در زبان برنامه‌نویسی انتخاب شده انقیاد نوع و مقدار چگونه و در چه زمانی انجام می‌شود؟ آیا تعاریف متغیرها ضمنی است یا صریح و یا هر دو نوع تعریف وجود دارد؟ با ذکر مثال توضیح داده شود.

انقیاد داده در SQL فرآیند مرتبط کردن یک نشانگر متغیر در یک دستور SQL با یک متغیر در برنامه است. انقیاد داده‌ها بسته به نوع دستور SQL و درایور مورد استفاده می‌تواند به‌صورت ایستا یا پویا انجام شود. اتصال داده‌های ایستا در زمان کامپایل انجام می‌شود، درحالی‌که اتصال داده‌های پویا در زمان اجرا انجام می‌شود. اتصال داده‌ها را می‌توان به‌صورت صریح یا ضمنی انجام داد، بسته به نحو دستور SQL و درایور مورد استفاده.

در زبان SQL، انقیاد نوع و مقدار به‌صورت ضمنی وجود دارد، به این معنی که شما در تعریف ستون‌ها و جداول نوع داده‌ها را مشخص نمی‌کنید. بلکه، بر اساس مقادیری که در ستون‌ها ذخیره می‌شوند، نوع داده به‌صورت ضمنی تشخیص داده می‌شود.

به‌طور معمول، در زمان ایجاد یک جدول، شما فقط نام ستون‌ها و نوع داده‌هایی که در آن‌ها ذخیره می‌شوند (مانند عددی، رشته، تاریخ و غیره) را مشخص می‌کنید. برای مثال، در ایجاد جدول زیر:

```
CREATE TABLE Employees (
  ID INT,
  Name VARCHAR(50),
  Age INT,
  Salary DECIMAL(10, 2)
);
```

در این مثال، برای ستون‌ها نوع داده‌های مشخص شده است. ستون "ID" به‌عنوان یک عدد صحیح (INT)، ستون "Name" به‌عنوان یک رشته با طول حداکثر ۵۰ کاراکتر (VARCHAR(50))، ستون "Age" به‌عنوان یک عدد صحیح (INT)، و ستون "Salary" به‌عنوان یک عدد اعشاری با ۲ رقم اعشار (DECIMAL(10,2)) تعریف شده‌اند.

به‌طور خلاصه، در زبان SQL، تعریف نوع داده به‌صورت صریح انجام نمی‌شود و نوع داده بر اساس مقادیر ورودی تشخیص داده می‌شود، که به‌صورت ضمنی است.

در زبان SQL، تعریف نوع داده به‌صورت صریح در مواردی مانند تعریف پارامترهای تابع‌ها، متغیرهای موقت (Temporary Variables) و دستورات دیگر انجام می‌شود. در این موارد، شما باید نوع داده‌ها را به‌صورت صریح مشخص کنید.

برای مثال، در تعریف یک پروسیجر (Stored Procedure) در SQL Server، می‌توانید نوع داده پارامترها را به‌صورت صریح مشخص کنید. در مثال زیر، یک پروسیجر به نام "GetEmployeeByID" تعریف شده است که با دریافت یک شناسه کارمند، اطلاعات کارمند متناظر را برمی‌گرداند:

```
CREATE PROCEDURE GetEmployeeByID
  @EmployeeID INT
AS
BEGIN
  SELECT * FROM Employees WHERE ID = @EmployeeID;
END;
```

در این مثال، پارامتر "EmployeeID" به‌عنوان یک عدد صحیح (INT) تعریف شده است.

به‌طور خلاصه، در زبان SQL، تعریف نوع داده به‌صورت صریح در برخی موارد از جمله تعریف پارامترها و متغیرهای موقت انجام می‌شود، درحالی‌که در تعریف ستون‌ها و جداول نوع داده به‌صورت ضمنی بر اساس مقادیر ورودی تشخیص داده می‌شود.

آیا در زبان SQL متغیرهای ایستا، پویا در پشته، پویا در هیپ به‌طور صریح، پویا در هیپ به‌طور ضمنی وجود دارند؟ با مثال توضیح داده شود. برای توصیف هر یک از موارد یک قطعه کد نوشته شود. همچنین توضیح داده شود که هر یک این متغیرها در این زبان چگونه پیاده‌سازی شده‌اند . آیا می‌توانید سرعت تخصیص این متغیرها را در این زبان مقایسه کنید؟

در زبان SQL، متغیرهای محلی وجود دارند که می‌توانند یک مقدار داده‌ای از یک نوع خاص را نگه دارند. متغیرهای محلی با استفاده از دستور DECLARE ایجاد می‌شوند و با استفاده از دستور SET یا SELECT مقداردهی می‌شوند.

متغیرهای محلی در SQL می‌توانند از نوع‌های داده‌ای مختلفی مانند int، varchar، date، xml و ... باشند. متغیرهای محلی در SQL به‌صورت ایستا، پویا در پشته، پویا در هیپ به‌طور صریح یا پویا در هیپ به‌طور ضمنی تعریف می‌شوند. در ادامه به توضیح این موارد با مثال می‌پردازیم.

