов, и он не принимает значение **NULL**.
- Это **address\_line\_2** столбец с переменными символами с максимальной длиной 100. Это **address\_line\_2** столбец, допускающий значение **NULL**, поэтому он может хранить значения **NULL**.
- И **city\_id** являются **state\_id** целочисленными столбцами. Они не принимают значения **NULL**.
- Столбец **zip\_code** представляет собой столбец переменных символов с максимальной длиной 6. Это столбец, допускающий значение **NULL**.
- **Это store\_id столбец первичного ключа таблицы stores**, заданный ограничением **PRIMARY KEY** в конце оператора. Это означает, что **store\_id** будет хранить уникальные значения, которые идентифицируют все строки таблицы.

## 2. Параметры идентификации столбцов в операторе CREATE, UPDATE or ALTER

column\_name **DATA\_TYPE**

**GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [( identity\_option ) ]**

- Необходимо указать **тип данных** для столбца идентификаторов. **Тип данных может быть SMALLINT, INT и BIGINT**.
- **Далее** – выберите (укажите в операторе) вариант генерации идентификаторов. **Вариантов генерации данных для столбца идентификации два - GENERATED ALWAYS, либо GENERATED BY DEFAULT.**

**GENERATED ALWAYS** - генерируется последовательное целое число для столбца идентификаторов. Любая попытка вставить, изменить значение в столбец идентификаторов с **GENERATED ALWAYS** параметром приведет к ошибке.

**GENERATED BY DEFAULT** - будет генерировать последовательное целое число только в том случае, если вы не укажете значение для столбца идентификаторов. То есть, если вручную ввести идентификатор в таблицу, это не приведет к ошибке.

- Далее необходимо указать параметры столбца идентификации (стартовые значения, максимальные и минимальные значения, что делать если лимит достигнут (мы указали макс 10 идентификаторов и на текущий момент лимит достигнут, база данных может обнулить значения))

**(START WITH** starting\_value  
**INCREMENT BY** increment\_value  
**[MINVALUE** min\_value]  
**[MAXVALUE** max\_value]  
**[CYCLE | NO CYCLE])**

Параметр **IDENTITY** позволяет указать начальное значение в **START WITH** предложении и значение приращения в файле **INCREMENT BY**.

Если значение **INCREMENT** положительное - будет восходящая последовательность, если оно отрицательное - будет нисходящая последовательность.

Опции **MINVALUE** и **MAXVALUE** позволяют указать минимальное и максимальное значения, которые будет генерировать система.

Параметр **CYCLE** или **NOCYCLE** определяет, должна ли система перезапустить значения после того, как были сгенерированы все возможные значения.

**Например**, при использовании **CYCLE** опции и последовательности 1, 2, 3, то система вернет 1, если она сгенерировала 3. Однако, если используется опция **NOCYCLE**, система вместо этого выдаст ошибку.

```
CREATE TABLE NEW_TABLE(  
    id INT GENERATED BY DEFAULT AS IDENTITY  
        (START WITH 10 INCREMENT BY 10),  
    COLUMN VARCHAR(10),  
    PRIMARY KEY(id)  
);
```

```
INSERT INTO  
    NEW_TABLE (COLUMN)  
VALUES  
    ('A'), ('B'), ('C');
```

#### СОЗДАТЬ ТАБЛИЦУ с именем **NEW\_TABLE**

**Id** имеет целочисленный тип значений и **генерируется если не было введено** как уникальный идентификатор, значения которого **начинаются с 10** и каждый раз при вставке новой строки **увеличиваются на 10**.

Имя столбца - **COLUMN** имеет тип изменяющихся символов и максимальную длину в 10 символов.

**Id** – первичный ключ

**SELECT \* FROM NEW\_TABLE;**

**ВСТАВИТЬ** в созданную таблицу **NEW\_TABLE** в столбец с именем **COLUMN** значения ('A'),('B'),('C'); (у нас этот столбец единственный, поэтому синтаксис верный. Итого вставляем 3 строки)

**ВЫБРАТЬ** все (\*) значения из таблицы с именем **NEW\_TABLE**

**Результатом будет следующая таблица:**

ID
10
20
30

COLUMN
A
B
C

**CREATE TABLE NEW\_TABLE2(**

id INT **GENERATED ALWAYS AS IDENTITY**

**(START WITH -1,**

**INCREMENT BY 1,**

**CYCLE,**

**MINVALUE -1,**

**MAXVALUE 2),**

COLUMN VARCHAR(10));

**INSERT INTO**

NEW\_TABLE2(COLUMN)

**VALUES('A'),('B'),('C'),('D'),('E'),('F');**

**SELECT \* FROM NEW\_TABLE2;**

**Результатом будет следующая таблица:**

**СОЗДАТЬ ТАБЛИЦУ** с именем **NEW\_TABLE2**

**Id** имеет целочисленный тип значений и **генерируется как уникальный идентификатор всегда** и не может быть введено иное, значения которого **НАЧИНАЮТСЯ С -1 (МИНИМАЛЬНОЕ ЗНАЧЕНИЕ)** и каждый раз при вставке новой строки **увеличиваются на 1, МАКСИМАЛЬНОЕ ЗНАЧЕНИЕ** идентификатора – 2, оно будет **ЦИКЛИЧНО** (обновляться при достижении максимума)

Имя столбца - **COLUMN** имеет тип изменяющихся символов и максимальную длину в 10 символов.

**Id** – первичный ключ

**ВСТАВИТЬ** в созданную таблицу **NEW\_TABLE2** в столбец с именем **COLUMN** значения ('A'),('B'),('C'),('D'),('E'),('F'); (у нас этот столбец единственный, поэтому синтаксис верный. Итого вставляем 6 строк)

**ВЫБРАТЬ** все (\*) значения из таблицы с именем **NEW\_TABLE2**

ID	COLUMN
-1	A
0	B
1	C
2	D
-1	E
0	F

Можно заметить, что при достижении значения MAXVALUE сработал оператор CYCLE и нумерация столбца ID начала генерироваться « заново ».

## СОЗДАНИЕ КОПИИ ТАБЛИЦЫ

**CREATE TABLE NEW\_TABLE3**

**LIKE NEW\_TABLE2;**

**INSERT INTO NEW\_TABLE3**

**SELECT \* FROM NEW\_TABLE2;**

**DESCRIBE TABLE NEW\_TABLE3;**

**СОЗДАТЬ ТАБЛИЦУ** с именем NEW\_TABLE3

**С ТАКИМИ ЖЕ ПАРАМЕТРАМИ**, как и созданная ранее таблица с именем NEW\_TABLE2

**ВНЕСТИ** в таблицу с именем NEW\_TABLE3 данные, **ВЫБРАННЫЕ ИЗ ТАБЛИЦЫ** с именем NEW\_TABLE2

**ПРОВЕРИТЬ** (ОБЪЯСНИТЬ) изменения таблицы NEW\_TABLE3

**Результатом будет следующая таблица** (таблица NEW\_TABLE2 изменилась в процессе написания гайда и имела текущие параметры):

NEW_TABLE3			
Имя	Тип данных	Допускает пустые значения	Длина
ID	INTEGER	N	
REQUESTED_DATE	DATE	N	4
STATUS	SMALLINT	N	
CREATED_DATE	TIMESTAMP	Y	10

## ИЗМЕНЕНИЕ ТАБЛИЦЫ (ALTER TABLE)

### ДОБАВЛЕНИЕ СТОЛБЦОВ В ТАБЛИЦУ:

**ALTER TABLE** NEW\_TABLE2

**ADD COLUMN** COLUMN1 **VARCHAR (10);**

**DESCRIBE TABLE** NEW\_TABLE2;

**ИЗМЕНИТЬ ТАБЛИЦУ** имеющуюся таблицу с именем NEW\_TABLE2,

**ДОБАВИТЬ СТОЛБЕЦ** с именем COLUMN1 и типом данных VARCHAR с максимальной длиной в 10 символов.

**ПРОВЕРИТЬ** изменения таблицы (как запрашивать изменения зависит от среды исполнения, так же, как и мелкие детали другого синтаксиса)

**Результатом будет следующая таблица:**

Имя	Тип данных	Допускает пустые значения	Длина
ID	INTEGER	N	
COLUMN	VARCHAR	Y	10
<b>COLUMN1</b>	VARCHAR	Y	10

**ALTER TABLE** NEW\_TABLE2

**ADD COLUMN** requested\_date **DATE NOT NULL DEFAULT CURRENT\_DATE**

**ADD COLUMN** status **SMALLINT NOT NULL DEFAULT 0;**

**ИЗМЕНИТЬ ТАБЛИЦУ** имеющуюся таблицу с именем NEW\_TABLE2,

**ДОБАВИТЬ СТОЛБЕЦ** с именем REQUESTED\_DATE и типом данных DATE (дата), не содержащий значений NULL (**пустых**) и генерирующий текущую дату по дефолту, если таковая не была введена.

**ДОБАВИТЬ СТОЛБЕЦ** с именем СТАТУС и типом данных небольшое целочисленное число (-32 768 до +32 767), так же не содержащий пустых значений. Если таковые появляются, система сгенерирует значение 0.

