Created by ZORINA

SQL PRACTICE

КРАТКОЕ СОДЕРЖАНИЕ И БАЗОВЫЙ СИНТАКСИС:

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

Команда	Синтаксис	Описание	Пример
CREATE TABLE	**CREATE TABLE** table_name (col1 datatype optional keyword, col2 datatype optional keyword, col3 datatype optional keyword,..., coln datatype optional keyword)	**CREATE TABLE** инструкция заключается в создании таблицы. Каждый столбец в таблице задается своим именем, типом данных и необязательным ключевым словом, которое может быть **PRIMARY KEY**, **NOT NULL** и т.д.,	**CREATE TABLE employee (** employee_id char(2) **PRIMARY KEY,** first_name varchar(30) **NOT NULL,** mobile int);
INSERT	**INSERT INTO** table_name (column1,column2,column3...) **VALUES**(value1,value2,value3...);	**INSERT INTO** 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.*	**INSERT INTO** placeofinterest (name,type,city,country,airport) **VALUES**('Niagara Waterfalls','Nature','Toronto','Canada' ,'Pearson');
UPDATE	**UPDATE** table_name **SET**[[column1]=[VALUES]] **WHERE** [condition];	**UPDATE** используется для обновления строк в таблице.	**UPDATE** placeofinterest **SET** name = 'Niagara Falls' **WHERE** name = "Niagara Waterfalls»;
DELETE	**DELETE FROM** table_name **WHERE** [condition];	**DELETE** используется для удаления из таблицы строк, указанных в операторе WHERE. Без указания **WHERE** оператор удалит все строки из таблицы	**DELETE FROM** placeofinterest **WHERE** city **IN** ('Rome','Vienna');
ALTER TABLE - ADD COLUMN	**ALTER TABLE** table_name **ADD COLUMN** column_name_1 datatype, **ADD COLUMN** …, **ADD COLUMN** column_name_n datatype;	ALTER TABLE – **ADD COLUMN** используется для добавления столбцов в таблицу.	**ALTER TABLE** employee **ADD COLUMN** income bigint;

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

ИЗВЛЕЧЕНИЕ ДАННЫХ:

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

LIMIT	**SELECT** * **FROM** table_name **LIMIT** number;	**LIMIT** это предложение для указания максимального количества строк, которые должен содержать набор результатов.	**SELECT** * **FROM** placeofinterest **WHERE** airport = "pearson" **LIMIT 5**;
LIKE	**SELECT** column1, column2, ... **FROM** table_name **WHERE** column **LIKE** pattern;	**LIKE** используется в предложении **WHERE** для поиска заданного шаблона в столбце. В сочетании с оператором **LIKE** часто используются два знака подстановки: • *знак процента (%) – заменяет содержимое начала (конца или середины) строки* • *знак подчеркивания (_) – заменяет 1 символ*	**SELECT** f_name , l_name **FROM** employees **WHERE** address **LIKE** '%Elgin,IL%';
AND	boolean_expression1 **AND** boolean_expression2	**Оператор AND—** это логический оператор, который объединяет два логических выражения или предиката**.** Требует истинности обоих логических выражений	**SELECT** title, rating, total_pages **FROM** books **WHERE** rating >= 4 **AND** rating <= 5 **ORDER BY** title;
OR	boolean_expression1 **OR** boolean_expression2	**Оператор OR—** это логический оператор, который объединяет два логических выражения или предиката**.** Требует истинности хотя бы одного их логических выражений	**SELECT** title, total_pages **FROM** books **WHERE** total_pages = 500 **OR** total_pages = 1000 **ORDER BY** total_pages;
IN	expression **IN (item1, item2, item2, …)**	**Оператор IN —** который ищет совпадение в списке значений, указанном в нем. Может быть одно или более истинных значений (совпадений значений столбца со значениями в списке)	**SELECT** title, total_pages **FROM** books **WHERE** total_pages **IN** (500, 1000) **ORDER BY** total_pages;

