2- نحو و معناشناسی

2-1- فهرستی از کلمات کلیدی SQL و شرحی در مورد کاربرد آنها

- SELECT

از این عبارت برای مشخص کردن ستون هایی که می خواهید داده های ان ها را از جدول مد نظر خود بازیابی و استخراج کنید را استفاده می شود.

در این مثال ستون های FirstName و LastName انتخاب شده اند.

```
SELECT FirstName, LastName
```

- FROM

از این عبارت برای مشخص کردن جدول یا جدول هایی که میخواهید توسط دستور SELECT از انها داده هایی را بازیابی و استخراج کنید استفاده می شود.

در این مثال نام جدولی که میخواهیم داده های ستون های FirstName و LastName ان را مشاهده کنیم، تعیین کرده ایم.

```
FROM Employees
```

- WHERE

از این عبارت برای فیلتر کردن، برقراری شرط ها و محدودیت ها روی فیلدهایی که میخواهید از جدول مد نظر خود بازیابی کنید استفاده می شود. یعنی تنها داده هایی نمایش داده می شوند که شرط برای ان ها برقرار بوده است.

در این مثال برای داده های FirstName و LastName که قرار است از جدول Employees استخراج و بازیابی شوند این شرط در نظر گرفته است که حتما Department آن ها معادل با Sales باشد.

```
WHERE Department = 'Sales';
```

- INSERT

از این عبارت برای برای افزودن یک رکورد جدید به صورت سطری به جدول استفاده می شود. یعنی هنگام استفاده از این دستور نام جدول، ستون ها، و مقادیری که میخواهید برای ان ها تعریف کنید را مشخص می کنید.

در این مثال ابتدا نام جدول که معادل با Customers است نوشته شده، سپس نام ستون هایی که میخواهیم داده ی انها را تعیین و به جدول اضافه کنیم را می نویسیم سپس از عبارت VALUES استفاده میکنیم و در نهایت مقادیر مد نظر خود را برای هر یکی از ستون ها یادداشت میکنیم.

```
INSERT INTO Customers (FirstName, LastName, Email, Phone)
VALUES ('John', 'Doe', 'johndoe@example.com', '123-456-
7890');
```

- UPDATE

از این عبارت برای به روزرسانی و اصلاح رکوردهای یک جدول استفاده می شود. یعنی هنگام استفاده از این دستور نام جدول، ستون ها، و مقادیر جدیدی که میخواهید برای ان ها تعریف کنید را مشخص می کنید.

در این مثال قیمت محصول با شماره ایدی 123 به 19.99 در جدول Products تغییر کرده است.

```
UPDATE Products
SET Price = 19.99
WHERE ProductID = 123;
```

- DELETE

از این عبارت برای حذف کردن یک یا چند رکورد از یک جدول استفاده می شود.

در این مثال سطری که شماره سفارش آن معادل 456 بوده است از جدول Orders حذف شده است.

```
DELETE FROM Orders
WHERE OrderID = 456;
```

- CREATE

از این عبارت برای ایجاد اشیا در پایگاه داده استفاده می شود. برای مثال برای ساخت و ایجاد یک جدول، تابع، procedure، teigger، و...

در این مثال یک جدول جدید به نام Products ساخته شده است. هنگام ساخت یک جدول جدید باید نام و نوع ستون های آن مشخص شده باشد.

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(255) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL,
);
```

- ALTER

از این عبارت برای اصلاح و ایجاد تغییرات در یک جدول از پایگاه داده استفاده می شود. نمونه ای از این تغیرات و اصلاحات می تواند افزودن، حذف کردن و... یک سطر از جدول باشد.

در این مثال یک ستون جدید به نام Discount به جدول Products اضافه شده است. هنگام ایجاد یک ستون جدید باید نوع ان مشخص باشد که در این مثال نوع ستون Discount، Decimal می باشد طول ان معادل 5 و دارای دو رقم اعشار می باشد.

```
ALTER TABLE Products
ADD Discount DECIMAL(5, 2);
```

- DROP

از این عبارت برای حذف اشیا پایگاه داده مانند جدول، ویو ها و.. استفاده می شود. بازگرداندن این عملیات غیر ممکن است. با حذف یک جدول تمامی داده های مربوط به ان از پایگاه داده حذف خواهند شد.

در این مثال جدول ObsoleteTable حذف شده است.

```
DROP TABLE ObsoleteTable;
```

- JOIN

از این عبارت برای برقراری ارتباط بین دو یا چند جدول استفاده می شود. برقراری این عبارت می تواند مشابه با ضرب کارتزین و یا با استفاده از یک ستون مشترک بین هر دو جدولی انجام شود.

در این مثال دو جدول Orders و Customers بر اساس ستون مشترکشان یعنی CustomerID با یکدیگر JOIN شده اند و ستون های Orders.OrderID و Customers.FirstName از بین تمامی ستون های انها انتخاب و مقادیر انها نمایش داده شده است.

```
SELECT Orders.OrderID, Customers.FirstName, Customers.LastName, Orders.OrderDate, Orders.TotalAmount
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- GROUP BY

از این عبارت برای خلاصه نشان دادن نتایج خروجی استفاده می شود. از این عبارت زمانی استفاده می شود که گروهی از سطر ها دارای مقدار داده ی یکسانی می باشند. هنگام استفاده از این دستور می توان از توابعی مانند sum, count و... استفاده کرد.

