لغة SQL

مقدمة

تقدم معظم نظم إدارة قواعد المعطيات التجارية لغة استفسار معيارية قريبة من المستثمر، وتُعتبر لغة SQL أكثر هذه اللغات انتشاراً. تتضمن لغة SQL تركيباً من الجبر العلاقاتي والحساب العلاقاتي، وتقدم إمكانات إضافية تسمح بتعريف بنية المعطيات وإضافة وتعديل المعطيات في قاعدة المعطيات، وتحديد طرق الوصول إليها.

تم تطوير النسخة الأصلية من لغة SQL في مخبر البحوث التابع لشركة IBM في بداية السبعينيات، وأطلق عليها اسم Sequel ، وكانت جزءاً من مشروع نظام إدارة قواعد المعطيات R الذي بدأ العمل عليه في العام 1970، ثم خضعت هذه اللغة إلى العديد من عمليات التحديث والتطوير، وتغير اسمها إلى SQL (Structured Query Language).

في عام 1986 نشر معهد المقاييس الأمريكي ANSI وهيئة المقاييس العالمية (ISO) لغة SQL المعيارية. سميت باسم SQL-86.

كما قامت شركة IBM بنشر معيار خاص بها للغة SQL وتوسيع لهذا المعيار المعياري عام 89، ثم اعتمدت الشركات التي تطور نظم قواعد المعطيات هذه اللغة، ومازالت معتمدة حتى اليوم.

سنعرض البنى والمفاهيم الأساسية للغة SQL ونترك التفاصيل الخاصة بكل نظام إدارة قواعد معطيات للقارئ الذي يمكنه الرجوع إلى دليل اللغة للاطلاع على هذه التفاصيل.

1. تعريف بلغة SQL

تضم لغة SQL طيفاً واسعاً من التعليمات، ويمكن اعتبارها مجموعة من اللغات المتكاملة:

- لغة الاستفسار **Query Language:** تسمح بتحديد المعطيات المطلوب استرجاعها.
- لغة تعريف المعطيات(**DDL(Data Definition Language**: تقدم التعليمات اللازمة لتعريف قاعدة المعطيات مثل تعريف وتعديل مخطط علاقة (جدول)، وحذف علاقة، وبناء الفهارس، وغيرها.
- لغة التعامل مع المعطيات **DML (Interactive Data Manipulation Language):** وتحوي التعليمات اللازمة لإضافة، وحذف وتعديل حدوديات (أسطر) في قاعدة المعطيات.

- التحكم في المناقلات **Transaction Control**: وتشمل التعليمات التي تسمح بتحديد بداية كل مناقلة ونهايتها، وتعليمات قفل المعطيات والتحكم في الوصول المتزامن.
- تعريف السماحيات: تحوي لغة **SQL** تعليمات لتحديد حقوق الوصول إلى محتويات قاعدة المعطيات.

في الأمثلة التالية سنستخدم قاعدة معطيات حول المصارف يتضمن مخططها العلاقات (الجداول) التالية:

```
Branch = (branch_name, branch_city, assets)
Customer = (customer_name, customer_street, city)
Loan = (branch_name, loan_number, amount)
Borrower = (customer_name, loan_number)
Account = (branch_name, account_number, balance)
Depositor = (customer_name, account_number)
```

2. لغة الاستعلام

1- البنية الأساسية للاستعلام في لغة SQL

تتألف البنية الأساسية للاستعلام في لغة SQL من ثلاثة أجزاء (عبارات) أساسية هي : **Select, From, Where .**

تُستخدم العبارة Select لتحديد الأعمدة التي نريد استرجاعها وهي تكافئ عملية الإسقاط في الجبر العلاقاتي، وتُستخدم العبارة From لتحديد العلاقات المستخدمة في عملية الاختيار وهي تكافئ الجداء الديكارتي في الجبر العلاقاتي، أما الجزء Where فيتضمن قضية منطقية يجب أن تحققها الواصفات الموجودة في العلاقات التي جرى تحديدها في جزء From وهي تكافئ الاختيار في الجبر العلاقاتي.

ويصبح الشكل العام للاستعلام في لغة SQL كمايلي:

```
select  A1 , A2 ,....,An
from    r1 ,r2 ,.....,rm
where   P
```

حيث: A_i هي واصفات.

r_i علاقات.

