فهرست

[1\_ مقدمه 2](#_Toc150760221)

[1\_1\_ تاریخچه زبان برنامه نویسی SQL 2](#_Toc150760222)

[1\_2\_ زبان برنامهنویسی SQL در ابتدا به چه منظوری تهیه شده است؟ 3](#_Toc150760223)

[1\_3\_ زبان برنامه‌نویسی SQL در چه زمینه‌ها و حوزه‌هایی کاربرد دارد؟ 4](#_Toc150760224)

[1\_4\_ این زبان برنامه نویسی برای رفع چه مشکلاتی ابداع شده و ابداع آن در جهت بهبود چه زبان هایی بوده است؟ 7](#_Toc150760225)

[1\_5\_ این زبان در ابتدای ظهور خود چه مشکلاتی را رفع می کرده است؟ 8](#_Toc150760226)

[1\_6\_ این زبان در مقایسه با زبان های دیگر و به خصوص زبان هایی که شبیه آن هستند چگونه ارزیابی می شود؟ 10](#_Toc150760227)

[1\_7\_ این زبان چه ویژگی های خاصی دارد که آن را از زبان های مشابه آن متمایز می کند؟ 11](#_Toc150760228)

[1\_8\_ این زبان را بر اساس خوانایی، قابلیت اطمینان، هزینه (کارایی و بهره‌وری و همچنین هزینه مورد نیاز برای یادگیری و برنامه‌نویسی) و سایر معیارها مانند قابلیت جابجایی چگونه می‌توان ارزیابی کرد؟ 14](#_Toc150760229)

[1\_9\_ آیا برای پیاده سازی زبان SQL از کامپایلر استفاده شده است یا مفسر یا پیاده سازی ترکیبی؟ 17](#_Toc150760230)

[1\_10\_ 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 شده اند؟ 18](#_Toc150760231)

[1\_11\_ مزیت هر یک از کامپایلر ها و یا مفسرهای نام برده شده چیست؟ 20](#_Toc150760232)

# 1\_ مقدمه

## 1\_1\_ تاریخچه زبان برنامه نویسی SQL

این زبان در دهه 1970 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 به عنوان زبان استاندارد برای دسترسی و مدیریت داده‌های پایگاه‌های داده‌های رابطه‌ای استفاده می‌شود.

تاریخچه زبان برنامه نویسی SQL را می‌توان به سه دوره تقسیم کرد:

* دوره اولیه (1986-1970)

در این دوره، SQL به عنوان یک زبان استاندارد برای دسترسی و مدیریت داده‌های پایگاه‌های داده‌های رابطه‌ای در حال توسعه بود. اولین نسخه SQL در سال 1974 منتشر شد و در سال‌های بعد با انتشار نسخه‌های جدید، ویژگی‌های جدیدی به آن اضافه شد.

* دوره رشد (2003-1986)

در این دوره، SQL به عنوان یک زبان استاندارد پذیرفته شد و به طور گسترده توسط سیستم‌های مدیریت پایگاه داده استفاده شد. در این دوره، ویژگی‌های جدیدی از جمله توابع، رویه‌ها و триггер‌ها به SQL اضافه شد.

* دوره پیشرفت (2003-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.

## 1\_2\_ 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 پایگاه داده 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 فیلتر کردن داده‌ها
* ایجاد گزارش‌ها

SQL به سرعت محبوب شد و امروزه به عنوان زبان استاندارد برای دسترسی و مدیریت داده‌های پایگاه‌های داده‌های رابطه‌ای استفاده می‌شود. این زبان توسط بسیاری از سیستم‌های مدیریت پایگاه داده (DBMS) از جمله MySQL، Oracle، Microsoft SQL Server و PostgreSQL پشتیبانی می‌شود.

برخی از مزایای استفاده از SQL عبارتند از:

* قدرتمند و انعطاف‌پذیر است.
* یادگیری آن آسان است (زیرا دستورات آن مشابه با جملات انگلیسی نوشته میشود.)
* توسط بسیاری از سیستم‌های مدیریت پایگاه داده پشتیبانی می‌شود.

