## **СИНТАКСИС ОСНОВНЫХ ОПЕРАТОРОВ**

#### SELECT

Инструкция SELECT используется для выбора данных из базы данных.

Возвращаемые данные хранятся в таблице результатов, называемой результирующим набором.

| SELECT column1, column2, ... FROM table\_name; |
| --- |

#### AND, OR и NOT

Выражение WHERE может быть объединено с операторами AND, OR и NOT.

* Операторы AND и OR используются для фильтрации записей на основе более чем одного условия:
* Оператор AND отображает запись, если все условия разделены и являются истинными.
* Оператор OR отображает запись, если любое из условий, разделенных OR, является истинным.
* Оператор NOT отображает запись, если условие(я) не является истинным.

**Синтаксис AND**

SELECT column1, column2, ...

FROM table\_name

WHERE condition1 AND condition2 AND condition3 ...;

**Синтаксис OR**

SELECT column1, column2, ...

FROM table\_name

WHERE condition1 OR condition2 OR condition3 ...;

**Синтаксис NOT**

SELECT column1, column2, ...

FROM table\_name

WHERE NOT condition;

#### ORDER BY

Ключевое слово ORDER BY используется для сортировки результирующего набора в порядке возрастания или убывания.

Ключевое слово ORDER BY по умолчанию сортирует записи в порядке возрастания. Чтобы отсортировать записи в порядке убывания, используйте ключевое слово DESC.

SELECT column1, column2, ...

FROM table\_name

ORDER BY column1, column2, ... ASC|DESC;

#### INSERT INTO

Инструкция INSERT INTO используется для вставки новых записей в таблицу.

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

#### UPDATE

Инструкция UPDATE используется для изменения существующих записей в таблице.

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

#### DELETE

Инструкция DELETE используется для удаления существующих записей в таблице.

DELETE FROM table\_name WHERE condition;

#### SELECT TOP

Инструкция SELECT TOP используется для указания количества возвращаемых записей.

SELECT TOP number column\_name(s)

FROM table\_name

WHERE condition;

#### MIN() и MAX()

Функция MIN() возвращает наименьшее значение выбранного столбца.

Функция MAX() возвращает наибольшее значение выбранного столбца.

**Синтаксис MIN()**

SELECT MIN(column\_name)

FROM table\_name

WHERE condition;

**Синтаксис MAX()**

SELECT MAX(column\_name)

FROM table\_name

WHERE condition;

#### COUNT(), AVG() и SUM()

Функция COUNT() возвращает количество строк, соответствующих заданному критерию.

Функция AVG() возвращает среднее значение числового столбца.

Функция SUM() возвращает общую сумму числового столбца.

**Синтаксис COUNT()**

SELECT COUNT(column\_name)

FROM table\_name

WHERE condition;

**Синтаксис AVG()**

SELECT AVG(column\_name)

FROM table\_name

WHERE condition;

**Синтаксис SUM()**

SELECT SUM(column\_name)

FROM table\_name

WHERE condition;

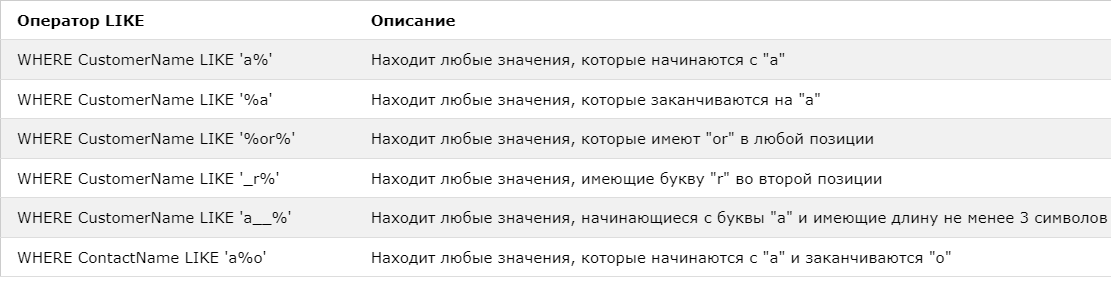
#### LIKE

Оператор LIKE используется в предложении WHERE для поиска указанного шаблона в столбце.

SELECT column1, column2, ...