P قضية.

سنناقش فيما يلي الأجزاء الثلاثة في استعلام بلغة SQL.

فقرة الاختيار Select

فقرة الاختيار **Select (Select clause)**: تُستخدم لتحديد قائمة الواصفات المطلوب إظهارها في نتيجة الاستعلام.

مثال: لإيجاد أسماء جميع الأفرع في علاقة القروض، نستخدم التعليمة:

```
select branch_name
from Loan;
```

والنتيجة هي علاقة تحوي واصفاً واحداً (branch_name)

تسمح لغة SQL بتكرار الحدوديات في العلاقة أو في نتيجة استعلام، ولحذف التكرار نضيف كلمة distinct بعد Select.

```
select distinct branch_name
from Loan;
```

يستخدم الرمز "*" للدلالة على اختيار جميع الواصفات من علاقة.

مثال:

```
select *
from Loan;
```

ويمكن أن تحوي فقرة select تعابير حسابية تستخدم العمليات + ، - ، * و /

مثال:

```
select branch_name, loan_number, amount * 100
from Loan;
```

فقرة Where

فقرة **Where** تُستخدم لتحديد الشروط التي يجب أن تحققها الحدوديات المختارة، فمثلاً لإيجاد أرقام القروض في الفرع **"perryridge"** والتي تزيد عن **1200** نكتب :

```
select loan_number
from Loan
where branch_name = "perryridge" and amount >1200;
```

تستخدم المعاملات المنطقية and، or و not في فقرة **where** وعمليات المقارنة <، <=، >=، = و <> و between و not between.

مثال:

```
where amount between 90000 and 100000 ;
```

فقرة From

فقرة **From** تعرف فقرة **From** العلاقات المراد اختيار الحدوديات منها، وعادة ما يتم التعبير الدمج من خلال وضع العلاقات المراد دمجها ضمن عبارة **From** ووضع شرط الدمج ضمن عبارة **Where**.

مثال: إذا أردنا دمج علاقتي القروض **Loan** والمقترضين **Borrower** فإننا نكتب :

```
select distinct customer_name, borrower. loan_number
from Borrower, Loan
where borrower.loan_number = loan.loan_number;
```

ويلاحظ في المثال السابق أننا استخدمنا relation_name.attribute_name في فقرة **select** بسبب وجود نفس اسم الواصف في أكثر من علاقة وذلك لتجنب الالتباس.

2- إعادة التسمية

تجري إعادة التسمية للعلاقات والواصفات باستخدام الفقرة as :

```
old_name as new_name
```

مثال: لإيجاد أسماء وأرقام قروض جميع الزبائن الـذين حصـلوا علـى قـرض مـن فـرع Perryridge، مع التعويض عن اسم العمود loan_number باسم loan-id نكتب:

```
select distinct customer_name, borrower.loan_number as loan-id
from Borrower, Loan
where Borrower.loan_number = Loan.loan_number and
        branch_name= "Perryridge"
```

2- إعادة التسمية، متحولات الحدودية

يمكن أيضاً استخدام عبارة "as" لتعريف متحولات من نمط حدودية، وهذا يفيد في أعطاء أسماء رديفة للعلاقات بهدف اختصار الكتابة أو استخدام عدة نسخ من العلاقة في الاستفسار نفسه.

مثال1: لإيجاد جميع الزبائن الذين حصلوا على قرض من المصرف (أسماء الزبـائن وأرقـام قروضهم)

```
select distinct customer_name, T.loan_number
from Borrower as T, Loan as S
where T.loan_number = S.loan_number
```

يلاحظ من المثال السابق أن المتحول الحدودي T يرمز إلى حدودية لا على التعيين من العلاقـة Borrower.

مثال2: لنفترض أننا نحتاج إلى معرفة جميع الفروع التي توجد في نفس المدينة التي يوجد فيها الفرع Perryridge. لحل هذا الاستعلام نستخدم متحوليين الأول S والثاني T يشير كلاهما إلـى العلاقة Branch-Schema.