## 1\_3\_ زبان برنامه‌نویسی SQL در چه زمینه‌ها و حوزه‌هایی کاربرد دارد؟

* تجارت و کسب و کار

SQL در بسیاری از کسب و کارها 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 تجزیه و تحلیل داده‌ها استفاده می‌کنند.

* آموزش و پرورش

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، فیلتر کردن، مرتب کردن 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 تجزیه و تحلیل داده‌ها

SQL برای ایجاد تجزیه و تحلیل داده‌ها استفاده می‌شود.

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

## 1\_4\_ این زبان برنامه نویسی برای رفع چه مشکلاتی ابداع شده و ابداع آن در جهت بهبود چه زبان هایی بوده است؟

زبان برنامه نویسی 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 برنامه نویسی 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 برنامه نویسی عمومی برای مدیریت داده‌های ذخیره شده در پایگاه‌های داده‌های رابطه‌ای عبارتند از:

* سادگی و یادگیری آسان
* قدرتمند و انعطاف‌پذیر
* سازگاری با بسیاری از سیستم‌های مدیریت پایگاه داده

## 1\_5\_ این زبان در ابتدای ظهور خود چه مشکلاتی را رفع می کرده است؟

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

1. بازیابی و دستکاری داده ها: قبل از SQL، دسترسی و دستکاری داده ها در پایگاه های داده یک کار پیچیده و مستعد خطا بود. SQL یک زبان استاندارد برای پرس و جو و دستکاری داده های ذخیره شده در پایگاه های داده رابطه ای 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.
2. استقلال داده ها: SQL سطح بالاتری از انتزاع را معرفی کرد و به کاربران اجازه داد تا بدون نیاز به درک مکانیسم های ذخیره سازی و دسترسی به داده های اساسی با پایگاه های 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.
3. یکپارچگی و سازگاری داده ها: SQL مکانیسم هایی را برای اعمال یکپارچگی و سازگاری داده ها در پایگاه های داده رابطه ای معرفی کرد. این ویژگی از تعریف محدودیت برای کلید اصلی، محدودیت برای یکتایی داده‌ها، روابط کلید خارجی و سایر قوانین یکپارچگی پشتیبانی می‌کند. این ویژگی ها به حفظ کیفیت و دقت داده های ذخیره شده در پایگاه های داده کمک کرد.
4. دسترسی همزمان و مدیریت تراکنش: SQL قابلیت های مدیریت تراکنش را معرفی کرد که به چندین کاربر اجازه می داد به طور همزمان به داده ها دسترسی داشته باشند و آنها را اصلاح کنند و در عین حال از ثبات و یکپارچگی پایگاه داده اطمینان حاصل کنند. مکانیسم هایی مانند قفل کردن و جداسازی سطوح را برای مدیریت دسترسی همزمان و جلوگیری از تناقضات داده ها ارائه کرد.
5. امنیت داده ها و کنترل دسترسی: SQL ویژگی هایی را برای اعمال امنیت و کنترل دسترسی در پایگاه های 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.
6. مقیاس پذیری داده ها و بهینه سازی عملکرد: SQL تکنیک های بهینه سازی و طرح های اجرای پرس و جو را برای بهبود عملکرد عملیات پایگاه داده ارائه می‌کند. به مدیران پایگاه داده و توسعه دهندگان این امکان را می‌داد تا کوئری ها، نمایه ها و ساختارهای پایگاه داده را تنظیم و بهینه سازی کنند تا مقیاس پذیری و عملکرد بهتری داشته باشند.

به طور کلی هدف SQL ساده‌سازی مدیریت داده‌ها، بهبود یکپارچگی و سازگاری داده‌ها، فعال کردن دسترسی همزمان، افزایش امنیت داده‌ها و بهینه‌سازی عملکرد عملیات پایگاه داده است. این قابلیت ها SQL را به زبانی قدرتمند و پرکاربرد برای کار با پایگاه های داده رابطه ای تبدیل کرد.

## 1\_6\_ این زبان در مقایسه با زبان های دیگر و به خصوص زبان هایی که شبیه آن هستند چگونه ارزیابی می شود؟

زبان برنامه نویسی 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، فیلتر کردن، مرتب کردن، ویرایش و ایجاد گزارش‌ها طراحی شده است.
* سازگاری با بسیاری از سیستم‌های مدیریت پایگاه داده: SQL توسط بسیاری از سیستم‌های مدیریت پایگاه داده محبوب پشتیبانی می‌شود. این امر توسعه‌دهندگان را قادر می‌سازد تا مهارت‌های خود را در SQL در سیستم‌های مدیریت پایگاه داده مختلف اعمال کنند.