**DESCRIBE TABLE** NEW\_TABLE2;

**ПРОВЕРИТЬ** (дословно: ОБЪЯСНИТЬ) изменения таблицы (как запрашивать изменения зависит от среды исполнения, так же, как и мелкие детали другого синтаксиса)

Результатом будет следующая таблица:

Имя	Тип данных	Допускает пустые значения	Длина
ID	INTEGER	N	
COLUMN	VARCHAR	Y	10
COLUMN1	VARCHAR	Y	10
REQUESTED_DATE	DATE	N	4
STATUS	SMALLINT	N	

### ИЗМЕНЕНИЕ (ALTER) СТОЛБЦА ТАБЛИЦЫ:

**ALTER TABLE** NEW\_TABLE2

**ALTER COLUMN** COLUMN

**SET DATA TYPE CHAR;**

**DESCRIBE TABLE** NEW\_TABLE2;

Результатом будет изменения типа данных столбца:

**ИЗМЕНИТЬ ТАБЛИЦУ** имеющуюся таблицу с именем NEW\_TABLE2,

**ИЗМЕНИТЬ СТОЛБЕЦ** с именем COLUMN (имеет до изменения тип данных VARCHAR (10))

**УСТАНОВИТЬ** тип данных CHAR

**ПРОВЕРИТЬ** изменения

Created by ZORUNA

Имя	↑↓	Тип данных	Допускает пустые значения	Длина
ID		INTEGER	N	
COLUMN		CHAR	Y	1
COLUMN1		VARCHAR	Y	10
REQUESTED_DATE		DATE	N	4
STATUS		SMALLINT	N	

**ALTER TABLE NEW\_TABLE2**

**ADD COLUMN created\_date TIMESTAMP;**

**ALTER TABLE NEW\_TABLE2**

**ALTER COLUMN created\_date**

**SET DEFAULT CURRENT\_TIMESTAMP;**

**DESCRIBE TABLE NEW\_TABLE2;**

**ИЗМЕНИТЬ ТАБЛИЦУ** имеющуюся таблицу с именем NEW\_TABLE2,  
**ДОБАВИТЬ СТОЛБЕЦ** с именем **CREATED\_DATE** и типом данных  
**TIMESTAMP**

**ИЗМЕНИТЬ ТАБЛИЦУ** имеющуюся таблицу с именем NEW\_TABLE2,  
**ИЗМЕНИТЬ СТОЛБЕЦ** с именем CREATED\_DATE, **ПО УМОЛЧАНИЮ**  
проставляющий метку текущего времени, если значение не было  
внесено

**ПРОВЕРИТЬ** изменения

Имя	Тип данных	Допускает пустые значения	Длина
ID	INTEGER	N	
COLUMN	CHAR	Y	1
COLUMN1	VARCHAR	Y	10
REQUESTED_DATE	DATE	N	4
STATUS	SMALLINT	N	
CREATED_DATE	TIMESTAMP	Y	10

При временном типе данных есть тип модификации CURRENT. Он не меняет сам тип данных (TIMESTAMP или DATE) но устанавливает тип генерируемых по умолчанию значений на текущие данные о времени

### ALTER TABLE - УДАЛЕНИЕ СТОЛБЦА:

```
ALTER TABLE NEW_TABLE2  
DROP COLUMN COLUMN  
DROP COLUMN COLUMN1;
```

```
DESCRIBE TABLE NEW_TABLE2;
```

**ИЗМЕНИТЬ ТАБЛИЦУ** имеющуюся таблицу с именем NEW\_TABLE2,  
**УДАЛИТЬ СТОЛБЕЦ** с именами **COLUMN** и **COLUMN1**

**ПРОВЕРИТЬ** изменения

**Результатом будет удаление двух, указанных в операторе столбцов:**

Имя	Тип данных	Допускает пустые значения	Длина
ID	INTEGER	N	
REQUESTED_DATE	DATE	N	4
STATUS	SMALLINT	N	
CREATED_DATE	TIMESTAMP	Y	10

### TRUNCATE TABLE - ОБРЕЗАТЬ ТАБЛИЦУ (НА ПРИМЕРЕ СОЗДАНИЯ КОПИИ ТАБЛИЦЫ JOBS):

```
CREATE TABLE JOBS1 LIKE JOBS;
```

```
INSERT INTO JOBS1 SELECT * FROM JOBS;
```

```
DESCRIBE TABLE JOBS1;
```

```
TRUNCATE TABLE JOBS1 IMMEDIATE;
```

```
DESCRIBE TABLE JOBS1;
```

**СОЗДАТЬ ТАБЛИЦУ** с именем JOBS1

**С ТАКИМИ ЖЕ ПАРАМЕТРАМИ**, как и созданная ранее таблица с именем JOBS

**ВНЕСТИ** в таблицу с именем JOBS1 данные, **ВЫБРАННЫЕ ИЗ ТАБЛИЦЫ** с именем JOBS

**ПРОВЕРИТЬ** (ОБЪЯСНИТЬ) изменения таблицы JOBS1

**УСЕЧЬ** все столбцы таблицы JOBS1 и снова ПРОВЕРИТЬ СОДЕРЖИМОЕ

**Результатом будет УСЕЧЕНИЕМ ВСЕХ столбцов, а точнее ПУСТАЯ таблица:**

**1. Создали таблицу с идентичными свойствами (которые были у исходной таблицы с именем JOBS)**

JOBS1

9 строки

Найти

Имя	Тип данных	Допускает пустые значения	Длина	Масштаб
JOB_IDENT	CHAR	N	9	0
JOB_TITLE	VARCHAR	Y	50	0
MIN_SALARY	DECIMAL	Y	10	2
MAX_SALARY	DECIMAL	Y	10	2

**2. Скопировали в таблицу JOBS1 содержимое исходной таблицы JOBS**

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
200	Sr. Software Developer	60000.00	80000.00
220	Sr. Designer	70000.00	90000.00
234	Sr. Designer	70000.00	90000.00
300	Jr. Software Developer	40000.00	60000.00
400	Jr. Software Developer	40000.00	60000.00
500	Jr. Architect	50000.00	70000.00
600	Lead Architect	70000.00	100000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00

**3. УСЕКЛИ столбцы таблицы (осталась только шапка, пропало все содержимое)**

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
 У вас сейчас нет никаких данных			

**RENAME - ПЕРЕИМЕНОВАНИЕ ТАБЛИЦ И СТОЛБЦОВ:**

**(ВЕРНЕМ В ТАБЛИЦУ СОДЕРЖИМОЕ. ROLLBACK. БУДЕМ ПЕРЕИМЕНОВЫВАТЬ ЭТУ ТАБЛИЦУ И ЕЕ СТОЛБЦЫ)**

Эти операции имеют немного разный синтаксис

**RENAME TABLE JOBS1**

**TO JOBS\_COPY;**

**Результатом будет ПЕРЕИМЕНОВАННАЯ таблица:**

Подробности о таблице					
JOBS_COPY					
9 строки					
Найти					
Имя	Тип данных	Допускает пустые значения	Длина	Масштаб	
JOB_IDENT	CHAR	N	9	0	
JOB_TITLE	VARCHAR	Y	50	0	
MIN_SALARY	DECIMAL	Y	10	2	
MAX_SALARY	DECIMAL	Y	10	2	

**ALTER TABLE JOBS\_COPY**

**RENAME COLUMN MAX\_SALARY**

**TO MAXIMUM\_SALARY;**

**Результатом будет ПЕРЕИМЕНОВАННЫЙ столбец таблицы:**

Подробности о таблице					
JOBS_COPY					
9 строки					
Найти					
Имя	Тип данных	Допускает пустые значения	Длина	Масштаб	
JOB_IDENT	CHAR	N	9	0	
JOB_TITLE	VARCHAR	Y	50	0	
MIN_SALARY	DECIMAL	Y	10	2	
MAXIMUM_SALARY	DECIMAL	Y	10	2	

## МАНИПУЛЯЦИЯ ДАННЫМИ:

После того как мы создали необходимые нам таблицы, внесли данные, все необходимые изменения, если это, конечно, было необходимо, можно приступить к DML – манипуляциям с данными, их извлечению с помощью DATA MANIPULATION LANGUAGE (это такой раздел команд в SQL).

### ОПЕРАТОР SELECT

Порядок выполнения в операторе:

1. Откуда выбирать **FROM**
2. Что выбирать **SELECT**

#### ПРИМЕР ЗАПРОСА

**SELECT \* FROM JOBS\_COPY;**

#### ОПИСАНИЕ

**ВЫБРАТЬ ВСЕ (\*) ИЗ** таблицы с именем JOBS\_COPY;

#### Результирующая таблица (все содержимое)

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
200	Sr. Software Developer	60000.00	80000.00
220	Sr. Designer	70000.00	90000.00
234	Sr. Designer	70000.00	90000.00
300	Jr. Software Developer	40000.00	60000.00
400	Jr. Software Developer	40000.00	60000.00
500	Jr. Architect	50000.00	70000.00
600	Lead Architect	70000.00	100000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00

**SELECT JOB\_IDENT, MAXIMUM\_SALARY FROM JOBS\_COPY;**

**ВЫБРАТЬ** содержимое только столбцов JOB\_IDENT и MAXIMUM\_SALARY  
**ИЗ** таблицы с именем JOBS\_COPY

#### Результирующая таблица (JOB\_IDENT и MAXIMUM\_SALARY)

JOB_ID	MAXIMUM_SALARY
200	80000.00
220	90000.00
234	90000.00
300	60000.00
400	60000.00
500	70000.00
600	100000.00
650	70000.00
660	70000.00

### ORDER BY - СОРТИРОВКА

Порядок выполнения:

1. Откуда выбирать **FROM**
2. Что выбирать **SELECT**
3. По какому принципу упорядочить (отсортировать) **ORDER BY**

Является необязательным предложением оператора SELECT. Он появляется только в конце оператора **SELECT**.

Имеет дефолтное значение **ASC** – по алфавитному порядку или по возрастанию (от меньшего к большему), если это касается числовых значений. При сортировке по умолчанию ключевое слово **ASC** можно не указывать. При указании ключевого слова **DESC** набор результатов сортируется в порядке убывания (от большего к меньшему). Принимает в качестве аргумента выражение или имя столбца. Может принимать в качестве аргумента **вместо имени** столбца его **порядковый номер**, но рекомендуется избегать порядкового расположения столбцов в ORDER BY предложении.