Branch_name	Branch_city	Assets	
perryridge			← S
Y			← T

وتُستخدم التعليمة:

```
select T.branch_name
from Branch  T,Branch S
where S.city = T.city and S.branch_name= "perryridge";
```

3- العمليات على سلسلة المحارف

تسمح SQL باستخدام توابع مختلفة لسلسلة الأحـرف مثـل وصـل سلسـلتين Concatenating باستخدام (“||”)، واستخراج جزء من السلسلة، وإيجاد طول سلسلة الأحـرف، والتحويـل بـين الأحرف الصغيرة والكبيرة، وغيرها. إن أكثر العمليات استخداماً هـي التشـابه الجزئـي like ونصف هنا التشابه باستخدام حرفين:

%: للدلالة على أي سلسلة أحرف جزئية.

under score: للدلالة على أي حرف.

مثال: للبحث عن الفروع التي تحوي سلسلة الأحرف idge في أي موقع من اسم الفرع نسـتخدم التعليمة:

```
select branch_name
from Branch
where branch_name Like "%idge%” ;
```

وتُستخدم "_ _ _" للدلالة على أي سلسلة أحرف مؤلفة من ثلاثة أحرف بالضبط

ويمكن استخدام التعبير not like لإيجاد عدم وجود تشابه ضمن سلسلة الأحرف.

4- ترتيب الحدوديات الناتجة

يمكن للمستثمر أن يتحكم في ترتيب الحدوديات في العلاقة الناتجة، وذلك باستخدام عبارة Order by. فمثلاً لاستخراج قائمة مرتبة ترتيباً أبجدياً بأسماء الزبائن، والذين حصلوا على قرض مـن الفرع “Perryridge” نكتب:

```
select distinct customer_name
from Borrower, Loan
where Borrower.loan_number = Loan.loan_number
      and branch_name = "perryridge"
      order by customer_name;
```

إن القائمة الناتجة ستكون مرتبة، ويمكن تحديد طريقة الترتيب تصـاعدياً asc أو تنازليـاً desc (descending ascending)، كما يمكن أن يطلب الترتيب على عدة واصفات.

مثال: لنفترض أننا نريد قائمة القروض مرتبة ترتيباً تنازلياً حسب مبلغ القـرض، وفـي حـال وجود عدة قروض لها نفس المبلغ، نقوم بترتيبها تصاعدياً حسب رقم القرض.

نكتب:

```
select *
from Loan
order by amount desc, loan_number asc
```

إن كلفة إجراء ترتيب على عدد كبير من الحدوديات هي كلفة عاليـة، ولـذلك يجـري إجـراء الترتيب فقط في حال الضرورة.

5- العمليات على المجموعات

تسمح لغة SQL بالعمليات union , intersect و except المقابلة للاجتماع والتقاطع والفرق فـي الجبر العلاقاتي. يجب أن تكون العلاقات التي تطبق عليها هذه العمليات متجانسـة (لهـا نفـس الواصفات).

سوف نبين كيفية استخدام هذه العمليات من خلال أمثلة.

1- الاجتماع

لإيجاد مجموعة كل الزبائن المتعاملين مع المصرف سواء مقترضين أو مودعين نكتب:

```
(select customer_name  from depositor)
union
(select customer_name  from borrower)
```

تحذف عملية الاجتماع الحدوديات المكررة آلياً مثل عملية الاختيار. فـي حـال الرغبـة فـي الاحتفاظ بالتكرار، نكتب (union all)، فتحوي النتيجة حدوديات مكررة بعدد ظهورها في كلٍ من العلاقتين.

2- التقاطع

لإيجاد الزبائن الذين حصلوا على قرض ولديهم حساب في المصرف نكتب:

```
(select distinct customer_name from depositor)
intersect
(select distinct customer_name from borrower)
```

وتُستخدم العبارة intersect all لإظهار الحدوديات المكررة، وعندها تحوي النتيجة الزبائن بعـدد القروض والإيداعات المشتركة.

3- الفرق

لإظهار جميع الزبائن الذين لديهم حساب في المصرف ولم يستفيدوا من قرض منه نكتب:

```
(select distinct customer_name from Depositor)
except
(select customer_name from Borrower)
```

يمكن استخدام العبارة "except all" لإظهار الحدوديات المكررة، وبذلك نستطيع الحصول علـى

أسماء الأشخاص الذين لديهم عدد من الحسابات أكبر من عدد القروض التي اقترضوها.

