2_1_1_ آیا زبان برنامه نویسی انتخاب شده برنامه نویسی تابعی را پشتیبانی می کند؟

بله، SQL از برنامه نویسی تابعی پشتیبانی می‌کند که باعث بهبود کارایی و خوانایی کد می‌شود همچنین باعث ساده‌سازی و کاهش خطا هم می‌شود.

SQL توابع داخلی (built-in function) که به شرح زیر هستند نیز پشتیبانی می‌کند:

- توابع ریاضیی مانندSUM ، AVG، MIN، MAX
- توابع منطقی مانندAND ، OR، NOT
- توابع متنی مانندCONCAT ، SUBSTR، LENGTH
- توابع زمان و تاریخ مانندNOW ، CURDATE، CURTIME

علاوه بر این SQL توابع کاربردی (user-defined function) که توسط کاربر پیاده سازی و تعریف می‌شوند هم پشتیبانی می‌کند، برای تعریف این توابع در SQL باید از ساختار زیر استفاده کرد:

```
FUNCTION <نام تابع>(<آرگومان ها>)
RETURNS <نوع بازگشت>
BEGIN
...
END;
```

برای استفاده از یک تابع پس از تعریف آن می‌توانید آن را در عبارت‌های SELECT، INSERT، UPDATE یا DELETE فراخوانی کنید.

مثال:

```
CREATE FUNCTION sum_column(column_name VARCHAR(255))
RETURNS INT
BEGIN
RETURN SUM(column_name);
END;
SELECT sum_column('column_name');
```

2_1_2_ در زبان sql در مورد سازوکارهایی که برای برنامه نویسی تابعی تهیه شده اند از جمله توابع لامبدا، ارسال تابع به تابع، بازگرداندن تابع از تابع، توابع نگاشت، فیلتر، کاهش و غیره. پیاده سازی این توابع چگونه انجام شده است؟

- توابع لامبدا

توابع لامبدا توابعی هستند که بدون نام تعریف می شوند. آنها می توانند برای انجام عملیات ساده بر روی داده ها استفاده شوند. برای تعریف یک تابع لامبدا در SQL، می توانید ترکیب‌های تو در تو دستور SELECT استفاده کنید:

مثال: دو SELECT تو در تو برای مقایسه مقادیر دو ستون

```
SELECT column_name
FROM table_name
WHERE column_name > (
  SELECT column_name
  FROM table_name
  WHERE condition
)
```

- ارسال تابع به تابع

ارسال تابع به تابع به شما امکان می دهد یک تابع را به عنوان آرگومان به یک تابع دیگر ارسال کنید.

مثال: تابع sum_column یک تابع داخلی را به عنوان آرگومان دریافت می کند. تابع داخلی یک تابع کاربری است که برای تبدیل یک رشته به یک عدد استفاده می شود.

```
CREATE FUNCTION sum_column(column_name VARCHAR(255),
func INT(VARCHAR(255)) RETURNS INT

BEGIN
RETURN SUM(func(column_name));
END;
```

- بازگردادندن تابع از تابع

ازگرداندن تابع از تابع به شما امکان می دهد یک تابع را از یک تابع دیگر بازگردانید. یعنی خروجی یک تابع، یک تابع باشد.

بازگرداندن تابع از تابع در SQL با استفاده از عبارت RETURN پیاده سازی می شود. عبارت RETURN به شما امکان می دهد یک مقدار را از یک تابع برگردانید.

مثال: در این قطعه کد خروجی تابع اول مجموع مقادیر می‌باشد که توسط تابع SUM محاسبه شده است.

```
CREATE FUNCTION sum_column(column_name VARCHAR(255))
RETURNS INT
BEGIN
RETURN SUM(column_name);
END;

SELECT sum_column('column_name');
```

- نگاشت

توابع نگاشت توابعی هستند که یک تابع را به هر عنصر از یک مجموعه اعمال می کنند.

توابع نگاشت در SQL با استفاده از عبارت MAP پیاده سازی می شوند. عبارت MAP به شما امکان می دهد یک تابع را به هر عنصر از یک مجموعه اعمال کنید.

مثال یک: در این قطعه کد با استفاده از تابع نگاشت مقادیر یک ستون دو برابر شده است.

```
SELECT column_name
FROM table_name
MAP(
  column_name,
  (x) => x * 2
);
```

مثال دو: در این قطعه کد تمام حروف ستون نام به تابع UPPER() فرستاده میشود.

```
SELECT map(column_name, UPPER);
```

- فیلتر

توابع فیلتر توابعی هستند که فقط عناصری از یک مجموعه را برمی گردانند که یک شرط را برآورده می کنند.