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<b>SELECT * FROM EMPLOYEES;</b>	<b>ВЫБРАТЬ ВСЕ из ТАБЛИЦЫ</b> EMPLOYEES (неотсортированная выборка)
<b>SELECT * FROM EMPLOYEES ORDER BY B_DATE DESC;</b>	<b>ВЫБРАТЬ ВСЕ из таблицы EMPLOYEES и ОТСОРТИРОВАТЬ ПО УБЫВАНИЮ</b> (от большего к меньшему)
<b>SELECT * FROM EMPLOYEES ORDER BY B_DATE;</b>	<b>ВЫБРАТЬ ВСЕ из таблицы EMPLOYEES и ОТСОРТИРОВАТЬ ПО ДЕФОЛТУ</b> (по возрастанию)

Created by ZORUNA

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1002	Alice	James	123457	1972-07-31	F	980 Berry Ln, Elgin,IL	200	80000.00	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000.00	30002	5
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora,IL	400	60000.00	30004	5
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva,IL	500	70000.00	30001	2
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600	90000.00	30001	2
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7
E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,IL	234	70000.00	30003	7
E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,Elgin,IL	220	70000.00	30004	5

## Результирующие таблицы

### 1. Неотсортированная выборка

Выборка, отсортированная по убыванию (DESC) – от большего к меньшему

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,IL	234	70000.00	30003	7
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora,IL	400	60000.00	30004	5
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7
E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,Elgin,IL	220	70000.00	30004	5
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva,IL	500	70000.00	30001	2
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000.00	30002	5
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600	90000.00	30001	2
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7
E1002	Alice	James	123457	1972-07-31	F	980 Berry Ln, Elgin,IL	200	80000.00	30002	5

### 2. Выборка, отсортированная по возрастанию (ASC) – от меньшего к большему

Created by ZORUNA

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1002	Alice	James	123457	1972-07-31	F	980 Berry Ln, Elgin,IL	200	80000.00	30002	5
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600	90000.00	30001	2
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000.00	30002	5
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva,IL	500	70000.00	30001	2
E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,Elgin,IL	220	70000.00	30004	5
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora,IL	400	60000.00	30004	5
E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,IL	234	70000.00	30003	7

## СОРТИРОВКА ИСХОДЯ ИЗ РЕЗУЛЬТАТОВ, ВОЗВРАЩАЕМЫХ ФУНКЦИЯМИ

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<pre>SELECT ADDRESS, L_NAME, EMP_ID FROM EMPLOYEES ORDER BY LENGTH(ADDRESS) DESC;</pre>	<p><b>ВЫБРАТЬ</b> столбцы с именами ADDRESS, L_NAME, EMP_ID из <b>ТАБЛИЦЫ</b> EMPLOYEES и отсортировать по длине (<b>LENGTH</b>) содержимого столбца ADDRESS <b>ПО УБЫВАНИЮ</b>.</p> <p>-----</p> <p>Функция <b>LENGTH ()</b> возвращает длину каждой строки выбранного столбца</p>
<p><b>Результирующая таблица отсортирована от самого длинного адреса к самому короткому:</b></p>	

ADDRESS	L_NAME	EMP_ID
111 Britany Springs, Elgin, IL	Jacob	E1010
145 Berry Ln, Naperville, IL	Gupta	E1008
511 Aurora Av, Aurora, IL	Kumar	E1004
216 Oak Tree, Geneva, IL	Hussain	E1005
120 Fall Creek, Gary, IL	Jones	E1009
980 Berry Ln, Elgin, IL	James	E1002
111 Green Pl, Elgin, IL	Allen	E1006
291 Springs, Gary, IL	Wells	E1003
100 Rose Pl, Gary, IL	Thomas	E1007

### СОРТИРОВКА РЕЗУЛЬТАТОВ, СОДЕРЖАЩИХ ЗНАЧЕНИЯ NULL

NULL – это особые маркеры, указывающие на пропущенные значения. При сортировке, список значений, состоящий из NULL значений, можно указать, следует ли рассматривать NULL значения как самые низкие или самые высокие значения, используя параметр **NULLS FIRST** или **NULLS LAST**. Для изучения этого примера возьмем таблицу, содержащую пропущенные значения

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<pre>SELECT COUNT (*) as NULL_VALUES FROM SCHOOLS WHERE ISAT_VALUE_ADD_MATH IS NULL;</pre>	<p><b>ПОСЧИТАТЬ</b> все NULL значения в столбце <b>ISAT_VALUE_ADD_MATH</b> таблицы <b>SCHOOLS</b></p> <p><b>УСЛОВИЕ</b> в предложении <b>WHERE</b> звучит дословно так: <b>ГДЕ</b> столбец <b>СОДЕРЖИТ</b> NULL (<b>пустое значение</b>) и представить результирующий столбец <b>КАК</b> <b>NULL_VALUES</b> (если не вписать, вместо него будет цифра 1 – порядковый номер столбцы в выборке)</p> <p><b>(их оказалось 98, нам подходит такой вариант для примера)</b></p>
<pre>SELECT NAME_OF_SCHOOL, STREET_ADDRESS, ISAT_VALUE_ADD_MATH FROM SCHOOLS ORDER BY ISAT_VALUE_ADD_MATH NULLS FIRST;</pre>	<p><b>ВЫБРАТЬ</b> столбцы с именами <b>NAME_OF_SCHOOL</b>, <b>STREET_ADDRESS</b>, <b>ISAT_VALUE_ADD_MATH</b> <b>ИЗ</b> таблицы <b>SCHOOLS</b></p> <p><b>СОРТИРОВАТЬ ПО</b> столбцу <b>ISAT_VALUE_ADD_MATH</b> <b>сначала будут идти пустые ячейки (NULLS FIRST);</b></p>
<p><b>Результирующая таблица отсортирована и сначала идут значения NULL</b></p>	

Created by ZORUNA

NAME_OF SCHOOL	STREET_ADDRESS	ISAT_VALUE_ADD_MATH
Air Force Academy High School	3630 S Wells St	
Albert G Lane Technical High School	2501 W Addison St	
Alcott High School for the Humanities	2957 N Hoyne Ave	
Al Raby High School	3545 W Fulton Blvd	
Austin Business and Entrepreneurship Academy High School	231 N Pine Ave	
Austin Polytechnical Academy High School	231 N Pine Ave	

### СОРТИРОВКА РЕЗУЛЬТАТОВ ПО НЕСКОЛЬКИМ СТОЛБЦАМ

#### ПРИМЕР ЗАПРОСА

```
SELECT * FROM JOBS;
```

```
SELECT MAX_SALARY, JOB_TITLE, MIN_SALARY  
FROM JOBS ORDER BY MAX_SALARY DESC, JOB_TITLE;
```

#### ОПИСАНИЕ

**ВЫБРАТЬ** все содержимое **ИЗ** таблицы с именем JOBS (для демонстрации исходного состояния таблицы)

**ВЫБРАТЬ** значения столбцов с именами MAX\_SALARY, JOB\_TITLE, MIN\_SALARY **ИЗ** таблицы JOBS **ОСОРТИРОВАННЫЕ ПО** столбцу по убыванию значений MAX\_SALARY (**DESC**) и по столбцу JOB\_TITLE в алфавитном порядке (**ASC** по умолчанию)

#### Начальная таблица

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
200	Sr. Software Developer	60000.00	80000.00
220	Sr. Designer	70000.00	90000.00
234	Sr. Designer	70000.00	90000.00
300	Jr. Software Developer	40000.00	60000.00
400	Jr. Software Developer	40000.00	60000.00
500	Jr. Architect	50000.00	70000.00
600	Lead Architect	70000.00	100000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00

**Результирующая таблица отсортирована в порядке убывания значений столбца MAX\_SALARY и алфавитном порядке столбца JOB\_TITLE**

MAX_SALARY	JOB_TITLE	MIN_SALARY
100000.00	Lead Architect	70000.00
90000.00	Sr. Designer	70000.00
90000.00	Sr. Designer	70000.00
80000.00	Sr. Software Developer	60000.00
70000.00	Jr. Architect	50000.00
70000.00	Jr. Designer	60000.00
70000.00	Jr. Designer	60000.00
60000.00	Jr. Software Developer	40000.00
60000.00	Jr. Software Developer	40000.00

## ПРЕДЛОЖЕНИЕ WHERE

Предложение **WHERE** является необязательным предложением оператора **SELECT**, **UPDATE** или **DELETE**. Предложение **WHERE** определяет **условие** поиска для строк, возвращаемых, изменяемых или удаляемых вышеуказанными операторами. **WHERE оценивает истинность условия** записанного после него, и выполняет оператор **SELECT (UPDATE или DELETE)** согласно этому условию.

Условия могут быть записаны в скобках для удобства, что может повлиять на порядок их исполнения и смысл предложения WHERE может существенно измениться при необходимости.

В условии можно использовать операторы **<>=**, диапазоны значений и тп.

Например,

**WHERE** (имя\_столбца > 4.7 **AND** имя\_столбца <= 5)

**WHERE** имя\_столбца **BETWEEN** '2018-01-01' **AND** '2018-12-31'

**WHERE** имя\_столбца **IN** (70000,90000)

Порядок выполнения:

1. Откуда выбирать **FROM**
2. Условие выбора **WHERE**

3. Что выбирать **SELECT (UPDATE или DELETE)**

4. По какому принципу упорядочить (отсортировать) **ORDER BY** или иные параметры, если таковые присутствуют

При работе со строками чаще всего используется ключевое слово **LIKE** (похож на, такой же как)

<b>ПРИМЕР ЗАПРОСА</b>	<b>ОПИСАНИЕ</b>
<pre><b>SELECT MAX_SALARY, JOB_TITLE, MIN_SALARY</b> <b>FROM JOBS WHERE UCASE(JOB_TITLE)</b> <b>LIKE '%DESIGNER' ORDER BY MAX_SALARY;</b></pre>	<p><b>ВЫБРАТЬ</b> столбцы с именами <b>MAX_SALARY, JOB_TITLE, MIN_SALARY</b></p> <p><b>ИЗ</b> таблицы <b>JOBS ГДЕ</b> значения в верхнем регистре столбца <b>JOB_TITLE</b></p> <p><b>КАК</b> (похожи на строку, заканчивающуюся словом) '<b>%DESIGNER</b>' сортировать по возрастанию содержимого столбца <b>MAX_SALARY</b>;</p> <p><b>%</b> - плейсхолдер, заменяющий <b>ВСЕ</b> содержимое строки, в месте его постановки</p>

**Результирующая таблица:**

MAX_SALARY	JOB_TITLE	MIN_SALARY
70000.00	Jr. Designer	60000.00
70000.00	Jr. Designer	60000.00
90000.00	% LIKE Sr. Designer	70000.00
90000.00	Sr. Designer	70000.00