6- التوابع التجميعية

تُطبق التوابع التجميعية على مجموعة من القيم وتعيد قيمة واحدة. أهم هذه التوابع مايلي:

دلالة التابع	التابع
المتوسط Average	Avg
الأصغر Minimum	Min
الأكبر Maximum	Max
المجموع Sum	Sum

أمثلة:

- لإظهار متوسط أرصدة الحسابات في الفرع "x" نكتب:

```
select avg (balance)
from Account
where branch_name = "x" ;
```

- ولإظهار أكبر قيمة قرض منحها فرع Perryridge نكتب:

```
select Max (amount)
from Loan
where branch_name= "Perryridge"
```

- ولإظهار مجموع أرصدة الحسابات المفتوحة في الفرع "x" نكتب:

```
select Sum(balance)
from Account
where branch_name ="x";
```

- ولإظهار قائمة بالأفرع ومتوسط الأرصدة في كل منها نكتب:

```
select branch_name, avg (balance)
from Account
group by branch_name ;
```

نلاحظ من المثال السابق أن الإبقاء على التكرار ضروري أثناء عملية حساب الوسطي avg.

7- معالجة القيم غير المعلومة

تسمح لغة SQL باستخدام القيمة غير المعلومة Null للدلالة على عدم توفر معلومات في واصف.

مثال: لإيجاد أرقام القروض التي لا نعرف قيمتها نكتب:

```
select loan_number
from Loan
where amount is null;
```

تعالج التوابع التجميعية القيم غير المعلومة بحيث تتجاهل وجود هذه القيم في مجموعة العناصر التي تُجرى عليها العملية التجميعية وتعطي النتيجة دون اعتبارها، ويقوم التابع Count بحصر

عدد الحدوديات التي لاتحوي قيماً غير معلومة.

8- تجزئة العلاقة

في الأمثلة السابقة كنا نطبق تابعاً تجميعياً على بعض أسطر العلاقة التي تحقق شرطاً معيناً مثل: "اسم الفرع=..." غير أننا لو أردنا مثلاً حساب وسطي الأرصدة في كل فرع فإننـا، وبحسـب الطريقة السابقة، سنضطر لتنفيذ تعليمة :

```
select Avg (balance)
from Account
where branch_name= "x";
```

عدداً من المرات يساوي عدد الفروع، وفي كل مرة نجعل x تأخذ اسم أحد الفروع.

طبعاً هذا الحل غير مجدٍ وخاصة عندما لا تكون لدينا فكرة سابقة عن أسماء كافة الفروع.

لحل هذه المشكلة حساب وسطي الأرصدة في كل فرع، تقدم لغة SQL إمكان تجزئة العلاقة وفق قيمة أحد الواصفات، ومن ثم إجراء عملية تجميع على واحد أو أكثر من الواصفات الباقية.

نحدد العمود الذي تجري التجزئة وفقه بالعبارة Group by. وهكذا نكتب:

```
select branch_name, avg(balance)
from Account
group by branch_name;
```

الذي يمكن تمثيله بالشكل التالي:

Account

Branch_name	Account_nb	Balance
Perryridge	X1	10000
National	X2	20000
Perryridge	X3	30000
Perryridge	X4	20000
Perryridge	X5	40000

Results

Branch_name	Avg(balance)
Perryridge	25000
National	20000

مثال:

- لإيجاد عدد المشتركين الذين لديهم حساب في كل فرع نكتب:

```
select branch_name, count (distinct customer_name)
from Depositor, Account
where Depositor.account_number = Account.account_number
group by branch_name ;
```

ويمكن وضع شروط تُطبق على نتيجة عملية التجميع باعتبارها علاقة جديدة.

مثال:

- لإيجاد أسماء الفروع التي متوسط الأرصدة فيها أكبر من 1200$ يستخدم التعبير having فـي SQL كالتالي:

```
select branch_name, avg (balance)
from Account
group by branch_name
having avg (balance) > 1200;
```

إذا وجد تعبير having و where في نفس الاستعلام فإن الشرط where يطبـق أولاً، وتوضـع الحدوديات المحققة للشرط في مجموعات (groups) بتطبيق group by ثم يطبق التعبيـر having على كل مجموعة. وتكون النتيجة المجموعات المستخدمة فـي تعليمـة Select والتـي تحقـق الاستفسار.

