دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

درس زبان‌های برنامه‌نویسی

استاد آرش شفیعی

اعضای گروه:

ملیکا آقاجانیان صباغ

مهلا زارع

مهدیس فتحی

معرفی و بررسی زبان SQL

پاییز ۱۴۰۲

فهرست

۱ _ مقدمه

۱_۱_ تاریخچه زبان برنامه‌نویسی SQL

این زبان در دهه ۱۹۷۰ میالدی توسط Chamberlin.D Donald و Boyce.F Raymond به همراه شرکت IBM طراحی شد. هدف از ایجاد این زبان، ایجاد یک زبان استاندارد برای دسترسی و مدیریت داده‌های ذخیره شده در پایگاه‌های داده‌های رابطه‌ای بود و امروزه به‌عنوان زبان استاندارد برای دسترسی و مدیریت داده‌های پایگاه‌های داده‌های رابطه‌ای استفاده می‌شود.

تاریخچه زبان برنامه‌نویسی SQL را می‌توان به سه دوره تقسیم کرد:

- دوره اولیه (۱۹۸۶-۱۹۷۰)

در این دوره، SQL به‌عنوان یک زبان استاندارد برای دسترسی و مدیریت داده‌های پایگاه‌های داده‌های رابطه‌ای در حال توسعه بود. اولین نسخه SQL در سال ۱۹۷۴ منتشر شد و در سال‌های بعد با انتشار نسخه‌های جدید، ویژگی‌های جدیدی به آن اضافه شد.

- دوره رشد (۲۰۰۳-۱۹۸۶)
- در این دوره، SQL به‌عنوان یک زبان استاندارد پذیرفته شد و به طور گسترده توسط سیستم‌های مدیریت پایگاه‌داده استفاده شد. در این دوره، ویژگی‌های جدیدی از جمله توابع، رویه‌ها و триггерها به SQL اضافه شد.
- دوره پیشرفت (۲۰۰۳-اکنون)

در این دوره، SQLهمچنان به رشد و توسعه خود ادامه داده است. در این دوره، ویژگی‌های جدیدی از جمله پشتیبانی از داده‌های جغرافیایی، داده‌های نیمه ساختار یافته و داده‌های بزرگ به SQL اضافه شده است.

۱_۲_ زبان برنامه‌نویسی SQL در ابتدا به چه منظوری تهیه شده است؟

زبان برنامه‌نویسی SQL در ابتدا برای مدیریت داده‌های ذخیره شده در پایگاه‌های داده‌های رابطه‌ای تهیه شده است. هدف از ایجاد این زبان، ایجاد یک زبان استاندارد برای دسترسی و مدیریت داده‌ها بود تا توسعه‌دهندگان بتوانند بدون نیاز به دانستن سیستم مدیریت پایگاه‌داده خاص، به داده‌ها دسترسی داشته باشند.

SQL یک زبان بسیار قدرتمند است که می‌توان از آن برای انجام طیف گسترده‌ای از وظایف مربوط به پایگاه‌های داده استفاده کرد. از جمله این وظایف می‌توان به موارد زیر اشاره کرد:

- ایجاد و حذف جداول
- افزودن، ویرایش و حذف داده‌ها از جداول
- جستجو و فیلترکردن داده‌ها
- ایجاد گزارش‌ها

SQLبه سرعت محبوب شد و امروزه به‌عنوان زبان استاندارد برای دسترسی و مدیریت داده‌های پایگاه‌های داده‌های رابطه‌ای استفاده می‌شود. این زبان توسط بسیاری از سیستم‌های مدیریت پایگاه‌داده (DBMS) از جملهMySQL ، Oracle، Microsoft SQL Serverو PostgreSQLپشتیبانی می‌شود .

برخی از مزایای استفاده ازSQL عبارت‌اند از:

- قدرتمند و انعطاف‌پذیر است.
- یادگیری آن آسان است (زیرا دستورات آن مشابه با جملات انگلیسی نوشته می‌شود.)

- توسط بسیاری از سیستم‌های مدیریت پایگاه‌داده پشتیبانی می‌شود.

۱_۳_ زبان برنامه‌نویسی SQL در چه زمینه‌ها و حوزه‌هایی کاربرد دارد؟

- تجارت و کسب‌وکار

SQL در بسیاری از کسب‌وکارها برای مدیریت داده‌های مشتریان، محصولات، فروش و سایر اطلاعات استفاده می‌شود.

برای مثال، شرکت‌ها از SQL برای ایجاد گزارش‌های فروش، ردیابی موجودی و مدیریت مشتریان استفاده می‌کنند.

- فناوری اطلاعات

SQL در بسیاری از محصولات و خدمات فناوری اطلاعات استفاده می‌شود.

برای مثال، سیستم‌های مدیریت محتوا (CMS) از SQL برای ذخیره و مدیریت محتوا استفاده می‌کنند. سیستم‌های مدیریت سفارش (CRM) از SQL برای ذخیره و مدیریت اطلاعات مشتریان استفاده می‌کنند.

- دولت و سازمان‌های دولتی

SQL در بسیاری از سازمان‌های دولتی برای مدیریت داده‌های شهروندان، مالیات، بودجه و سایر اطلاعات استفاده می‌شود.

برای مثال، دولت‌ها از SQL برای ایجاد پایگاه‌های داده هویت شهروندان، ردیابی درآمد مالیاتی و مدیریت بودجه استفاده می‌کنند.

- علم و تحقیقات

SQL در بسیاری از زمینه‌های علمی و تحقیقاتی برای مدیریت داده‌های آزمایشگاهی، نتایج تحقیقات و سایر اطلاعات استفاده می‌شود.

برای مثال، دانشمندان از SQL برای ذخیره و مدیریت داده‌های تحقیقاتی، ایجاد نمودارها و تجزیه‌وتحلیل داده‌ها استفاده می‌کنند.

- آموزش‌وپرورش

SQL در بسیاری از مدارس و دانشگاه‌ها برای مدیریت داده‌های دانشجویان، نمرات، کلاس‌ها و سایر اطلاعات استفاده می‌شود.

برای مثال، مدارس از SQL برای ایجاد پایگاه‌های داده دانش‌آموزان، ردیابی نمرات دانش‌آموزان و مدیریت کلاس‌ها استفاده می‌کنند.

- ایجاد و مدیریت پایگاه‌های داده

SQL برای ایجاد و مدیریت پایگاه‌های داده‌های رابطه‌ای استفاده می‌شود.

برای مثال، می‌توان از SQL برای ایجاد جداول، افزودن ستون‌ها، حذف جداول و غیره استفاده کرد.

- دسترسی و مدیریت داده‌ها

SQL برای دسترسی و مدیریت داده‌های ذخیره شده در پایگاه‌های داده استفاده می‌شود.

برای مثال، می‌توان از SQL برای جستجو، فیلترکردن، مرتب‌کردن و ویرایش داده‌ها استفاده کرد.

- ایجاد گزارش‌ها

SQL برای ایجاد گزارش‌های مبتنی بر داده استفاده می‌شود.

برای مثال، می‌توان از SQL برای ایجاد گزارش‌های فروش، گزارش‌های مالی و گزارش‌های دیگر استفاده کرد.

- ایجاد تجزیه‌وتحلیل داده‌ها

SQL برای ایجاد تجزیه‌وتحلیل داده‌ها استفاده می‌شود.