در این مثال مجموع کل مقادیر ستون Revenue از جدول Sales توسط تابع Sum محاسبه شده و مطابق با Category ها گروه بندی شده اند. یعنی خروجی این قطعه کد شامل مجموع در امد کتگوری های مختلف می باشد.

```
SELECT Category, SUM(Revenue) AS TotalRevenue
FROM Sales
GROUP BY Category;
```

- HAVING

این عبارت مانند عبارت WHERE می باشد و برای فیلتر کردن و برقراری شرط ها و محدودیت ها روی داده های استخراج شده استفاده می شود. تنها تفاوت این است که عبارت HAVING همواره همراه با عبارت GROUP BY ظاهر می شود.

در این مثال مجموع کل مقادیر ستون TotalSales از جدول Sales توسط تابع Sum محاسبه شده و مطابق با Category و Month گروه بندی شده اند. یعنی خروجی این قطعه کد شامل کل فروش بر اساس کتگوری ها و ماه های مختلف می باشد با این تفاوت که خروجی تنها شامل نتایجی می باشد که مجموع کل فروش آن ها از 10000 بیشتر است.

```
SELECT Category, Month, SUM(TotalSales) AS MonthlyTotalSales
FROM Sales
GROUP BY Category, Month
HAVING SUM(TotalSales) > 10000;
```

- ORDER BY

از این عبارت برای مرتب کردن نتایج به دست آمده به صورت صعودی، نزولی یا ترکیبی از هر دوی ان ها روی یک ستون استفاده می شود.

در این مثال ستون های ProductID, ProductName و Price از جدول Products انتخاب شده و خروجی بر اساس قیمت کالا ها به صورت نزولی مرتب شده است.

```
SELECT ProductID, ProductName, Price
FROM Products
ORDER BY Price DESC;
```

- DISTINCT

از این عبارت برای عدم نمایش سطر های تکراری از یک ستون استفاده می شود.

در این مثال مقادیر ستون City از جدول Customers انتخاب و بازیابی شده اند استفاده از عبارت DISTINCT باعث می شود که اگر مشتری ها از شهر های مشترک

بودند نام هر شهر فقط یک بار نمایش داده شود یعنی در خروجی نتیجه تکراری وجود نداشته باشد.

```
SELECT DISTINCT City
FROM Customers;
```

- AS

از این عبارت برای ساخت نام مستعار برای ستون و جدول ها استفاده می شود. استفاده از این دستور باعث افزایش خوانایی و نتایج بهتر در هنگام جست و جو خواهد شد.
در این مثال زمانی که خروجی به کاربر نمایش داده می شود نام ستونی که مقدار فیلد SUM(Revenue) را معادل با TotalRevenue نمایش می دهد.

```
SELECT Category, SUM(Revenue) AS TotalRevenue
FROM Sales
GROUP BY Category;
```

2-2-یک گرامر برای زیرمجموعه ای از زبان SQL

گرامر مجموعه ای از قوانین است که نحو و ساختار یک زبان را تعریف می کند. زیربرنامه یک بلوک کد نامگذاری شده است که می تواند توسط قسمت های دیگر برنامه اجرا شود.

SQL زبانی برای پرس و جو و دستکاری داده ها در پایگاه داده های رابطه ای است. یک زیربرنامه از SQL می تواند یک رویه ذخیره شده یا یک تابع باشد. رویه ذخیره شده زیربرنامه ای است که وظیفه خاصی را انجام می دهد و می تواند مقدار صفر یا بیشتر را برگرداند. تابع یک زیربرنامه است که یک مقدار واحد را برمی گرداند و می تواند در عبارات SQL استفاده شود .

یکی از راه‌های ممکن برای نوشتن گرامر برای یک زیر برنامه از SQL به صورت زیر است:

```
Subprogram ::= procedure | function
procedure ::= CREATE PROCEDURE name (parameters) AS BEGIN statements END
function ::= CREATE FUNCTION name (parameters) RETURNS type AS BEGIN RETURN
expression END
parameters ::= parameter | parameter, parameters
parameter ::= name type
statements ::= statement | statement; statements
statement ::= assignment | control | query | call
assignment ::= name := expression
control ::= IF condition THEN statements ELSE statements END IF | WHILE
condition LOOP statements END LOOP
query ::= SELECT columns FROM tables WHERE condition
call ::= name (arguments)
columns ::= column | column, columns
column ::= name | name AS alias
tables ::= table | table, tables
table ::= name | name AS alias
condition ::= expression comparison expression | condition AND condition |
condition OR condition | NOT condition
comparison ::= = | <> | < | > | <= | >=
expression ::= term | term + term | term - term | term * term | term / term
| term % term | (expression) | name | literal | function (arguments)
term ::= name | literal | function (arguments)
arguments ::= argument | argument, arguments
argument ::= expression
type ::= INT | FLOAT | CHAR | VARCHAR | DATE | BOOLEAN
name ::= identifier
alias ::= identifier
identifier ::= letter | letter identifier
letter ::= A | B | ... | Z | a | b | ... | z
literal ::= number | string | date | boolean
number ::= digit | digit number
digit ::= 0 | 1 | ... | 9
string ::= 'character' | 'character string'
character ::= any printable ASCII character
date ::= 'YYYY-MM-DD'
boolean ::= TRUE | FALSE
```

از این دستور زبان می توان برای ایجاد زیربرنامه های SQL استفاده کرد که از نحو و ساختار تعریف شده توسط قوانین پیروی می کنند. به عنوان مثال، زیر یک زیربرنامه معتبر از SQL است که از این دستور زبان استفاده می کند:

```
CREATE FUNCTION average_salary (dept_id INT) RETURNS FLOAT AS
BEGIN
  RETURN (SELECT AVG(salary) FROM employees WHERE department_id = dept_id);
END
```

2-3- توضیحات گرامر:

- خط اول قانون زیربرنامه را تعریف می کند که می گوید یک زیربرنامه می تواند یک رویه یا یک تابع باشد.