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

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

	SELECT select_list (columns) **FROM** Table1_name **FULL OUTER JOIN** Table2_name **ON** join_condition**;** **Для более чем 2 таблиц:** **SELECT** select_list (columns) **FROM** Table1_name **FULL JOIN** Table2_name **ON** join_condition2 **FULL JOIN** Table3_name **ON** join_condition3...;	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. Ключевое **OUTER** слово является необязательным и его можно опускать**.**	p.name **AS** publisher **FROM** books b **FULL OUTER JOIN** publishers p **ON** p.publisher_id = **b.**publisher_id **ORDER BY** b.title **NULLS FIRST,** publisher**;**
CROSS JOIN	**SELECT** select_list (columns) **FROM** Table1_name **CROSS JOIN** Table2_name Или стоит просто забыть условие соединения: **SELECT** select_list (columns) **FROM** Table1_name**,** Table2_name**;**	**Объединяет CROSS JOIN** 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 n и m, CROSS JOINбудут возвращены n*m строки. *Такой оператор соединения стоит использовать с особой осторожностью из-за большого потребления ресурсов Однако, если вы забудете* ***join_condition****, любое соединение превратится в CROSS JOIN*	**SELECT** **c1,** **c2** **FROM** **t1** **CROSS JOIN t2;**
INTERSECT	**SELECT** column1, column2, ... **FROM** table_name **INTERSECT** **SELECT** column1, column2, ... **FROM** table2_name;	**INTERSECT** используется для поиска совпадающих данных в указанных столбцах 2 таблиц. Фактически это объединение двух и более **SELECT** запросов в один. Тип данных столбцов (или выражений) в списке выбора	**SELECT** name **FROM** customers **INTERSECT** **SELECT**

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

СОЗДАНИЕ ТАБЛИЦ:

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

Created by ZORINA

CREATE TABLE	**Общий синтаксис** **CREATE TABLE** [**schema_name**.]**table_name** (**column_1 data_type NOT NULL**, **column_2 data_type DEFAULT VALUE**, **column_3 DATA_TYPE CHECK**(**expression**), …, **table_constraints**);	• Можно указать схему ([schema_name.]), к которой принадлежит таблица. • Список столбцов таблицы. • Столбец связан с определенным типом данных и может иметь ограничение (**CHECK**(expression), такое как not null и, например, primary key, foreign key и **check ограничения** (об этом позднее).	**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));

РЕЗУЛЬТАТ – ПУСТАЯ ТАБЛИЦА, СОДЕРЖАНИЕ ОПРЕДЕЛЕННЫЕ СТОЛБЦЫ С КАКИМИ УКАЗАННЫМИ ХАРАКТЕРИСТИКАМИ:

STORE_ID	STORE_NAME	ADDRESS_LINE_1	ADDRESS_LINE_2	CITY_ID	STATE_ID	ZIP_CODE

У вас сейчас нет никаких данных

PRYMARY KEY

Команда	Синтаксис	Описание	Пример
	CREATE TABLE contacts(contact_id **INT NOT NULL GENERATED ALWAYS AS IDENTITY**, first_name VARCHAR(100) NOT NULL, last_name VARCHAR(100) NOT NULL, **PRIMARY KEY**(contact_id)); **CREATE TABLE** phones(phone_id **INT NOT NULL GENERATED ALWAYS AS IDENTITY**, phone_no VARCHAR(20) NOT NULL, phone_type VARCHAR(10) NOT NULL, contact_id INT NOT NULL, **PRIMARY KEY**(phone_id));	**PRIMARY KEY** означает, что **столбец** будет хранить уникальные значения, которые уникально идентифицируют **каждую строку конкретной таблицы.** Предложение **GENERATED BY DEFAULT AS IDENTITY** помечает **store_id** столбец как столбец идентификаторов, поэтому, когда вы вставляете новую строку в **stores** таблицу, Db2 автоматически генерирует последовательное целое число для **store_id** столбца. Ограничение **NOT NULL гарантирует**, что **store_id** не будет принимать никаких значений NULL.	**CREATE TABLE** favorite_books(member_id **INT NOT NULL**, book_id **INT NOT NULL**, **FOREIGN KEY** (book_id) **REFERENCES** books(book_id) **ON UPDATE RESTRICT** **ON DELETE CASCADE**, **FOREIGN KEY** (member_id) **REFERENCES** members(member_id) **ON UPDATE RESTRICT** **ON DELETE CASCADE**);

FOREIGN KEY

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

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	COLUMN
10	A
20	B
30	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;	**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**

Результатом будет следующая таблица:

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 (ОБЬЯСНИТЬ) изменения таблицы NEW_TABLE3

Результатом будет следующая таблица (*таблица NEW_TABLE2 изменялась 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

Created by ZORINA

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
COLUMN1	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;**

DESCRIBE TABLE NEW_TABLE2;

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.