برای مثال، می‌توان از SQL برای محاسبه میانگین، مقادیر حداقل و حداکثر و سایر آمار داده‌ها استفاده کرد.

۱_۴_ این زبان برنامه‌نویسی برای رفع چه مشکلاتی ابداع شده و ابداع آن در جهت بهبود چه زبان هایی بوده است؟

زبان برنامه‌نویسی SQL برای رفع مشکلاتی در زمینه مدیریت داده‌های ذخیره شده در پایگاه‌های داده‌های رابطه‌ای ابداع شده است. قبل از ظهور SQL، برای دسترسی و مدیریت داده‌ها در پایگاه‌های داده‌های رابطه‌ای از زبان‌های برنامه‌نویسی عمومی مانند COBOL و PL/I استفاده می‌شد. این زبان‌ها برای مدیریت داده‌ها در پایگاه‌های داده‌های رابطه‌ای طراحی نشده بودند و استفاده از آنها برای این کار دشوار و پیچیده بود.

برخی از مزایای استفاده از SQL نسبت به زبان‌های برنامه‌نویسی عمومی برای مدیریت داده‌های ذخیره شده در پایگاه‌های داده‌های رابطه‌ای عبارت‌اند از:

- سادگی و یادگیری آسان
- قدرتمند و انعطاف‌پذیر
- سازگاری با بسیاری از سیستم‌های مدیریت پایگاه‌داده

۱_۵_ این زبان در ابتدای ظهور خود چه مشکلاتی را رفع می کرده است؟

هنگامی که SQL برای اولین‌بار معرفی شد، هدف آن حل چندین مشکل مربوط به مدیریت و دست‌کاری داده‌ها بود. در اینجا برخی از مشکلات کلیدی که SQL به حل آن‌ها پرداخته است را مرور می‌کنیم:

۱. بازیابی و دست‌کاری داده‌ها: قبل از SQL، دسترسی و دست‌کاری داده‌ها در پایگاه‌های داده یک کار پیچیده و مستعد خطا بود. SQL یک زبان استاندارد برای پرس‌وجو و دست‌کاری داده‌های ذخیره شده در پایگاه‌های داده رابطه‌ای ارائه کرد. این یک روش ساده و شفاف برای تعیین اینکه چه داده‌هایی باید بازیابی شوند و چگونه آن‌ها را فیلتر، مرتب‌سازی و جمع‌آوری کنیم، ارائه کرد.

۲. استقلال داده‌ها: SQL سطح بالاتری از انتزاع را معرفی کرد و به کاربران اجازه داد تا بدون نیاز به درک مکانیسم‌های ذخیره‌سازی و دسترسی به داده‌های اساسی با پایگاه‌های داده تعامل داشته باشند. این امر استقلال داده‌ها را فراهم کرد و کار با پایگاه‌های داده را آسان‌تر کرد، زیرا کاربران می‌توانستند به‌جای جزئیات پیاده‌سازی فیزیکی، بر ساختار و عملیات منطقی تمرکز کنند.

۳. یکپارچگی و سازگاری داده‌ها: SQL مکانیسم‌هایی را برای اعمال یکپارچگی و سازگاری داده‌ها در پایگاه‌های داده رابطه‌ای معرفی کرد. این ویژگی از تعریف محدودیت برای کلید اصلی، محدودیت برای یکتایی داده‌ها، روابط کلید خارجی و سایر قوانین یکپارچگی پشتیبانی می‌کند. این ویژگی‌ها به حفظ کیفیت و دقت داده‌های ذخیره شده در پایگاه‌های داده کمک کرد.

۴. دسترسی هم‌زمان و مدیریت تراکنش: SQL قابلیت‌های مدیریت تراکنش را معرفی کرد که به چندین کاربر اجازه می‌داد به طور هم‌زمان به داده‌ها دسترسی داشته باشند و آنها را اصلاح کنند و درعین‌حال از ثبات و یکپارچگی پایگاه‌داده اطمینان حاصل کنند. مکانیسم‌هایی مانند قفل‌کردن و جداسازی سطوح را برای مدیریت دسترسی هم‌زمان و جلوگیری از تناقضات داده‌ها ارائه کرد.

۵. امنیت داده‌ها و کنترل دسترسی: SQL ویژگی‌هایی را برای اعمال امنیت و کنترل دسترسی در پایگاه‌های داده معرفی کرد. این به مدیران اجازه می‌داد تا نقش‌ها، امتیازات و مجوزهای دسترسی را تعریف کنند و اطمینان حاصل شود که فقط کاربران مجاز می‌توانند به داده‌ها دسترسی داشته باشند و آن‌ها را دست‌کاری کنند.

۶. مقیاس‌پذیری داده‌ها و بهینه‌سازی عملکرد: SQL تکنیک‌های بهینه‌سازی و طرح‌های اجرای پرس‌وجو را برای بهبود عملکرد عملیات پایگاه‌داده ارائه می‌کند. به مدیران پایگاه‌داده و توسعه‌دهندگان این امکان را می‌داد تا کوئری‌ها، نمایه‌ها و ساختارهای پایگاه‌داده را تنظیم و بهینه‌سازی کنند تا مقیاس‌پذیری و عملکرد بهتری داشته باشند.

به‌طورکلی هدف SQL ساده‌سازی مدیریت داده‌ها، بهبود یکپارچگی و سازگاری داده‌ها، فعال‌کردن دسترسی هم‌زمان، افزایش امنیت داده‌ها و بهینه‌سازی عملکرد عملیات پایگاه‌داده است. این قابلیت‌ها SQL را به زبانی قدرتمند و پرکاربرد برای کار با پایگاه‌های داده رابطه‌ای تبدیل کرد.

۱_۶_ این زبان در مقایسه با زبان‌های دیگر و به خصوص زبان هایی که شبیه آن هستند چگونه ارزیابی می‌شود؟

زبان برنامه‌نویسی SQL در مقایسه با زبان‌های دیگر، از جمله زبان‌های شبیه به آن، دارای مزایای زیر است:

- سادگی و یادگیری آسان SQL :یک زبان ساده و مختصر است که یادگیری آن آسان است. این زبان از دستورات و عبارات ساده‌ای استفاده می‌کند که یادگیری آنها برای توسعه‌دهندگان تازه‌کار آسان است.
- قدرتمند و انعطاف‌پذیر SQL :یک زبان قدرتمند و انعطاف‌پذیر است که می‌توان از آن برای انجام طیف گسترده‌ای از وظایف مربوط به مدیریت داده‌ها استفاده کرد. این زبان برای انجام عملیات پیچیده‌ای؛ مانند جستجو، فیلترکردن، مرتب‌کردن، ویرایش و ایجاد گزارش‌ها طراحی شده است.
- سازگاری با بسیاری از سیستم‌های مدیریت پایگاه‌داده SQL :توسط بسیاری از سیستم‌های مدیریت پایگاه‌داده محبوب پشتیبانی می‌شود. این امر توسعه‌دهندگان را قادر می‌سازد تا مهارت‌های خود را در SQL در سیستم‌های مدیریت پایگاه‌داده مختلف اعمال کنند.