توابع فیلتر در SQL با استفاده از عبارت FILTER پیاده سازی می شوند. عبارت FILTER به شما امکان می دهد فقط عناصری از یک مجموعه را برگردانید که یک شرط را برآورده می کنند.

مثال: در این قطعه کد از تابع فیلتر برای انتخاب عناصری که بزرگ تر از 10 هستند استفاده شده است.

```
SELECT column_name
FROM table_name
FILTER(
  column_name,
  (x) => x > 10
);
```

- کاهش

توابع کاهش توابعی هستند که یک مقدار را از یک مجموعه با اعمال یک تابع به هر عنصر از مجموعه محاسبه می کنند. برای تعریف یک تابع کاهش در SQL، می‌توانید از ساختارهایی مشابه با مثال زیر استفاده کنید:

مثال: محاسبه محموع مقادیر یک ستون با پیاده سازی تابع کاهش.

```
SELECT SUM(column_name)
FROM table_name;
```

2-2-استفاده از توابع نگاشت و فیلتر و کاهش باعث افزایش کارایی برنامه می شوند؟ آیا می توانید سرعت اجرای این توابع را با برنامه نویسی رویه ای توسط حلقهٔ تکرار مقایسه کنید؟

بله، استفاده از توابع نگاشت و فیلتر و کاهش در زبان SQL می تواند باعث افزایش کارایی برنامه شود. این توابع به ما اجازه می دهند که بر روی دنباله هایی از عناصر عملیات انجام دهیم بدون اینکه نیاز به نوشتن حلقه های تکرار یا شرط های کنترل جریان داشته باشیم. این توابع همچنین می توانند به صورت موازی اجرا شوند و از منابع پردازشی بهتری استفاده کنند. برای مثال، اگر بخواهیم تعداد کاراکترهای هر خط در یک جدول را بدست آوریم، می توانیم از تابع map استفاده کنیم:

```
SELECT map(line, length(line)) FROM table;
```

این تابع برای هر خط در جدول یک جفت از خط و طول آن را برمی گرداند. اگر بخواهیم فقط خط هایی را که بیشتر از 10 کاراکتر دارند را انتخاب کنیم، می توانیم از تابع filter استفاده کنیم:

```
SELECT filter(map(line, length(line)), (k, v) -> v > 10) FROM table;
```

این تابع برای هر جفت خط و طول آن یک شرط را بررسی می کند و فقط جفت هایی را که شرط را برآورده می کنند را برمی گرداند. اگر بخواهیم مجموع طول همه خط ها را محاسبه کنیم، می‌توانیم از تابع reduce استفاده کنیم:

```
SELECT reduce(map(line, length(line)), 0, (acc, x) -> acc + x) FROM table;
```

این تابع یک مقدار اولیه (0) و یک تابع دوم (جمع) را گرفته و برای هر جفت خط و طول آن مقدار اولیه را با طول خط جمع می کند و نتیجه را برمی گرداند.

اگر بخواهیم همین کار را با برنامه نویسی رویه ای توسط حلقه تکرار انجام دهیم، باید کد بیشتری بنویسیم و ممکن است با خطاهای بیشتری روبرو شویم. برای مثال، اگر بخواهیم مجموع طول همه خط ها را محاسبه کنیم، باید چنین کدی بنویسیم:

```
DECLARE @total INT = 0;
DECLARE @line VARCHAR(100);
DECLARE @length INT;
DECLARE cursor CURSOR FOR SELECT line FROM table;
OPEN cursor;
FETCH NEXT FROM cursor INTO @line;
WHILE @@FETCH_STATUS = 0
BEGIN
 SET @length = LEN(@line);
 IF @length > 10
 BEGIN
  SET @total = @total + @length;
 END
 FETCH NEXT FROM cursor INTO @line;
END
CLOSE cursor;
DEALLOCATE cursor;
SELECT @total;
```

همانطور که می بینید، این کد خوانایی کمتری دارد و از ساختارهای پیچیده تری مانند مکانیزم cursor استفاده می کند. همچنین این کد به صورت ترتیبی اجرا می شود و از منابع پردازشی کمتری بهره می برد. بنابراین، استفاده از توابع نگاشت و فیلتر و کاهش در زبان SQL می تواند باعث افزایش کارایی برنامه شود.