- متغیرهای ایستا: این متغیرها در زمان کامپایل تعریف و مقداردهی می‌شوند و طول عمر آن‌ها تا پایان بچ یا رویه‌ای که در آن تعریف شده‌اند است. این متغیرها در حافظه پشته قرار می‌گیرند و سرعت تخصیص و دسترسی به آن‌ها بالاست. برای مثال، در کد زیر یک متغیر ایستا به نام @MyCounter با نوع int تعریف و مقداردهی شده است:

```
-- Declare and initialize a static variable
DECLARE @MyCounter INT = 0;
-- Print the value of the variable
PRINT @MyCounter;
```

- متغیرهای پویا در پشته: این متغیرها در زمان اجرا تعریف و مقداردهی می‌شوند و طول عمر آن‌ها تا پایان بچ یا رویه‌ای که در آن تعریف شده‌اند است. این متغیرها نیز در حافظه پشته قرار می‌گیرند و سرعت تخصیص و دسترسی به آن‌ها بالاست. برای مثال، در کد زیر یک متغیر پویا در پشته به نام @MyName با نوع varchar تعریف و مقداردهی شده است:

```
-- Declare a dynamic variable in stack
 DECLARE @MyName VARCHAR(50);
 -- Assign a value to the variable at run time
 SET @MyName = 'Bing';
 -- Print the value of the variable
 PRINT @MyName;
```

•متغیرهای پویا در هیپ به‌طور صریح: این متغیرها در زمان اجرا تعریف و مقداردهی می‌شوند و طول عمر آن‌ها تا پایان بچ یا رویه‌ای که در آن تعریف شده‌اند است. این متغیرها در حافظه هیپ قرار می‌گیرند و سرعت تخصیص و دسترسی به آن‌ها کمتر از متغیرهای پشته است. برای مثال، در کد زیر یک متغیر پویا در هیپ به‌طور صریح به نام @MyDate با نوع date تعریف و مقداردهی شده است:

```
 -- Declare an explicit dynamic variable in heap
DECLARE @MyDate DATE;
-- Assign a value to the variable at run time
SELECT @MyDate = GETDATE();
-- Print the value of the variable
PRINT @MyDate;
```

• متغیرهای پویا در هیپ به‌طور ضمنی: این متغیرها در زمان اجرا تعریف و مقداردهی می‌شوند و طول عمر آن‌ها تا پایان دستوری که در آن تعریف شده‌اند است. این متغیرها نیز در حافظه هیپ قرار می‌گیرند و سرعت تخصیص و دسترسی به آن‌ها کمتر از متغیرهای پشته است. برای مثال، در کد زیر یک متغیر پویا در هیپ به‌طور ضمنی به نام @MyNumber با نوع int تعریف و مقداردهی شده است:

```
 -- Declare an implicit dynamic variable in heap
SELECT @MyNumber = 10;
-- Print the value of the variable
PRINT @MyNumber;
```

به‌طورکلی، سرعت تخصیص متغیرها در SQL بستگی به نوع متغیر و نحوه استفاده از آن‌ها دارد. متغیرهای ایستا به دلیل تخصیص یک‌باره سریع‌تر از متغیرهای پویا تخصیص داده می‌شوند.

آیا حوزه تعریف در این زبان ایستا است یا پویا؟ با ذکر مثال توضیح دهید.

حوزه تعریف در زبان SQL یک زیرزبان از SQL است که برای ایجاد، تغییر و حذف ساختارهای داده مانند جداول، ستون‌ها، کلیدها، اندیس‌ها و محدودیت‌ها در پایگاه داده استفاده می‌شود. این زیرزبان شامل دستوراتی مانند CREATE، ALTER و DROP می‌شود که به ترتیب برای ساخت، تغییر و حذف ساختارهای داده مورد نظر استفاده می‌شوند. حوزه تعریف در زبان SQL هم می‌تواند ایستا باشد و هم پویا. ایستا بودن به این معناست که ساختارهای داده پس از ایجاد، تغییر یا حذف نمی‌کنند و برای تغییر آن‌ها باید دستورات جدیدی اجرا شود. پویا بودن به این معناست که ساختارهای داده می‌توانند بر اساس شرایط خاصی که در دستورات تعریف شده‌اند، تغییر کنند. به عنوان مثال، می‌توان یک جدول را به گونه‌ای تعریف کرد که هر بار که یک رکورد به آن اضافه می‌شود، یک ستون جدید هم به آن اضافه شود. این یک مثال از حوزه تعریف پویا در زبان SQL است. برای این کار می‌توان از دستور زیر استفاده کرد:

```
CREATE TABLE test ( id INT PRIMARY KEY, name
VARCHAR(50));

CREATE TRIGGER add_column AFTER INSERT ON test FOR
EACH ROW
BEGIN
    DECLARE new_column VARCHAR(50);
    SET new_column = CONCAT('col', NEW.id);
    SET @sql = CONCAT('ALTER TABLE test ADD COLUMN ',
new_column, ' VARCHAR(50)');
    PREPARE stmt FROM @sql;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END;
```

این دستور یک جدول به نام test با دو ستون id و name ایجاد می‌کند. سپس یک trigger به نام add_column تعریف می‌کند که بعد از هر INSERT بر روی جدول test اجرا می‌شود. این trigger یک متغیر به نام new_column تعریف می‌کند و مقدار آن را برابر با یک رشته که شامل عبارت col و مقدار id رکورد جدید است، قرار می‌دهد. سپس یک متغیر دیگر به نام @sql تعریف می‌کند و مقدار آن را برابر با یک دستور ALTER TABLE که برای اضافه کردن یک ستون جدید با نام new_column به جدول test است، قرار می‌دهد. سپس این دستور را با استفاده از دستورات PREPARE، EXECUTE و DEALLOCATE PREPARE اجرا می‌کند. به این ترتیب، هر بار که یک رکورد جدید به جدول test اضافه می‌شود، یک ستون جدید هم با نام متناسب با id آن رکورد به جدول اضافه می‌شود.