زبان‌های برنامه‌نویسی دیگری مبتنی بر SQL و بر اساس آن ساخته شده‌اند، و دارای قابلیت‌های ویژه و خاصی هستند از قبیل PL/SQL, T-SQL, PLpgSQL اما هنوز زبان SQL به‌عنوان یک زبان استاندارد برای مدیریت داده‌های پایگاه‌های داده‌های رابطه‌ای شناخته می‌شود.

۱_۷_ این زبان چه ویژگی‌های خاصی دارد که آن را از زبان‌های مشابه آن متمایز می‌کند؟

۱. زبان اظهاری: SQL یک زبان اظهاری است، به این معنی که شما مشخص می‌کنید که چه چیزی را می‌خواهید بازیابی کنید یا از یک پایگاه‌داده دست‌کاری کنید بدون اینکه مشخص کنید چگونه آن را انجام دهید. شما نتایج مورد نظر را توصیف می‌کنید و سیستم مدیریت پایگاه‌داده (DBMS) کارآمدترین راه را برای اجرای دست‌کاری تعیین می‌کند.

۲. عملیات مبتنی بر مجموعه: SQL به‌جای عناصر منفرد، بر روی مجموعه داده‌ها عمل می‌کند. این ویژگی به شما این امکان را می‌دهد تا با استفاده از عملیات‌های مجموعه‌ای قدرتمند مانند اجتماع، اشتراک و تفاضل، عملیات را روی کل جداول یا زیرمجموعه جداول انجام دهید.

۳. زبان تعریف داده‌ها (DDL): SQL شامل یک DDL است که شما را قادر می‌سازد تا ساختار پایگاه‌داده را تعریف کرده و تغییر دهید. با دستورات DDL مانند CREATE، ALTER و DROP می‌توانید جداول ایجاد کنید، ساختار آن‌ها را تغییر دهید و آن‌ها را حذف کنید.

۴. زبان دست‌کاری داده‌ها (DML): SQL یک DML ارائه می‌دهد که به شما امکان می‌دهد داده‌ها را از پایگاه‌داده وارد کنید، به‌روز کنید، حذف کنید و بازیابی کنید. عبارات DML مانند SELECT، INSERT، UPDATE و DELETE شما را قادر می‌سازد تا این عملیات را انجام دهید.

۵. اتصالات و روابط: SQL از قابلیت پیوستن جداول بر اساس ستون‌های مشترک پشتیبانی می‌کند و شما را قادر می‌سازد تا داده‌ها را از چندین جدول در یک کد بازیابی کنید. این ویژگی برای مدیریت روابط پیچیده بین موجودیت‌ها در یک پایگاه‌داده رابطه‌ای بسیار مهم است.

۶. یکپارچگی داده‌ها و محدودیت‌ها: SQL شامل مکانیسم‌هایی برای اعمال یکپارچگی داده‌ها با تعریف محدودیت‌ها در جداول است. محدودیت‌ها تضمین می‌کنند که داده‌ها با شرایط مشخصی مانند مقادیر منحصربه‌فرد، یکپارچگی ارجاعی و محدودیت‌های نوع داده مطابقت دارند.

۷. نمایه‌سازی: SQL به ایجاد نمایه‌هایی روی جداول اجازه می‌دهد که عملکرد کوئری را با تسهیل بازیابی سریع‌تر داده‌ها بهبود می‌بخشد. شاخص‌ها را می‌توان بر روی ستون‌های خاص یا ترکیبی از ستون‌ها برای سرعت‌بخشیدن به جستجو و مرتب‌سازی ایجاد کرد.

۸. تراکنش‌ها و کنترل هم‌زمان: SQL قابلیت‌های تراکنشی را فراهم می‌کند و به شما امکان می‌دهد چندین عملیات را در یک واحد اتمی گروه‌بندی کنید. این تضمین می‌کند که یا تمام تغییرات در یک تراکنش انجام می‌شود یا هیچ یک از آن‌ها انجام نمی‌شود. علاوه بر این، SQL از مکانیسم‌های کنترل هم‌زمانی برای مدیریت دسترسی هم‌زمان به پایگاه‌داده توسط چندین کاربر یا فرایند پشتیبانی می‌کند.

۹. بهینه‌سازی کوئری: هدف بهینه‌سازهای SQL ایجاد کارآمدترین برنامه‌های اجرایی برای کوئری‌ها است. DBMS نحو و ترکیب کوئری‌ها، آمار جداول، و دیگر موارد را تجزیه‌وتحلیل می‌کند تا بهینه‌ترین راه برای اجرای یک کوئری کمینه‌کردن منابع مورد استفاده‌ی برنامه و زمان اجرای آن را مشخص کند.

۱۰. قابلیت حمل: SQL یک زبان استاندارد است که توسط اکثر سیستم‌های مدیریت پایگاه‌داده رابطه‌ای پشتیبانی می‌شود. این قابلیت حمل به شما امکان می‌دهد کد SQL بنویسید که می‌تواند در پلتفرم‌های مختلف پایگاه‌داده با حداقل تغییرات اجرا شود.

این ویژگی‌ها در مجموع SQL را به زبانی قدرتمند و انعطاف‌پذیر برای کار با پایگاه‌های داده رابطه‌ای تبدیل می‌کند که امکان بازیابی، دست‌کاری و مدیریت کارآمد داده‌ها را فراهم می‌کند.

۱_۸_ این زبان را بر اساس خوانایی، قابلیت اطمینان، هزینه (کارایی و بهره‌وری و همچنین هزینه موردنیاز برای یادگیری و برنامه‌نویسی) و سایر معیارها مانند قابلیت جابجایی چگونه می‌توان ارزیابی کرد؟

۱. قابلیت اطمینان: SQL یک زبان قابل‌اعتماد برای کار با پایگاه‌داده‌های رابطه‌ای است، زیرا از استانداردی پیروی می‌کند که به طور گسترده توسط بسیاری از سیستم‌های مدیریت پایگاه‌داده پشتیبانی می‌شود. بااین‌حال، گویش‌های مختلف SQL ممکن است دارای تغییرات و محدودیت‌هایی باشند که بر قابلیت اطمینان کوئری‌ها تأثیر می‌گذارد. برای مثال، SQL Server قوانین سخت‌گیرانه‌تری برای انواع داده‌ها و شناسه‌ها نسبت به MySQL دارد. برخی از زبان‌هایی که شبیه SQL هستند، مانند NoSQL، برای پایگاه‌های داده غیرمرتبطی طراحی شده‌اند که انعطاف‌پذیری و مقیاس‌پذیری بیشتری را ارائه می‌دهند، اما ممکن است بخشی از قابلیت اطمینان و ثبات را قربانی کنند.

۲. خوانایی: SQL یک زبان خوانا است، زیرا از کلمات کلیدی و نحوی استفاده می‌کند که نزدیک به زبان طبیعی هستند و به‌راحتی قابل درک هستند. بااین‌حال، کوئری‌های SQL زمانی که شامل چندین جدول، پیوست، کوئری‌های فرعی و توابع می‌شوند، می‌توانند پیچیده شده و خواندن آن‌ها دشوار شود. برخی از زبان‌هایی که شبیه SQL هستند، مانند LINQ، با سایر زبان‌های برنامه‌نویسی مانند C#ادغام می‌شوند و از مفاهیم شی‌گرا و عبارات لامبدا برای جستجوی داده‌ها استفاده می‌کنند که ممکن است خوانایی و قابلیت نگهداری کد را بهبود بخشد.