- خط دوم قانون رویه را تعریف می‌کند که می‌گوید یک رویه با کلمه کلیدی CREATE PROCEDURE شروع می‌شود و به دنبال آن یک نام، لیستی از پارامترهای داخل پرانتز، کلمه کلیدی AS، کلمه کلیدی BEGIN، دنباله ای از عبارات و کلمه کلیدی END قرار می گیرد. .

- خط سوم قانون تابع را تعریف می کند که می گوید یک تابع با کلمه کلیدی CREATE FUNCTION شروع می شود و پس از آن یک نام، لیستی از پارامترهای داخل پرانتز، کلمه کلیدی RETURNS، یک نوع، کلمه کلیدی AS، کلمه کلیدی BEGIN، کلمه کلیدی RETURN، یک عبارت و کلمه کلیدی END .

- خط چهارم قانون پارامترها را تعریف می کند که می گوید یک لیست از پارامترها می تواند یک پارامتر باشد یا یک پارامتر به دنبال یک کاما و یک لیست دیگر از پارامترها.

- خط پنجم قانون پارامتر را تعریف می کند که می گوید یک پارامتر از یک نام و یک نوع تشکیل شده است.

- خط ششم قاعده عبارات را تعریف می کند که می گوید دنباله ای از دستورات می تواند یک دستور منفرد یا یک دستور به دنبال آن یک نقطه ویرگول و دنباله ای دیگر از دستورات باشد.

- خط هفتم قانون دستور را تعریف می کند که می گوید یک دستور می تواند یک انتساب، یک کنترل، یک پرس و جو یا یک فراخوانی باشد.

- خط هشتم قانون انتساب را تعریف می کند که می گوید یک انتساب از یک نام تشکیل شده است که توسط عملگر انتساب := و یک عبارت دنبال می شود.

- خط نهم قانون کنترل را تعریف می کند که می گوید یک دستور کنترل می تواند یک دستور IF یا یک دستور WHILE باشد.

- سطر دهم عبارت IF را تعریف می کند که می گوید یک دستور IF با کلمه کلیدی IF شروع می شود و پس از آن یک شرط، کلمه کلیدی THEN، دنباله ای از عبارات، کلمه کلیدی ELSE، دنباله ای دیگر از عبارات و کلمه کلیدی END IF قرار می گیرد.

- خط یازدهم عبارت WHILE را تعریف می کند که می گوید یک دستور WHILE با کلمه کلیدی WHILE شروع می شود و به دنبال آن یک شرط، کلمه کلیدی LOOP، دنباله ای از عبارات و کلمه کلیدی END LOOP قرار می گیرد.

- خط دوازدهم قانون query را تعریف می کند که می گوید یک query از کلمه کلیدی SELECT، لیستی از ستون ها، کلمه کلیدی FROM، لیست جداول، کلمه کلیدی WHERE و یک شرط تشکیل شده است.

- خط سیزدهم قانون فراخوانی (Call) را تعریف می‌کند که می‌گوید فراخوانی از یک نام تشکیل شده و به دنبال آن فهرستی از آرگومان‌های داخل پرانتز قرار می‌گیرد.

- خط چهاردهم قانون ستون‌ها(columns) را تعریف می‌کند که می‌گوید فهرست ستون‌ها می‌تواند تک ستونی یا ستونی باشد که با کاما و لیست دیگری از ستون‌ها همراه باشد.

- خط پانزدهم قانون هر ستون را تعریف می کند، که می گوید یک ستون می تواند یک نام باشد یا یک نام به دنبال آن کلمه کلیدی AS و نام مستعار.

- خط شانزدهم قانون جداول را تعریف می کند، که می گوید لیست جداول می تواند یک جدول باشد یا یک جدول به دنبال یک کاما و لیست دیگری از جداول.

- خط هفدهم قانون هر جدول را تعریف می کند که می گوید یک جدول می تواند یک نام باشد یا یک نام به دنبال آن کلمه کلیدی AS و نام مستعار.

- خط هجدهم قاعده شرط (condition) را تعریف می کند که می گوید شرط می تواند یا عبارتی باشد که بعد از آن یک عملگر مقایسه و یک عبارت دیگر قرار می گیرد یا یک شرط بعد از یک عملگر منطقی (AND, OR, NOT) و شرط دیگری.

- خط نوزدهم قانون مقایسه (comparison) را تعریف می کند که می گوید عملگر مقایسه می تواند یکی از نمادهای زیر باشد: = یا <> یا < یا > یا <= یا >= .