<b>ПРИМЕР ЗАПРОСА</b>	<b>ОПИСАНИЕ</b>
<pre><b>SELECT * FROM JOBS</b> <b>WHERE JOB_TITLE LIKE '%DEVELOPER'</b> <b>AND MAX_SALARY BETWEEN 40000 AND 90000</b> <b>ORDER BY MAX_SALARY;</b></pre>	<p><b>ВЫБРАТЬ</b> все содержимое <b>ИЗ</b> таблицы <b>JOBS</b></p> <p><b>ГДЕ</b> содержимое столбца <b>JOB_TITLE</b></p> <p><b>КАК</b> (похожи на строку, заканчивающуюся словом) '<b>%Developer</b>' сортировать по возрастанию содержимого столбца <b>MAX_SALARY</b>,</p> <p><b>И MAX_SALARY МЕЖДУ</b> 40000 <b>И</b> 90000</p>

**Результирующая таблица:**

Created by ZORUNA

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
300	Jr. Software Developer	40000.00	60000.00
400	Jr. Software Developer %	40000.00	60000.00
200	Sr. Software Developer	60000.00	80000.00

**ПРИМЕР ЗАПРОСА**

```
SELECT * FROM JOBS
WHERE MAX_SALARY IN (70000,90000)
ORDER BY MAX_SALARY;
```

**ОПИСАНИЕ**

**ВЫБРАТЬ** все **ИЗ** таблицы с именем JOBS  
**ГДЕ** значения столбца MAX\_SALARY **есть В СПИСКЕ** (70000 , 90000,...)  
**СОРТИРОВАТЬ ПО УБЫВАНИЮ** MAX\_SALARY;  
Если искомое значение все-таки находится в списке **IN** (val,val2,val3,val4, и так далее), оператор вернет его в виде выборки

**Результирующая таблица:**

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
500	Jr. Architect	50000.00	70000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00
220	Sr. Designer	70000.00	90000.00
234	Sr. Designer	70000.00	90000.00

**SELECT \* FROM JOBS
WHERE JOB\_TITLE LIKE '%DEVELOPER'
OR MAX\_SALARY = 90000
ORDER BY JOB\_IDENT;**

**ВЫБРАТЬ** все **ИЗ** таблицы с именем JOBS  
**ГДЕ** содержимое столбца JOB\_TITLE **КАК** (содержит) '%DEVELOPER'  
**ИЛИ** содержимое MAX\_SALARY **эквивалентно** 90000  
**СОРТИРОВАТЬ ПО УБЫВАНИЮ** JOB\_IDENT;

**Результирующая таблица:**

JOB_IDENT	JOB_TITLE		MIN_SALARY	MAX_SALARY
200	% Sr. Software Developer	LIKE	60000.00	80000.00
220	Sr. Designer		70000.00	90000.00
234	Sr. Designer		70000.00	90000.00
300	Jr. Software Developer	LIKE	40000.00	60000.00
400	% Jr. Software Developer		40000.00	60000.00

### SELECT DISTINCT – ВЫБОР УНИКАЛЬНЫХ ЗНАЧЕНИЙ

Ключевое **DISTINCT** слово появляется после **SELECT**, но перед любым столбцом или выражением в списке выбора. Если столбец содержит несколько **NULL** значений, в результирующем наборе **DISTINCT** останется только одно значение **NULL**

В нашей исходной таблице Jobs есть повторяющиеся должности, зарплаты. Поэтому она подходит для примера работы ключевого слова **DISTINCT** (в переводе с английского – отдельный, отличный от остального)

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
200	Sr. Software Developer	60000.00	80000.00
220	Sr. Designer	70000.00	90000.00
234	Sr. Designer	70000.00	90000.00
300	Jr. Software Developer	40000.00	60000.00
400	Jr. Software Developer	40000.00	60000.00
500	Jr. Architect	50000.00	70000.00
600	Lead Architect	70000.00	100000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00

**SELECT DISTINCT JOB\_TITLE, MAX\_SALARY**  
**FROM JOBS;**

**ВЫБРАТЬ УНИКАЛЬНОЕ СОДЕРЖИМОЕ** из столбца **JOB\_TITLE**, а также **MAX\_SALARY** для этих значений **ИЗ** таблицы с именем **JOBS**

JOB_TITLE	MAX_SALARY
Jr. Software Developer	60000.00
Jr. Architect	70000.00
Jr. Designer	70000.00
Sr. Software Developer	80000.00
Sr. Designer	90000.00
Lead Architect	100000.00

## ОПЕРАТОР AND

Оператор **AND** — это логический оператор, который объединяет два логических выражения или предиката. **AND** используется для того, чтобы указать, что поиск должен удовлетворять **обоим условиям**. На практике возможно использование нескольких логических операторов в одном предложении.

*Логическое\_выражение1 AND Логическое\_выражение2*

Эта таблица отображает результат при объединении истинных, ложных и неизвестных значений с помощью **AND** оператора:

	ИСТИНА	ЛОЖЬ	НЕИЗВЕСТНО
ИСТИНА	ИСТИНА	ЛОЖЬ	НЕИЗВЕСТНО
ЛОЖЬ	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ
НЕИЗВЕСТНО	НЕИЗВЕСТНО	ЛОЖЬ	НЕИЗВЕСТНО

### ПРИМЕР ЗАПРОСА

```
SELECT * FROM JOBS
WHERE JOB_TITLE LIKE '%DEVELOPER'
AND MAX_SALARY BETWEEN 40000 AND 90000
ORDER BY MAX_SALARY;
```

### ОПИСАНИЕ

**ВЫБРАТЬ** все содержимое **ИЗ** таблицы JOBS  
**ГДЕ** содержимое столбца **JOB\_TITLE**  
**КАК** (похожи на строку, заканчивающуюся словом) '**%Developer**'  
 сортировать по возрастанию содержимого столбца **MAX\_SALARY**,  
**И MAX\_SALARY МЕЖДУ** 40000 **И** 90000  
**РЕЗУЛЬТАТ ЭТОГО ЗАПРОСА МЫ ВИДЕЛИ РАНЕЕ**

## ОПЕРАТОР OR

Оператор **OR** – это логический оператор, который объединяет два логических выражения или предиката. Оператор **OR** часто используется в **WHERE** предложениях операторов **SELECT**, **UPDATE** и **DELETE** для указания условия поиска для строк, которые необходимо выбрать, обновить и удалить.

В таблице истинности показан результат объединения истинных, ложных и неизвестных значений с помощью **OR** оператора:

	ИСТИНА	ЛОЖЬ	НЕИЗВЕСТНО
ИСТИНА	ИСТИНА	ИСТИНА	ИСТИНА
ЛОЖЬ	ИСТИНА	ЛОЖЬ	НЕИЗВЕСТНО
НЕИЗВЕСТНО	ИСТИНА	НЕИЗВЕСТНО	НЕИЗВЕСТНО

Чтобы получить **НЕ** (выражение **ИЛИ** выражение) – используем буквально **NOT** (boolean\_expression1 **OR** boolean\_expression2), но на основании законов Де Моргана **НЕ** (**А ИЛИ В**) эквивалентно (**НЕ А И НЕ В**). Можно одного и того же результата добиться использую различные операторы. В этом случае эквивалентны следующие условия:

**NOT** (имя\_столбца1 > 4 **OR** имя\_столбца2 < 1000)

и

(имя\_столбца1 <= 4 **AND** имя\_столбца2 >= 1000)

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<b>SELECT * FROM JOBS WHERE JOB_TITLE LIKE '%Designer' OR JOB_TITLE LIKE '%ARCHITECT';</b>	<b>ВЫБРАТЬ</b> все содержимое <b>ИЗ</b> таблицы JOBS <b>ГДЕ</b> содержимое столбца <b>JOB_TITLE</b> <b>КАК</b> (похожи на строку, заканчивающуюся словом) '%Designer' <b>ИЛИ</b> <b>JOB_TITLE</b> содержит текст '%ARCHITECT'

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
220	Sr. Designer	70000.00	90000.00
234	Sr. Designer	70000.00	90000.00
500	Jr. Architect	50000.00	70000.00
600	Lead Architect	70000.00	100000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00

### ЛОГИЧЕСКИЙ ОПЕРАТОР BETWEEN (МЕЖДУ)

**BETWEEN** — это логический оператор, определяющий, находится ли значение **между двумя значениями, указанными в порядке возрастания**. Как и любой другой логический оператор, может использоваться с ключевым словом **NOT**

**WHERE** имя\_столбца **NOT BETWEEN** 1000 **AND** 3000

#### ПРИМЕР ЗАПРОСА

**SELECT \* FROM JOBS**

**WHERE JOB\_TITLE LIKE '%DEVELOPER'**

**AND MAX\_SALARY BETWEEN 40000 AND 90000**

**ORDER BY MAX\_SALARY;**

#### ОПИСАНИЕ

**ВЫБРАТЬ** все содержимое **ИЗ** таблицы JOBS

**ГДЕ** содержимое столбца **JOB\_TITLE**

**КАК** (похожи на строку, заканчивающуюся словом) '**%Developer**'  
сортировать по возрастанию содержимого столбца **MAX\_SALARY**,

**И MAX\_SALARY МЕЖДУ** 40000 **И** 90000

**РЕЗУЛЬТАТ ЭТОГО ЗАПРОСА МЫ ВИДЕЛИ РАНЕЕ**

### ЛОГИЧЕСКИЙ ОПЕРАТОР LIKE (такой как, похожий)

**LIKE** — это логический оператор, который возвращает значение **true**, если строка содержит определенный шаблон, паттерн, который мы указываем в качестве условия для поиска. Шаблон представляет собой строку (текстовые данные, которые содержит столбец), может включать в себя обычные символы и специальные символы, называемые подстановочными знаками.