FROM table\_name

WHERE columnN LIKE pattern;



#### IN

Оператор IN позволяет указать несколько значений в предложении WHERE.

**Синтаксис IN**

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (value1, value2, ...);

или:

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (SELECT STATEMENT);

#### BETWEEN

Оператор BETWEEN выбирает значения в заданном диапазоне. Эти значения могут быть числами, текстом или датами.

SELECT column\_name(s)

FROM table\_name

WHERE column\_name BETWEEN value1 AND value2;

#### JOIN

Предложение JOIN используется для объединения строк из двух или более таблиц на основе связанного столбца между ними.

Ключевое слово INNER JOIN выбирает записи, имеющие совпадающие значения в обеих таблицах.

Ключевое слово LEFT JOIN возвращает все записи из левой таблицы (table1) и соответствующие записи из правой таблицы (table2). Результат будет нулевым с правой стороны, если нет совпадения.

Ключевое слово RIGHT JOIN возвращает все записи из правой таблицы (table2) и совпадающие записи из левой таблицы (table1). Результат равен нулю с левой стороны, когда нет совпадения.

Ключевое слово FULL OUTER JOIN возвращает все записи, когда есть совпадение в записях таблицы left (table1) или right (table2).

**Синтаксис INNER JOIN**

SELECT column\_name(s)

FROM table1

INNER JOIN table2

ON table1.column\_name = table2.column\_name;

**Синтаксис LEFT JOIN**

SELECT column\_name(s)

FROM table1

LEFT JOIN table2

ON table1.column\_name = table2.column\_name;

**Синтаксис RIGHT JOIN**

SELECT column\_name(s)

FROM table1

RIGHT JOIN table2

ON table1.column\_name = table2.column\_name;

**Синтаксис FULL JOIN**

SELECT column\_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column\_name = table2.column\_name

WHERE condition;

#### UNION

Оператор UNION используется для объединения результирующего набора из двух или более заявлений SELECT.

* Каждый оператор SELECT в UNION должен иметь одинаковое количество столбцов
* Столбцы также должны иметь схожие типы данных
* Столбцы в каждом заявлении SELECT также должны располагаться в том же порядке

SELECT column\_name(s) FROM table1

UNION

SELECT column\_name(s) FROM table2;

#### GROUP BY

Заявление GROUP BY группирует строки с одинаковыми значениями в сводные строки, например "найти количество клиентов в каждой стране".

SELECT column\_name(s)

FROM table\_name

WHERE condition

GROUP BY column\_name(s)

ORDER BY column\_name(s);

#### HAVING

Инструкция HAVING была добавлена в SQL, поскольку ключевое слово WHERE не могло использоваться с агрегатными функциями.

**Пример**

В следующей инструкции SQL указано количество клиентов в каждой стране. Только включите страны с более чем 5 клиентами:

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5;

#### EXISTS

Оператор EXISTS используется для проверки наличия любой записи в подзапросе.

Оператор EXISTS возвращает true, если подзапрос возвращает одну или несколько записей.

SELECT column\_name(s)

FROM table\_name

WHERE EXISTS

(SELECT column\_name FROM table\_name WHERE condition);

#### SELECT INTO

Инструкция SELECT INTO копирует данные из одной таблицы в новую.

SELECT \*

INTO newtable [IN externaldb]

FROM oldtable

WHERE condition;

#### CASE

Инструкция CASE проходит через условия и возвращает значение, когда выполняется первое условие (например, инструкция IF-THEN-ELSE). Таким образом , как только условие истинно, оно прекратит чтение и вернет результат.

**Пример**

SELECT OrderID, Quantity,

CASE

WHEN Quantity > 30 THEN "Количество больше чем 30"

WHEN Quantity = 30 THEN "Количество составляет 30"

ELSE "Количество до 30"

END AS QuantityText --название столбца

FROM OrderDetails;

#### ALTER TABLE

Инструкция ALTER TABLE используется для добавления, удаления или изменения столбцов в существующей таблице.

--добавление столбца

ALTER TABLE table\_name

ADD column\_name datatype;

--удаление столбца

ALTER TABLE table\_name

DROP COLUMN column\_name;

--изменение столбца

ALTER TABLE table\_name

ALTER COLUMN column\_name datatype;

#### FOREIGN KEY

FOREIGN KEY - это поле (или набор полей) в одной таблице, которое ссылается на первичный ключ в другой таблице.