- خط بیستم قانون عبارت (expression) را تعریف می کند، که می گوید یک عبارت می‌تواند یک جمله باشد یا یک اصطلاح به دنبال یک عملگر حسابی (z+ ، - ، * ، / ، ٪) و یک عبارت دیگر، یا یک عبارت محصور در پرانتز، یا یک نام، یا یک کلمه، یا یک تابع به دنبال فهرستی از آرگومان ها در پرانتز.

- خط بیست و یکم اصطلاح قانون (term) را تعریف می کند که می گوید یک اصطلاح می‌تواند یک نام باشد یا یک کلمه یا یک تابع و به دنبال آن فهرستی از آرگومان های داخل پرانتز.

- خط بیست و دوم قانون آرگومان ها را تعریف می کند، که می گوید لیستی از آرگومان ها می تواند یک آرگومان واحد باشد یا یک آرگومان به دنبال یک کاما و یک لیست دیگر از آرگومان ها.

- خط بیست و سوم قانون هر آرگومان را تعریف می‌کند که می‌گوید آرگومان یک عبارت است.

- خط بیست و چهارم قانون نوع (type) را تعریف می کند که می گوید یک نوع می تواند یکی از کلیدواژه های زیر باشد:

INT ،FLOAT ،CHAR ،VARCHAR ،DATE ،BOOLEAN.

- خط بیست و پنجم قانون نام را تعریف می کند که می گوید یک نام یک شناسه (identifier) است.

- خط بیست و ششم قانون مستعار (alias) را تعریف می کند که می‌گوید نام مستعار یک شناسه است.

- خط بیست و هفتم قاعده هر شناسه (identifier) را تعریف می کند که می گوید یک شناسه می تواند یک حرف باشد یا یک حرف و به دنبال آن یک شناسه دیگر.

- خط بیست و هشتم قانون حرف (letter) را تعریف می کند که می گوید یک حرف می تواند هر حرف بزرگ یا کوچکی از A تا Z باشد.

- خط بیست و نهم قاعده لفظی(literal) را تعریف می‌کند که می‌گوید لفظ می تواند عدد باشد یا رشته یا تاریخ یا بولی.

- خط سی ام قاعده اعداد (number) را تعریف می کند که می گوید یک عدد می‌تواند یک رقم باشد یا یک رقم به دنبال آن یک عدد دیگر.

- خط سی و یکم قانون رقمی (digit) را تعریف می کند که می گوید یک رقم می تواند هر رقمی از 0 تا 9 باشد.

- خط سی و دوم قانون رشته (String) را تعریف می کند، که می گوید یک رشته می تواند یک کاراکتر منفرد باشد که در گیومه های تکی محصور شده است یا یک کاراکتر به دنبال رشته دیگری.

- خط سی و سوم قانون کاراکتر (character) را تعریف می کند که می گوید یک کاراکتر می تواند هر یک از کاراکترهای جدول ASCII باشد.

- خط سی و چهارم قانون تاریخ (date) را تعریف می کند که می گوید تاریخ یک سال چهار رقمی، یک ماه دو رقمی و یک روز دو رقمی است که با خط فاصله از هم جدا شده و در گیومه های تکی قرار می گیرد.

- خط سی و پنجم قانون بولی (Boolean) را تعریف می کند که می‌گوید یک بولی می‌تواند کلمه کلیدی TRUE یا کلمه کلیدی FALSE باشد.

2-4- نوشتن برنامه‌ای در SQL:

یک برنامه در زبان SQL که شامل کلمات کلیدی آن است به شرح زیر است:

```
-- Create a database called Books
CREATE DATABASE Books;

 -- Use the Books database
 USE Books;

 -- Create a table called Authors with four columns: id, name, country, and
birth_year
 CREATE TABLE Authors ( id INT PRIMARY KEY, name VARCHAR(50) NOT NULL,
country VARCHAR(50), birth_year INT );

 -- Insert some data into the Authors table
 INSERT INTO Authors (id, name, country, birth_year)
```

```sql
 VALUES (1, 'George Orwell', 'UK', 1903), (2, 'Maya Angelou', 'USA', 1928),
(3, 'Yuval Noah Harari', 'Israel', 1976),
 (4, 'J.K. Rowling', 'UK', 1965), (5, 'Rupi Kaur', 'Canada', 1992);

 -- Create a table called Books with five columns: id, title, author_id,
price, and rating
 CREATE TABLE Books ( id INT PRIMARY KEY, title VARCHAR(100) NOT NULL,
author_id INT NOT NULL, price DECIMAL(5,2), rating INT CHECK (rating BETWEEN
1 AND 5),
 FOREIGN KEY (author_id) REFERENCES Authors (id) );

 -- Insert some data into the Books table
 INSERT INTO Books (id, title, author_id, price, rating)
 VALUES (1, '1984', 1, 9.99, 5),
 (2, 'Animal Farm', 1, 7.99, 4), (3, 'I Know Why the Caged Bird Sings', 2,
8.99, 5),
 (4, 'Sapiens: A Brief History of Humankind', 3, 12.99, 5), (5, 'Harry Potter
and the Philosopher''s Stone', 4, 6.99, 5),
 (6, 'Milk and Honey', 5, 9.99, 4);

  -- Alter the Authors table to add a column called genre
 ALTER TABLE Authors ADD genre VARCHAR(50) CHECK (genre IN ('Fiction', 'Non-
fiction', 'Poetry'));

 -- Select the name and country of the authors who write fiction
 SELECT name, country FROM Authors as au WHERE au.genre = 'fiction';

 -- Select the title and price of the books that have a rating of 5
 SELECT title, price FROM Books WHERE rating = 5;

 -- Select the name and title of the authors and their books using a join
 SELECT Authors.name, Books.title FROM Authors INNER JOIN Books ON
Authors.id = Books.author_id;

 -- Update the price of the book '1984' to 10.99
 UPDATE Books SET price = 10.99 WHERE title = '1984';

 -- Delete the book 'Milk and Honey' from the Books table
 DELETE FROM Books WHERE title = 'Milk and Honey';

 -- Alter the Authors table to drop the column birth_year
 ALTER TABLE Authors DROP COLUMN birth_year;

 -- Drop the Books table
 DROP TABLE Books;

 -- Drop the Books database
 DROP DATABASE Books;

 -- Added code starts here
 -- Create a database called Movies
 CREATE DATABASE Movies;

 -- Use the Movies database
```

```sql
USE Movies;

-- Create a table called Actors with three columns: id, name, and gender
CREATE TABLE Actors ( id INT PRIMARY KEY, name VARCHAR(50) NOT NULL, gender
VARCHAR(10) CHECK (gender IN ('Male', 'Female', 'Other')) );

-- Insert some data into the Actors table
INSERT INTO Actors (id, name, gender) VALUES (1, 'Tom Hanks', 'Male'), (2,
'Meryl Streep', 'Female'),
(3, 'Will Smith', 'Male'), (4, 'Emma Watson', 'Female'), (5, 'Elliot Page',
'Other');

-- Create a table called Movies with four columns: id, title, year, and
genre
CREATE TABLE Movies ( id INT PRIMARY KEY, title VARCHAR(100) NOT NULL, year
INT, genre VARCHAR(50) );

-- Insert some data into the Movies table
INSERT INTO Movies (id, title, year, genre)
VALUES (1, 'Forrest Gump', 1994, 'Drama'), (2, 'The Devil Wears Prada',
2006, 'Comedy'),
(3, 'Men in Black', 1997, 'Sci-Fi'), (4, 'Beauty and the Beast', 2017,
'Fantasy'), (5, 'Inception', 2010, 'Thriller');

-- Create a table called Casts with three columns: movie_id, actor_id, and
role
CREATE TABLE Casts ( movie_id INT, actor_id INT, role VARCHAR(50), PRIMARY
KEY (movie_id, actor_id),
FOREIGN KEY (movie_id) REFERENCES Movies (id), FOREIGN KEY (actor_id)
REFERENCES Actors (id) );

-- Insert some data into the Casts table
INSERT INTO Casts (movie_id, actor_id, role) VALUES (1, 1, 'Forrest Gump'),
(2, 2, 'Miranda Priestly'),
(3, 3, 'Agent J'), (4, 4, 'Belle'), (5, 5, 'Ariadne');

-- Select the title and genre of the movies that were released after 2000
SELECT title, genre FROM Movies WHERE year > 2000;

-- Select the name and gender of the actors who played in 'Men in Black'
SELECT Actors.name, Actors.gender

FROM Actors INNER JOIN Casts ON Actors.id = Casts.actor_id WHERE
Casts.movie_id = (SELECT id FROM Movies WHERE title = 'Men in Black');

-- Select the title and role of the movies that Tom Hanks played in
SELECT Movies.title, Casts.role FROM Movies
INNER JOIN Casts ON Movies.id = Casts.movie_id WHERE Casts.actor_id =
(SELECT id FROM Actors WHERE name = 'Tom Hanks');

-- Select the distinct genres of the movies in the Movies table
SELECT DISTINCT genre FROM Movies;
```

```
 -- Select the name and count of the movies that each actor played in,
grouped by actor name
 SELECT Actors.name, COUNT(Casts.movie_id) AS movie_count FROM Actors
 LEFT JOIN Casts ON Actors.id = Casts.actor_id GROUP BY Actors.name;

 -- Select the name and average rating of the movies that each actor played
in, grouped by actor name and ordered by rating in descending order
 SELECT Actors.name, AVG(Movies.rating) AS avg_rating FROM Actors
 LEFT JOIN Casts ON Actors.id = Casts.actor_id LEFT JOIN Movies ON
Casts.movie_id = Movies.id GROUP BY Actors.name ORDER BY avg_rating DESC;

 -- Select the name and gender of the actors who played in more than one
movie, having a movie count greater than 1
 SELECT Actors.name, Actors.gender FROM Actors LEFT JOIN Casts ON Actors.id
= Casts.actor_id
 GROUP BY Actors.name HAVING COUNT(Casts.movie_id) > 1;

 -- Added code ends here
```

2-5- رسم درخت تجزیه:

2-6- تقدم عملگرها و همچنین وابستگی عملگرها در زبان sql :

تقدم عملگرها و وابستگی عملگرها در زبان SQL مفاهیم مهمی هستند که برای نوشتن پرس و جوهای صحیح و بهینه لازم است بدانیم. تقدم عملگرها به این معناست که در صورت وجود چند عملگر مختلف در یک عبارت، کدام عملگر اولویت بالاتری دارد و زودتر اجرا می شود. وابستگی عملگرها به این معناست که در صورت وجود چند عملگر یکسان در یک عبارت، کدام عملگر از سمت چپ یا راست شروع به اجرا می شود. برای مثال، در عبارت زیر:

```
SELECT * FROM student WHERE id = 1 + 2 * 3
```