زبان های برنامه نویسی دیگری مبتنی بر SQL و براساس آن ساخته شده اند، و دارای قابلیت های ویژه و خاصی می‌باشند از قبیل PL/SQL, T-SQL, PLpgSQL اما هنوز زبان SQL به عنوان یک زبان استاندارد برای مدیریت داده‌های پایگاه‌های داده‌های رابطه‌ای شناخته می‌شود.

## 1\_7\_ این زبان چه ویژگی های خاصی دارد که آن را از زبان های مشابه آن متمایز می کند؟

1. زبان اظهاری: SQL یک زبان اظهاری است، به این معنی که شما مشخص می‌کنید که چه چیزی را می‌خواهید بازیابی کنید یا از یک پایگاه داده دستکاری کنید بدون اینکه مشخص کنید چگونه آن را انجام دهید. شما نتایج مورد نظر را توصیف می‌کنید و سیستم مدیریت پایگاه داده (DBMS) کارآمدترین راه را برای اجرای دستکاری تعیین می‌کند.
2. 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.
3. زبان تعریف داده‌ها (DDL): SQL شامل یک DDL است که شما را قادر می‌سازد تا ساختار پایگاه داده 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.
4. زبان دستکاری داده‌ها (DML): SQL یک DML ارائه می‌دهد که به شما امکان می‌دهد داده‌ها را از پایگاه داده 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.
5. 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 پایگاه داده رابطه‌ای بسیار مهم است.
6. 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.
7. 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 سرعت بخشیدن به جستجو و مرتب‌سازی ایجاد کرد.
8. 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 پایگاه داده توسط چندین کاربر یا فرآیند پشتیبانی می‌کند.
9. بهینه‌سازی کوئری: هدف بهینه‌سازهای SQL ایجاد کارآمدترین برنامه‌های اجرایی برای کوئری‌ها است. DBMS نحو و ترکیب کوئری‌ها، آمار جداول، و دیگر موارد را تجزیه و تحلیل می‌کند تا بهینه‌ترین راه برای اجرای یک کوئری کمینه کردن منابع مورد استفاده‌ی برنامه و زمان اجرای آن را مشخص کند.
10. قابلیت حمل: SQL یک زبان استاندارد است که توسط اکثر سیستم‌های مدیریت پایگاه داده رابطه‌ای پشتیبانی می‌شود. این قابلیت حمل به شما امکان می‌دهد کد SQL بنویسید که می‌تواند در پلتفرم‌های مختلف پایگاه داده 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.

## 1\_8\_ این زبان را بر اساس خوانایی، قابلیت اطمینان، هزینه (کارایی و بهره‌وری و همچنین هزینه مورد نیاز برای یادگیری و برنامه‌نویسی) و سایر معیارها مانند قابلیت جابجایی چگونه می‌توان ارزیابی کرد؟

1. قابلیت اطمینان: SQL یک زبان قابل اعتماد برای کار با پایگاه داده‌های رابطه‌ای است، زیرا از استانداردی پیروی می‌کند که به‌طور گسترده توسط بسیاری از سیستم‌های مدیریت پایگاه داده 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.
2. خوانایی: SQL یک زبان خوانا است، زیرا از کلمات کلیدی و نحوی استفاده می‌کند که نزدیک به زبان طبیعی هستند و به راحتی 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.
3. 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 می توانند ناکارآمد باشند اگر به‌خوبی نوشته یا بهینه نشده باشند. برخی از عواملی که می‌توانند بر کارایی کوئری‌های SQL تأثیر بگذارند، اندازه و ساختار پایگاه داده، استفاده از فهرست‌ها، پیوست‌ها، کوئری‌های فرعی و توابع هستند. برخی از زبان‌هایی که شبیه SQL هستند، مانند DAX، برای تجزیه و تحلیل داده‌ها و گزارش‌دهی تخصصی هستند و از موتور محاسباتی و نحو متفاوتی استفاده می‌کنند که می‌تواند کارایی و دقت درخواست‌ها را بهبود بخشد.
4. زمان و منابع مورد نیاز برای یادگیری: SQL یک زبان نسبتا 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 گسترش‌ یافته‌ی 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.
5. 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 پایگاه داده 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 همه کاره برای کار با داده‌ها، به‌خصوص داده‌های رابطه‌ای است. SQL به شما امکان ایجاد، دستکاری، نوشتن کوئری و تجزیه و تحلیل 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.