CREATE TABLE Orders (

OrderID int NOT NULL,

OrderNumber int NOT NULL,

PersonID int,

PRIMARY KEY (OrderID),

FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)

);

ALTER TABLE Orders

ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

#### CHECK

CHECK используется для ограничения диапазона значений, который может быть помещен в столбец.

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int CHECK (Age>=18)

);

#### CREATE INDEX

Инструкция CREATE INDEX используется для создания индексов в таблицах.

Индексы используются для более быстрого извлечения данных из базы данных, чем в других случаях.Пользователи не могут видеть индексы, они просто используются для ускорения поиска/запросов.

CREATE INDEX index\_name

ON table\_name (column1, column2, ...);

#### AUTO INCREMENT

Автоматическое приращение позволяет автоматически генерировать уникальное число при вставке новой записи в таблицу.

CREATE TABLE Persons (

Personid int IDENTITY(1,1) PRIMARY KEY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int

);

#### DATE

DATE - формат YYYY-MM-DD

DATETIME - формат: YYYY-MM-DD HH:MI:SS

SMALLDATETIME - формат: YYYY-MM-DD HH:MI:SS

TIMESTAMP - формат: уникальное число

#### CREATE VIEW

В SQL представление - это виртуальная таблица, основанная на результирующем наборе инструкции SQL.

Представление содержит строки и столбцы, как и настоящая таблица. Поля в представлении - это поля из одной или нескольких реальных таблиц в базе данных.

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

## **ЗАДАЧИ SQL**

1. **Создание таблиц и атрибутов:**
   * Создайте таблицу "Студенты" с атрибутами: идентификатор студента, имя, фамилия, дата рождения и средний балл.
   * Создайте таблицу "Курсы" с атрибутами: идентификатор курса, название, длительность и преподаватель.
   * Создайте таблицу "Результаты" с атрибутами: идентификатор результата, идентификатор студента, идентификатор курса и оценка.
2. **Выборка данных:**
   * Напишите SQL-запрос для выборки всех студентов из таблицы "Студенты".
   * Напишите SQL-запрос для выборки курсов, которые длительностью больше 6 месяцев.
   * Напишите SQL-запрос для выборки студентов, у которых средний балл выше 4.5.
3. **Создание индексов:**
   * Создайте индекс на атрибуте "Идентификатор студента" в таблице "Результаты".
   * Создайте индекс на атрибуте "Название курса" в таблице "Курсы".
   * Создайте составной индекс на атрибутах "Идентификатор студента" и "Идентификатор курса" в таблице "Результаты".
4. **Создание представлений:**
   * Создайте представление "Студенты\_с\_высоким\_баллом", которое будет отображать студентов с баллом выше 4.5.
   * Создайте представление "Информация\_о\_курсе", которое будет объединять данные из таблиц "Курсы" и "Результаты" для предоставления информации о курсе и связанных результатах.
5. **Разработка концептуальных и логических моделей:**
   * Разработайте концептуальную модель базы данных для университета, учитывая основные сущности, такие как студенты, курсы и результаты.
   * Преобразуйте концептуальную модель в логическую модель, определив таблицы, атрибуты и связи между ними.
   * Добавьте первичные и внешние ключи в логическую модель, чтобы обеспечить связи между таблицами.

## **РЕШЕНИЕ**

[SQL Файл](https://drive.google.com/file/d/1Vig7RWiod1rzgjtt8np9ouXMPyu1bjfN/view?usp=sharing)