عملگر * اولویت بالاتری از عملگر + دارد و ابتدا اجرا می شود. بنابراین، عبارت بالا معادل با عبارت زیر است:

```
SELECT * FROM student WHERE id = 1 + 6
```

همچنین، در عبارت زیر:

```
SELECT * FROM student ORDER BY name DESC, age ASC
```

عملگر ORDER BY وابسته به چپ است و ابتدا از سمت چپ به راست اجرا می شود. بنابراین، عبارت بالا معادل با عبارت زیر است:

```
SELECT * FROM (SELECT * FROM student ORDER BY name DESC) ORDER BY age ASC
```

برای بیان تقدم و وابستگی عملگرها در زبان SQL، می توان از جدول زیر استفاده کرد. این جدول بر اساس استاندارد SQL-92 تهیه شده است و ممکن است در برخی از سیستم های مدیریت پایگاه داده متفاوت باشد. در این جدول، عملگرهایی که در یک سطر قرار دارند، هم تقدم دارند و عملگرهایی که در سطرهای بالاتر قرار دارند، اولویت بالاتری دارند. همچنین، در هر سطر، عملگرهایی که وابسته به چپ هستند، قبل از عملگرهایی که وابسته به راست هستند، نوشته شده اند.

وابستگی	عملگر	تقدم
وابسته نیست	()	1
وابسته به چپ	/ *	2
وابسته به چپ	- +	3
وابسته به چپ	= <> < > <= =>	4
وابسته به راست	NOT	5
وابسته به چپ	AND	6
وابسته به چپ	OR	7
وابسته به چپ	BETWEEN IN LIKE IS NULL	8
وابسته به چپ	ALL ANY SOME EXISTS UNIQUE	9
وابسته به راست	SELECT	10
وابسته به چپ	AS	11
وابسته به راست	FROM	12
وابسته به راست	WHERE	13
وابسته به چپ	GROUP BY	14
وابسته به راست	HAVING	15
وابسته به چپ	ORDER BY	16
وابسته به چپ	UNION EXCEPT INTERSECT	17

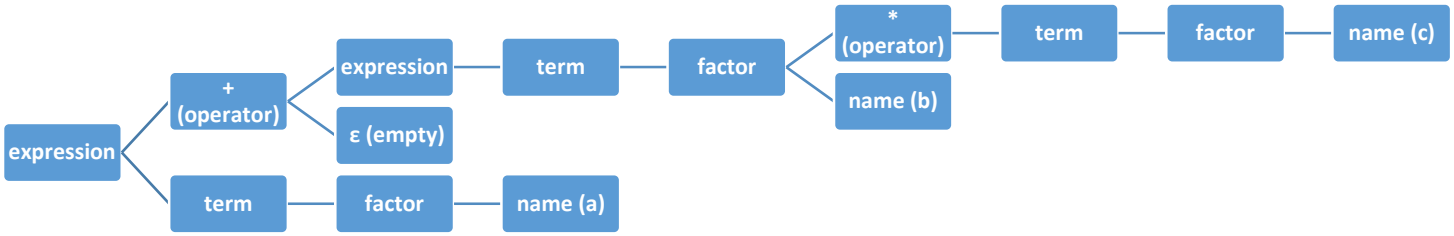
2-7- نحوهٔ توصیف گرامر برای پیروی از تقدم های ذکر شده:

```
ubprogram ::= procedure | function
procedure ::= CREATE PROCEDURE name (parameters) AS BEGIN statements END
function ::= CREATE FUNCTION name (parameters) RETURNS type AS BEGIN RETURN
expression END
parameters ::= parameter | parameter, parameters
parameter ::= name type
statements ::= statement | statement; statements
statement ::= assignment | control | query | call
assignment ::= name := expression
control ::= IF condition THEN statements ELSE statements END IF | WHILE
condition LOOP statements END LOOP
query ::= SELECT columns FROM tables WHERE condition
call ::= name (arguments)
columns ::= column | column, columns
column ::= name | name AS alias
tables ::= table | table, tables
table ::= name | name AS alias
condition ::= expression comparison expression | condition AND condition |
condition OR condition | NOT condition
comparison ::= = | <> | < | > | <= | >=

expression ::= term | term + expression | term - expression
term ::= factor | factor * term | factor / term
factor ::= (expression) | name | literal | function (arguments)

arguments ::= argument | argument, arguments
argument ::= expression
type ::= INT | FLOAT | CHAR | VARCHAR | DATE | BOOLEAN
name ::= identifier
alias ::= identifier
identifier ::= letter | letter identifier
letter ::= A | B | ... | Z | a | b | ... | z
literal ::= number | string | date | boolean
number ::= digit | digit number
digit ::= 0 | 1 | ... | 9
string ::= 'character' | 'character string'
character ::= any printable ASCII character
date ::= 'YYYY-MM-DD'
boolean ::= TRUE | FALSE
CREATE FUNCTION average_salary (dept_id INT) RETURNS FLOAT AS
BEGIN
  RETURN (SELECT AVG(salary) FROM employees WHERE department_id = dept_id);
END
```