در صورتی که زبان حوزه تعریف ایستا را پشتیبانی می کند و بخواهیم امکان استفاده از حوزه تعریف پویا به آن بیافزاییم، چه تغییراتی در زبان باید ایجاد کنیم؟

برای اضافه کردن امکان استفاده از حوزه تعریف پویا به زبان SQL، باید چندین تغییر در زبان ایجاد کنیم. برخی از این تغییرات عبارتند از:

- اضافه کردن یک کلمه کلیدی جدید به نام DYNAMIC به زبان SQL که برای تعیین نوع حوزه تعریف استفاده می‌شود. به عنوان مثال، برای ایجاد یک جدول با حوزه تعریف پویا، می‌توان از دستور زیر استفاده کرد:

```
CREATE DYNAMIC TABLE test (
id INT PRIMARY KEY,
name VARCHAR(50)
);
```

- اضافه کردن یک ساختار جدید به نام RULE به زبان SQL که برای تعریف قواعدی که برای تغییر ساختار داده بر اساس شرایط خاصی اعمال می‌شوند، استفاده می‌شود. به عنوان مثال، برای اضافه کردن یک ستون جدید به جدول test هر بار که یک رکورد جدید اضافه می‌شود، می‌توان از دستور زیر استفاده کرد:

```
CREATE RULE add_column ON test
AFTER INSERT
FOR EACH ROW
BEGIN
DECLARE new_column VARCHAR(50);
SET new_column = CONCAT('col', NEW.id);
ALTER TABLE test ADD COLUMN new_column VARCHAR(50);
END;
```

- اضافه کردن یک کلمه کلیدی جدید به نام NEW به زبان SQL که برای ارجاع به رکورد جدیدی که به جدول اضافه شده است، استفاده می‌شود. به عنوان مثال، در دستور بالا، NEW.id به معنای مقدار id رکورد جدید است.

- اضافه کردن یک کلمه کلیدی جدید به نام OLD به زبان SQL که برای ارجاع به رکورد قبلی که از جدول حذف شده است، استفاده می‌شود. به عنوان مثال، برای حذف یک ستون از جدول test هر بار که یک رکورد از آن حذف می‌شود، می‌توان از دستور زیر استفاده کرد:

```
CREATE RULE drop_column ON test
AFTER DELETE
FOR EACH ROW
BEGIN
DECLARE old_column VARCHAR(50);
SET old_column = CONCAT('col', OLD.id);
ALTER TABLE test DROP COLUMN old_column;
END;
```

اینها فقط برخی از تغییرات ممکن برای اضافه کردن امکان استفاده از حوزه تعریف پویا به زبان SQL هستند و ممکن است راه‌حل‌های دیگری هم وجود داشته باشند.

پس از تغییر زبان، قطعه کدی نوشته شود که توسط آن حوزهٔ تعریف پویا استفاده شود. همچنین اگر زبان هر دو حوزهٔ تعریف را پشتبانی می کند، هر دو مورد توصیف شوند.

برای استفاده از حوزه تعریف پویا در زبان SQL، باید از کلمه کلیدی DYNAMIC و ساختار RULE استفاده کنیم. به عنوان مثال، قطعه کد زیر یک جدول با حوزه تعریف پویا ایجاد می‌کند که هر بار که یک رکورد جدید به آن اضافه می‌شود، یک ستون جدید هم به آن اضافه می‌شود:

```
CREATE DYNAMIC TABLE test (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);
CREATE RULE add_column ON test
AFTER INSERT
FOR EACH ROW
BEGIN
    DECLARE new_column VARCHAR(50);
    SET new_column = CONCAT('col', NEW.id);
    ALTER TABLE test ADD COLUMN new_column
VARCHAR(50);
END;
```

اگر زبان SQL هر دو حوزه تعریف ایستا و پویا را پشتیبانی کند، می‌توانیم با استفاده از کلمه کلیدی STATIC یک جدول با حوزه تعریف ایستا ایجاد کنیم. به عنوان مثال، قطعه کد زیر یک جدول با حوزه تعریف ایستا ایجاد می‌کند که ساختار آن پس از ایجاد، تغییر نمی‌کند:

```
CREATE STATIC TABLE test (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);
```

بلوک ها در این زبان چگونه تعریف شده اند؟ آیا کلمات کلیدی ویژه ای برای اعمال تغییر در حوزهٔ تعریف متغیرها وجود دارند؟

بلوک ها در زبان SQL به عنوان یک واحد منطقی از دستورات تعریف شده اند که می توانند در یک تراکنش یا یک برنامه اجرا شوند. بلوک ها می توانند شامل متغیرها، ثابت ها، مقادیر پیش فرض، توابع، زیر برنامه ها، دستورات کنترل جریان و خطاها باشند. بلوک ها می توانند درون یکدیگر تودرتو شوند و محدوده متغیرها را تعیین کنند. برای شروع و پایان یک بلوک، از کلمات کلیدی BEGIN و END استفاده می شود. برای مثال، بلوک زیر یک متغیر به نام x را تعریف می کند و مقدار آن را به ۱۰ تغییر می دهد:

```
BEGIN
    DECLARE x INT DEFAULT 0;
    SET x = 10;
END
```

برای اعمال تغییر در حوزه تعریف متغیرها، می توان از کلمات کلیدی مختلفی مانند DECLARE، SET، DEFAULT، LOCAL و GLOBAL استفاده کرد.

- DECLARE:

این کلمه کلیدی برای تعریف یک متغیر در SQL استفاده می شود. برای تعریف یک متغیر، باید نام و نوع آن را مشخص کنید. مثلا:

```
DECLARE @x INT; -- Define an integer value named x
```

- SET:

این کلمه کلیدی برای اختصاص یا تغییر مقدار یک متغیر در SQL استفاده می شود. برای اختصاص یا تغییر مقدار یک متغیر، باید نام و مقدار جدید آن را مشخص کنید. مثلا:

```
SET @x = 10; -- Assign the value of 10 to variable x
```

- DEFAULT:

این کلمه کلیدی برای تعیین یک مقدار پیش فرض برای یک ستون در SQL استفاده می شود. مقدار پیش فرض برای یک ستون، آن مقداری است که در صورت عدم ورود مقدار توسط کاربر، به آن ستون اختصاص داده می شود. مثلا:

```
CREATE TABLE Users (
UserID INT NOT NULL,
UserName NVARCHAR(50) NOT NULL,
Email NVARCHAR(100) DEFAULT 'example@example.com' --
Set the default value for the email column
);
```

- LOCAL:

این کلمه کلیدی برای مشخص کردن حوزه یک متغیر در SQL استفاده می شود. یک متغیر محلی، آن متغیری است که فقط در بلوکی که تعریف شده است، قابل دسترسی است. برای تعریف یک متغیر محلی، باید قبل از نام آن یک علامت @ قرار دهید. مثلا:

```
BEGIN
DECLARE @x INT; -- Define a local variable named  x
SET @x = 10;
```

```
PRINT @x; -- Print x
END
```

- GLOBAL:

این کلمه کلیدی برای مشخص کردن حوزه یک متغیر در SQL استفاده می شود. یک متغیر جهانی، آن متغیری است که در همه بلوک های یک اتصال، قابل دسترسی است. برای تعریف یک متغیر جهانی، باید قبل از نام آن دو علامت @@ قرار دهید. مثلا:

```
DECLARE @@x INT; -- Define a global variable named  x
SET @@x = 10;
BEGIN
    PRINT @@x; --Print x
END
```

در مورد همهٔ نوع های داده ای در زبان SQL توضیح داده شود. با ذکر مثال و قطعه کد نوع های داده ای توضیح داده شوند و پیاده سازی آنها شرح داده شود. هر یک از نوع ها چه ویژگی هایی دارند و در چه مواردی استفاده میشوند؟

انتخاب نوع داده مناسب برای یک ستون، متغیر یا پارامتر، اهمیت زیادی دارد:

- نوع داده باید بتواند محدوده داده‌هایی را که قرار است در آن ذخیره شوند، به طور دقیق مشخص کند.
- نوع داده باید فضای ذخیره‌سازی مناسبی را اشغال کند.
- نوع داده باید با نیازهای برنامه‌نویسی مطابقت داشته باشد.

انواع داده در SQL به دو دسته کلی تقسیم می‌شوند:

- داده‌های پایه (Primitive):

این نوع داده‌ها، داده‌های اصلی و ساده‌ای هستند که می‌توان آنها را به طور مستقیم در یک ستون، متغیر یا پارامتر ذخیره کرد.

- داده‌های مشتق (Derived):

این نوع داده‌ها، از ترکیب چند نوع داده پایه ایجاد می‌شوند.

از این نوع داده ها برای ذخیره مقادیر، انجام محاسبات و تجزیه و تحلیل داده ها استفاده می شود.

داده‌های پایه در SQL به شرح زیر هستند:

داده‌های متنی (Character): این نوع داده‌ها، برای ذخیره متن استفاده می‌شوند.

- char(n)

رشته متنی با طول ثابت n ، طول رشته هنگام تعریف ستون یا متغیر مشخص میشود. اگر طول داده وارد شده از متغیر بیشتر باشد کاراکترهای اضافه حذف میشوند.

مثال: در این مثال یک جدول با یک ستون به نام name از نوع char ساخته شده است.

```
CREATE TABLE customers (
  id INT,
  name CHAR(10)
);

INSERT INTO customers (id, name) VALUES (1, 'John Doe');
```

- varchar(n)

رشته متنی با طول متغیر n ، طول رشته، در هنگام وارد کردن مقدار برای رشته مشخص می‌شود. اگر مقدار وارد شده برای این ستون، کوتاه‌تر از ۲۰ کاراکتر باشد، هیچ کاراکتری به آن اضافه نمی‌شود. اگر مقدار وارد شده برای این ستون، بلندتر از ۲۰ کاراکتر باشد، رشته بدون تغییر باقی می‌ماند.

مثال: در این مثال یک جدول با یک ستون به نام name از نوع varchar ساخته شده است.

```
CREATE TABLE customers (
  id INT,
  name VARCHAR(20)
);

INSERT INTO customers (id, name) VALUES (1, 'John Doe');
```

- nchar(n)

رشته متنی با طول ثابت n که از کاراکترهای Unicode استفاده می‌کند. طول رشته، در هنگام تعریف ستون یا متغیر مشخص می‌شود. اگر مقدار وارد شده برای رشته، از طول مشخص شده بیشتر باشد، رشته کوتاه می‌شود و کاراکترهای اضافی حذف می‌شوند.

مثال: در این مثال یک جدول با یک ستون به نام name از نوع nchar ساخته شده است.

```
CREATE TABLE customers (
  id INT,
  name NCHAR(10)
);

INSERT INTO customers (id, name) VALUES (1, 'John Doe');
```

- nvarchar(n)

رشته متنی با طول متغیر n که از کاراکترهای Unicode استفاده می‌کند. طول رشته، در هنگام وارد کردن مقدار برای رشته مشخص می‌شود.

مثال: در این مثال یک جدول با یک ستون به نام name از نوع nvarchar ساخته شده است.

```
CREATE TABLE customers (
  id INT,
  name NVARCHAR(20)
);

INSERT INTO customers (id, name) VALUES (1, 'John Doe');
```

داده‌های عددی (Numeric): این نوع داده‌ها، برای ذخیره اعداد استفاده می‌شوند.

- Tinyint

عدد صحیح بین ۰ تا ۲۵۵

مثال: در این مثال یک جدول با ستون age از نوع TINYINT ساخته شده است، یعنی داده های این ستون میتواند مقادیری بین ۰ تا ۲۵۵ باشد.

```
CREATE TABLE customers (
  id INT,
  age TINYINT
);

INSERT INTO customers (id, age) VALUES (1, 20);
```

- Smallint

عدد صحیح بین ۳۲۷۶۸- تا ۳۲۷۶۷

مثال: در این مثال یک جدول با ستون age از نوع SMALLINT ساخته شده است، یعنی داده های این ستون میتواند مقادیری بین ۳۲۷۶۸- تا ۳۲۷۶۷ باشد.

```
CREATE TABLE customers (
  id INT,
  age SMALLINT
);

INSERT INTO customers (id, age) VALUES (1, 20000);
```

- Int

عدد صحیح بین ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷

مثال: در این مثال یک جدول با ستون age از نوع INT ساخته شده است، یعنی داده های این ستون میتواند مقادیری بین ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ باشد.

```
CREATE TABLE customers (
  id INT,
  age INT
);

INSERT INTO customers (id, age) VALUES (1, 20000000);
```