#### ПРИМЕР ЗАПРОСА

**SELECT \* FROM JOBS**

**WHERE JOB\_TITLE LIKE '%Designer';**

#### ОПИСАНИЕ

**ВЫБРАТЬ** все содержимое **ИЗ** таблицы JOBS

**ГДЕ** содержимое столбца **JOB\_TITLE**

**КАК** (похожи на строку, заканчивающуюся словом) '**% Designer**'

JOB_IDEN	JOB_TITLE	MIN_SALARY	MAX_SALARY
220	Sr. Designer	70000.00	90000.00
234	Sr. Designer	70000.00	90000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00

<b>ПРИМЕР ЗАПРОСА</b>	<b>ОПИСАНИЕ</b>
<pre>SELECT * FROM JOBS WHERE JOB_TITLE LIKE '%Des__n%'; (там 3 нижних подчеркивания, заменяющие каждый по одному символу, который я не помню в названии профессии)</pre>	<b>ВЫБРАТЬ</b> все содержимое <b>ИЗ</b> таблицы JOBS <b>ГДЕ</b> содержимое столбца <b>JOB_TITLE</b> <b>КАК 'НЕПОМНЮС ЧЕГО НАЧИНАЕТСЯDesНЕПОМНЮЗСИМВОЛА ВНУТРИСЛОВАнНЕПОМНЮЧЕМЗАКАНЧИВАЕТСЯ'</b>

JOB_IDEN	JOB_TITLE	MIN_SALARY	MAX_SALARY
220	Sr. Designer	70000.00	90000.00
234	Sr. Designer	70000.00	90000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00

<b>ЛОГИЧЕСКИЙ ОПЕРАТОР IN (В)</b>	<b>IN (ЗНАЧЕНИЕ, ЗНАЧЕНИЕ, ЗНАЧЕНИЕ, ...)</b>
<p>Оператор <b>IN</b> возвращает <b>true</b>, если условие поиска совпадает с одним из значений <b>в списке</b>, указанном в скобках.</p> <p>Список значений может быть литеральным значением или набором результатов запроса.</p>	
<b>ПРИМЕР ЗАПРОСА</b>	<b>ОПИСАНИЕ</b>
<pre>SELECT * FROM JOBS WHERE MAX_SALARY IN (60000,70000) ORDER BY MAX_SALARY;</pre>	<b>ВЫБРАТЬ</b> все <b>ИЗ</b> таблицы с именем JOBS <b>ГДЕ</b> значения столбца MAX_SALARY <b>В СПИСКЕ ЗНАЧЕНИЙ</b> (60000 и 70000) <b>СОРТИРОВАТЬ ПО УБЫВАНИЮ</b> MAX_SALARY; <p>В данном случае, у нас есть оба эти значения, но оператор будет искать любое истинное значение в списке</p>

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
300	Jr.Software Developer	40000.00	60000.00
400	Jr.Software Developer	40000.00	60000.00
500	Jr. Architect	50000.00	70000.00
650	Jr. Designer	60000.00	70000.00
660	Jr. Designer	60000.00	70000.00

Бывают более сложные запросы, где логическим операторам приходится давать в качестве аргумента целые наборы результатов (sub-select), которые заключаются в скобки

```
WHERE publisher_id IN
(
    SELECT
        publisher_id
    FROM
        publishers
    WHERE name LIKE 'Addison Wesley%'
);
```

### LIMIT:

Предложение **LIMIT** позволяет ограничить количество строк, возвращаемых запросом. Предложение LIMIT является расширением оператора **SELECT**. Часто используется, когда выборка слишком объёмная и необходимо получить только **ТОП** лузеров или **ТОП** лидеров. Чаще используется вместе с сортировкой.

**SELECT** столбцы

**FROM** имя\_таблицы

**ORDER BY** условие сортировки, если таковое есть

**LIMIT** n [**OFFSET** m]; // более короткий синтаксис **LIMIT** m, n;

**ГДЕ:**

n - количество возвращаемых строк.

m - это количество строк, которые необходимо пропустить перед возвратом n строк (**OFFSET** является необязательным параметром)

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ																
<pre>SELECT * FROM JOBS ORDER BY JOB_TITLE LIMIT 3</pre>	<b>ВЫБРАТЬ</b> все <b>ИЗ</b> таблицы с именем JOBS <b>ОСОРТИРОВАТЬ</b> по алфавиту в столбце JOB_TITLE <b>ВЫВЕСТИ</b> (вернуть в качестве результата) <b>ТОЛЬКО 3</b> строки																
<table border="1"> <thead> <tr> <th>JOB_IDENT</th><th>JOB_TITLE</th><th>MIN_SALARY</th><th>MAX_SALARY</th></tr> </thead> <tbody> <tr> <td>500</td><td>Jr. Architect</td><td>50000.00</td><td>70000.00</td></tr> <tr> <td>650</td><td>Jr. Designer</td><td>60000.00</td><td>70000.00</td></tr> <tr> <td>660</td><td>Jr. Designer</td><td>60000.00</td><td>70000.00</td></tr> </tbody> </table>		JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY	500	Jr. Architect	50000.00	70000.00	650	Jr. Designer	60000.00	70000.00	660	Jr. Designer	60000.00	70000.00
JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY														
500	Jr. Architect	50000.00	70000.00														
650	Jr. Designer	60000.00	70000.00														
660	Jr. Designer	60000.00	70000.00														
ПРИМЕР ЗАПРОСА	ОПИСАНИЕ																
<pre>SELECT * FROM JOBS ORDER BY JOB_TITLE LIMIT 3 OFFSET 5</pre>	<b>ВЫБРАТЬ</b> все <b>ИЗ</b> таблицы с именем JOBS <b>ОСОРТИРОВАТЬ</b> по алфавиту в столбце JOB_TITLE <b>ВЫВЕСТИ</b> (вернуть в качестве результата) <b>ТОЛЬКО 3</b> строки <b>НО</b> начиная с 6 строки ( <i>6,7,8 строки будут в выборке, 5 предыдущих будут пропущены</i> )																
<table border="1"> <thead> <tr> <th>JOB_IDENT</th><th>JOB_TITLE</th><th>MIN_SALARY</th><th>MAX_SALARY</th></tr> </thead> <tbody> <tr> <td>600</td><td>Lead Architect</td><td>70000.00</td><td>100000.00</td></tr> <tr> <td>220</td><td>Sr. Designer</td><td>70000.00</td><td>90000.00</td></tr> <tr> <td>234</td><td>Sr. Designer</td><td>70000.00</td><td>90000.00</td></tr> </tbody> </table>		JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY	600	Lead Architect	70000.00	100000.00	220	Sr. Designer	70000.00	90000.00	234	Sr. Designer	70000.00	90000.00
JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY														
600	Lead Architect	70000.00	100000.00														
220	Sr. Designer	70000.00	90000.00														
234	Sr. Designer	70000.00	90000.00														
<b>FETCH</b>																	
<p>Позволяет <b>ограничить количество строк</b>, возвращаемых запросом и получить небольшое, указанное в предложении <b>FETCH</b> подмножество строк</p> <p>Используется следующим образом:</p> <p><b>OFFSET n ROWS</b></p>																	

## **FETCH {FIRST | NEXT } m {ROW | ROWS} ONLY**

N - количество строк, которые необходимо пропустить.

m- количество возвращаемых строк. И FIRST, и NEXT взаимозаменямы соответственно. Они используются для смысловой цели.

Подобно предложению **LIMIT**, FETCH всегда использует ORDER BY, чтобы получить возвращаемые строки в указанном порядке.

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ																								
<b>SELECT * FROM JOBS</b>	<b>ВЫБРАТЬ</b> все <b>ИЗ</b> таблицы с именем JOBS																								
<b>ORDER BY JOB_TITLE</b>	<b>ОСОРТИРОВАТЬ</b> по алфавиту в столбце JOB_TITLE																								
<b>FETCH FIRST 5 ROWS ONLY;</b>	<b>ВЫБРАТЬ ТОЛЬКО ПЕРВЫЕ 5 СТРОК</b>																								
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">JOB_IDENT</th><th style="text-align: left; padding: 2px;">JOB_TITLE</th><th style="text-align: left; padding: 2px;">MIN_SALARY</th><th style="text-align: left; padding: 2px;">MAX_SALARY</th></tr> </thead> <tbody> <tr> <td style="padding: 2px;">500</td><td style="padding: 2px;">Jr. Architect</td><td style="padding: 2px;">50000.00</td><td style="padding: 2px;">70000.00</td></tr> <tr> <td style="padding: 2px;">650</td><td style="padding: 2px;">Jr. Designer</td><td style="padding: 2px;">60000.00</td><td style="padding: 2px;">70000.00</td></tr> <tr> <td style="padding: 2px;">660</td><td style="padding: 2px;">Jr. Designer</td><td style="padding: 2px;">60000.00</td><td style="padding: 2px;">70000.00</td></tr> <tr> <td style="padding: 2px;">300</td><td style="padding: 2px;">Jr. Software Developer</td><td style="padding: 2px;">40000.00</td><td style="padding: 2px;">60000.00</td></tr> <tr> <td style="padding: 2px;">400</td><td style="padding: 2px;">Jr. Software Developer</td><td style="padding: 2px;">40000.00</td><td style="padding: 2px;">60000.00</td></tr> </tbody> </table>		JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY	500	Jr. Architect	50000.00	70000.00	650	Jr. Designer	60000.00	70000.00	660	Jr. Designer	60000.00	70000.00	300	Jr. Software Developer	40000.00	60000.00	400	Jr. Software Developer	40000.00	60000.00
JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY																						
500	Jr. Architect	50000.00	70000.00																						
650	Jr. Designer	60000.00	70000.00																						
660	Jr. Designer	60000.00	70000.00																						
300	Jr. Software Developer	40000.00	60000.00																						
400	Jr. Software Developer	40000.00	60000.00																						
<b>SELECT * FROM JOBS</b>	<b>ВЫБРАТЬ</b> все <b>ИЗ</b> таблицы с именем JOBS																								
<b>ORDER BY JOB_TITLE</b>	<b>ОСОРТИРОВАТЬ</b> по алфавиту в столбце JOB_TITLE																								
<b>OFFSET 3 ROWS</b>	<b>ПРОПУСТИТЬ 3 СТРОКИ</b> выборки																								
<b>FETCH NEXT 2 ROWS ONLY;</b>	<b>ВЫБРАТЬ ТОЛЬКО СЛЕДУЮЩИЕ ЗА НИМИ 2 СТРОКИ</b>																								
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">JOB_IDENT</th><th style="text-align: left; padding: 2px;">JOB_TITLE</th><th style="text-align: left; padding: 2px;">MIN_SALARY</th><th style="text-align: left; padding: 2px;">MAX_SALARY</th></tr> </thead> <tbody> <tr> <td style="padding: 2px;">300</td><td style="padding: 2px;">Jr. Software Developer</td><td style="padding: 2px;">40000.00</td><td style="padding: 2px;">60000.00</td></tr> <tr> <td style="padding: 2px;">400</td><td style="padding: 2px;">Jr. Software Developer</td><td style="padding: 2px;">40000.00</td><td style="padding: 2px;">60000.00</td></tr> </tbody> </table>		JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY	300	Jr. Software Developer	40000.00	60000.00	400	Jr. Software Developer	40000.00	60000.00												
JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY																						
300	Jr. Software Developer	40000.00	60000.00																						
400	Jr. Software Developer	40000.00	60000.00																						

## ПСЕВДОНИМЫ (ALIASES)

При использовании оператора **SELECT**, он возвращает данные используя заголовки столбцов или порядковые номера (особенно в случае использования подзапросов и математических расчетов). Для лучшей «картины» происходящего иногда нужно присвоить в выборке другие имена **столбцов и таблиц (при операциях с несколькими таблицами)** – это называется **Псевдонимы**

Присваиваются они с помощью ключевого слова **AS (как)** и заключаются в кавычки "", в случае если они содержат **пробелы**.