مکانیزم Cursor:

مکانیزم cursor در زبان SQL یک اشیای پایگاه داده است که به ما اجازه می دهد تا بر روی سطوح نتیجه یک جستجوی کار کنیم. با استفاده از cursor، ما می توانیم هر سطر را به صورت جداگانه معامله کنیم و از منابع پردازشی بهتری استفاده کنیم. برای استفاده از cursor، ما باید چند مرحله را طی کنیم:

- اول، یک cursor را تعریف کنیم. برای این کار، ما باید نام cursor را بعد از کلمه DECLARE با نوع CURSOR و یک جمله SELECT را تعریف کنیم. جمله SELECT باید نتیجه جستجوی خود را تعریف کند.

- دوم، cursor را باز و پر دهیم. برای این کار، ما باید جمله SELECT را در حالت OPEN قرار دهیم.

- سوم، هر سطر را از cursor در یک یا چند متغیر دریافت کنیم. برای این کار، ما باید جملات FETCH NEXT FROM و INTO را در حالت WHILE قرار دهیم.

- چهارم، cursor را بسته و منصوب کنیم. برای این کار، شما باید جملات CLOSE و DEALLOCATE را در حالت WHILE قرار دهیم.

برای مثال، فرض کنید بخواهید نام و سن همه دانش آموزانی را که در کلاس 10 هستند را از یک جدول به نام students بدست آورید. برای این کار، می توانید از یک cursor به نام student_cursor استفاده کنید:

```
-- تعریف cursor
DECLARE student_cursor CURSOR FOR
SELECT name, age FROM students
WHERE class = 10;

-- باز و پر کردن cursor
OPEN student_cursor;

-- ایجاد متغیرهای محلی
DECLARE @name VARCHAR(50);
DECLARE @age INT;

-- دریافت هر سطر از cursor
FETCH NEXT FROM student_cursor INTO @name, @age;

-- حلقه تکرار برای پردازش هر سطر
WHILE @@FETCH_STATUS = 0
```

```
BEGIN
```

-- نمایش نام و سن دانش آموز

```
PRINT @name + ' is ' + CAST(@age AS VARCHAR) + ' years
old.';
```

-- دریافت سطر بعدی از cursor

```
FETCH NEXT FROM student_cursor INTO @name, @age;
END
```

-- بسته و منصوب کردن cursor

```
CLOSE student_cursor;
DEALLOCATE student_cursor;
```

این کد برای هر سطر در جدول students که در کلاس 10 هستند، نام و سن آنها را نمایش می دهد. اما اگر بخواهیم همین کار را با استفاده از تابع map انجام دهیم، می توانیم از یک جمله SQL ساده تر استفاده کنیم:

-- استفاده از تابع map

```
SELECT map(name, age) FROM students
WHERE class = 10;
```

این جمله برای هر سطر در جدول students که در کلاس 10 هستند، یک جفت از نام و سن آنها را برمی گرداند. همانطور که می بینید، استفاده از تابع map کد را خواناتر و کوتاه تر می کند.

3-2-آیا زبان SQL برنامه‌نویسی رویه‌ای را پشتیبانی می‌کند؟ در این صورت در مورد زیربرنامه‌ها، روش‌های ارسال متغیرها به توابع، برنامه‌نویسی عمومی (توابعی که نوع ورودی‌های آن‌ها عمومی است)، و غیره توضیح دهید.

SQL در درجه اول یک زبان اعلامی (declarative) است که برای مدیریت و دستکاری پایگاه‌های داده رابطه‌ای طراحی شده است. به‌طور گسترده برای کارهایی مانند ساخت کوئری، درج، به‌روز رسانی و حذف داده‌ها در پایگاه داده استفاده می‌شود. SQL یک نحو استاندارد و مجموعه‌ای از دستورات را ارائه می‌دهد که توسط اکثر سیستم‌های مدیریت پایگاه داده رابطه‌ای (RDBMS) پشتیبانی می‌شود.

اگرچه SQL به عنوان یک زبان برنامه‌نویسی رویه‌ای طبقه‌بندی نمی‌شود، برخی از سیستم‌های مدیریت پایگاه داده SQL را برای پشتیبانی از ساختارهای رویه‌ای و توانایی تعریف توابع و رویه‌های ذخیره شده گسترش داده‌اند. این پسوندها به توسعه دهندگان اجازه می‌دهد تا کد رویه‌ای را در خود سیستم پایگاه داده بنویسند.