- Bigint

عدد صحیح بین ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۸- تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۷

مثال: در این مثال یک جدول با ستون age از نوع BIGINT ساخته شده است، یعنی داده های این ستون میتواند مقادیری بین ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۸- تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۷ باشد.

```
CREATE TABLE customers (
  id INT,
  age BIGINT
);

INSERT INTO customers (id, age) VALUES (1,
20000000000000000);
```

- Float

عدد اعشاری با دقت ۶ یا ۲۴ رقم

مثال: در این مثال یک جدول با ستون price از نوع FLOAT ساخته شده است.

```
CREATE TABLE customers (
  id INT,
  price FLOAT
);

INSERT INTO customers (id, price) VALUES (1, 123.456);
```

- Real

عدد اعشاری با دقت ۷ رقم

مثال: در این مثال یک جدول با ستون price از نوع REAL ساخته شده است.

```
CREATE TABLE customers (
  id INT,
  price REAL
);

INSERT INTO customers (id, price) VALUES (1,
123.456789);
```

- decimal(p,s)

عدد اعشاری با دقت p رقم و s رقم اعشار

مثال: در این مثال یک جدول با ستون price از نوع DECIMAL ساخته شده است. بنابراین، مقدار عدد اعشاری که می‌توان برای این ستون وارد کرد، باید بین ۰ تا ۹۹۹۹۹۹۹۹۹۹,۹۹ باشد. دقت عدد اعشاری، برابر با ۲ رقم است.

```
CREATE TABLE customers (
  id INT,
  price DECIMAL(10,2)
);

INSERT INTO customers (id, price) VALUES (1, 123.45);
```

داده‌های تاریخ و زمان (Datetime): این نوع داده‌ها، برای ذخیره تاریخ و زمان استفاده می‌شوند.

- Date

تاریخ، سال به صورت چهار رقمی، ماه به صورت دو رقمی و روز نیز به صورت دو رقمی ذخیره می شود. بنابراین، مقدار تاریخ بدون زمان که می‌توان برای این ستون وارد کرد، باید مطابق با فرمت YYYY-MM-DD باشد.

مثال: در مثال زیر یک جدول با ستون birth_date از نوع date تعریف شده است.

```
CREATE TABLE customers (
  id INT,
  birth_date DATE
);

INSERT INTO customers (id, birth_date) VALUES (1,
'2023-08-02');
```

- Time

زمان، ساعت به صورت دو رقمی، دقیقه به صورت دو رقمی، ثانیه به صورت دو رقمی ذخیره می شود. بنابراین، مقدار زمان بدون تاریخ که می‌توان برای این ستون وارد کرد، باید مطابق با فرمت HH:MM:SS باشد.

مثال: در این مثال یک جدول با ستون order_time از نوع TIME تعریف شده است.

```
CREATE TABLE customers (
  id INT,
  order_time TIME
);

INSERT INTO customers (id, order_time) VALUES (1,
'12:34:56');
```

- Datetime

تاریخ و زمان، این نوع داده، از ترکیب دو نوع داده date و time تشکیل شده است.

مثال: در این مثال یک جدول با ستون last_login از نوع DATETIME تعریف شده است.

```
CREATE TABLE customers (
  id INT,
  last_login DATETIME
);

INSERT INTO customers (id, last_login) VALUES (1,
'2023-08-20 12:34:56');
```

- datetimeoffset

تاریخ و زمان با دقت میلی‌ثانیه و اختلاف ساعت با گرینویچ. نابراین، مقدار تاریخ و زمانی که می‌توان برای این ستون وارد کرد، باید بین ۰۰۰۱-۰۱-۰۱ ۰۰:۰۰:۰۰ تا ۹۹۹۹-۱۲-۳۱ ۲۳:۵۹:۵۹ باشد. دقت تاریخ و زمان، برابر با ۷ رقم است. اختلاف زمانی، در قالب +HH:mm یا -HH:mm وارد می‌شود.

مثال: در این مثال یک جدول با ستون order_date از نوع DATETIMEOFFSET تعریف شده است.

```
CREATE TABLE customers (
  id INT,
  order_date DATETIMEOFFSET
);

INSERT INTO customers (id, order_date) VALUES (1,
'2023-07-20 12:00:00 +04:00');
```

داده‌های منطقی (Boolean): این نوع داده‌ها، برای ذخیره مقادیر منطقی true یا false استفاده می‌شوند. برای true مقدار ۱ و برای false مقدار ۰ در نظر گرفته می شود.

- Bit

یک بیت

مثال: در مثال زیر یک جدول با ستون active با نوع داده BIT تعریف شده است.

```
CREATE TABLE customers (
  id INT,
  active BIT
);

INSERT INTO customers (id, active) VALUES (1, 1);
```

- Boolean

یک عدد صحیح ۰ یا ۱

مثال: در مثال زیر یک جدول با ستون is_active با نوع داده BOOLEAN تعریف شده است.

```
CREATE TABLE customers (
  id INT,
  is_active BOOLEAN
);

INSERT INTO customers (id, is_active) VALUES (1,
TRUE);
```

داده‌های مشتق در SQL به شرح زیر هستند:

- داده‌های آرایه (Array)

این نوع داده‌ها، مجموعه‌ای از داده‌های مشابه هستند که می‌توانند در یک ستون، متغیر یا پارامتر ذخیره شوند.

مثال: در مثال زیر یک جدول با ستون names با نوع داده ARRAY تعریف شده است. بنابراین، مقداری که می‌توان برای این ستون وارد کرد، باید یک آرایه از رشته‌ها باشد.

```
CREATE TABLE customers (
  id INT,
  names ARRAY(VARCHAR(255))
);

INSERT INTO customers (id, names) VALUES (1,
ARRAY['John Doe', 'Jane Doe']);
```

- داده‌های (XML)

این نوع داده‌ها، برای ذخیره داده‌های XML استفاده می‌شوند.