## 1\_9\_ آیا برای پیاده سازی زبان SQL از کامپایلر استفاده شده است یا مفسر یا پیاده سازی ترکیبی؟

برای پیاده سازی این زبان هم از کامپایلر و هم ازمفسر استفاده شده است. کامپایلرها کد SQL را به ماشین کد تبدیل کرده تامستقیما توسط CPU قابل درک و اجرا باشد. با این حال کامپایل کردن کد SQL در صورت پیچیده بودن کد میتواند فرایندی زمان بر باشد.

اما مفسر ها کد SQL را مستقیما بدون اینکه آن را به ماشین کد تبدیل کنند اجرا میکنند. اما اینکار بسیار کند است زیرا مفسر ها باید کد SQL را به صورت خط به خط تفسیر کند، با این حال مفسر ها سریع تر از کامپایلر ها کد SQL را توسعه میدهند.

پیاده سازی ترکیبی کامپایلر و مفسر نیز ممکن است که در بسیار از موارد یکی از بهترین گزینه ها است. در این نوع پیاده سازی اگر کد SQL پیچیده باشد توسط مفسرها به صورت مستقیم اجرا میشود در غیر این صورت کد SQL به ماشین کد تبدیل شده و سپس کامپایل می شود. یعنی اجرای سریع تر در کنار توسعه سریع تر برای قطعه کدهای مختلف وجود دارد.

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

* سرعت مورد نیاز اجرای کد SQL
* پیچیدگی کد SQL
* زمان توسعه مورد نیاز

## 1\_10\_ 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 شده اند؟

برخی از محبوب ترین کامپایلرها برای زبان SQL عبارتند از:

* Microsoft SQL Server

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

ویژگی ها: پشتیبانی از انواع داده های پیشرفته، عملکرد بالا، قابلیت اطمینان

قیمت: پولی

پیچیدگی: بالا

* Oracle Database

توسط تیم توسعه Oracle Database در شرکت اوراکل توسعه یافته است.

ویژگی ها: پشتیبانی از انواع داده های پیشرفته، عملکرد بالا، قابلیت اطمینان

قیمت: پولی

پیچیدگی: بالا

* MySQL

توسط تیم توسعه MySQL در شرکت Oracle توسعه یافته است.

ویژگی ها: پشتیبانی از انواع داده های پیشرفته، عملکرد خوب، قیمت مناسب

قیمت: رایگان یا پولی

پیچیدگی: متوسط

* PostgreSQL

توسط تیم توسعه PostgreSQL در بنیاد PostgreSQL توسعه یافته است.

ویژگی ها: پشتیبانی از انواع داده های پیشرفته، عملکرد خوب، قابلیت اطمینان بالا

قیمت: رایگان یا پولی

پیچیدگی: بالا

برخی از محبوب ترین مفسرها برای زبان SQL عبارتند از:

این مفسرها معمولا سبک تر و سریع تر از کامپایلر ها هستند، اما ویژگی پیشرفته ای ارائه نمیدهند.

* SQLite

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

ویژگی ها: سبک وزن، سریع، رایگان

قیمت: رایگان

پیچیدگی: پایین

* Firebird

توسط تیم توسعه Firebird در شرکت Firebird Foundation توسعه یافته است.