به عنوان مثال، MySQL از ایجاد رویه‌های ذخیره شده با استفاده از زبانی به نام SQL/PSM (SQL/Persistent Stored Modules) پشتیبانی می‌کند. PostgreSQL از رویه‌های ذخیره شده با استفاده از PL/pgSQL که یک زبان رویه‌ای مشابه PL/SQL در اوراکل است، پشتیبانی می‌کند. Microsoft SQL Server از T-SQL (Transact-SQL) برای تعریف رویه‌ها و توابع ذخیره شده استفاده می‌کند.

رویه‌های ذخیره شده، بلوک‌های کد قابل استفاده مجدد هستند که در پایگاه داده ذخیره می‌شوند و می‌توانند از یک برنامه کاربردی یا سایر دستورات SQL فراخوانی شوند. آن‌ها می‌توانند پارامترها را بپذیرند، محاسبات را انجام دهند، کوئری‌های SQL را اجرا کنند و نتایج را برگردانند. رویه‌های ذخیره شده با امکان دسترسی کنترل‌شده به داده‌های زیربنایی، کپسوله‌سازی و امنیت را فراهم می‌کنند.

وقتی صحبت از برنامه‌نویسی عمومی می‌شود، SQL به تنهایی قابلیت‌های یک زبان برنامه‌نویسی همه منظوره را ندارد. در درجه اول بر روی عملیات پایگاه داده متمرکز است. برای عملیات پیچیده و وظایف برنامه‌نویسی عمومی، اغلب استفاده از یک زبان برنامه‌نویسی مانند جاوا، پایتون، سی‌شارپ یا سایر زبان‌ها مناسب‌تر است. این زبان‌های برنامه‌نویسی طیف وسیع‌تری از ویژگی‌ها، کتابخانه‌ها و ابزارها را برای توسعه اپلیکیشن ارائه می‌دهند.

به‌طور خلاصه، SQL یک زبان قدرتمند برای مدیریت و دستکاری پایگاه‌های داده است. معمولاً برای عملیات مربوط به داده استفاده می‌شود و می‌تواند با ساختارهای رویه‌ای برای پشتیبانی از توابع و رویه‌های ذخیره شده گسترش یابد. بااین‌حال، برای کارهای برنامه‌نویسی گسترده‌تر و کلی‌تر، توصیه می‌شود از یک زبان برنامه‌نویسی اختصاصی در ارتباط با SQL برای تعامل با پایگاه داده استفاده شود.

2-4-آیا زبان برنامه نویسی SQL برنامه نویسی شیء گرا را پشتیبانی می کند؟ در اینصورت در مورد ساختارهای موجود و روش های پیاده سازی اشیاء در حافظه، چندریختی، وراثت، و غیره توضیح دهید.

خیر زبان SQL به طور کامل از برنامه نویسی شیء گرا پشتیبانی نمی کند. اما در SQL، دو ساختار داده وجود دارد که می توانند برای شبیه سازی اشیاء شیء گرا استفاده شوند:

1. توابع کاربری: برای تعریف کلاس ها و نمونه های شیء استفاده شوند.
2. توابع تعریف شده توسط کاربر(UDF) : برای تعریف کلاس ها و نمونه های شیء، و همچنین برای تعریف توابع و روش های شیء استفاده شوند.

- پیاده سازی اشیا در حافظه

SQL از ساختاری به نام جدول برای ذخیره داده ها استفاده می کند. یک جدول از سطرها و ستون ها تشکیل شده است. هر سطر یک رکورد را نشان می دهد و هر ستون یک فیلد را

نشان می دهد. اشیاء شیء گرا در SQL به صورت رکوردهای جدول ذخیره می شوند. رکوردهای جدول حاوی داده هایی هستند که مربوط به یک شیء خاص هستند.