مثال: در مثال زیر یک جدول با ستون profile با نوع داده XML تعریف شده است. بنابراین، مقداری که می‌توان برای این ستون وارد کرد، باید یک سند XML باشد.

```
CREATE TABLE customers (
  id INT,
  profile XML
);

INSERT INTO customers (id, profile) VALUES (1,
'<profile><name>John
Doe</name><age>30</age></profile>');
```

- داده‌های USER-DEFINED TYPE

نوع داده‌ای که توسط کاربر تعریف شده است.

مثال: در مثال زیر نوع داده my_type تعریف شده است. این نوع داده، از دو فیلد id و name تشکیل شده است. سپس، ستون name با نوع داده my_type تعریف شده است. بنابراین، مقداری که می‌توان برای این ستون وارد کرد، باید یک مقدار از نوع داده my_type باشد.

```
CREATE TYPE my_type AS (
  id INT,
  name VARCHAR(255)
);

CREATE TABLE customers (
  id INT,
  name my_type
);

INSERT INTO customers (id, name) VALUES (1, my_type(1,
'John Doe'));
```

هر یک از این نوع های داده‌ای چگونه در حافظه تخصیص و چگونه پیاده سازی شده اند؟

- داده‌های متنی (Character)

داده‌های متنی، در حافظه به صورت آرایه‌ای از کاراکترها ذخیره می‌شوند. طول آرایه، برابر با طول رشته متنی است. هر کاراکتر، با یک کد عددی مشخص می‌شود. این کد عددی، معمولاً در یک جدول کد (Code Page) تعریف می‌شود.

برای مثال، اگر طول رشته متنی برابر با ۱۰ باشد، آرایه متنی، ۱۰ کاراکتر را در حافظه اشغال خواهد کرد.

- داده‌های عددی (Numeric)

داده‌های عددی، در حافظه به صورت آرایه‌ای از اعداد ذخیره می‌شوند. طول آرایه، برابر با تعداد ارقام عدد است. هر عدد، با یک کد عددی مشخص می‌شود. این کد عددی، معمولاً در یک قالب عددی (Numeric Format) تعریف می‌شود.

برای مثال، اگر عدد برابر با ۱۲۳۴۵۶ باشد، آرایه عددی، ۶ عدد را در حافظه اشغال خواهد کرد.

- داده‌های تاریخ و زمان (Datetime)

داده‌های تاریخ و زمان، در حافظه به صورت آرایه‌ای از اعداد ذخیره می‌شوند. طول آرایه، برابر با تعداد ارقام تاریخ و زمان است. هر عدد، با یک کد عددی مشخص می‌شود. این کد عددی، معمولاً در یک قالب تاریخ و زمان (Datetime Format) تعریف می‌شود.

برای مثال، اگر تاریخ و زمان برابر با ۲۰-۰۸-۲۰۲۳ ۱۲:۴۵:۰۰ باشد، آرایه تاریخ و زمان، ۱۴ عدد را در حافظه اشغال خواهد کرد. شش عدد اول آرایه، برای ذخیره تاریخ استفاده می‌شوند. هشت عدد بعدی آرایه، برای ذخیره زمان استفاده می‌شوند.

- داده‌های منطقی (Boolean)

داده‌های منطقی، در حافظه به صورت یک بیت ذخیره می‌شوند. مقدار بیت، برابر با مقدار منطقی true یا false است.

برای مثال، مقدار منطقی true با مقدار بیت ۱ و مقدار منطقی false با مقدار بیت ۰ ذخیره می‌شود.

نوع داده	اندازه در حافظه
char(n)	n بایت
varchar(n)	n بایت
nchar(n)	n بایت
nvarchar(n)	n بایت
tinyint	1 بایت
smallint	2 بایت
int	4 بایت
bigint	8 بایت
float	4 یا 8 بایت
real	4 بایت
decimal(p,s)	p + s + 2 بایت

چه عملگرهایی برای این نوع ها تعریف شده اند؟

برای انواع داده‌های پایه در SQL، عملگرهای زیر تعریف شده‌اند:

- عملگرهای مقایسه (Comparison Operators)

این عملگرها برای مقایسه دو مقدار از یک نوع داده استفاده می‌شوند. نتیجه مقایسه، یک مقدار منطقی true یا false است.

  - = مساوی
  - <> نابرابر
  - > کوچکتر از
  - => کوچکتر یا مساوی
  - < بزرگتر از
  - =< بزرگتر یا مساوی

- عملگرهای منطقی (Logical Operators)

این عملگرها برای ترکیب دو یا چند عبارت منطقی استفاده می‌شوند. نتیجه ترکیب، یک مقدار منطقی true یا false است.

  - AND و
  - OR یا
  - NOT نه

- عملگرهای تخصیص(Assignment Operators)

این عملگرها برای تخصیص یک مقدار به یک متغیر یا ستون استفاده می‌شوند.

  - = تخصیص
  - =+ جمع و تخصیص
  - =- تفریق و تخصیص
  - =* ضرب و تخصیص
  - =/ تقسیم و تخصیص

- عملگرهای دودویی (Binary Operators)

این عملگرها برای انجام عملیات ریاضی دوتایی بر روی دو مقدار استفاده می‌شوند.

  - + جمع
  - تفریق
  - ضرب
  - / تقسیم
  - % باقی‌مانده
  - ** توان

- عملگرهای یک‌تایی (Unary Operators)

این عملگرها برای انجام عملیات ریاضی یک‌تایی بر روی یک مقدار استفاده می‌شوند.

  - + مثبت
  - - منفی
  - ! معکوس منطقی

برای انواع داده‌های مشتق، عملگرهای زیر تعریف شده است:

- برای انواع داده‌های آرایه عملگرهای زیر تعریف شده است:
  - []: دسترسی به عنصر یک آرایه
  - @: طول یک آرایه

- برای انواع داده‌های XML، عملگرهای زیر تعریف شده‌اند:
  - . : دسترسی به عنصر یک سندXML
  - // : دسترسی به تمام عناصر یک سند XML که با یک الگوی خاص مطابقت دارند.