	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;	**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

Результатом будет изменения типа данных столбца:

Created by ZORINA

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

Created by ZORINA

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 (ОБЬЯСНИТЬ) 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**;**	**ПЕРЕИМЕНОВАТЬ** таблицу с именем JOBS1 **НА** 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**;**	**ИЗМЕНИТЬ ТАБЛИЦУ** с именем JOBS_COPY **ПЕРЕИМЕНОВАТЬ СТОЛБЕЦ** под названием MAX_SALARY **НА** 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	ОПИСАНИЕ
SELECT * FROM JOBS_COPY**;**	**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_IDENT	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
SELECT * FROM EMPLOYEES**;**	**ВЫБРАТЬ ВСЕ** из **ТАБЛИЦЫ** EMPLOYEES (неотсортированная выборка)
SELECT * FROM EMPLOYEES **ORDER BY** B_DATE **DESC;**	**ВЫБРАТЬ ВСЕ** из таблицы EMPLOYEES и **ОТСОРТИРОВАТЬ ПО УБЫВАНИЮ** (от большего к меньшему)
SELECT * FROM EMPLOYEES **ORDER BY** B_DATE**;**	**ВЫБРАТЬ ВСЕ** из таблицы EMPLOYEES и **ОТСОРТИРОВАТЬ ПО ДЕФОЛТУ** (по возрастанию)

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 ZORINA

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
SELECT ADDRESS, L_NAME, EMP_ID **FROM** EMPLOYEES **ORDER BY LENGTH(**ADDRESS**) DESC;**	**ВЫБРАТЬ** столбцы с именами ADDRESS, L_NAME, EMP_ID из **ТАБЛИЦЫ** EMPLOYEES и отсортировать по длине (**LENGTH**) содержимого столбца ADDRESS **ПО УБЫВАНИЮ.** --- Функция **LENGTH ()** возвращает длину каждой строки выбранного столбца

Результирующая таблица отсортирована от самого длинного адреса к самому короткому:

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

Результирующая таблица отсортирована и сначала идут значения NULL

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 **ОТСОРТИРОВАННЫЕ 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)

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

Результирующая таблица:

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

по возрастанию

% LIKE

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

Результирующая таблица:

Created by ZORINA

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

% LIKE

по возрастанию

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
SELECT * FROM JOBS **WHERE** MAX_SALARY **IN (**70000,90000**)** **ORDER BY** MAX_SALARY**;**	**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 OR =
234	Sr. Designer	70000.00	90000.00
300	Jr.Software Developer	40000.00	60000.00
400	% Jr.Software Developer LIKE	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 нам 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_выражение2 **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	ОПИСАНИЕ
SELECT * FROM JOBS **WHERE** JOB_TITLE **LIKE** '%DEVELOPER' **AND** MAX_SALARY **BETWEEN** 40000 **AND** 90000 **ORDER BY** MAX_SALARY;	**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**
SELECT * FROM JOBS **WHERE** JOB_TITLE **LIKE** ‘%Designer’ **OR** JOB_TITLE **LIKE** ‘%ARCHITECT’**;**	**ВЫБРАТЬ** все содержимое **ИЗ** таблицы JOBS **ГДЕ** содержимое столбца **JOB_TITLE** **КАК** (похожи на строку, заканчивающуюся словом) '**%** Designer' **ИЛИ** JOB_TITLE содержит текст '%ARCHITECT'