۳. کارایی: SQL یک زبان کارآمد است، زیرا به شما امکان می‌دهد عملیات مختلفی را روی‌داده‌ها با یک کوئری انجام دهید. SQL همچنین دارای بسیاری از توابع و ویژگی‌های داخلی است که می‌تواند عملکرد و سرعت کوئری را بهینه کند. بااین‌حال، کوئری‌های

SQL نیز می‌توانند ناکارآمد باشند اگر به‌خوبی نوشته یا بهینه نشده باشند. برخی از عواملی که می‌توانند بر کارایی کوئری‌های SQL تأثیر بگذارند، اندازه و ساختار پایگاه‌داده، استفاده از فهرست‌ها، پیوست‌ها، کوئری‌های فرعی و توابع هستند. برخی از زبان‌هایی که شبیه SQL هستند، مانند DAX، برای تجزیه‌وتحلیل داده‌ها و گزارش‌دهی تخصصی هستند و از موتور محاسباتی و نحو متفاوتی استفاده می‌کنند که می‌تواند کارایی و دقت درخواست‌ها را بهبود بخشد.

۴. زمان و منابع موردنیاز برای یادگیری: SQL یک زبان نسبتاً آسان برای یادگیری است، زیرا دارای یک نحو ساده و منطقی و تعداد محدودی از کلمات کلیدی و دستورات است. بااین‌حال، SQL همچنین دارای بسیاری از ویژگی‌ها و توابع پیشرفته است که نیاز به زمان و منابع بیشتری برای تسلط دارند. برخی از موضوعاتی که یادگیری آن‌ها در SQL می‌تواند چالش‌برانگیز باشد، مدل‌سازی داده‌ها، نرمال‌سازی، تراکنش‌ها، هم‌زمانی، امنیت و بهینه‌سازی است. برخی از زبان‌هایی که شبیه SQL هستند، مانند PL/SQL، نسخه‌های گسترش‌یافته‌ی SQL هستند که عملکرد و پیچیدگی بیشتری را اضافه می‌کنند، مانند برنامه‌نویسی رویه‌ای، متغیرها، حلقه‌ها، شرایط، استثناها و راه‌اندازها. یادگیری این زبان‌ها می‌تواند سخت‌تر و زمان‌برتر باشد، اما همچنین قدرتمندتر و همه‌کاره‌تر است.

۵. قابلیت جابه‌جایی: SQL یک زبان قابل‌حمل است، زیرا می‌توان از آن در پلتفرم‌ها و سیستم‌عامل‌های مختلف استفاده کرد. بااین‌حال، SQL همچنین وابسته به سیستم مدیریت پایگاه‌داده است که آن را پیاده‌سازی می‌کند، و گویش‌های مختلف SQL ممکن است تفاوت‌ها و ناسازگاری‌هایی داشته باشند که بر تحرک کوئری‌ها تأثیر می‌گذارد. به‌عنوان مثال، برخی از توابع و کلمات کلیدی که در MySQL کار می‌کنند ممکن است در SQL Server کار نکنند و بالعکس. برخی از زبان‌هایی که شبیه SQL هستند، مانند HiveQL، برای پلتفرم‌ها یا چارچوب‌های خاصی مانند Hadoop طراحی شده‌اند و ممکن است با سیستم‌های دیگر سازگار نباشند.

به‌طورکلی SQL یک زبان قدرتمند و همه‌کاره برای کار با داده‌ها، به‌خصوص داده‌های رابطه‌ای است. SQL به شما امکان ایجاد، دست‌کاری، نوشتن کوئری و تجزیه‌وتحلیل داده‌ها را به روشی ساختاریافته و کارآمد می‌دهد. SQL همچنین دارای ویژگی‌ها و توابع بسیاری است که می‌تواند به شما در بهینه‌سازی و تقویت کوئری‌ها کمک کند. بااین‌حال، SQL همچنین دارای محدودیت‌ها و چالش‌هایی مانند مسائل سازگاری، پیچیدگی و خطرات امنیتی است؛ بنابراین، یادگیری و استفاده صحیح و مسئولانه از SQL بسیار مهم است. SQL تنها زبان برای داده‌ها نیست، اما یکی از پرکاربردترین و محبوب‌ترین زبان‌هاست.

۱_۹_ آیا برای پیاده‌سازی زبان SQL از کامپایلر استفاده شده است یا مفسر یا پیاده‌سازی ترکیبی؟

برای پیاده‌سازی این زبان هم از کامپایلر و هم از مفسر استفاده شده است. کامپایلرها کد SQL را به ماشین کد تبدیل کرده تا مستقیماً توسط CPU قابل درک و اجرا باشد. بااین‌حال کامپایل کردن کد SQL در صورت پیچیده بودن کد می‌تواند فرایندی زمان بر باشد.

اما مفسرها کد SQL را مستقیماً بدون اینکه آن را به ماشین کد تبدیل کنند اجرا می‌کنند. اما این کار بسیار کند است؛ زیرا مفسرها باید کد SQL را به‌صورت خط‌به‌خط تفسیر کند، بااین‌حال مفسرها سریع‌تر از کامپایلرها کد SQL را توسعه می‌دهند.

پیاده‌سازی ترکیبی کامپایلر و مفسر نیز ممکن است که در بسیار از موارد یکی از بهترین گزینه‌ها است. در این نوع پیاده‌سازی اگر کد SQL پیچیده باشد توسط مفسرها به‌صورت مستقیم اجرا می‌شود در غیر این صورت کد SQL به ماشین کد تبدیل شده و سپس کامپایل می‌شود. یعنی اجرای سریع‌تر در کنار توسعه سریع‌تر برای قطعه کدهای مختلف وجود دارد.

در نتیجه انتخاب بین این سه مورد وابسته موارد زیر است:

- سرعت موردنیاز اجرای کد SQL
- پیچیدگی کد SQL
- زمان توسعه موردنیاز

۱_۱۰_ چه کامپایلرها یا مفسرهایی در حال حاضر برای این زبان وجود دارند؟ این کامپایلرها یا مفسرها توسط چه تیم هایی یا چه شرکت هایی تهیه شده‌اند؟

برخی از محبوب‌ترین کامپایلرها برای زبان SQL عبارت‌اند از:

- Microsoft SQL Server

این کامپایلر توسط تیم تحقیقاتی Microsoft SQL Server در شرکت Microsoft توسعه‌یافته است.

ویژگی‌ها: پشتیبانی از انواع داده‌های پیشرفته، عملکرد بالا، قابلیت اطمینان

قیمت: پولی

پیچیدگی: بالا

- Oracle Database

توسط تیم توسعه Oracle Database در شرکت اوراکل توسعه‌یافته است.

ویژگی‌ها: پشتیبانی از انواع داده‌های پیشرفته، عملکرد بالا، قابلیت اطمینان

قیمت: پولی

پیچیدگی: بالا

- MySQL

توسط تیم توسعه MySQL در شرکت Oracle توسعه‌یافته است.

ویژگی‌ها :پشتیبانی از انواع داده‌های پیشرفته، عملکرد خوب، قیمت مناسب

قیمت :رایگان یا پولی

پیچیدگی :متوسط

- PostgreSQL

توسط تیم توسعه PostgreSQL در بنیاد PostgreSQL توسعه‌یافته است.