مثال: در قطعه کد زیر، ابع Person() یک کلاس Person را تعریف می کند که حاوی دو فیلد است name :و .age تابع Person.GetAge() یک روش شیء است که مقدار فیلد ageرا بازمی گرداند. تابع Person.SetAge() یک روش شیء است که مقدار فیلد age را تنظیم می کند.
در عبارتSELECT ، یک شیء Person ایجاد می شود و سپس روش GetAge() آن شیء فراخوانی می شود. این عبارت مقدار فیلد age را که برابر با 30 است بازمی گرداند.

```
CREATE FUNCTION Person(name VARCHAR(255), age INT)
RETURNS TABLE
AS
(
    SELECT
        name AS name,
        age AS age
);

CREATE FUNCTION Person.GetAge()
RETURNS INT
AS
BEGIN
    RETURN age;
END;

CREATE FUNCTION Person.SetAge(new_age INT)
AS
BEGIN
```

```
    age = new_age;
END;

SELECT
    Person('John Doe', 30).GetAge();
```

- چندریختی

چندریختی در SQL به صورت چندریختی نوع داده انجام می شود. این بدان معناست که یک تابع می تواند با انواع داده های مختلف فراخوانی شود.

مثال: قطعه کد زیر برای اعداد به صورت تابع ریاضی و برای کاراکترها به صورت کانکت عمل می‌کند، مثلا خروجی SELECT اول معادل 20 و SELECT دوم معادل HelloHello می باشد.

```
CREATE FUNCTION double(value ANY TYPE)
RETURNS ANY TYPE
BEGIN
RETURN 2 * value;
END;

SELECT double(10); --20
SELECT double('Hello'); --HelloHello
```

- وراثت

وراثت در SQL به صورت وراثت ساختار داده انجام می شود. این بدان معناست که یک جدول می تواند از جدول دیگری ارث ببرد. جدول وارث تمام ستون های جدول پایه را به علاوه ستون های اضافی خود دارد.

مثال: در قطعه کد زیر، جدول Employee از جدول Person به ارث می برد، زیرا تمام ستون های آن را دارا می باشد، با مشخص کردن کلیدهای اصلی و خارجی وراثت وضوح بیشتری خواهد داشت.

```
CREATE TABLE Person (
    name VARCHAR(255),
    age INT
);

CREATE TABLE Employee (
    name VARCHAR(255),
    age INT,
    job VARCHAR(255)
);
```

5-2-آیا زبان برنامه نویسی انتخاب شده برنامه نویسی همروند را پشتیبانی می کند؟ در اینصورت در مورد سازوکارهای موجود از جمله سمافورها، قفل ها، مکانیزم های ارسال پیام، و ریسه ها، و دیگر سازوکارهای موجود برای پشتیبانی برنامه نویسی همروند توضیح دهید.

sql به طور مستقیم برنامه‌نویسی همروند را پشتیبانی نمی‌کند، اما برخی از سیستم‌های مدیریت پایگاه داده که sql را پیاده‌سازی می‌کنند، قابلیت‌هایی را برای اجرای همروند برنامه‌های sql فراهم می‌کنند.

برنامه‌نویسی همروند به این معنی است که چندین برنامه یا فرآیند به طور همزمان یا موازی اجرا شوند. این می‌تواند منجر به افزایش کارایی و عملکرد برنامه‌ها شود، اما همچنین می‌تواند مشکلاتی را در مورد هماهنگی، همزمانی و امنیت داده‌ها ایجاد کند.

به عنوان مثال، اگر چندین برنامه sql به طور همزمان به یک پایگاه داده دسترسی داشته باشند و بخواهند داده‌های یکسان را بخوانند یا تغییر دهند، ممکن است با مشکلاتی مانند رقابت، تداخل، بلوکه شدن یا مردود شدن مواجه شوند. برای حل این مشکلات، سیستم‌های مدیریت پایگاه داده مکانیزم‌هایی را برای کنترل دسترسی همروند به داده‌ها ارائه می‌دهند. این مکانیزم‌ها عبارتند از:

- قفل‌ها: قفل‌ها ابزاری هستند که برای جلوگیری از دسترسی همزمان به یک داده یا منبع توسط چندین برنامه استفاده می‌شوند. قفل‌ها می‌توانند از نوع انحصاری یا اشتراکی باشند. قفل انحصاری به این معنی است که فقط یک برنامه می‌تواند به داده دسترسی داشته باشد و هیچ برنامه دیگری نمی‌تواند آن را بخواند یا تغییر دهد. قفل اشتراکی به این معنی است

که چندین برنامه می‌توانند به داده دسترسی داشته باشند و آن را بخوانند، اما هیچ برنامه‌ای نمی‌تواند آن را تغییر دهد.