Created by ZORINA

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	ОПИСАНИЕ
SELECT * FROM JOBS **WHERE** JOB_TITLE **LIKE** '%DEVELOPER' **AND** MAX_SALARY **BETWEEN** 40000 **AND** 90000 **ORDER BY** MAX_SALARY;	**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	ОПИСАНИЕ
SELECT * FROM JOBS **WHERE** JOB_TITLE **LIKE '%**Designer**';**	**ВЫБРАТЬ** все содержимое **ИЗ** таблицы JOBS **ГДЕ** содержимое столбца **JOB_TITLE** **КАК** (похожи на строку, заканчивающуюся словом) '**%** Designer'

Created by ZORINA

JOB_IDENT	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

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
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)*	**ВЫБРАТЬ** все содержимое **ИЗ** таблицы JOBS **ГДЕ** содержимое столбца **JOB_TITLE** **КАК 'НЕПОМНЮС ЧЕГО НАЧИНАЕТСЯDesНЕПОМНЮЗСИМВОЛА ВНУТРИСЛОВАnНЕПОМНЮЧЕМЗАКАНЧИВАЕТСЯ'**

JOB_IDENT	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

ЛОГИЧЕСКИЙ ОПЕРАТОР IN (В)

IN (ЗНАЧЕНИЕ, ЗНАЧЕНИЕ, ЗНАЧЕНИЕ, ...)

Оператор **IN** возвращает **true**, если условие поиска совпадает с одним из значений **в списке,** указанном в скобках.

Список значений может быть литеральным значением или набором результатов запроса.

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
SELECT * FROM JOBS **WHERE** MAX_SALARY **IN (**60000,70000**)** **ORDER BY** MAX_SALARY**;**	**ВЫБРАТЬ** все **ИЗ** таблицы с именем JOBS **ГДЕ** значения столбца MAX_SALARY **В СПИСКЕ ЗНАЧЕНИЙ (**60000 и 70000**)** **СОРТИРОВАТЬ ПО УБЫВАНИЮ** MAX_SALARY**;** В данном случае, у нас есть оба эти значения, но оператор будет искать любое истинное значение в списке

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**

Created by ZORINA

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
SELECT * FROM JOBS **ORDER BY** JOB_TITLE **LIMIT** 3	**ВЫБРАТЬ** все **ИЗ** таблицы с именем JOBS **ОТСОРТИРОВАТЬ** по алфавиту в столбце JOB_TITLE **ВЫВЕСТИ** (вернуть в качестве результата) **ТОЛЬКО 3** строки

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
SELECT * FROM JOBS **ORDER BY** JOB_TITLE **LIMIT** 3 **OFFSET** 5	**ВЫБРАТЬ** все **ИЗ** таблицы с именем JOBS **ОТСОРТИРОВАТЬ** по алфавиту в столбце JOB_TITLE **ВЫВЕСТИ** (вернуть в качестве результата) **ТОЛЬКО 3** строки **НО** начиная с 6 строки *(6,7,8 строки будут в выборке, 5 предыдущих будут пропущены)*

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

FETCH

Позволяет **ограничить количество строк**, возвращаемых запросом и получить небольшое, указанное в предложении **FETCH** подмножество строк

Используется следующим образом:

OFFSET n **ROWS**

Created by ZORINA

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

N - количество строк, которые необходимо пропустить.

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

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

ПРИМЕР ЗАПРОСА	ОПИСАНИЕ
SELECT * FROM JOBS **ORDER BY** JOB_TITLE **FETCH FIRST** 5 **ROWS ONLY;**	**ВЫБРАТЬ** все **ИЗ** таблицы с именем JOBS **ОТСОРТИРОВАТЬ** по алфавиту в столбце JOB_TITLE **ВЫБРАТЬ ТОЛЬКО ПЕРВЫЕ 5 СТРОК**

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

SELECT * FROM JOBS **ORDER BY** JOB_TITLE **OFFSET** 3 **ROWS** **FETCH NEXT** 2 **ROWS ONLY;**	**ВЫБРАТЬ** все **ИЗ** таблицы с именем JOBS **ОТСОРТИРОВАТЬ** по алфавиту в столбце JOB_TITLE **ПРОПУСТИТЬ** 3 **СТРОКИ** выборки **ВЫБРАТЬ ТОЛЬКО СЛЕДУЮЩИЕ ЗА НИМИ** 2 **СТРОКИ**

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 (ALIASSES)