| CREATE DATABASE DB\_EXAM  --Создайте таблицу "Студенты" с атрибутами: идентификатор студента, имя, фамилия, дата рождения и средний балл. CREATE TABLE Students (  ID\_Student INT NOT NULL PRIMARY KEY IDENTITY(1,1), --Identity служит для автоматического увеличения айди  St\_Name VARCHAR(15) NOT NULL,  St\_Surname VARCHAR(30) NOT NULL,  DateBirth DATE NOT NULL,  Avg\_Mark DECIMAL(5,1) );  CREATE TABLE Teachers (  ID\_Teacher INT NOT NULL PRIMARY KEY IDENTITY(1,1),  T\_Name VARCHAR(15) NOT NULL,  T\_Surname VARCHAR(30) NOT NULL,  DateBirth DATE NOT NULL,  EmpDate DATE NOT NULL,  WorkExp AS DATEDIFF(YEAR, EmpDate, GETDATE()) );  --Создайте таблицу "Курсы" с атрибутами: идентификатор курса, название, длительность и преподаватель. CREATE TABLE Courses (  ID\_Course INT NOT NULL PRIMARY KEY IDENTITY(1,1),  CourseName VARCHAR(50) NOT NULL,  Duration INT NOT NULL, --длительность в часах  Teacher INT NOT NULL,  FOREIGN KEY (Teacher) REFERENCES Teachers(ID\_Teacher) );  --Создайте таблицу "Результаты" с атрибутами: идентификатор результата, идентификатор студента, идентификатор курса и оценка. CREATE TABLE Results (  ID\_Result INT NOT NULL PRIMARY KEY IDENTITY(1,1),  Student INT NOT NULL,  FOREIGN KEY (Student) REFERENCES Students(ID\_Student),  Course INT NOT NULL,  FOREIGN KEY (Course) REFERENCES Courses(ID\_Course),  Mark DECIMAL(5,1) );  INSERT INTO Courses VALUES ('ultricies',77,4), ('lacinia.',75,3), ('arcu vel',55,1), ('sagittis augue,',80,3), ('arcu. Sed',63,4);  INSERT INTO Teachers VALUES ('Jason','Barlow','1994-01-23', '2022-07-27'), ('Brian','Mann','1989-03-09','2023-03-18'), ('Emma','Lindsay','1991-01-08','2022-08-01'), ('Clark','Golden','1975-12-16','2023-10-03'), ('Kevyn','Moses','1967-12-28','2022-11-10');  INSERT INTO Students VALUES ('Christopher','Robles','2023-11-25',2.1), ('Montana','Ray','2023-01-21', 4.2), ('Chandler','Farley','2022-09-11',3.5), ('Devin','Swanson','2022-12-03',4.7), ('Aimee','Levine','2023-10-09',3.8);  INSERT INTO Results VALUES (5, 3, 3.2), (7, 3, 4.7), (6, 5, 4.9), (9, 3,3.4), (8, 3,2.7);    --Напишите SQL-запрос для выборки всех студентов из таблицы "Студенты". SELECT \* FROM Students;  --Напишите SQL-запрос для выборки курсов, которые длительностью больше 6 месяцев. SELECT C.CourseName AS [Название курса] FROM Courses AS C WHERE (C.Duration/3)/4 >=6 --длительность курса хранится в академических часах, допустим, что в неделю проходит 3 ак.часа, соответственно делим на 3 => получаем количество недель  --Напишите SQL-запрос для выборки студентов, у которых средний балл выше 4.5. SELECT S.St\_Name, S.St\_Surname FROM Students AS S WHERE S.Avg\_Mark >= 4.5  --Создайте индекс на атрибуте "Идентификатор студента" в таблице "Результаты". CREATE INDEX ix\_StudentResult ON Results(Student)  --Создайте индекс на атрибуте "Название курса" в таблице "Курсы". CREATE INDEX ix\_CourseName ON Courses(CourseName)  --Создайте составной индекс на атрибутах "Идентификатор студента" и "Идентификатор курса" в таблице "Результаты". CREATE INDEX ix\_StudentCourses ON Results(Student, Course)  --Создайте представление "Студенты\_с\_высоким\_баллом", которое будет отображать студентов с баллом выше 4.5. CREATE VIEW [Студенты\_с\_высоким\_баллом] AS SELECT S.St\_Name, S.St\_Surname FROM Students AS S WHERE S.Avg\_Mark >=4.5  --Создайте представление "Информация\_о\_курсе", которое будет объединять данные из таблиц "Курсы" и "Результаты" для предоставления информации о курсе и связанных результатах. CREATE VIEW [Информация\_о\_курсе] AS SELECT C.CourseName, R.Mark FROM Courses AS C JOIN Results AS R ON C.ID\_Course = R.Course GROUP BY C.CourseName, R.Mark; |
| --- |