مثال:

- لإيجاد متوسط أرصدة كل زبون يعيش في مدينة "x" ويملك على الأقل ثلاثة حسابات نكتب:

```
select depositor, customer_name, avg (balance)
from Depositor, Account, Customer
where depositor.account_number = account.account_number
and   depositor.customer_name = customer.customer- name and customer-
city = "x";
group by (depositor, customer_name)
having count (distinct depositor.account_number > = 3)
```

9- الاستفسارات الجزئية المضمنة

تقدم لغة SQL تقانات لتنفيذ استفسارات جزئية مضمنة، الاستفسار الجزئي هو تعبير من الشكل select -from -where ويكون مضمناً في استفسار آخر.

مثال: الانتماء إلى مجموعة

لإيجاد جميع الزبائن الذين لديهم قرض وحساب في المصرف، يمكننا الوصول إلـى المطلـوب باستخدام عملية التقاطع بين مجموعتين: مجموعة الزبـائن المقترضـين و مجموعـة الزبـائن المودعين. كما يمكن استخدام منحى آخر كإيجاد جميع المودعين في المصرف الذين ينتمون إلى مجموعة المقترضين من المصرف.

نكتب:

```
select distinct customer_name
from Borrower
where customer_name in (select customer_name from Depositor)
```

مثال: مقارنة المجموعات

تسمح لغة SQL باستخدام المقارنة بين مجموعة حدوديات ومجموعة حدوديات أخرى.

مثال: لإيجاد أسماء جميع الفروع التي قيم موجوداتها أكبر من قيم موجودات فرع واحـد علـى الأقل من الفروع الموجودة في مدينة "x".

يمكن صياغة الاستعلام بالشكل التالي:

```
select distinct T.branch_name
from Branch as T, Branch as S
where T.assets > S.assets and S.branch-city = “x”
```

تسمح SQL بالصياغة بشكل مختلف وباستخدام عبارة بعض "some" التي تـدل علـى بعـض عناصر المجموعة، والعبارة all التي تدل على كل عناصر المجموعة.

مثال:

```
select branch_name
from Branch
where assets > some
      (select assets
       from branch
       where branch-city = “*”)
```

ويمكن استخدام عبارات المقارنة مع جزء من المجموعة التالية:

>some , <some , >=some, < >some , =some , <= some

وكذلك >all , <all , >=all , < >all , =all , <=all

مثال:

لإيجاد الفروع التي لديها متوسط الأرصدة أعظمي (أي متوسط الأرصدة فيها أكبر مـن جميـع متوسطات الأرصدة في باقي الفروع) نكتب:

```
select branch_name
from Account
group by branch_name
having avg (balance) >= all
      (select avg (balance)
       from Account
       group by branch_name)
```

اختبار العلاقات الفارغة

يمكن اختبار وجود عناصر (حدوديات) في علاقة باستخدام عبارة يوجد "exists" التي تعيد قيمة صح "true" في حال وجود هذه العناصر.

مثال: لإيجاد أسماء الزبائن الذين لهم قرض وحساب في المصرف نكتب

```
select customer_name
from Borrower
where exists (select *
                  from depositor
              where depositor.customer_name =
                      borrower.customer_name)
```

اختبار عدم وجود تكرار في الحدوديات

يجري ذلك باستخدام التعبير "Unique" الذي يعيد القيمة true إذا لم يحتـو الاسـتعلام الجزئـي تكراراً في الحدوديات.

مثال: لإيجاد أسماء الزبائن الذين لديهم حساب واحد في الفرع “x” نكتب:

```
select T.customer_name
from Depositor as T
where unique (select R.customer_name
from Account, Depositor as R
where T.customer_name = R.customer_name and
      R.account_number = Account.account_number
      and Account.branch_name = "x")
```

3. المناظير

يعرف المنظار بلغة SQL بالشكل:

```
create view as (query expression)
```

مثال:

```
create view branch-total-loan
(branch_name, total-loan)
as select branch_name, sum (amount)
from Loan
group by branch_name
```

ولحذف منظار نكتب:

```
drop view view_name
```

4. تعديل قاعدة المعطيات

يتضمن تعديل قاعدة المعطيات عمليات الإضافة والحذف والتعديل على قاعدة المعطيات ويجري التعبير عنها باستخدام لغة SQL كما هو موضح في الفقرات اللاحقة.