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 ZORINA

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
SELECT JOB_TITLE **AS "Job Position",** MIN_SALARY **AS "Minimum Salary"** **FROM** JOBS **LIMIT** 4**;**	**ВЫБРАТЬ** JOB_TITLE **и переименовать его в текущей выборке на «Job Position»**, так же выбрать столбец MIN_SALARY и **переименовать его в текущей выборке на «Minimum Salary»** **ИЗ** таблицы JOBS **Ограничить выборку** 4 строками

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
SELECT ROUND(AVG(MAX_SALARY**),**2**)** **AS "AVERAGE SALARY"** **FROM** JOBS**;**	**ВЫБРАТЬ ИЗ** таблицы JOBS округленное (**ROUND()**) до 2 знаков после запятой среднее значение(**AVG()**) столбца MAX_SALARY и назвать его в текущей выборке **«AVERAGE SALARY»**

AVERAGE SALARY
76666.670000000000000000000000

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
SELECT JOB_TITLE **AS JOBPOSITION FROM** JOBS **ORDER BY JOBPOSITION** **LIMIT** 3**;**	**ВЫБРАТЬ** JOB_TITLE **и переименовать его в текущей выборке на JOBPOSITION ИЗ** таблицы JOBS **ОТСОРТИРОВАТЬ** по столбцу с псевдонимом **JOBPOSITION** **Ограничить выборку** 3 строками

Created by ZORINA

РЕЗУЛЬТАТ ВЫБОРКИ:

JOBPOSITION
Jr. Architect
Jr. Designer
Jr. Designer

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

РЕЗЛЬТАТ:

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
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;	**То же самое можно сделать в упрощенной форме, присвоив имени таблицы удобный и понятный, желательно, псевдоним.** **FROM JOBS → J, EMPLOYEES → E** где **J псевдоним таблицы JOBS**, а **E псевдоним таблицы EMPLOYEES**. Для того, чтобы система понимала какой запрашиваемый столбец принадлежит какой таблице, используется следующий синтаксис: ***псевдоним.имя_столбца_этой_таблицы*** *(между полным названием либо псевдонимом таблицы **всегда стоит ТОЧКА**)* ***При видоизменении кода результат выборки никак не изменится, вопрос удобства.***

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 обьединять 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)*

В следующем примере существуют 2 очень простые таблицы: Contacts и Costumers со следующим содержимым

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

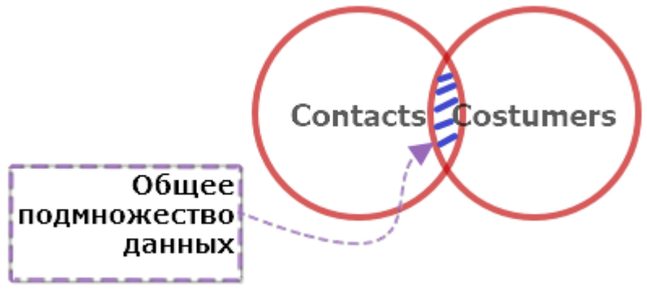
ПРИМЕР ЗАПРОСА	**ОПИСАНИЕ**
SELECT **CO.**CONTACT_ID**,** **CO.**NAME **as** CONTACT_NAME**,** **CU.**CUSTOMER_ID**,** **CU.**NAME **as** CUSTOMER_NAME **FROM**	**ВЫБРАТЬ** следующие столбцы CONTACT_ID, **принадлежащий таблице с псевдонимом** CO NAME, **принадлежащий таблице с псевдонимом** CO и назвать 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 так же имеет название **LEFT OUTER** (внешнее) **JOIN**

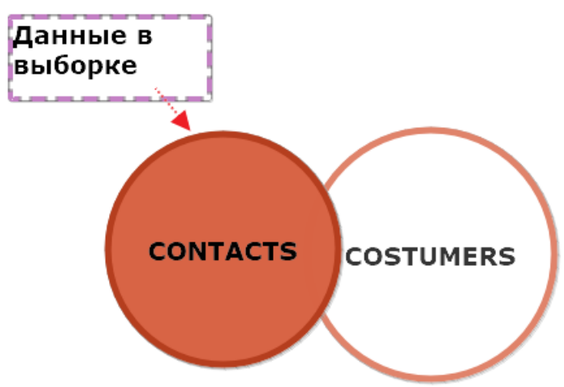
SELECT CO.CONTACT_ID, CO.NAME **AS** CONTACT_NAME,	**ВЫБРАТЬ** следующие столбцы CONTACT_ID, **принадлежащий таблице с псевдонимом** CO