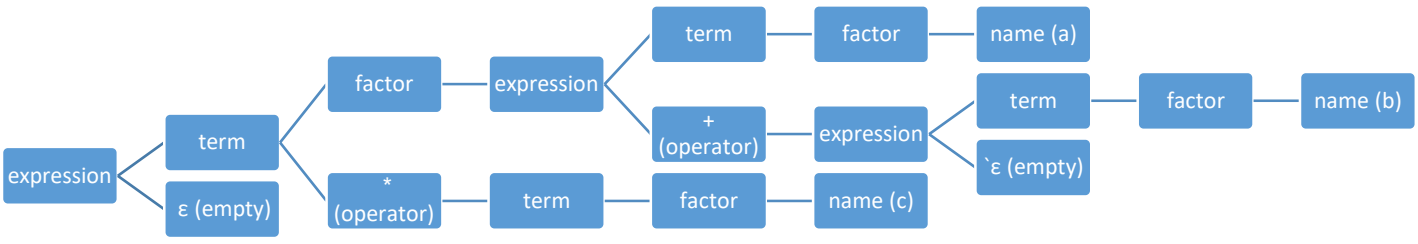
قسمتی که با آبی هایلایت شده برای نشان دادن تقدن عملگرها بجای دستورات زیر قرار گرفته است:

```
expression ::= term | term + term | term - term | term * term | term / term
| term % term | (expression) | name | literal | function (arguments)
term ::= name | literal | function (arguments)
```

تصحیح و جایگذاری انجام شده در توصیف گرامر برای پیروی از تقدم‌های مختلف، شامل ساخت یک گرامر بدون ابهام برای زبان است. این روش به این صورت عمل می‌کند که با استفاده از توسعه یا تغییر گرامر، تقدم و هم‌سطحی عملگرها را در درخت‌های نحوی خود منعکس می‌کند. به عبارت دیگر، این روش با ایجاد سطوح مختلف برای عبارات، اولویت ارزیابی آن‌ها را مشخص می‌کند. برای مثال، در این گرامر عبارت a + b * c به شکل زیر درخت نحوی می‌سازد:



این درخت نشان می‌دهد که عبارت b * c اول ارزیابی می‌شود و سپس نتیجه‌ی آن با a جمع می‌شود. به همین ترتیب، عبارت (a + b) * c به شکل زیر درخت اشتقاق می‌سازد:



این درخت نشان می‌دهد که عبارت a + b اول ارزیابی می‌شود و سپس نتیجه‌ی آن با c ضرب می‌شود. این روش باعث می‌شود که هر عبارت فقط یک درخت اشتقاق داشته باشد و ابهامی در تفسیر آن وجود نداشته باشد.

8-2- توصیف تعدادی از ساختارهای SQL به یک زبان سطح پایین توسط معناشناسی عملیاتی:

معناشناسی عملیاتی یک روش از معناشناسی زبان‌های برنامه‌نویسی است که به‌صورت قاعده‌مند نحوه اجرای یک برنامه را با استفاده از گام‌های محاسباتی توصیف می‌کند. در این روش، می‌توان از زبان دیگری مانند C برای نشان دادن گام‌های محاسباتی استفاده کرد.

- ساختار SELECT:

این ساختار برای انتخاب داده‌های موردنظر از یک یا چند جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورت یک تابع که یک پارامتر ورودی به‌عنوان شرط و یک پارامتر خروجی به‌عنوان نتیجه دارد پیاده‌سازی کرد.

برای مثال، ساختار زیر را می‌توان به‌صورت زیر توصیف کرد:

```
SELECT * FROM students WHERE age > 20
```

```
// Define a struct for student records
struct student
{
    int id; char name[50];
    int age;
};

// Define a function for selecting students with age greater than 20
void select_students (
    struct student *condition,
    struct student *result) {

    // Declare a variable for the نمایه of the result array
```

```
    int index = 0;
    // Loop through the condition array
    for (int i = 0; i < sizeof(condition) / sizeof(condition[0]); i++)
    {
    // Check if the age of the current student is greater than 20
        if (condition[i].age > 20)
            { // Copy the current student to the result array
                result[index] = condition[i];
                // Increment the index of the result array
                index++;
            }
    }
}
```

- ساختار INSERT:

این ساختار برای درج داده‌های جدید به یک جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورت یک تابع که یک پارامتر ورودی به‌عنوان داده‌های جدید و یک پارامتر خروجی به‌عنوان جدول به‌روزرسانی شده دارد پیاده‌سازی کرد.

برای مثال، ساختار زیر را می‌توان به این صورت پیاده‌سازی کرد:

```
INSERT INTO students (id, name, age) VALUES (4, 'Ali',19)
```

```
// Define a struct for student records
struct student {
    int id;
    char name[50];
    int age;
};

// Define a function for inserting a new student to the table
void insert_student(struct student *new_record, struct student *table) {
    // Declare a variable for the size of the table
    int size = sizeof(table) / sizeof(table[0]);

    // Append the new record to the end of the table
    table[size] = *new_record;
}
```

- ساختار UPDATE:

این ساختار برای به‌روزرسانی داده‌های موجود در یک جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورت یک تابع که دو پارامتر ورودی به‌عنوان شرط و عمل به‌روزرسانی و یک پارامتر خروجی به‌عنوان جدول به‌روزرسانی شده دارد پیاده‌سازی کرد.