1- الحذف

الشكل العام لتعليمة حذف حدوديات من علاقة هو:

```
delete from tablename where condition
```

أمثلة:

1- لحذف جميع الحسابات العائدة للفرع " Perryridge " نكتب:

```
delete from Account
where branch_name= “ Perryridge ”
```

2- لحذف جميع الحسابات في جميع الفروع الموجودة في مدينة "Needham" نكتب :

```
delete from Account
where branch_name in (select branch_name
                      from Branch
                      where branch_city=  “ Needham ” )
```

3- لحذف كافة المودعين الذين فتحوا حسابات في فروع تقع في مدينة"Needham" نكتب:

```
delete from Depositor
where account_number in (select account_number
                                    from Branch, Account
                                    where branch_city=  “ Needham ”
                                    and Branch.branch_name=
                                    Account.branch_name)
)
```

4- لحذف جميع تسجيلات الحسابات التي رصيدها أقل من وسطي الأرصـدة فـي المصـرف نكتب:

```
delete from Account
where balance < (select avg ( balance) from Account)
```

المشكلة التي تظهر في هذا المثال هي أن حذف حدوديات من العلاقة account يغير مـن قيمـة وسطي الأرصدة في المصرف، وهنا تقوم لغة SQL بحل هذا الإشكال كالتالي:

في البداية تقوم بحساب متوسط الأرصدة وإيجاد جميع الحدوديات التي يجب أن تُحذف، ثُم تقوم بحذف جميع الحدوديات الموجودة سـابقاً (دون إعـادة حسـاب المتوسـط أو إعـادة اختبـار الحدوديات).

2- الإضافة

تأخذ عملية إضافة حدودية إلى علاقة الشكل العام التالي:

```
insert into rel_name
values ( attribute values)
```

أو

```
insert into rel_name
(select  ….From….Where ….)
```

وفي هذه الحالة يجري تنفيذ عملية الاختيار أولاً، ثم عملية الإضافة.

أمثلة:

1- لإضافة حدودية جديدة إلى علاقة الحسابات نكتب:

```
insert into Account
values ( “ Perryridge ” , A-9732, 1200)
```

أو الشكل المكافئ:

```
insert into Account( branch_name, balance, account_number)
values ( “ Perryridge ” , 1200, A-9732)
```

2- لإضافة حدودية جديدة إلى علاقة الحسابات مع أن الرصيد للحساب مجهول أو غير معلـوم نكتب:

```
insert into Account
values ( “ Perryridge ” , A-777, null)
```

3- لإضافة حساب مصرفي إلى جميع المقترضين من المصرف من الفرع "Perryridge" بقيمـة

200$ وبحيث يعتمد رقم القرض رقماً للحساب الجديد، نكتب :

```
insert into Account
select branch_name, loan_number, 200
from Loan
where branch_name=  " Perryridge "

insert into  Depositor
select customer_name, loan_number
from Loan, Borrower
where branch_name=  " Perryridge "
       and Loan.account_number= Borrower.account_number
```

3- التعديل

الشكل العام لتعليمة التعديل هو:

```
update rel_name
set attribute= new-values
where condition
```

مثال: لتعديل أرصدة الحسابات المصرفية بإضافة 6% إلى الحسابات التي يزيد رصـيدها عـن 10.000$ وإضافة 5% إلى البقية، نكتب:

```
update Account
set balance = balance *  1.06
where balance>  10000

update Account
set balance = balance* 1.05
where balance <= 10000
```

ويمكن إجراء التعديل على قاعدة المعطيات بواسطة التعديل على المنظار المعرف علـى هـذه القاعدة، ولكن قد تظهر بعض المشاكل المتعلقة بالتعامل مع القيم غير المعلومة.