ویژگی ها: عملکرد خوب، ویژگی های پیشرفته، رایگان

قیمت: رایگان

پیچیدگی: متوسط

* H2

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

ویژگی ها: سبک وزن، سریع، رایگان

قیمت: رایگان

پیچیدگی: پایین

## 1\_11\_ مزیت هر یک از کامپایلر ها 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، ویژگی های پیشرفته
* معایب: قیمت بالا، پیچیدگی

Oracle Database

* مزیت: عملکرد بالا، قابلیت اطمینان، ویژگی های پیشرفته
* معایب: قیمت بالا، پیچیدگی

MySQL

* مزیت: عملکرد خوب، ویژگی های پیشرفته، قیمت مناسب
* معایب: قابلیت اطمینان کمتر از Microsoft SQL Server و Oracle Database

PostgreSQL

* مزیت: عملکرد خوب، ویژگی های پیشرفته، قابلیت اطمینان بالا
* معایب: پیچیدگی بیشتر از MySQL

SQLite

* مزیت: سبک وزن، سریع، رایگان
* معایب: ویژگی های پیشرفته کمتر از کامپایلرها و مفسرهای دیگر

Firebird

* مزیت: عملکرد خوب، ویژگی های پیشرفته، رایگان
* معایب: پیچیدگی بیشتر از SQLite

H2

* مزیت: سبک وزن، سریع، رایگان
* معایب: ویژگی های پیشرفته کمتر از کامپایلرها و مفسرهای دیگر

# 3\_1\_ در زبان برنامه‌نویسی انتخاب شده انقیاد نوع و مقدار چگونه و در چه زمانی انجام می‌شود؟ آیا تعاریف متغیرها ضمنی است یا صریح و یا هر دو نوع تعریف وجود دارد؟ با ذکر مثال توضیح داده شود.

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

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

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

CREATE TABLE Employees (

ID INT,

Name VARCHAR(50),

Age INT,

Salary DECIMAL(10, 2)

);

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

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

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

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

CREATE PROCEDURE GetEmployeeByID

@EmployeeID INT

AS

BEGIN

SELECT \* FROM Employees WHERE ID = @EmployeeID;

END;

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

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

# 3\_2\_ آیا در زبان SQL متغیرهای ایستا، پویا در پشته، پویا در هیپ به‌طور صریح، پویا در هیپ به‌طور ضمنی وجود دارند؟ با مثال توضیح داده شود. برای توصیف هر یک از موارد یک قطعه کد نوشته شود. همچنین توضیح داده شود که هر یک این متغیرها در این زبان چگونه پیاده‌سازی شده‌اند.

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

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

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

-- Declare and initialize a static variable

DECLARE @MyCounter INT = 0;

-- Print the value of the variable

PRINT @MyCounter;

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

-- Declare a dynamic variable in stack

DECLARE @MyName VARCHAR(50);

-- Assign a value to the variable at run time

SET @MyName = 'Bing';

-- Print the value of the variable

PRINT @MyName;

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

-- Declare an explicit dynamic variable in heap

DECLARE @MyDate DATE;

-- Assign a value to the variable at run time

SELECT @MyDate = GETDATE();

-- Print the value of the variable

PRINT @MyDate;

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

-- Declare an implicit dynamic variable in heap

SELECT @MyNumber = 10;

-- Print the value of the variable

PRINT @MyNumber;

# 3\_3\_ 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.

# 3\_4\_ آیا حوزه تعریف در این زبان ایستا است یا پویا؟ با ذکر مثال توضیح دهید.

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

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

CREATE TRIGGER add\_column AFTER INSERT ON test FOR EACH ROW

BEGIN

DECLARE new\_column VARCHAR(50);

SET new\_column = CONCAT('col', NEW.id);

SET @sql = CONCAT('ALTER TABLE test ADD COLUMN ', new\_column, ' VARCHAR(50)');

PREPARE stmt FROM @sql;

EXECUTE stmt;

DEALLOCATE PREPARE stmt;

END;

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

# 

# 3\_5\_ در صورتی که زبان حوزه تعریف ایستا را پشتیبانی می کند و بخواهیم امکان استفاده از حوزه تعریف پویا به آن بیافزاییم، چه تغییراتی در زبان باید ایجاد کنیم؟

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

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

CREATE DYNAMIC TABLE test (

id INT PRIMARY KEY,

name VARCHAR(50)

);

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

CREATE RULE add\_column ON test

AFTER INSERT

FOR EACH ROW

BEGIN

DECLARE new\_column VARCHAR(50);

SET new\_column = CONCAT('col', NEW.id);

ALTER TABLE test ADD COLUMN new\_column VARCHAR(50);

END;

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

CREATE RULE drop\_column ON test

AFTER DELETE

FOR EACH ROW

BEGIN

DECLARE old\_column VARCHAR(50);

SET old\_column = CONCAT('col', OLD.id);

ALTER TABLE test DROP COLUMN old\_column;

END;

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

# 3\_6\_ پس از تغییر زبان، قطعه کدی نوشته شود که توسط آن حوزۀ تعریف پویا استفاده شود. همچنین اگر زبان هر دو حوزۀ تعریف را پشتبانی می کند، هر دو مورد توصیف شوند.