*Created by ZORVNA*

**Синтаксис присвоения псевдонимов следующий: {столбец | выражение} AS “псевдоним\_столбца”. Если мы не заключаем псевдоним в кавычки, то его необходимо писать в CAMMELCASE или с нижним подчеркиванием (AS **AVERAGE\_SALARY**);**

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<pre>SELECT JOB_TITLE AS "Job Position", MIN_SALARY AS "Minimum Salary" FROM JOBS LIMIT 4;</pre>	<p><b>ВЫБРАТЬ</b> JOB_TITLE и переименовать его в текущей выборке на «<b>Job Position</b>», так же выбрать столбец MIN_SALARY и переименовать его в текущей выборке на «<b>Minimum Salary</b>»</p> <p><b>ИЗ</b> таблицы JOBS</p> <p><b>Ограничить выборку</b> 4 строками</p>

Job Position	Minimum Salary
Sr. Software Developer	60000.00
Sr. Designer	70000.00
Sr. Designer	70000.00
Jr. Software Developer	40000.00

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<pre>SELECT ROUND(AVG(MAX_SALARY),2) AS "AVERAGE SALARY"  FROM JOBS;</pre>	<p><b>ВЫБРАТЬ ИЗ</b> таблицы JOBS округленное (<b>ROUND()</b>) до 2 знаков после запятой среднее значение(<b>AVG()</b>) столбца MAX_SALARY и назвать его в текущей выборке <b>«AVERAGE SALARY»</b></p>

В данном случае, если бы псевдоним не был присвоен, столбец выборки имел бы только порядковый номер (в данном случае – 1) и это могло бы стать проблемой в интерпретации информации в дальнейшем, так как было бы не совсем ясно какая именно информация хранится в выборке

После назначения псевдонимов столбцов на них можно ссылаться в *ORDER BY* предложении

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<pre>SELECT JOB_TITLE AS <b>JOBPOSITION</b> FROM JOBS ORDER BY <b>JOBPOSITION</b> LIMIT 3;</pre>	<p><b>ВЫБРАТЬ</b> JOB_TITLE и <b>переименовать</b> его в текущей выборке на <b>JOBPOSITION</b> из таблицы JOBS</p> <p><b>ОСОРТИРОВАТЬ</b> по столбцу с псевдонимом <b>JOBPOSITION</b></p> <p><b>Ограничить</b> выборку 3 строками</p>

**РЕЗУЛЬТАТ ВЫБОРКИ:**

JOBPOSITION
Jr. Architect
Jr. Designer
Jr. Designer

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<pre> <b>SELECT</b> <b>JOBS.JOB_TITLE AS "JOB POSITION",</b> <b>JOBS.JOB_IDENT AS "ID",</b> <b>EMPLOYEES.L_NAME AS "LAST NAME",</b> <b>EMPLOYEES.F_NAME AS "FIRST NAME",</b> <b>EMPLOYEES.ADDRESS</b> <b>FROM JOBS, EMPLOYEES</b> <b>WHERE JOBS.JOB_IDENT = EMPLOYEES.JOB_ID</b> <b>ORDER BY "JOB POSITION"</b> <b>LIMIT 5;</b> </pre>	<p>Это вариант запроса данных из 2 таблиц с использованием полного имени таблицы (<b>Имя_Таблицы.Имя_Столбца_Этой_Таблицы</b>)</p> <p><b>ВЫБРАТЬ</b> из таблицы <b>JOBS</b> столбец JOB_TITLE <b>назвать его в текущей выборке</b> «JOB POSITION»,</p> <p>из таблицы JOBS столбец JOB_IDENT <b>назвать его в текущей выборке</b> «ID»,</p> <p>из таблицы <b>EMPLOYEES</b> получить столбец L_NAME <b>назвать его в текущей выборке</b> «LAST NAME»,</p> <p>из таблицы <b>EMPLOYEES</b> получить столбец F_NAME <b>назвать его в текущей выборке</b> «FIRST NAME»,</p> <p>из таблицы <b>EMPLOYEES</b> получить столбец ADDRESS</p> <p><b>ГДЕ</b> идентификатор работника таблицы <b>JOBS</b> (JOBS.JOB_IDENT) эквивалентен идентификатору работника таблицы <b>EMPLOYEES</b> (EMPLOYEES.JOB_ID)</p> <p><b>ОСОРТИРОВАТЬ</b> по столбцу с псевдонимом «JOB POSITION»</p> <p><b>ОГРАНИЧИТЬ</b> выборку 5 строками;</p>

**РЕЗЛЪТАТ:**

Job Position	ID	LAST NAME	FIRST NAME	ADDRESS
Jr. Architect	500	Hussain	Ahmed	216 Oak Tree, Geneva,IL
Jr. Designer	650	Thomas	Mary	100 Rose Pl, Gary,IL
Jr. Designer	660	Gupta	Bharath	145 Berry Ln, Naperville,IL
Jr. Software Developer	300	Wells	Steve	291 Springs, Gary,IL
Jr. Software Developer	400	Kumar	Santosh	511 Aurora Av, Aurora,IL

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
<pre> SELECT J.JOB_TITLE AS "JOB POSITION", J.JOB_IDENT AS "ID", E.L_NAME AS "LAST NAME", E.F_NAME AS "FIRST NAME", E.ADDRESS FROM JOBS J, EMPLOYEES E WHERE J.JOB_IDENT = E.JOB_ID ORDER BY "JOB POSITION" LIMIT 5; </pre>	<p>То же самое можно сделать в упрощенной форме, присвоив имени таблицы удобный и понятный, желательно, псевдоним.</p> <p><b>FROM JOBS → J, EMPLOYEES → E</b></p> <p>где <b>J</b> псевдоним таблицы <b>JOBS</b>, а <b>E</b> псевдоним таблицы <b>EMPLOYEES</b>.</p> <p>Для того, чтобы система понимала какой запрашиваемый столбец принадлежит какой таблице, используется следующий синтаксис: <b>псевдоним.имя_столбца_этой_таблицы</b> (между полным названием либо псевдонимом таблицы <b>всегда стоит ТОЧКА</b>)</p> <p><b>При видоизменении кода результат выборки никак не изменится, вопрос удобства.</b></p>

Job Position	ID	LAST NAME	FIRST NAME	ADDRESS
Jr. Architect	500	Hussain	Ahmed	216 Oak Tree, Geneva,IL
Jr. Designer	650	Thomas	Mary	100 Rose Pl, Gary,IL
Jr. Designer	660	Gupta	Bharath	145 Berry Ln, Naperville,IL
Jr. Software Developer	300	Wells	Steve	291 Springs, Gary,IL
Jr. Software Developer	400	Kumar	Santosh	511 Aurora Av, Aurora,IL

## JOIN ОПЕРАТОРЫ

### INNER JOIN (ВНУТРЕННЕЕ ОБЪЕДИНЕНИЕ)

**INNER JOIN** используется для получения строк из таблиц **JOBS** и **EMPLOYEES** с одинаковыми **ID**, в данном случае. Однако, это могут быть любые идентичные друг другу по смысловой нагрузке столбцы (имена, фамилии, адреса, продублированные в двух или более таблицах вашей базы данных). Собственно, все зависит от структуры базы данных.

Существует возможность объединять две и более таблиц. Необходимо точное указание какие столбцы принадлежат какой таблице

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
----------------	----------

**SELECT**

```
J.JOB_TITLE AS "JOB POSITION",
J.JOB_IDENT AS "ID",
E.L_NAME AS "LAST NAME",
E.F_NAME AS "FIRST NAME",
E.ADDRESS
FROM JOBS J INNER JOIN EMPLOYEES E
ON J.JOB_IDENT = E.JOB_ID
ORDER BY "JOB POSITION"
LIMIT 5;
```

Опять же, код делает то же самое что и два предыдущих запроса.

**FROM JOBS → J, EMPLOYEES → E**

где **J псевдоним таблицы JOBS, а E псевдоним таблицы EMPLOYEES.**

Существует одно различие в синтаксисе - здесь используется **ОБЪЕДИНЕНИЕ ТАБЛИЦ**.