مثال: ليكن لدينا المنظار branch-loan المعرف على قاعدة معطيات المصرف، والـذي يظهـر معطيات جميع القروض مع إخفاء المعطيات المتعلقة بكمية هذه القروض.

```
create view branch-loan as
select branch_name, loan_number
from Loan
```

نضيف حدودية جديدة إلى المنظار:

```
insert into branch-loan
values ( " Perryridge " ,  " L-307 " )
```

تتمثل عملية الإضافة هذه بعملية إضافة للحدودية التالية إلى العلاقة loan:

```
("Perryridge " ,  " L-307 " , null)
```

التعديل على المعطيات من خلال مناظير أكثر تعقيداً، وفي بعض الأحيـان تسـتحيل ترجمتـه لمعرفة التعديلات الواجب إجراؤها على العلاقات الأساسية.

5. دمج العلاقات

دمج العلاقات عملية ثنائية تأخذ علاقتين كدخل وتُعطي علاقة كخرج. تُستخدم هـذه العمليـات الإضافية كتعابير استفسار جزئي في فقرة From ضمن الاستفسار.

شرط الدمج : يحدد الحدوديات المختارة من العلاقتين والواصفات التي تظهر في نتيجة الدمج.

نوع الدمج : يحدد طريقة دمج أو عدم دمج الحدوديات من كل علاقة مع العلاقة الأخرى.

أنواع الدمج
دمج داخلي
دمج خارجي يساري
دمج خارجي يميني
دمج خارجي تام

شروط الدمج
طبيعي
يحقق <قضية>
باستخدام (A1,A2,…,An)

مثال: لتكن لدينا العلاقتان التاليتان:

العلاقة Loan

branch-name	loan-number	amount
Downtown	L-170	3000
Redwood	L-230	4000
Perryridge	L_260	1700

العلاقة Borrower

customer_name	loan-number
Jones	L-170
Smith	L-230
Hayes	L_155

تجري عملية الدمج الداخلي بين العلاقتين بالعملية التالية :

```
Loan inner join Borrower on loan.loan_number =borrower.loan_number
```

ونحصل على النتيجة :

branch_name	loan_number	amount	customer_name	loan_number
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230

وعملية الدمج الخارجي اليساري تُعطي:

```
Loan left outer join Borrower on
Loan.loan_number = Borrower.loan_number
```

branch_name	loan_number	amount	customer_name	loan_number
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
Perryridge	L_260	1700	null	null

وعملية الدمج الطبيعي:

```
Loan natural inner join Borrower on
Loan.loan_number = Borrower.loan_number
```

branch_name	loan_number	amount	customer_name
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Perryridge	L_260	1700	null

وعملية الدمج الخارجي اليميني الطبيعي

```
Loan natural right outer join Borrower
```

branch_name	loan_number	amount	customer_name
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
null	L-155	null	Hayes

وعملية الدمج الخارجي الكامل:

```
Loan full outer join Borrower using ( loan_number )
```

branch_name	loan_number	amount	customer_name
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
null	L-155	null	Hayes
Perryridge	L-260	1700	null

مثال: لإيجاد الزبائن الذين يتعاملون مع المصرف بنوع واحـد مـن التعامـل (مقترضـون أو مودعون) ولكن ليس بالنوعين معاً، نكتب :

```
select customer_name
from ( Depositor natural full outer join Borrower)
where account_number is null
or loan_number is null
```

6. لغة تعريف المعطيات DDL

تسمح لغة SQL بتعريف العلاقات التي تتكون منها قاعدة المعطيات، وبتحديد معلومات عن كل علاقة بما يتضمن:

- المخطط العلاقاتي لكل علاقة.

- مجال تعريف القيم المرتبطة بكل واصف.
- شروط التكامل.
- مجموعة المؤشرات المرتبطة بكل علاقة.
- أمن المعلومات وسماحيات الوصول.
- بنية التخزين الفيزيائي.

1- تعريف المجالات بلغة SQL

يمكن تعريف أنماط مختلفة للمجالات بلغة SQL منها:

char(n): لتعريف سلسلة أحرف ذات طول ثابت. حيث n طول ثابت يحدده المستثمر.

varchar(n): لتعريف سلسلة أحرف ذات طول متغير. حيثn الطول الأعظمي الـذي يحـدده المستثمر.

Int: عدد صحيح.

number(p,d): رقم ممثل بالنقطة الثابتة.

real, double precision: رقم ممثل بالفاصلة العائمة مع دقة مضاعفة.

Date: التاريخ يحوي 4 خانات للسنة.