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

CREATE DYNAMIC TABLE test (

id INT PRIMARY KEY,

name VARCHAR(50)

);

CREATE RULE add\_column ON test

AFTER INSERT

FOR EACH ROW

BEGIN

DECLARE new\_column VARCHAR(50);

SET new\_column = CONCAT('col', NEW.id);

ALTER TABLE test ADD COLUMN new\_column VARCHAR(50);

END;

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

CREATE STATIC TABLE test (

id INT PRIMARY KEY,

name VARCHAR(50)

);

# 

# 3\_7\_ بلوک ها در این زبان چگونه تعریف شده اند؟ آیا کلمات کلیدی ویژه ای برای اعمال تغییر در حوزۀ تعریف متغیرها وجود دارند؟

بلوک ها در زبان SQL به عنوان یک واحد منطقی از دستورات تعریف شده اند که می توانند در یک تراکنش یا یک برنامه اجرا شوند. بلوک ها می توانند شامل متغیرها، ثابت ها، مقادیر پیش فرض، توابع، زیر برنامه ها، دستورات کنترل جریان و خطاها باشند. بلوک ها می توانند درون یکدیگر تودرتو شوند و محدوده متغیرها را تعیین کنند. برای شروع و پایان یک بلوک، از کلمات کلیدی BEGIN و END استفاده می شود. برای مثال، بلوک زیر یک متغیر به نام x را تعریف می کند و مقدار آن را به 10 تغییر می دهد:

BEGIN

DECLARE x INT DEFAULT 0;

SET x = 10;

END

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

* DECLARE:

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

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

* SET:

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

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

* DEFAULT:

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

CREATE TABLE Users (

UserID INT NOT NULL,

UserName NVARCHAR(50) NOT NULL,

Email NVARCHAR(100) DEFAULT 'example@example.com' -- Set the default value for the email column

);

* LOCAL:

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

BEGIN

DECLARE @x INT; -- Define a local variable named x

SET @x = 10;

PRINT @x; -- Print x

END

* GLOBAL:

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

DECLARE @@x INT; -- Define a global variable named x

SET @@x = 10;

BEGIN

PRINT @@x; --Print x

END

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

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

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

انواع داده در SQL به دو دسته کلی تقسیم می‌شوند:

* داده‌های پایه :(Primitive)

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

* داده‌های مشتق (Derived):

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

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

داده‌های پایه در SQL به شرح زیر هستند:

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

* char(n)

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

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

CREATE TABLE customers (

id INT,

name CHAR(10)

);

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

* varchar(n)

رشته متنی با طول متغیر n ، طول رشته، در هنگام وارد کردن مقدار برای رشته مشخص می‌شود. اگر مقدار وارد شده برای این ستون، کوتاه‌تر از 20 کاراکتر باشد، هیچ کاراکتری به آن اضافه نمی‌شود. اگر مقدار وارد شده برای این ستون، بلندتر از 20 کاراکتر باشد، رشته بدون تغییر باقی می‌ماند.

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

CREATE TABLE customers (

id INT,

name VARCHAR(20)

);

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

* nchar(n)

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

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

CREATE TABLE customers (

id INT,

name NCHAR(10)

);

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

* nvarchar(n)

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

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

CREATE TABLE customers (

id INT,

name NVARCHAR(20)

);

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

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

* Tinyint

عدد صحیح بین 0 تا 255

مثال: در این مثال یک جدول با ستون age از نوع TINYINT ساخته شده است، یعنی داده های این ستون میتواند مقادیری بین 0 تا 255 باشد.

CREATE TABLE customers (

id INT,

age TINYINT

);

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

* Smallint

عدد صحیح بین 32768- تا 32767

مثال: در این مثال یک جدول با ستون age از نوع SMALLINT ساخته شده است، یعنی داده های این ستون میتواند مقادیری بین 32768- تا 32767 باشد.