اگر لیست ها و رشته ها و آرایه های انجمنی در زبان SQL تعریف شده اند، پیاده سازی آنها چگونه است؟

در زبان SQL، لیست ها، رشته ها و آرایه های انجمنی به عنوان نوع داده های تعریف نشده وجود ندارند. با این حال، می توان آنها را با استفاده از توابع و عبارات SQL پیاده سازی کرد.

- لیست ها

لیست ها می توانند با استفاده از تابع ARRAY() پیاده سازی شوند .این تابع یک آرایه از مقادیر را ایجاد می کند .به عنوان مثال، کد زیر یک لیست از اعداد صحیح ایجاد می کند:

```
SELECT ARRAY(1, 2, 3, 4, 5);
```

- رشته ها

رشته ها می توانند با استفاده از تابع CONCAT() پیاده سازی شوند .این تابع رشته های داده را به یکدیگر متصل می کند .به عنوان مثال، کد زیر دو رشته را به یکدیگر متصل می کند:

```
SELECT CONCAT('Hello', 'World');
```

- آرایه های انجمنی

آرایه های انجمنی می توانند با استفاده از تابعMAP() پیاده سازی شوند .این تابع یک آرایه از جفت های کلید-مقدار را ایجاد می کند .به عنوان مثال، کد زیر یک آرایه انجمنی از نام و سن افراد ایجاد می کند:

```
SELECT MAP('name', 'John Doe', 'age', 30);
```

اگر اشاره گرها و متغیرهای مرجع در زبان SQL تعریف شده اند، پیاده سازی آنها چگونه است؟

در زبان SQL ، اشاره گرها و متغیرهای مرجع به عنوان نوع داده های تعریف نشده وجود ندارند. با این حال، می توان آنها را با استفاده از توابع و عبارات SQL پیاده سازی کرد.

- اشاره گرها

اشاره گرها می توانند با استفاده از تابع ROW() پیاده سازی شوند. این تابع یک ردیف از یک جدول را به عنوان یک اشاره گر بازمی گرداند. به عنوان مثال، کد زیر یک اشاره گر به ردیف اول جدول PERSON ایجاد می کند:

```
SELECT ROW(1, 'John Doe', 30) FROM PERSON;
```

- متغیرهای مرجع

متغیرهای مرجع می توانند با استفاده از تابع REF() پیاده سازی شوند. این تابع یک اشاره گر به یک متغیر را ایجاد می کند. به عنوان مثال، کد زیر یک متغیر مرجع به متغیر x ایجاد می کند:

```
DECLARE x INT;
SELECT REF(x);
```

در زبان SQL چه سازوکارهایی برای رفع مشکلات نشتی حافظه و اشاره گر معلق وجود دارد؟

در زبانSQL ، دو سازوکار اصلی برای رفع مشکلات نشتی حافظه و اشاره گر معلق وجود دارد:

- حذف متغیرها و اشاره گرهای استفاده شده

این سازوکار ساده ترین و موثرترین روش برای جلوگیری از مشکلات نشتی حافظه و اشاره گر معلق است. به طور کلی، هر متغیر یا اشاره گر که دیگر استفاده نمی شود باید بلافاصله حذف شود. این کار می تواند با استفاده از دستورDROP انجام شود.

به عنوان مثال، کد زیر یک متغیر x ایجاد می کند و سپس آن را حذف می کند:

```
DECLARE x INT;
x = 1;
DROP x;
```

- استفاده از توابع FREE() و CLOSE()

این توابع در سیستم های مدیریت پایگاه داده (DBMS) مختلف برای آزاد کردن حافظه تخصیص یافته به متغیرها و اشاره گرها استفاده می شوند.

به عنوان مثال، کد زیر یک اشاره گر p به یک ردیف از جدول PERSON ایجاد می کند و سپس از تابع FREE() برای آزاد کردن حافظه تخصیص یافته به آن استفاده می کند:

```
DECLARE p ROW;
SELECT ROW(1, 'John Doe', 30) INTO p FROM PERSON;
FREE p;
```

سایر روش ها:

- استفاده از متغیرهای محلی

متغیرهای محلی تنها در محدوده بلوک کدی که در آن تعریف شده اند قابل دسترسی هستند. بنابراین، استفاده از متغیرهای محلی به جلوگیری از نشتی حافظه کمک می کند.

- عدم استفاده از اشاره گرهای بی‌نیاز

اشاره گرها به طور خودکار آزاد نمی شوند. بنابراین، باید از اشاره گرهایی که دیگر استفاده نمی شوند استفاده نکنید.

- استفاده از ابزارهای عیب‌یابی

ابزارهای عیب‌یابی می توانند به شما کمک کنند تا مشکلات نشتی حافظه و اشاره گر معلق را شناسایی کنید.

آیا بازیافت کننده حافظه وجود دارد و در صورت وجود چگونه پیاده سازی شده است؟ در صورتی که بازیافت کنندهٔ حافظه وجود ندارد، زبان با یک زبان دیگر که بازیافت کنندهٔ حافظه دارد مقایسه شود.

بازیافت کننده حافظه یا Garbage Collector یک مکانیزمی است که به صورت خودکار حافظه‌ای را که توسط برنامه‌ها در حال استفاده نیست، آزاد می‌کند. این کار باعث می‌شود که برنامه‌نویس نیازی نداشته باشد به صورت دستی حافظه را مدیریت کند و از اتلاف و نشت حافظه جلوگیری شود.

زبان SQL به طور مستقیم از بازیافت کننده حافظه استفاده نمی‌کند، زیرا این زبان برای پرس و جو و دستکاری داده‌های موجود در پایگاه‌های داده رابطه‌ای طراحی شده است و نه برای مدیریت حافظه. SQL یک زبان دکلاراتیو است که به برنامه‌نویس اجازه می‌دهد که بگوید چه چیزی را می‌خواهد و نه چگونه آن را بدست آورد. بنابراین، جزئیات پیاده‌سازی و مدیریت حافظه به عهده سیستم مدیریت پایگاه داده است که SQL را اجرا می‌کند.

