موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

درس و کنکور ارشد

پایگاه داده‌ها

(حل سوالات کنکور ارشد ۱۴۰۳)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی کامپیوتر و IT

براساس کتب مرجع

آبراهام سیلبر شاتز، رامز المصری، راما کریشنان، سی جی دیت، توماس کانلی و کارولین بگ

ارسطو خلیلی‌فر

تست‌های فصل هفتم: SQL دستورات DDL

۱۰۹- تفاوت اصلی بین دیدهای پذیرا (Updatable Views) و دیدهای ناپذیرا (Non-Updatable Views) در پایگاه داده چیست؟

(مهندسی کامپیوتر-دولتی۱۴۰۳)

۱) هر دو نوع دید قابلیت به‌روزرسانی داده‌ها را دارند، اما دیدهای ناپذیرا، محدودیت‌های بیشتری دارند.

۲) دیدهای ناپذیرا، امکان به‌روزرسانی داده‌های موجود در آنها را فراهم می‌کنند، درحالی‌که دیدهای پذیرا، فقط برای خواندن داده‌ها استفاده می‌شوند.

۳) دیدهای پذیرا، امکان به‌روزرسانی داده‌های موجود در آنها را فراهم می‌کنند، درحالی‌که دیدهای ناپذیرا، فقط برای خواندن داده‌ها استفاده می‌شوند.

۴) هیچ تفاوتی بین دیدهای پذیرا و دیدهای ناپذیرا وجود ندارد و هر دو فقط برای خواندن داده‌ها استفاده می‌شوند.

استاد ارسطو خلیلی فر
مولف کتاب پایگاه داده انتشارات بابان

پاسخ تست‌های هفتم: SQL دستورات DDL

۱۰۹- گزینه (۳) صحیح است.

پاسخ سریع: به طور کلی دو عمل **ذخیره‌سازی** (درج، حذف و بروزرسانی) و **بازیابی** (خواندن) بر روی دیدها (جدول مجازی) انجام می‌شود. که عمل ذخیره‌سازی با قید و شرط انجام می‌شود یعنی نه همیشه، اما عمل بازیابی بی‌قید و شرط انجام می‌شود یعنی همیشه امکان‌پذیر است.

در مفهوم دید، **پذیرا از نظر تئوری** یعنی در ساختار دید، قوانین مدل رابطه‌ای رعایت شود و **پذیرا از نظر عملی** یعنی در **ذخیره‌سازی** (درج، حذف و بروزرسانی) قوانین مدل رابطه‌ای رعایت شود. دقت کنید که وقتی تئوری ذخیره‌سازی روی دید امکان‌پذیر باشد ذخیره‌سازی عملی آن ممکن است امکان‌پذیر باشد یا نباشد. اما وقتی تئوری ذخیره‌سازی روی دید امکان‌پذیر نباشد به تبع قطعا ذخیره‌سازی عملی آن هم امکان‌پذیر نیست.

پاسخ تشریحی:

دستور Create View

این دستور جهت ایجاد یک دید (جدول مجازی) در یک پایگاه داده مورد استفاده قرار می‌گیرد.

ساختار کلی این دستور به صورت زیر است:

```
Create View name
   AS  Select …
```

دید (View) یا جدول مجازی، خودش به خودی خود وجود خارجی و فیزیکی ندارد. اما می‌توان به آن جهت بازیابی رکوردهای اطلاعاتی از جداول پایه و ذخیره‌سازی رکوردهای اطلاعاتی روی جداول پایه دسترسی داشت. هدف اصلی از مفهوم دید، نمایشی محدود، خلاصه و فیلترشده از اطلاعات جداول پایه است. دید به جداول پایه متصل می‌شود و همانند یک فیلتر روی جداول پایه عمل می‌کند. هنگامی که توسط دید قصد بازیابی اطلاعات را داریم، از آنجاکه دید خودش به خودی خود وجود خارجی و فیزیکی ندارد، بنابراین ابتدا دید به جداول پایه متصل می‌شود و همانند یک فیلتر روی جداول پایه عمل می‌کند و در نهایت فقط بخشی از اطلاعات جداول پایه (سطر و ستون مورد نظر) را استخراج و به نمایش می‌گذارد. همچنین هنگامی که توسط دید قصد ذخیره‌سازی اطلاعات را داریم، از آنجاکه دید خودش به خودی خود وجود خارجی و فیزیکی ندارد، بنابراین ابتدا دید به جداول پایه متصل می‌شود و در نهایت رکوردهای ورودی روی خود جداول پایه ذخیره‌سازی می‌شوند. به بیان دیگر جداول مجازی، استقلال وجودی ندارند و به جداول پایه متکی و متصل هستند، به گونه‌ای که هر عملیاتی که روی جدول مجازی انجام می‌شود، در واقع در نهایت روی جداول پایه مرتبط و متصل به آن اعمال می‌گردد.

جداول مجازی (دیدها) تصویری از جداول حقیقی (پایه) هستند، یعنی توسط سیستم به جداول پایه متصل می‌شوند و وجود خارجی ندارند. دسترسی به آنها از دید کاربر مستقیم ولی از دید سیستم غیرمستقیم است، یعنی سیستم، هرگونه استخراج اطلاعات را از روی جداول پایه انجام می‌دهد. محتوای دید در لحظه اجرای دید، تولید می‌شود. این محتوای بازیابی شده از سوی دید، در جایی ذخیره نمی‌گردد، بلکه در هر لحظه بر اساس ساختار دید از روی جداول پایه، استخراج می‌گردد و به نمایش گذاشته می‌شود.