Time: الوقت في اليوم ،ساعة، دقيقة، ثانية.

ملاحظات:

- يسمح باستخدام القيم غير المعلومة في جميع أنماط المجالات. والتصريح بكون واصفة لا تقبل قيماً غير معلومة يحرم استخدام القيم غير المعلومة لهذه الواصفة.
- يمكن للمستثمر أن يعرف مجالات خاصة به باستخدام لغة SQL كما في المثال التالي:

```
create domain person_name char (20) not null
```

2- تعريف المخطط العلاقاتي

يعرف مخطط علاقة بلغة SQL باستخدام التعليمة التالية:

```
create table r ( A1 D1 ,A2 D2 , ..., An Dn ,
integrity-constraint l i  ,
...,
integrity-constraint k i  )
```

حيث:

r اسم العلاقة.

Ai اسم الواصفة رقم I من مخطط العلاقة r.

Di مجال تعريف الواصفة Ai.

li, …,ki شروط تكامل معرف على الجدول.

مثال:

```
create table Branch
 ( branch_name char(15) not null ,
 branch_city char(30),
 assets integer)
```

تُعرف شروط التكامل على المخطط العلاقاتي والمتضمنة:

- عدم احتوائه على قيم غير معلومة not null.
- تعريف المفتاح الرئيسي primary key (A1 ,...,An)
- تعريف قضية check (P) حيث P قضية.

مثال:

لتعريف مخطط علاقاتي Branch بحيث تكون الواصفة branch_name مفتاحاً رئيسـياً و قـيم الواصفة assets غير سالبة، نكتب:

```
create table branch
(
branch_name char(15) not null,
branch_city char(30),
assets integer,
primary key ( branch_name), check ( assets>=0)
)
```

إن تعريف المفتاح الرئيسي على واصفة يجعل اختبار عدم احتوائها على قيم غير معلومة آلياً.

3- حذف مخطط علاقة بلغة SQL

تسمح تعليمة **drop table** بحذف جميع المعلومات المتعلقة بالعلاقة من قاعدة المعطيات، وتسمح تعليمة **alter table** بإضافة واصفات إلى مخطط علاقة موجودة، وبحيث تأخذ هذه الواصفة قيمة غير معلومة في جميع الحدوديات الموجودة سابقاً في العلاقة.

مثال: لإضافة واصف باسم A ومجاله D إلى العلاقة r نكتب:

```
alter table r add A D
```

ويمكن استخدام تعليمة **alter table** لحذف واصف من العلاقة.

مثال:

```
alter table r drop A
```

حيث A واصف ضمن العلاقة r.

تمرين

لتكن لدينا العلاقات التالية :

```
Employee  (Id, name, job, manager-id, salary, dept-id )
Department  (dept-id, name, city)
```

التي تتضمن معلومات عن الموظفين في شركة تمتلك عدة فروع.

المطلوب:

أ. اكتب التعليمات اللازمة لإنشاء هاتين العلاقتين مع تحديد الشروط التالية:

- رقم الموظف وحيد.
- لايمكن أن يكون اسم الموظف مجهولاً.
- يعمل كل موظف في قسم واحد.

ب. هل يمكن اعتبار رقم الرئيس المباشر إجبارياً.

ت. هل يمكن اعتبار متضمناً في قائمة أرقام موظفي الشركة.

ث. اكتب الاستفسارات التالية بلغة SQL:

- عدد العاملين في مدينة دمشق مرتبة أبجدياً.
- أسماء ورواتب العاملين في قسم التسويق (Marketing) مرتبة تنازلياً وفق الراتب.
- اسم المدير العام للشركة.
- الموظفون الذين يتبعون مباشرة إلى المدير العام.
- الموظف (أو الموظفون) الذي يحصل على أعلى راتب في الشركة.
- أعداد العاملين في كل قسم من أقسام الشركة.
- اسم القسم الذي يحوي أكبر عدد من العاملين.
- اسم القسم الذي يزيد مجموع رواتب العاملين فيه عن 100000 ل. س.
- أسماء ورواتب العاملين الذين يزيد راتبهم عن راتب رئيسهم المباشر.
- القسم الذي يزيد متوسط رواتب العاملين عن متوسط رواتب العاملين في الشركة.