- معاملات: معاملات واحدهای منطقی از عملیات sql هستند که باید به طور کامل اجرا شوند یا اصلا اجرا نشوند. معاملات باید چهار خاصیت ACID را داشته باشند: اتمی بودن، سازگاری، ایزولاسیون و دوام. اتمی بودن به این معنی است که یا تمام عملیات معامله انجام می‌شوند یا هیچکدام انجام نمی‌شوند. سازگاری به این معنی است که معامله باید داده‌ها را از یک حالت سازگار به حالت سازگار دیگر ببرد. ایزولاسیون به این معنی است که معامله باید از تأثیر معاملات دیگر جدا شود. دوام به این معنی است که نتایج معامله باید به طور دائمی در پایگاه داده ذخیره شوند.

- سطح ایزولاسیون معامله: سطح ایزولاسیون معامله مشخص می‌کند که چه میزان تداخل بین معاملات مجاز است. سطح‌های مختلف ایزولاسیون می‌توانند مشکلات مختلف همروندی را حل یا ایجاد کنند. مشکلات همروندی عبارتند از: خواندن ناپایدار، خواندن ناتمام، خواندن فریبنده و نوشتن فریبنده. سطح‌های ایزولاسیون معامله عبارتند از: خواندن نامتعارف، خواندن تکرار شونده، خواندن تأیید شده و سریال.

- مکانیزم‌های دیگر: برخی از سیستم‌های مدیریت پایگاه داده مکانیزم‌های دیگری را برای پشتیبانی برنامه‌نویسی همروند در sql ارائه می‌دهند. برای مثال،
Oracle E-Business Suite از APIهای پردازش همروند استفاده می‌کند که امکان اجرای همروند برنامه‌های sql را با استفاده از صف‌های پیام و سمافورها فراهم می‌کند. همچنین، برخی از سیستم‌ها از روش‌هایی مانند برچسب‌زمان، چندنسخه‌ای و اعتبارسنجی برای کنترل همروندی استفاده می‌کنند.

- ریسه: ریسه‌ها در SQL به عنوان یک مکانیزم قفل‌گذاری عمل می‌کنند که می‌توانند به فرایندها یا ریسه‌ها اجازه دهند که به بخش‌های مختلف یک منبع دسترسی پیدا کنند. برای مثال، اگر یک فرایند یا ریسه بخواهد یک ستون خاص را در یک جدول به روز رسانی کند، می‌تواند یک ریسه را بر روی آن ستون قرار دهد. این کار باعث می‌شود که هیچ فرایند یا ریسه دیگری نتواند به آن ستون دسترسی پیدا کند و تغییری در آن ایجاد کند. اما این کار مانع از دسترسی به بقیه ستون‌ها یا ردیف‌های جدول نمی‌شود. این روش می‌تواند از تداخل بین عملیات‌های مختلف جلوگیری کند و از ایجاد تراکنش‌های ناقص جلوگیری کند.

- سمافور: سمافورها در SQL به عنوان یک متغیر صحیح عمل می‌کنند که تعداد فرایندها یا ریسه‌هایی را نشان می‌دهند که می‌توانند به یک منبع مشترک دسترسی پیدا کنند. برای مثال، اگر یک سمافور مقدار ۳ داشته باشد، به این معنی است که فقط ۳ فرایند یا ریسه می‌توانند به یک جدول یا یک ستون خاص در پایگاه داده دسترسی داشته باشند. وقتی یک فرایند یا ریسه به منبع مورد نظر دسترسی پیدا می‌کند، سمافور یک واحد کاهش می‌یابد. وقتی یک فرایند یا ریسه از منبع استفاده کرده و آن را رها می‌کند، سمافور یک واحد افزایش می‌یابد. اگر سمافور مقدار صفر داشته باشد، به این معنی است که هیچ فرایند یا ریسه دیگری نمی‌تواند به منبع دسترسی پیدا کند و باید منتظر بماند تا یکی از فرایندها یا ریسه‌های فعال منبع را رها کند. این روش می‌تواند از رقابت برای دسترسی به منابع جلوگیری کند و از ایجاد داده‌های ناهماهنگ جلوگیری کند.

منابع:

- https://www.sqlshack.com
- https://dba.stackexchange.com
- https://docs.oracle.com
- https://www.sqlshack.com
- https://reintech.io

- https://ocw.mit.edu
- https://learn.microsoft.com
- https://developerhowto.com
- https://stackoverflow.com
- https://docs.databricks.com/en/sql
- https://www.sqlservertutorial.net
- https://www.sqlshack.com
- https://poe.com/