**Имя\_таблицы1 INNER JOIN имя\_таблицы2 ON (по совпадению значений идентификаторов этих двух таблиц (JOB\_IDENT и JOB\_ID это один и тот же параметр уникально определяющий номер проекта, в котором участвуют работники и является связующим звеном между этими таблицами)**

CONTACT_ID	NAME	CUSTOMER_ID	NAME
1	Amelia	1	Amelia
2	Olivia	2	Isla
3	Isla	3	Jessica
4	Emily	4	Lily

Объединим их по именам, которые совпадают в обеих таблицах для получения всей информации в общей выборке

**ПРИМЕР ЗАПРОСА**

**ОПИСАНИЕ**

**SELECT**

```
CO.CONTACT_ID,
CO.NAME as CONTACT_NAME,
CU.CUSTOMER_ID,
CU.NAME as CUSTOMER_NAME
```

**FROM**

**ВЫБРАТЬ** следующие столбцы

**CONTACT\_ID, принадлежащий таблице с псевдонимом CO NAME, принадлежащий таблице с псевдонимом CO и назвать его в текущей выборке именем CONTACT\_NAME,**  
**CUSTOMER\_ID, принадлежащий таблице с псевдонимом CU NAME, принадлежащий таблице с псевдонимом CU и назвать его в текущей выборке именем CUSTOMER\_NAME**

CONTACTS CO

INNER JOIN CUSTOMERS CU

ON CU.NAME = CO.NAME;

ИЗ

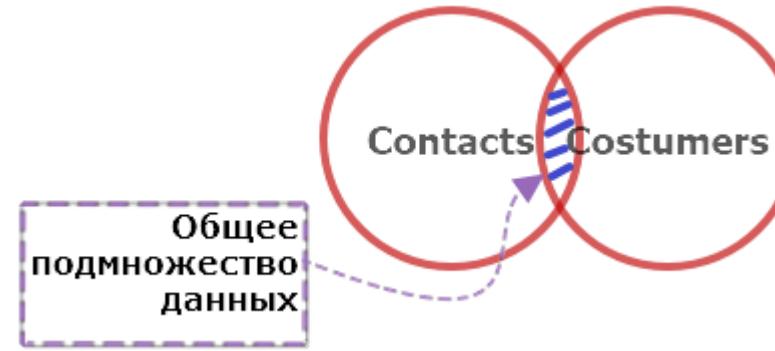
ОБЪЕДИНЕННЫХ ТАБЛИЦ (CUSTOMERS CU) и CONTACTS с псевдонимом CO

По принципу эквивалентности имен (CU.NAME = CO.NAME);

В результате получим ОБЩЕЕ ПОДМНОЖЕСТВО имен, при объединении мы не получим в выборке имена ОЛИВИЯ, ЭМИЛИ, ДЖЕССИКА и ЛИЛЛИ, так как они находятся только какой-то одной из таблиц и не эквивалентны именам в другой таблице

CONTACT_ID	↓	CONTACT_NAME	CUSTOMER_ID	CUSTOMER_NAME
1		Amelia	1	Amelia
3		Isla	2	Isla

Объединение подмножеств строк происходит по следующей схеме (заштриховано):



### LEFT JOIN (ЛЕВОЕ ОБЪЕДИНЕНИЕ)

LEFT JOIN выбирает данные, начиная с левой таблицы, и сопоставляет строки в правой таблице. Подобно внутреннему соединению, левое соединение возвращает все строки из левой таблицы и соответствующие строки из правой таблицы. Кроме того, если строка в левой таблице не имеет соответствующей строки в правой таблице, в столбцах правой таблицы будут пустые (NULL) значения. Этот тип соединения таблиц также имеет название LEFT OUTER (внешнее) JOIN

SELECT

CO.CONTACT\_ID,  
CO.NAME AS CONTACT\_NAME,

ВЫБРАТЬ следующие столбцы

CONTACT\_ID, принадлежащий таблице с псевдонимом CO

Created by ZORUNA

```
CU.CUSTOMER_ID,  
CU.NAME AS CUSTOMER_NAME  
FROM  
CONTACTS CO  
LEFT JOIN CUSTOMERS CU  
ON CU.NAME = CO.NAME;
```

NAME, принадлежащий таблице с псевдонимом CO и назвать его в текущей выборке именем CONTACT\_NAME,

CUSTOMER\_ID, принадлежащий таблице с псевдонимом CU

NAME, принадлежащий таблице с псевдонимом CU и назвать его в текущей выборке именем CUSTOMER\_NAME

ИЗ

Таблицы CONTACTS с псевдонимом CO

**ОБЪЕДИНИТЬ ЛЕВУЮ (целиком) ТАБЛИЦУ (CONTACTS) с правой таблицей CUSTOMERS (только эквивалентные ее значения) с псевдонимом CU**

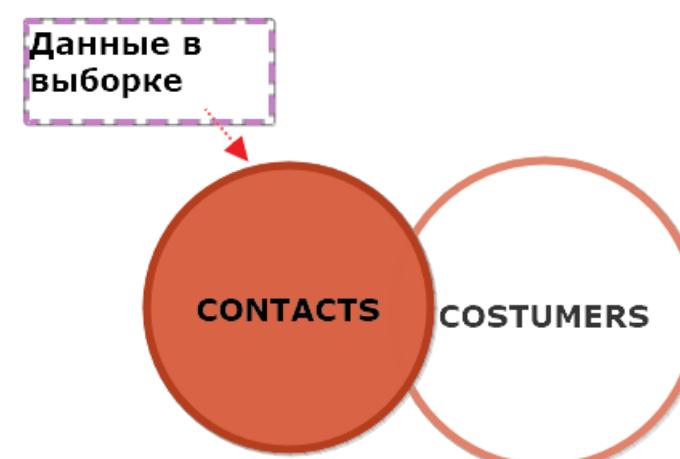
По принципу эквивалентности имен (CU.NAME = CO.NAME);

**В результирующей таблице** будут **ВСЕ** имена из левой таблицы (CONTACTS) и только 2 имени, из правой, те, которые **идентичны в двух таблицах**

CONTACT_ID	CONTACT_NAME	CUSTOMER_ID	CUSTOMER_NAME
1	Amelia	1	Amelia
3	Isla	2	Isla
4	Emily		
2	Olivia		

RIGHT TABLE NULL

Объединение подмножеств строк происходит по следующей схеме - ЛЕВАЯ таблица полностью выведена и захватывает лишь идентичную часть данных правой (диаграмма Венна):

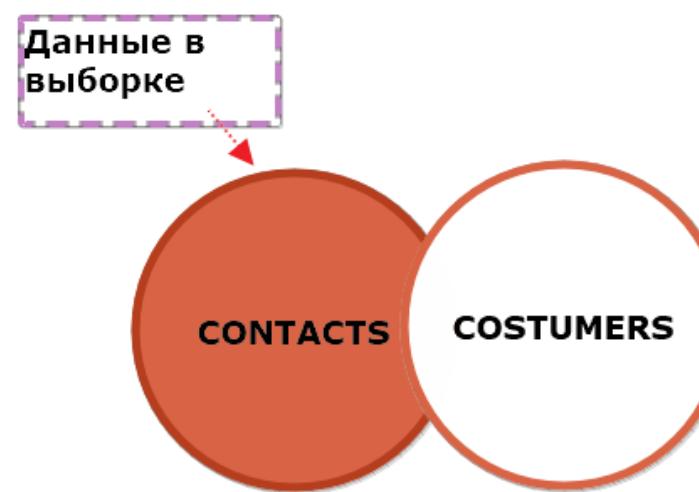


На схеме видно, что главная (левая) таблица выводится полностью и только идентичная часть правой попадет в выборку.

Чтобы получить строки, которые доступны только в левой таблице, но не в правой, вы добавляете в конец предыдущего запроса предложение **WHERE (WHERE cu.name IS NULL)**:

CONTACT_ID	CONTACT_NAME	CUSTOMER_ID	CUSTOMER_NAME
4	Emily		
2	Olivia		<b>RIGHT TABLE NULL</b>

Результатом будут, по сути, уникальные имена из левой таблицу, которых нет нигде (в данном случае, нет в правой таблице). На диаграмме Венна это выглядит так:



### RIGHT JOIN (ПРАВОЕ ОБЪЕДИНЕНИЕ)

**RIGHT JOIN** объединение работает как обратное LEFT JOIN и выбирает **данные, начиная с правой таблицы, и сопоставляет строки в левой таблице**. **ПРАВОЕ соединение возвращает все строки из** правой таблицы и соответствующие строки из левой таблицы. Кроме того, если строка в правой таблице не имеет соответствующей строки в левой таблице, в столбцах левой таблицы **будут пустые (NULL) значения**. Этот тип соединения таблиц так же имеет название **RIGHT OUTER** (внешнее) JOIN

**SELECT**

```
CO.CONTACT_ID,  
CO.NAME AS CONTACT_NAME,  
CU.CUSTOMER_ID,  
CU.NAME AS CUSTOMER_NAME
```

**FROM**

**ВЫБРАТЬ** следующие столбцы

```
CONTACT_ID, принадлежащий таблице с псевдонимом CO  
NAME, принадлежащий таблице с псевдонимом CO и называть его в  
текущей выборке именем CONTACT_NAME,  
CUSTOMER_ID, принадлежащий таблице с псевдонимом CU  
NAME, принадлежащий таблице с псевдонимом CU и называть его в  
текущей выборке именем CUSTOMER_NAME
```

CONTACTS CO

RIGHT JOIN CUSTOMERS CU  
ON CU.NAME = CO.NAME

ИЗ

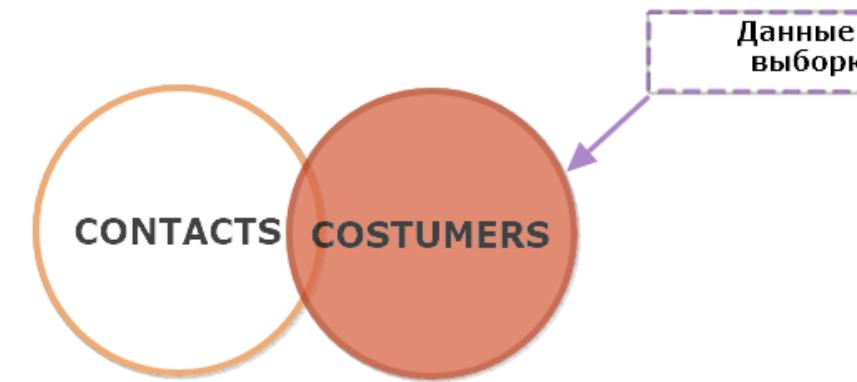
ОБЪЕДИНЕННЫХ ПРАВОЙ ТАБЛИЦЫ (CUSTOMERS CU) с левой  
таблицей CONTACTS с псевдонимом CO

По принципу эквивалентности имен (CU.NAME = CO.NAME);

**В результирующей таблице** будут **ВСЕ** имена из **ПРАВОЙ** таблицы (**CUSTOMERS**) и **только 2** имени, из **левой** (**CONTACTS**), те, которые **идентичны** в **двух** таблицах

CONTACT_ID	CONTACT_NAME	CUSTOMER_ID	CUSTOMER_NAME
1	Amelia	1	Amelia
3	Isla	2	Isla
LEFT TABLE NULL		4	Lily
		3	Jessica

Объединение подмножеств строк происходит по следующей схеме правая таблица полностью выведена и захватывает лишь идентичную часть левой (диаграмма Венна):



Чтобы получить строки, которые доступны только в правой таблице (**CUSTOMERS**), но не в левой, вы добавляете в конец предыдущего запроса предложение **WHERE (WHERE co.name IS NULL)**:

CONTACT_ID	CONTACT_NAME	CUSTOMER_ID	CUSTOMER_NAME
LEFT TABLE NULL		4	Lily
		3	Jessica

На диаграмме эта выборка будет выглядеть так:



### FULL JOIN (ПОЛНОЕ ВНЕШНЕЕ ОБЪЕДИНЕНИЕ)

**Полное объединение** возвращает результатирующий набор, **включающий все строки из левой и правой таблиц**, с соответствующими строками из обеих сторон, если они существуют (та самая идентичность имен). В случае отсутствия совпадения отсутствующая таблица будет иметь **нулевое(NULL) значение**, так как их не с кем (имя) сопоставить. То есть, это разные люди.