ویژگی‌ها: پشتیبانی از انواع داده‌های پیشرفته، عملکرد خوب، قابلیت اطمینان بالا

قیمت: رایگان یا پولی

پیچیدگی: بالا

برخی از محبوب‌ترین مفسرها برای زبان SQL عبارت‌اند از:

این مفسرها معمولاً سبک‌تر و سریع‌تر از کامپایلرها هستند، اما ویژگی پیشرفته‌ای ارائه نمی‌دهند.

- SQLite

توسط تیم توسعه SQLite در شرکت D. Richard Hipp توسعه‌یافته است.

ویژگی‌ها :سبک‌وزن، سریع، رایگان

قیمت :رایگان

پیچیدگی :پایین

- Firebird

توسط تیم توسعه Firebird در شرکت Firebird Foundation توسعه‌یافته است.

ویژگی‌ها :عملکرد خوب، ویژگی‌های پیشرفته، رایگان

قیمت :رایگان

پیچیدگی :متوسط

- H2

توسط تیم توسعه H2 در شرکت H2 Database Engine توسعه‌یافته است.

ویژگی‌ها :سبک‌وزن، سریع، رایگان

قیمت :رایگان

پیچیدگی :پایین

۱_۱۱_ مزیت هر یک از کامپایلرها و یا مفسرهای نام برده شده چیست؟

مزیت هر یک از کامپایلرها یا مفسرهای SQL به عوامل مختلفی بستگی دارد، از جمله:

- نیازهای عملکردی
- نیازهای توسعه
- بودجه

Microsoft SQL Server

- مزیت :عملکرد بالا، قابلیت اطمینان، ویژگی‌های پیشرفته
- معایب :قیمت بالا، پیچیدگی

Oracle Database

- مزیت :عملکرد بالا، قابلیت اطمینان، ویژگی‌های پیشرفته
- معایب :قیمت بالا، پیچیدگی

MySQL

- مزیت :عملکرد خوب، ویژگی‌های پیشرفته، قیمت مناسب
- معایب :قابلیت اطمینان کمتر از Microsoft SQL Server وOracle Database

PostgreSQL

- مزیت :عملکرد خوب، ویژگی‌های پیشرفته، قابلیت اطمینان بالا
- معایب :پیچیدگی بیشتر ازMySQL

SQLite

- مزیت :سبک‌وزن، سریع، رایگان
- معایب :ویژگی‌های پیشرفته کمتر از کامپایلرها و مفسرهای دیگر

Firebird

- مزیت :عملکرد خوب، ویژگی‌های پیشرفته، رایگان
- معایب :پیچیدگی بیشتر ازSQLite

H2

- مزیت :سبک‌وزن، سریع، رایگان
- معایب :ویژگی‌های پیشرفته کمتر از کامپایلرها و مفسرهای دیگر

۲- نحو و معناشناسی

۲-۱- 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 ۱۲۳ به ۱۹,۹۹ 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 ۴۵۶ 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 ۵ 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 ۱۰۰۰۰ 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;
```

۲-۲-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
```

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

۳-۲- 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 (+ز ، - ، * ، / ، ٪) 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 ۰ تا ۹ 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.

۴-۲- 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)
 VALUES (1, 'George Orwell', 'UK', 1903), (2, 'Maya Angelou', 'USA', 1928),
(3, 'Yuval Noah Harari', 'Israel', 1976),
```

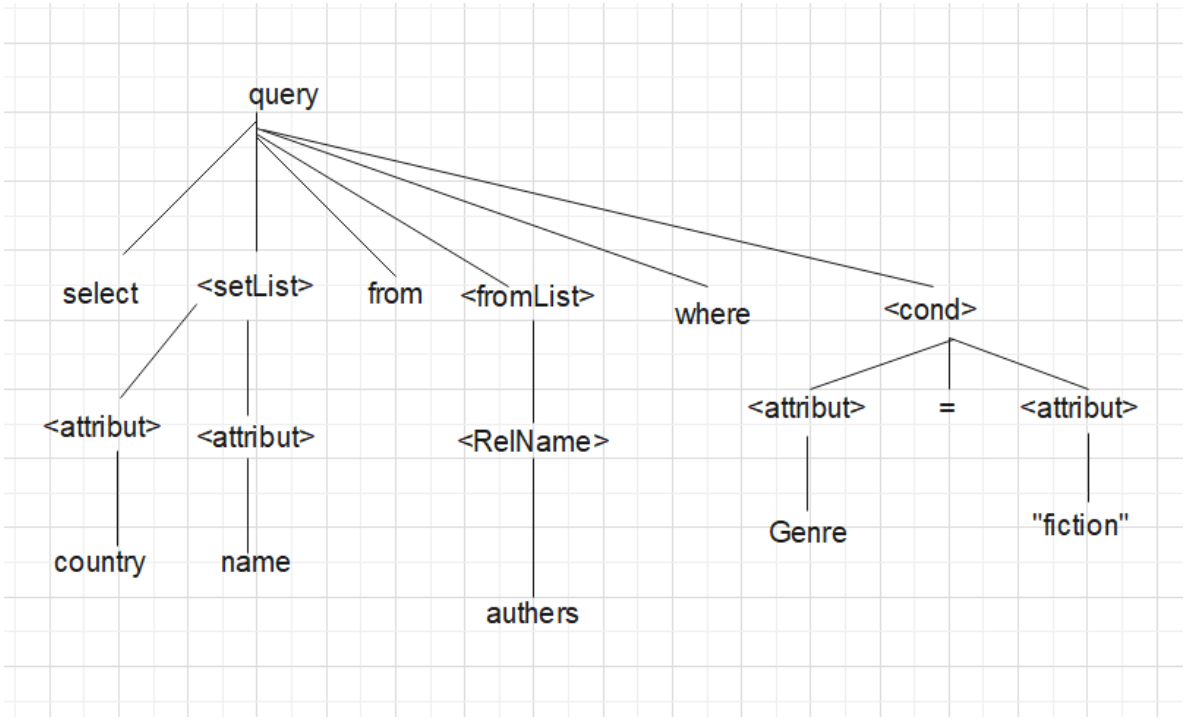
```sql
 (4, 'J.K. Rowling', 'UK', 1965), (5, 'Rupi Kaur', 'Canada', 1992);