به عنوان مثال، اگر از MySQL برای اجرای SQL استفاده کنیم، می‌توانیم از دستوراتی مانند SHOW ENGINE INNODB STATUS یا SHOW ENGINE PERFORMANCE_SCHEMA STATUS برای مشاهده وضعیت حافظه و عملکرد موتور پایگاه داده استفاده کنیم. این دستورات به ما اطلاعاتی مانند حافظه مصرفی، حافظه آزاد، تعداد تراکنش‌ها و غیره را نشان می‌دهند. اما این دستورات جزئی از زبان SQL نیستند و بستگی به نوع موتور پایگاه داده دارند.

از طرف دیگر، برخی از زبان‌های برنامه‌نویسی مانند جاوا، پایتون، روبی و غیره از بازیافت کننده حافظه به صورت خودکار استفاده می‌کنند. این زبان‌ها به عنوان زبان‌های امپراتیو شناخته می‌شوند که به برنامه‌نویس اجازه می‌دهند که چگونگی انجام کار را مشخص کند. این زبان‌ها می‌توانند با استفاده از کتابخانه‌ها یا درایورهای مختلف با پایگاه‌های داده رابطه‌ای ارتباط برقرار کنند و دستورات SQL را اجرا کنند. اما در هنگام نوشتن برنامه، برنامه‌نویس نیازی ندارد که به

صورت دستی حافظه را آزاد کند، زیرا بازیافت کننده حافظه این کار را به صورت پشت پرده انجام می‌دهد.

به عنوان مثال، در زبان جاوا، می‌توانیم از کلاس `java.sql.Connection` برای برقراری ارتباط با پایگاه داده و اجرای دستورات SQL استفاده کنیم. این کلاس یک شیء را ایجاد می‌کند که حافظه‌ای را برای ذخیره اطلاعات مربوط به ارتباط اختصاص می‌دهد. اما وقتی این شیء دیگر مورد استفاده قرار نگیرد، بازیافت کننده حافظه جاوا این حافظه را آزاد می‌کند و از اتلاف و نشت حافظه جلوگیری می‌کند.

به این ترتیب، می‌توان گفت که زبان SQL و بازیافت کننده حافظه دو مفهوم متفاوت هستند که برای اهداف مختلف طراحی شده‌اند. SQL برای کار با داده‌های رابطه‌ای و بازیافت کننده حافظه برای مدیریت حافظه در زبان‌های برنامه‌نویسی استفاده می‌شود.

#دکلاراتیو(declarative) بودن یک زبان برنامه نویسی به چه معناست؟

دکلاراتیو بودن یک زبان برنامه‌نویسی به این معناست که زبان برنامه‌نویسی به برنامه‌نویس اجازه می‌دهد که بگوید چه چیزی را می‌خواهد و نه چگونه آن را بدست آورد. به عبارت دیگر، زبان برنامه‌نویسی دکلاراتیو تمرکز خود را بر روی هدف یا نتیجه قرار می‌دهد و جزئیات پیاده‌سازی یا روش را به عهده مفسر یا کامپایلر می‌گذارد. برای مثال، زبان SQL یک زبان برنامه‌نویسی دکلاراتیو است که برای کار با داده‌های رابطه‌ای طراحی شده است. در SQL، می‌توان با استفاده از دستوراتی مانند SELECT، INSERT، UPDATE و DELETE پرس و جوها و دستکاری‌های مورد نظر را بر روی داده‌ها انجام داد، بدون اینکه نیاز باشد که بگوییم چگونه این کارها را انجام دهیم. مثلاً برای انتخاب تمام رکوردهایی که در جدول `customers` وجود دارند، می‌توان از دستور زیر استفاده کرد:

```
SELECT * FROM customers;
```

این دستور به ما می‌گوید که چه چیزی را می‌خواهیم (تمام رکوردهای جدول `customers`) ولی نمی‌گوید که چگونه آن را بدست آوریم. این کار به عهده سیستم مدیریت پایگاه داده است

که SQL را اجرا می‌کند و می‌تواند از الگوریتم‌های مختلفی برای انجام این پرس و جو استفاده کند. بنابراین، SQL یک زبان برنامه‌نویسی دکلاراتیو است که تنها هدف را مشخص می‌کند و جزئیات را پنهان می‌کند.

#امپراتیو (Imperative) بودن یک زبان برنامه نویسی به چه معناست؟

برخلاف زبان‌های برنامه‌نویسی دکلاراتیو، زبان‌های برنامه‌نویسی امپراتیو به برنامه‌نویس اجازه می‌دهند که بگوید چگونه چیزی را بدست آورد. به عبارت دیگر، زبان برنامه‌نویسی امپراتیو تمرکز خود را بر روی روش یا الگوریتم قرار می‌دهد و جزئیات هدف یا نتیجه را به عهده برنامه‌نویس می‌گذارد. برای مثال، زبان C یک زبان برنامه‌نویسی امپراتیو است که برای کار با داده‌های سطح پایین و مدیریت حافظه طراحی شده است. در C، می‌توان با استفاده از دستوراتی مانند `for`، `if`، `switch` و `malloc` عملیات مورد نظر را بر روی داده‌ها انجام داد، با این شرط که بگوییم چگونه این کارها را انجام دهیم.

منابع:

- https://dev.mysql.com
- https://docs.oracle.com
- https://azaronline.com
- https://Poe.com
- https://Bing.com
- https://stackoverflow.com
- https://ostovaee.ir
- https://www.w3schools.com/sql
- https://www.roxo.ir/series
- https://blog.faradars.org
- https://www.mongard.ir
- https://sariasan.com
- https://liangroup.net
- https://sitedesign-co.com
- https://sabzdanesh.com
- https://maktabkhooneh.org/learn/d