برای مثال، ساختار زیر را می‌توان به این صورت پیاده‌سازی کرد:

```
UPDATE students SET age = age + 1 WHERE id = 2
```

```
// Define a struct for student records
struct student {
    int id;
    char name[50];
    int age;
};

// Define a function for updating the age of a student with id 2
void update_student( struct student *condition, struct student *action, struct
student *table) {
    // Loop through the table
    for (int i = 0; i < sizeof(table) / sizeof(table[0]); i++)
    {
        // Check if the id of the current student matches the condition
        if (table[i].id == condition->id)

        { // Perform the action on the age of the current student
        table[i].age = table[i].age + action->age;
        }
    }
}
```

- ساختار DELETE:

این ساختار برای حذف داده‌های موردنظر از یک جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورتی نشان داد که یک پارامتر ورودی به‌عنوان شرط حذف و یک پارامتر خروجی به‌عنوان جدول به‌روزرسانی شده دارد. برای مثال، ساختار زیر را می‌توان به این صورت پیاده‌سازی کرد:

```
DELETE FROM students WHERE id = 3
```

```
  // Define a struct for student records
  struct student {
    int id;
    char name[50];
    int age;
};
// Define a function for deleting a student with id 3 from the table
void delete_student(struct student *condition, struct student *table) {
    // Declare a variable for the index of the table
    int index = 0;
    // Loop through the table
    for (int i = 0; i < sizeof(table) / sizeof(table[0]); i++) {
        // Check if the id of the current student matches the condition
        if (table[i].id == condition->id) {
            // Skip the current student and shift the remaining students to
the left
            continue;
        }
        // Copy the current student to the new index of the table
        table[index] = table[i];
        // Increment the index of the table
        index++;
    }
}
```

- ساختار ALTER:

این ساختار برای تغییر ساختار یک جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورت یک تابع نشان داد که یک پارامتر ورودی به‌عنوان عمل تغییر و یک پارامتر خروجی به‌عنوان جدول به‌روزرسانی شده دارد. برای مثال، ساختار زیر در زبان C به این صورت پیاده‌سازی می‌شود:

```
ALTER TABLE students ADD email VARCHAR(50)
```

```
// Define a struct for student records
struct student {
    int id;
    char name[50];
    int age;
};

// Define a function for adding an email column to the table
void alter_table(struct student *action, struct student *table) {
    // Declare a variable for the size of the table
    int size = sizeof(table) / sizeof(table[0]);
    // Loop through the table
    for (int i = 0; i < size; i++) {
        // Allocate memory for the new column
        table[i].email = malloc(action->email);
        // Assign a default value to the new column
        strcpy(table[i].email, "N/A");
    }
}
```

- ساختار CREATE:

این ساختار یک جدول جدید با نام و ستون‌های مشخص‌شده ایجاد می‌کند. این دستور را می‌توان با تعریف یک ساختار داده‌ای در C شبیه‌سازی کرد. برای مثال، دستور sql زیر را می‌توان به این شکل در زبان C پیاده‌سازی کرد:

```
creat table student (id int, name varchar(20), age int);
```

```
struct student {
    int id;
    char name[20];
    int age;
};
```

- ساختار JOIN:

دستور join دو یا چند جدول را بر اساس یک شرط اتصال باهم ترکیب می‌کند. این دستور را می‌توان با استفاده از حلقه‌های تودرتو در C پیاده‌سازی کرد. برای مثال، دستور sql زیر را می‌توان به شکل زیر در زبان C پیاده‌سازی کرد:

```
select * from student join course on student.id = course.student_id;
```

```
for (int i = 0; i < student_count; i++) {

    for (int j = 0; j < course_count; j++) {

        if (student[i].id == course[j].student_id) {
            printf("%d %s %d %s %d\n", student[i].id, student[i].name,
student[i].age, course[j].name, course[j].grade);
        }
    }
}
```

- ساختار GROUP BY:

دستور group by یک جدول را بر اساس یک یا چند ستون گروه‌بندی می‌کند و امکان اجرای توابع تجمیعی را بر روی هر گروه فراهم می‌کند. این دستور را می‌توان با استفاده از یک آرایه از ساختارهای داده‌ای در C شبیه‌سازی کرد. برای مثال، دستور sql زیر در زبان C به‌صورت زیر پیاده‌سازی می‌شود:

```
select name, avg(grade) from student join course on student.id = course.student_id group by name;
```

```
struct group {
    char name[20];
    int grade_sum;
    int grade_count;
};

struct group groups[student_count];

// initialize the groups array
for (int i = 0; i < student_count; i++) {
    strcpy(groups[i].name, student[i].name);
    groups[i].grade_sum = 0;
    groups[i].grade_count = 0;
}

// iterate over the joined table and update the groups array
for (int i = 0; i < student_count; i++) {

    for (int j = 0; j < course_count; j++) {
        if (student[i].id == course[j].student_id) {
            groups[i].grade_sum += course[j].grade; groups[i].grade_count++;
        }
    }
}

// print the groups array with the average grade
for (int i = 0; i < student_count; i++) {
    if (groups[i].grade_count > 0) {
        printf("%s %f\n", groups[i].name, (float)groups[i].grade_sum /
groups[i].grade_count);
    }
}
```

منابع:

- https://www.roxo.ir
- https://alotamrin.ir
- https://fa.wikipedia.org
- https://www.parlike.com

- https://join.skype.com/bot/cf0e6215-34fe-409b-9e4b-135d7f3aa13b