 -- 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
 USE Movies;
```

```sql
-- 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
```

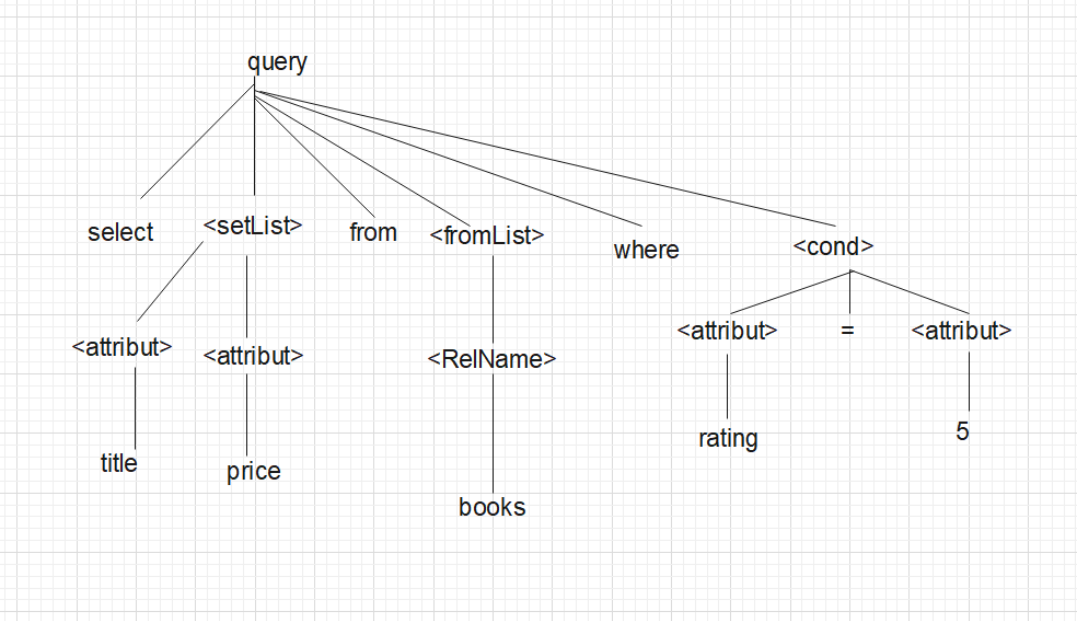
```
 LEFT JOIN Casts ON Actors.id = Casts.actor_id GROUP BY Actors.name;

 -- 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
```

۵-۲- رسم درخت تجزیه:

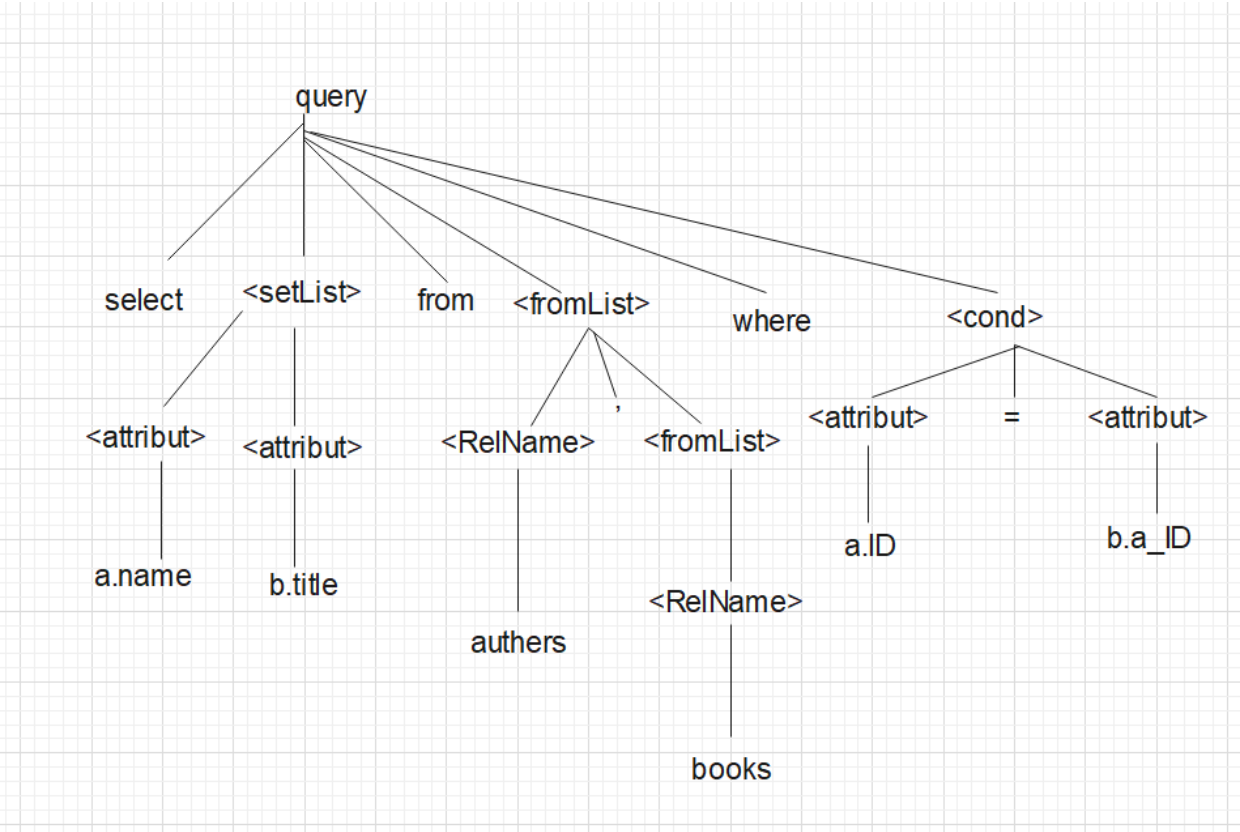
```
SELECT name, country FROM Authors as au WHERE au.genre =
'fiction';
```



```
SELECT title, price FROM Books WHERE rating = 5;
```



```
SELECT Authors.name, Books.title FROM Authors INNER JOIN
Books ON Authors.id = Books.author_id;
```



۶-۲- 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.

تقدم	عملگر	وابستگی
۱	()	وابسته نیست
۲	* /	وابسته به چپ
۳	- +	وابسته به چپ
۴	<> = < > <= =>	وابسته به چپ
۵	NOT	وابسته به راست
۶	AND	وابسته به چپ
۷	OR	وابسته به چپ
۸	BETWEEN IN LIKE IS NULL	وابسته به چپ
۹	ALL ANY SOME EXISTS UNIQUE	وابسته به چپ
۱۰	SELECT	وابسته به راست
۱۱	AS	وابسته به چپ
۱۲	FROM	وابسته به راست
۱۳	WHERE	وابسته به راست
۱۴	GROUP BY	وابسته به چپ
۱۵	HAVING	وابسته به راست
۱۶	ORDER BY	وابسته به چپ
۱۷	UNION EXCEPT INTERSECT	وابسته به چپ

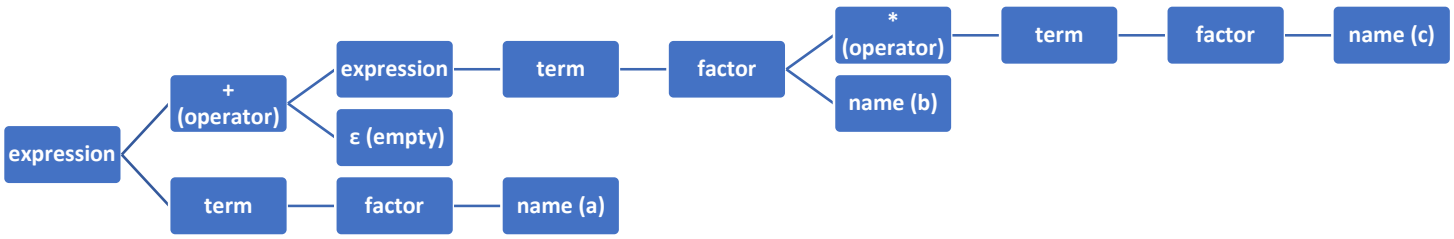
۷-۲- نحوهٔ توصیف گرامر برای پیروی از تقدم های ذکر شده:

```
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
```

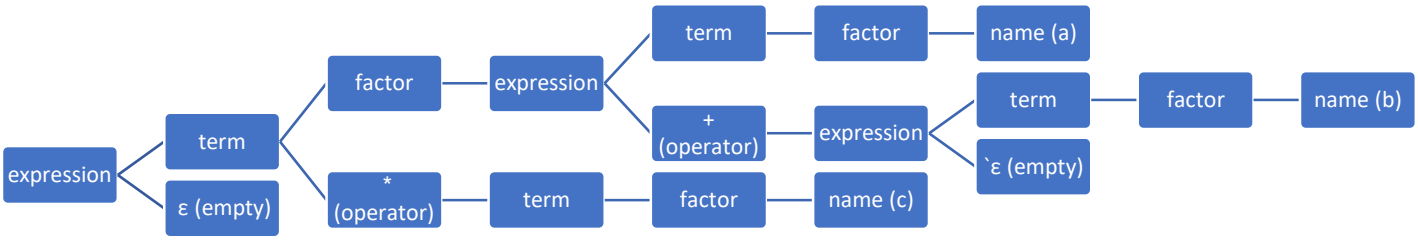
قسمتی که با آبی هایلایت شده برای نشان دادن تقدن عملگرها بجای دستورات زیر قرار گرفته است:

```
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.

۸-۲- 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)
```

```c
// 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
```

```c
// 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:

```sql
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);
    }
}
```

۳_ متغیرها و نوع داده‌ای

۳_۱_ 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 فرایند 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.

به طور 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 یک‌رشته 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.

به طور 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.

به طور 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.

۳_۲_ آیا در زبان SQL متغیرهای ایستا، پویا در پشته، پویا در هیپ به طور صریح، پویا در هیپ به طور 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.

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 به طور صریح یا پویا در هیپ به طور 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 به طور 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 به طور 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 به طور 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 به طور 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;
```

۳_۳_ آیا می‌توانید سرعت تخصیص این متغیرها را در این زبان مقایسه کنید؟

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.

۳_۴_ آیا حوزه تعریف در این زبان ایستا است یا پویا؟ با ذکر مثال توضیح دهید.

حوزه تعریف در زبان SQL یک زیر زبان از SQL است که برای ایجاد، تغییر و حذف ساختارهای داده؛ مانند جداول، ستون‌ها، کلیدها، اندیس‌ها و محدودیت‌ها در پایگاه‌داده استفاده می‌شود. این زیر زبان 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. به‌عنوان مثال، می‌توان یک جدول را به‌گونه‌ای 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 اضافه‌کردن 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.

۳_۵_ در صورتی که زبان حوزه تعریف ایستا را پشتیبانی می‌کند و بخواهیم امکان استفاده از حوزه تعریف پویا به آن بیافزاییم، چه تغییراتی در زبان باید ایجاد کنیم؟

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

- اضافه‌کردن یک کلمه کلیدی جدید به نام DYNAMIC به زبان SQL که برای تعیین نوع حوزه تعریف استفاده می‌شود. به‌عنوان 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)
);
```

- اضافه‌کردن 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. به‌عنوان مثال، برای اضافه‌کردن 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;
```

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

- اضافه‌کردن یک کلمه کلیدی جدید به نام OLD به زبان SQL که برای ارجاع به رکورد قبلی که از جدول حذف شده است، استفاده می‌شود. به‌عنوان 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 اضافه‌کردن امکان استفاده از حوزه تعریف پویا به زبان SQL هستند و ممکن است راه‌حل‌های دیگری هم وجود داشته باشند.

۳_۶_ 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 می‌کند، هر دو مورد توصیف شوند.

برای استفاده از حوزه تعریف پویا در زبان SQL، باید از کلمه کلیدی DYNAMIC و ساختار RULE استفاده کنیم. به‌عنوان 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. به‌عنوان 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)
);
```

۳_۷_ بلوک ها در این زبان چگونه تعریف شده‌اند؟ آیا کلمات کلیدی ویژه ای برای اعمال تغییر در حوزهٔ تعریف متغیرها وجود دارند؟

بلوک ها در زبان SQL به‌عنوان یک واحد منطقی از دستورات تعریف شده‌اند که می‌توانند در یک تراکنش یا یک برنامه اجرا شوند. بلوک ها می‌توانند شامل متغیرها، ثابت ها، مقادیر پیش فرض، توابع، زیر برنامه ها، دستورات کنترل جریان و خطاها باشند. بلوک ها می‌توانند درون یکدیگر تودرتو شوند و محدوده متغیرها را تعیین کنند. برای شروع و پایان یک بلوک، از کلمات کلیدی BEGIN و END استفاده می‌شود. برای مثال، بلوک زیر یک متغیر به نام x را تعریف می‌کند 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، می‌توان از کلمات کلیدی مختلفی مانند DECLARE، SET، DEFAULT، LOCAL و GLOBAL استفاده کرد.

- DECLARE:

این کلمه کلیدی برای تعریف یک متغیر در SQL استفاده می‌شود. 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 می‌شود. 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 می‌شود. مقدار پیش فرض برای یک ستون، آن مقداری است که در صورت عدم ورود مقدار توسط کاربر، به آن ستون اختصاص داده می‌شود. 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 می‌شود. یک متغیر محلی، آن متغیری است که فقط در بلوکی که تعریف شده است، قابل‌دسترسی 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 می‌شود. یک متغیر جهانی، آن متغیری است که در همه بلوک های یک اتصال، قابل‌دسترسی 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
```

۳_۸_ 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 پیاده‌سازی آنها شرح داده شود. هر یک از نوع ها چه ویژگی‌هایی 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 دودسته 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 داده‌ها برای ذخیره مقادیر، انجام محاسبات و تجزیه‌وتحلیل داده‌ها استفاده می‌شود.

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 واردکردن 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 واردکردن 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 داده‌های 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 داده‌های 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 داده‌های 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 داده‌های 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 بادقت 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 بادقت 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 بادقت 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 به‌صورت چهاررقمی، ماه به‌صورت دورقمی و روز نیز به‌صورت دورقمی ذخیره می‌شود؛ 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 به‌صورت دورقمی، دقیقه به‌صورت دورقمی، ثانیه به‌صورت دورقمی ذخیره می‌شود. 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 بادقت میلی‌ثانیه و اختلاف ساعت با گرینویچ؛ بنابراین، 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 می‌شود.

- 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')) ;
```

۳_۹_ هر یک از این نوع های داده‌ای چگونه در حافظه تخصیص و چگونه پیاده‌سازی شده‌اند؟

- داده‌های متنی (Character)

داده‌های متنی، در حافظه به‌صورت 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 به‌صورت 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 به‌صورت 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 ۲۰۲۳-۰۸-۲۰ 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 به‌صورت 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

۳_۱۰_ چه عملگرهایی برای این نوع ها تعریف شده‌اند؟

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.

۳_۱۱_ اگر لیست‌ها و رشته‌ها و آرایه‌های انجمنی در زبان SQL تعریف شده‌اند، پیاده‌سازی آنها چگونه است؟

در زبان SQL، لیست‌ها، رشته‌ها و آرایه‌های انجمنی به‌عنوان نوع داده‌های تعریف نشده وجود ندارند. بااین‌حال، می‌توان آنها را با استفاده از توابع و عبارات SQL پیاده‌سازی کرد.