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	RIGHT TABLE NULL	
2	Olivia		

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

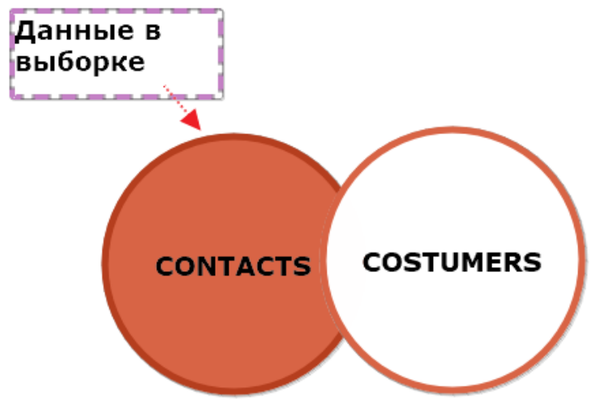


На схеме видно, что главная (левая) таблица выведется полностью и только идентичная часть правой попадет в выборку.

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	RIGHT TABLE NULL	
2	Olivia		

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**

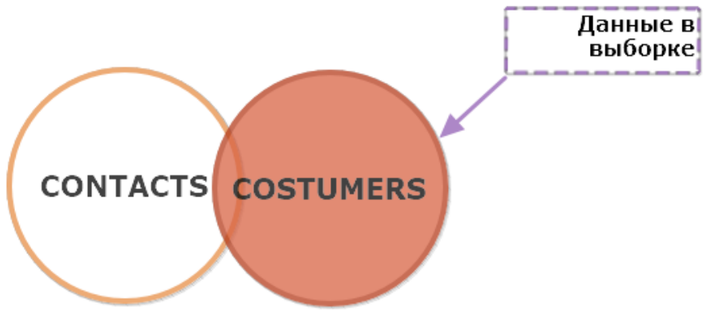
SELECT	**ВЫБРАТЬ** следующие столбцы
CO.CONTACT_ID**,**	CONTACT_ID, **принадлежащий таблице с псевдонимом** CO
CO.NAME AS CONTACT_NAME,	NAME, **принадлежащий таблице с псевдонимом** CO и назвать его в текущей выборке именем CONTACT_NAME**,**
CU.CUSTOMER_ID**,**	CUSTOMER_ID**, принадлежащий таблице с псевдонимом** CU
CU.NAME **AS** CUSTOMER_NAME	NAME, **принадлежащий таблице с псевдонимом** CU и назвать его в текущей выборке именем CUSTOMER_NAME
FROM	

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 (COSTUMERS)** 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 (COSTUMERS), 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 результирующий 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	
LEFT 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** NULL **OR** cu.name **IS** NULL**)**

CONTACT_ID	CONTACT_NAME	CUSTOMER_ID	CUSTOMER_NAME
2	Olivia	RIGHT TABLE NULL	
4	Emily		
LEFT 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,	**ВЫБРАТЬ СТОЛБЕЦ** JOB_TITLE
COUNT(JOB_TITLE) AS COUNT	**ПОСЧИТАТЬ СКОЛЬКО** В СТОЛЦЕ ЗНАЧЕНИЙ КАЖДОЙ ПРОФЕССИИ
FROM JOBS	**В ТАБЛИЦЕ** JOBS
GROUP BY JOB_TITLE;	**СГРУППИРОВАТЬ ПО** JOB_TITLE

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	COUNT
Jr. Designer	2
Jr.Software Developer	2
Sr. Designer	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

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
8002131	HT233595	2011-04-04	043XX S WABASH AVE	820	DOMESTIC BATTERY SIMPLE
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

$500 AND UNDER

Элементов на странице: 50 1 – 50 из 533 элементов

FORCIBLE ENTRY

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 ZORINA