CREATE TABLE customers (

id INT,

age SMALLINT

);

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

* Int

عدد صحیح بین 2147483648- تا 2147483647

مثال: در این مثال یک جدول با ستون age از نوع INT ساخته شده است، یعنی داده های این ستون میتواند مقادیری بین 2147483648- تا 2147483647 باشد.

CREATE TABLE customers (

id INT,

age INT

);

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

* Bigint

عدد صحیح بین 922337203685477588- تا 9223372036854775807

مثال: در این مثال یک جدول با ستون age از نوع BIGINT ساخته شده است، یعنی داده های این ستون میتواند مقادیری بین 922337203685477588- تا 9223372036854775807 باشد.

CREATE TABLE customers (

id INT,

age BIGINT

);

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

* Float

عدد اعشاری با دقت 6 یا 24 رقم

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

CREATE TABLE customers (

id INT,

price FLOAT

);

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

* Real

عدد اعشاری با دقت 7 رقم

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

CREATE TABLE customers (

id INT,

price REAL

);

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

* decimal(p,s)

عدد اعشاری با دقت p رقم و s رقم اعشار

مثال: در این مثال یک جدول با ستون price از نوع DECIMAL ساخته شده است. بنابراین، مقدار عدد اعشاری که می‌توان برای این ستون وارد کرد، باید بین 0 تا 99999999999.99 باشد. دقت عدد اعشاری، برابر با 2 رقم است.

CREATE TABLE customers (

id INT,

price DECIMAL(10,2)

);

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

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

* Date

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

مثال: در مثال زیر یک جدول با ستون birth\_date از نوع date تعریف شده است.

CREATE TABLE customers (

id INT,

birth\_date DATE

);

INSERT INTO customers (id, birth\_date) VALUES (1, '2023-08-02');

* Time

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

مثال: در این مثال یک جدول با ستون order\_time از نوع TIME تعریف شده است.

CREATE TABLE customers (

id INT,

order\_time TIME

);

INSERT INTO customers (id, order\_time) VALUES (1, '12:34:56');

* Datetime

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

مثال: در این مثال یک جدول با ستون last\_login از نوع DATETIME تعریف شده است.

CREATE TABLE customers (

id INT,

last\_login DATETIME

);

INSERT INTO customers (id, last\_login) VALUES (1, '2023-08-20 12:34:56');

* datetimeoffset

تاریخ و زمان با دقت میلی‌ثانیه و اختلاف ساعت با گرینویچ. نابراین، مقدار تاریخ و زمانی که می‌توان برای این ستون وارد کرد، باید بین 0001-01-01 00:00:00 تا 9999-12-31 23:59:59 باشد. دقت تاریخ و زمان، برابر با 7 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 1 و برای false مقدار 0 در نظر گرفته می شود.

* Bit

یک بیت

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

CREATE TABLE customers (

id INT,

active BIT

);

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

* Boolean

یک عدد صحیح 0 یا 1

مثال: در مثال زیر یک جدول با ستون is\_active با نوع داده BOOLEAN تعریف شده است.

CREATE TABLE customers (

id INT,

is\_active BOOLEAN

);

INSERT INTO customers (id, is\_active) VALUES (1, TRUE);

داده‌های مشتق در SQL به شرح زیر هستند:

* داده‌های آرایه (Array)

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

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

CREATE TABLE customers (

id INT,

names ARRAY(VARCHAR(255))

);

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

* داده‌های (XML)

این نوع داده‌ها، برای ذخیره داده‌های XML استفاده می‌شوند.

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

CREATE TABLE customers (

id INT,

profile XML

);

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

* داده‌های USER-DEFINED TYPE

نوع داده‌ای که توسط کاربر تعریف شده است.

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

CREATE TYPE my\_type AS (

id INT,

name VARCHAR(255)

);

CREATE TABLE customers (

id INT,

name my\_type

);

INSERT INTO customers (id, name) VALUES (1, my\_type(1, 'John Doe'));

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

* داده‌های متنی (Character)

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

برای مثال، اگر طول رشته متنی برابر با 10 باشد، آرایه متنی، 10 کاراکتر را در حافظه اشغال خواهد کرد.