**SELECT**

```
CO.CONTACT_ID,  
CO.NAME AS CONTACT_NAME,  
CU.CUSTOMER_ID,  
CU.NAME AS CUSTOMER_NAME
```

**FROM**

```
CONTACTS CO  
FULL OUTER JOIN CUSTOMERS CU  
ON CU.NAME = CO.NAME
```

**ВЫБРАТЬ** следующие столбцы

```
CONTACT_ID, принадлежащий таблице с псевдонимом CO  
NAME, принадлежащий таблице с псевдонимом CO и назвать его в  
текущей выборке именем CONTACT_NAME,  
CUSTOMER_ID, принадлежащий таблице с псевдонимом CU  
NAME, принадлежащий таблице с псевдонимом CU и назвать его в  
текущей выборке именем CUSTOMER_NAME
```

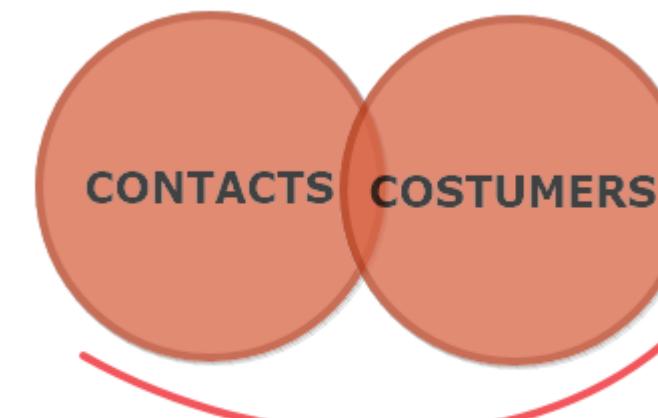
**ИЗ**

**ПОЛНОСТЬЮ ОБЪЕДИНЕННЫХ ТАБЛИЦ** CONTACTS с псевдонимом  
CO таблицы CUSTOMERS с псевдонимом CU

**По принципу эквивалентности имен (CU.NAME = CO.NAME)** –  
наших ключевых значений и идентичных по смысловой нагрузке,  
уникально идентифицирующих людей в нашей Бд;

CONTACT_ID	CONTACT_NAME	CUSTOMER_ID	CUSTOMER_NAME
1	Amelia	1	Amelia
2	Olivia		RIGHT TABLE NULL
3	Isla	2	Isla
4	Emily		RIGHT TABLE NULL
		4	Lily
		3	Jessica

На диаграмме Венна это выглядит так:



В выборке будут ВСЕ  
данные из обеих таблиц

Чтобы выбрать строки, доступные либо в левой, либо в правой таблице, вы исключаете строки, общие для обеих таблиц, добавляя предложение **WHERE** к приведенному выше запросу:

(**WHERE** co.name **IS NOT NULL OR** cu.name **IS NOT NULL**)

CONTACT_ID	CONTACT_NAME	CUSTOMER_ID	CUSTOMER_NAME
2	Olivia		RIGHT TABLE NULL
4	Emily		RIGHT TABLE NULL
		4	Lily
		3	Jessica

На диаграмме выборка будет выглядеть так. Место пересечения двух таблиц (совпадающие данные) не будут включены в выборку:



### GROUP BY (ГРУППИРОВКА РЕЗУЛЬТАТОВ ВЫБОРКИ)

Когда вы используете **SELECT** оператор для запроса данных, вы получаете набор результатов, состоящий из множества строк. Чтобы разделить эти строки на группы, вы используете **GROUP BY**. Этот оператор нуждается в количественном определении. Отвечает на вопросы как много одинакового содержимого в столбце, сколько таких (одинаковых школ, цифр, имен и тп) значений содержится в столбце. Используется с агрегатными функциями, такими как AVG(), COUNT().

Этот оператор делит строки, полученные из **FROM** предложения, на группы по одному или нескольким выражениям столбца (column1, column2, ...), указанным в **GROUP BY**. Для фильтрации сгруппированных запросов используется ключевое слово **HAVING** (используется только с **GROUP BY**)

ВСЕ ЗАПРАШИВАЕМЫЕ СТОЛБЦЫ В СЕЛЕКТ ЛИСТЕ, НЕ ВКЛЮЧЕННЫЕ В АГГРЕГАТНЫЕ ФУНКЦИИ, ДОЛЖНЫ БЫТЬ УКАЗАНЫ В ОПЕРАТОРЕ **GROUP BY**

**SELECT** JOB\_TITLE,

**COUNT**(JOB\_TITLE) AS COUNT

**FROM** JOBS

**GROUP BY** JOB\_TITLE;

**ВЫБРАТЬ СТОЛБЕЦ** JOB\_TITLE

**ПОСЧИТАТЬ СКОЛЬКО** в столце значений каждой профессии

**В ТАБЛИЦЕ** JOBS

**СГРУППИРОВАТЬ ПО** JOB\_TITLE

Created by ZORUNA

JOB_TITLE	COUNT
Jr. Architect	1
Jr. Designer	2
Jr. Software Developer	2
Lead Architect	1
Sr. Designer	2
Sr. Software Developer	1

Для фильтрации сгруппированных запросов используется ключевое слово **HAVING** (используется только с **GROUP BY**)

```
SELECT JOB_TITLE,  
COUNT(JOB_TITLE) AS COUNT  
FROM JOBS  
GROUP BY JOB_TITLE  
HAVING COUNT>1;
```

**ВЫБРАТЬ СТОЛБЕЦ** JOB\_TITLE  
**ПОСЧИТАТЬ СКОЛЬКО** в столце значений каждой профессии  
**В ТАБЛИЦЕ** JOBS  
**СГРУППИРОВАТЬ ПО** JOB\_TITLE  
И вывести только строки, которые **СОДЕРЖАТ** количество должностей больше 1

JOB_TITLE
Jr. Designer
Jr. Software Developer
Sr. Designer

COUNT
2
2
2

```
SELECT DESCRIPTION, COUNT(PRIMARY_TYPE)  
AS COUNT_TYPE_OF_CRIME  
FROM CRIMEDATA  
GROUP BY DESCRIPTION  
ORDER BY COUNT_TYPE_OF_CRIME DESC;
```

**ВЫБРАТЬ СТОЛБЕЦ** DESCRIPTION  
**ПОСЧИТАТЬ СКОЛЬКО** существует типов преступлений (PRIMARY\_TYPE)  
**В ТАБЛИЦЕ** CRIMEDATA  
**СГРУППИРОВАТЬ ПО** DESCRIPTION (описанию)  
**СОРТИРОВАТЬ** по убыванию значений количества типов преступлений

Created by ZORUNA

**В РЕЗУЛЬТИРУЮЩЕМ СЕТЕ** мы видим, что большинство преступлений за 2019 год – это мелкие правонарушения и кражи, и домашнее насилие.

DESCRIPTION	COUNT_TYPE_OF_CRIME
SIMPLE	59
\$500 AND UNDER	48
DOMESTIC BATTERY SIMPLE	37
TO PROPERTY	28
OVER \$500	26
POSS: CANNABIS 30GMS OR LESS	25
TO VEHICLE	25
FORCIBLE ENTRY	20

МАЛО ТОГО, МЫ СИЛЬНО УМЕНЬШИЛИ РЕЗУЛЬТИРУЮЩИЙ НАБОР (**ИЗ 533 СТРОК ЗНАЧЕНИЙ МЫ ПОЛУЧИЛИ 103 СТРОКИ, КОТОРЫЕ В ЦЕЛОМ ПОКАЗЫВАЮТ КАКИЕ ПРЕСТУПЛЕНИЯ СОВЕРШАЛИСЬ ЧАЩЕ ВСЕГО ЗА ОТЧЕТНЫЙ ПЕРИОД**)

3406613	HK456306	2004-06-26	009XX N CENTRAL PARK AVE	820	\$500 AND UNDER	\$500 AND UNDER
8002131	HT233595	2011-04-04	043XX S WABASH AVE	820	DOMESTIC BATTERY SIMPLE	\$500 AND UNDER
7903289	HT133522	2010-12-30	083XX S KINGSTON AVE	840	TO PROPERTY	
10402076	HZ138551	2016-02-02	033XX W 66TH ST	820	OVER \$500	
7732712	HS540106	2010-09-29	006XX W CHICAGO AVE	810	POSS: CANNABIS 30GMS OR LESS	
10769475	HZ534771	2016-11-30	050XX N KEDZIE AVE	810	TO VEHICLE	
←		FORCIBLE ENTRY		←		
Элементов на странице: 50		1 – 50 из 533 элементов		Элементов на странице: 50		1 – 50 из 103 элементов

## ФУНКЦИИ

Агрегатная функция принимает несколько строк в качестве входных данных и возвращает одно значение для этих строк.

Некоторыми часто используемыми агрегатными функциями являются **AVG ()**, **COUNT ()**, и, например, функция **COUNT ()** возвращает количество строк для каждой группы.

**AVG ()** - Функция возвращает среднее значение всех значений в группе

**MIN (), MAX(), SUM(), COUNT(), AVG()**

*Created by ZORVN4*