- لیست‌ها

لیست‌ها می‌توانند با استفاده از تابع ARRAY() پیاده‌سازی شوند .این تابع یک آرایه از مقادیر را ایجاد می‌کند .به‌عنوان مثال، کد زیر یک لیست از اعداد صحیح ایجاد می‌کند:

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

- رشته‌ها

رشته‌ها می‌توانند با استفاده از تابع CONCAT() پیاده‌سازی شوند .این تابع رشته‌های داده را به یکدیگر متصل می‌کند .به‌عنوان مثال، کد زیر دو رشته را به یکدیگر متصل می‌کند:

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

- آرایه‌های انجمنی

آرایه‌های انجمنی می‌توانند با استفاده از تابع MAP() پیاده‌سازی شوند .این تابع یک آرایه از جفت‌های کلید - مقدار را ایجاد می‌کند .به‌عنوان مثال، کد زیر یک آرایه انجمنی از نام و سن افراد ایجاد می‌کند:

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

۳_۱۲_ اگر اشاره‌گرها و متغیرهای مرجع در زبان SQL تعریف شده‌اند، پیاده‌سازی آنها چگونه است؟

در زبان SQL ، اشاره‌گرها و متغیرهای مرجع به‌عنوان نوع داده‌های تعریف نشده وجود ندارند. بااین‌حال، می‌توان آنها را با استفاده از توابع و عبارات SQL پیاده‌سازی کرد.

- اشاره‌گرها

اشاره‌گرها می‌توانند با استفاده از تابع ROW() پیاده‌سازی شوند. این تابع یک ردیف از یک جدول را به‌عنوان یک اشاره‌گر بازمی‌گرداند. به‌عنوان مثال، کد زیر یک اشاره‌گر به‌ردیف اول جدول PERSON ایجاد می‌کند:

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

- متغیرهای مرجع

متغیرهای مرجع می‌توانند با استفاده از تابع REF() پیاده‌سازی شوند. این تابع یک اشاره‌گر به یک متغیر را ایجاد می‌کند. به‌عنوان مثال، کد زیر یک متغیر مرجع به متغیر x ایجاد می‌کند:

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

۳_۱۳_ در زبان SQL چه سازوکارهایی برای رفع مشکلات نشتی حافظه و اشاره‌گر معلق وجود دارد؟

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

- حذف متغیرها و اشاره‌گرهای استفاده شده

این سازوکار ساده‌ترین و مؤثرترین روش برای جلوگیری از مشکلات نشتی حافظه و اشاره‌گر معلق است. به‌طورکلی، هر متغیر یا اشاره‌گر که دیگر استفاده نمی‌شود باید بلافاصله حذف شود. این کار می‌تواند با استفاده از دستور DROP انجام شود.

به‌عنوان مثال، کد زیر یک متغیر x ایجاد می‌کند و سپس آن را حذف می‌کند:

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

- استفاده از توابع FREE() و CLOSE()

این توابع در سیستم‌های مدیریت پایگاه‌داده (DBMS) مختلف برای آزادکردن حافظه تخصیص‌یافته به متغیرها و اشاره‌گرها استفاده می‌شوند.

به‌عنوان مثال، کد زیر یک اشاره‌گر p به یک ردیف از جدول PERSON ایجاد می‌کند و سپس از تابع FREE() برای آزادکردن حافظه تخصیص‌یافته به آن استفاده می‌کند:

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

سایر روش‌ها:

- استفاده از متغیرهای محلی

متغیرهای محلی تنها در محدوده بلوک کدی که در آن تعریف شده‌اند قابل‌دسترسی هستند. بنابراین، استفاده از متغیرهای محلی به جلوگیری از نشتی حافظه کمک می‌کند.

- عدم استفاده از اشاره‌گرهای بی‌نیاز

اشاره‌گرها به طور خودکار آزاد نمی‌شوند؛ بنابراین، باید از اشاره‌گرهایی که دیگر استفاده نمی‌شوند استفاده نکنید.

- استفاده از ابزارهای عیب‌یابی

ابزارهای عیب‌یابی می‌توانند به شما کمک کنند تا مشکلات نشتی حافظه و اشاره‌گر معلق را شناسایی کنید.

۳_۱۴_ آیا بازیافت‌کننده حافظه وجود دارد و در صورت وجود چگونه پیاده‌سازی شده است؟ در صورتی که بازیافت کنندهٔ حافظه وجود ندارد، زبان با یک زبان دیگر که بازیافت کنندهٔ حافظه دارد مقایسه شود.

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

زبان SQL به طور مستقیم از بازیافت‌کننده حافظه استفاده نمی‌کند، زیرا این زبان برای پرس‌وجو 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 به دست آورد. بنابراین، جزئیات پیاده‌سازی و مدیریت حافظه به عهده سیستم مدیریت پایگاه‌داده است که SQL را اجرا می‌کند.

به‌عنوان 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 پایگاه‌داده 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 پایگاه‌داده دارند.

از طرف دیگر، برخی از زبان‌های برنامه‌نویسی مانند جاوا، پایتون، روبی و غیره از بازیافت‌کننده حافظه به‌صورت خودکار استفاده می‌کنند. این زبان‌ها به‌عنوان 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 به‌صورت دستی حافظه را آزاد کند، زیرا بازیافت‌کننده حافظه این کار را به‌صورت پشت پرده انجام می‌دهد.

به‌عنوان مثال، در زبان جاوا، می‌توانیم از کلاس `java.sql.Connection` برای برقراری ارتباط با پایگاه‌داده 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، بازیافت‌کننده حافظه جاوا این حافظه را آزاد می‌کند و از اتلاف و نشت حافظه جلوگیری می‌کند.

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

۳_۱۵_ دکلاراتیو(declarative) بودن یک زبان برنامه‌نویسی 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 به دست 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 پرس‌وجوها 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 به دست آوریم. این کار به عهده سیستم مدیریت پایگاه‌داده است که SQL را اجرا می‌کند و می‌تواند از الگوریتم‌های مختلفی برای انجام این پرس‌وجو 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 برنامه‌نویسی به چه معناست؟

برخلاف زبان‌های برنامه‌نویسی دکلاراتیو، زبان‌های برنامه‌نویسی امپراتیو به برنامه‌نویس اجازه می‌دهند که بگوید چگونه چیزی را به دست آورد. سبرای 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
- https://openai.com
- https://fa.wikipedia.org/wiki/%D8%A7%D8%B3%E2%80%8C%DA%A9%DB%8C%D9%88%D8%A7%D9%84
- https://blog.faradars.org/
- https://www.roxo.ir/
- https://blog.faradars.org/
- https://bing.com/
- https://stackoverflow.com/
- https://towardsdatascience.com/
- https://learnsql.com/
- https://poe.com
- https://www.datacamp.com/blog/sql-server-postgresql-mysql-whats-the-difference-where-do-i-start
- https://www.coursera.org/articles/sql-vs-mysql
- https://www.dataquest.io/blog/sql-vs-t-sql/
- https://emeritus.org/in/learn/data-science-sql-vs-python/
- https://www.datacamp.com/blog/r-vs-sql-which-to-choose
- https://www.datacamp.com/blog/sql-server-postgresql-mysql-whats-the-difference-where-do-i-start
- Bard (google.com)
- 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