* داده‌های عددی (Numeric)

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

برای مثال، اگر عدد برابر با 123456 باشد، آرایه عددی، 6 عدد را در حافظه اشغال خواهد کرد.

* داده‌های تاریخ و زمان (Datetime)

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

برای مثال، اگر تاریخ و زمان برابر با 2023-08-20 12:45:00 باشد، آرایه تاریخ و زمان، 14 عدد را در حافظه اشغال خواهد کرد. شش عدد اول آرایه، برای ذخیره تاریخ استفاده می‌شوند. هشت عدد بعدی آرایه، برای ذخیره زمان استفاده می‌شوند.

* داده‌های منطقی (Boolean)

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

برای مثال، مقدار منطقی true با مقدار بیت 1 و مقدار منطقی false با مقدار بیت 0 ذخیره می‌شود.



چه عملگرهایی برای این نوع ها تعریف شده اند؟

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

* عملگرهای مقایسه (Comparison Operators)

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

* = مساوی
* <> نابرابر
* < کوچکتر از
* <= کوچکتر یا مساوی
* < بزرگتر از
* >= بزرگتر یا مساوی
* عملگرهای منطقی (Logical Operators)

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

* AND و
* OR یا
* NOT نه
* عملگرهای تخصیص (Assignment Operators)

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

* = تخصیص
* += جمع و تخصیص
* -= تفریق و تخصیص
* \*= ضرب و تخصیص
* /= تقسیم و تخصیص
* عملگرهای دودویی (Binary Operators)

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

* + جمع
* تفریق
* ضرب
* / تقسیم
* % باقی‌مانده
* \*\* توان
* عملگرهای یک‌تایی (Unary Operators)

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

* + + مثبت
  + - منفی
  + ! معکوس منطقی

برای انواع داده‌های مشتق، عملگرهای زیر تعریف شده است:

* برای انواع داده‌های آرایه عملگرهای زیر تعریف شده است:
* []: دسترسی به عنصر یک آرایه
* @: طول یک آرایه
* برای انواع داده‌های XML، عملگرهای زیر تعریف شده‌اند:
* . : دسترسی به عنصر یک سند XML
* // : دسترسی به تمام عناصر یک سند XML که با یک الگوی خاص مطابقت دارند.

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

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

* لیست ها

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

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

* رشته ها

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

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

* آرایه های انجمنی

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

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

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

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

* اشاره گرها

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

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

* متغیرهای مرجع

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

DECLARE x INT;

SELECT REF(x);

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

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

* حذف متغیرها و اشاره گرهای استفاده شده

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

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

DECLARE x INT;

x = 1;

DROP x;

* استفاده از توابع FREE() و CLOSE()

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

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

DECLARE p ROW;

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

FREE p;

سایر روش ها:

* استفاده از متغیرهای محلی

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

* عدم استفاده از اشاره گرهای بی‌نیاز

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

* استفاده از ابزارهای عیب‌یابی

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

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

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

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

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

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

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

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

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

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

SELECT \* FROM customers;

این دستور به ما می‌گوید که چه چیزی را می‌خواهیم (تمام رکوردهای جدول customers) ولی نمی‌گوید که چگونه آن را بدست آوریم. این کار به عهده سیستم مدیریت پایگاه داده است که SQL را اجرا می‌کند و می‌تواند از الگوریتم‌های مختلفی برای انجام این پرس و جو استفاده کند. بنابراین، SQL یک زبان برنامه‌نویسی دکلاراتیو است که تنها هدف را مشخص می‌کند و جزئیات را پنهان می‌کند.

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

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

# منابع:

* <https://dev.mysql.com>
* https://docs.oracle.com
* https://azaronline.com
* https://Poe.com
* https://Bing.com
* https://stackoverflow.com
* https://ostovaee.ir
* https://www.w3schools.com/sql
* https://www.roxo.ir/series
* https://blog.faradars.org
* https://www.mongard.ir
* https://sariasan.com
* https://liangroup.net
* https://sitedesign-co.com
* https://sabzdanesh.com
* https://maktabkhooneh.org/learn/d
* <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://bard.google.com/chat/d0b20b3f587eb0c6?hl=en)