به طور کلی دو عمل **ذخیره‌سازی** (درج، حذف و بروزرسانی) و **بازیابی** بر روی دیدها انجام می‌شود. که عمل ذخیره‌سازی با قید و شرط انجام می‌شود یعنی نه همیشه، اما عمل بازیابی بی‌قید و شرط انجام می‌شود یعنی همیشه امکان‌پذیر است. بنابراین مشکل اصلی در دیدها، عمل ذخیره‌سازی (درج، حذف و بروزرسانی) است که قید و شرط دارد و همیشه قابل انجام نیست. هر عمل ذخیره‌سازی (درج، حذف و بروزرسانی) که روی جداول مجازی انجام می‌گیرد در واقع روی جداول پایه متصل به آن اعمال می‌شود، بنابراین فقط و فقط در صورتی می‌توان عمل ذخیره‌سازی را روی جداول مجازی انجام داد که عمل مورد نظر (درج، حذف و بروزرسانی) قوانین جامعیت داخی و خارجی بانک اطلاعات را نقض نکند. به طور کلی، پذیرا بودن ذخیره‌سازی (درج، حذف و بروزرسانی) بر روی دیدها به سه دسته زیر تقسیم می‌گردد:

پذیرا از نظر تئوری و عملی

در این حالت دید از نظر **تئوری** پذیرای ذخیره‌سازی است، یعنی قوانین مدل رابطه‌ای همچون داشتن حداقل یک کلید کاندید و داشتن ستون‌های واقعی و نه مجازی حفظ می‌گردد و همچنین تناظر سطر به سطر مابین دید و جدول پایه حفظ می‌گردد. و از نظر **عملی** هم پذیرای ذخیره‌سازی است به دلیل رعایت قوانین جامعیت بانک در مدل رابطه‌ای در حین اجرا.

موارد زیر پذیرا بودن ذخیره‌سازی بر روی دید را از نظر تئوری تضمین می‌کند:

– در تعریف دید فقط از یک جدول پایه با حداقل یک کلید کاندید استفاده شده باشد.

اگر رعایت نشود حفظ قوانین مدل رابطه‌ای همچون داشتن حداقل یک کلید کاندید و حفظ تناظر سطر به سطر مابین دید و جدول پایه نقض می‌شود.

توجه مهم: عملیاتی همچون الحاق طبیعی، اشتراک، اجتماع و تفاضل میان دو جدول که خروجی آن یک جدول با کلید کاندید واضح و مشخص است، پذیرا بودن ذخیره‌سازی بر روی دید را از نظر تئوری تضمین می‌کند.

– در تعریف دید از دستور select distinct استفاده نشده باشد.

اگر رعایت نشود حفظ تناظر سطر به سطر مابین دید و جدول پایه نقض می‌شود.

– در تعریف دید از دستور group by استفاده نشده باشد.

اگر رعایت نشود حفظ تناظر سطر به سطر مابین دید و جدول پایه نقض می‌شود.

– در تعریف دید از توابع آماری استفاده نشده باشد.

اگر رعایت نشود حفظ قوانین مدل رابطه‌ای همچون داشتن ستون‌های واقعی و نه ستون مجازی (محاسبه شدنی) نقض می‌شود.

– در تعریف دید از زیر پرس‌وجوهای تو در تو استفاده نشده باشد.

اگر رعایت نشود حفظ تناظر سطر به سطر مابین دید و جدول پایه نقض می‌شود.

– در عمل Select کلید اصلی جدول حذف نشده باشد.

– در تعریف دید از عمل تقسیم استفاده نشده باشد.

مورد زیر پذیرا بودن ذخیره‌سازی بر روی دید را از نظر عملی تضمین می‌کند:

– در هنگام اجرای ذخیره‌سازی بر روی دید، قوانین جامعیت داخلی و خارجی بانک نقض نگردد.

سه جدول S، P و SP در سیستم تولیدکنندگان و قطعات با مقادیر زیر را در نظر بگیرید:

S#	Sname	City
S1	Sn1	C1
S2	Sn2	C2
S3	Sn3	C3

جدول S

(تولیدکنندگان)

S#	P#	QTY
S1	P1	10
S1	P2	20
S2	P1	30

جدول SP

(تولید)

P#	Pname	Color
P1	Pn1	Red
P2	Pn2	Blue

جدول P

(قطعات)

مثال: دید زیر را در نظر بگیرید:

```
Create View Report1
AS Select S# , Sname
    From S
```

توجه: اجرای دستور فوق منجر به ایجاد دید Report1 در پایگاه داده prodsuppdb می‌شود.

در تعریف دید Report1 از یک جدول پایه استفاده شده است، بدنه دید Report1 به صورت زیر است:

S#	Sname
S1	Sn1
S2	Sn2
S3	Sn3

آنچه واضح است، این است که ساختار دید Report1 از نظر تئوری پذیرای ذخیره‌سازی است، یعنی قوانین مدل رابطه‌ای همچون داشتن حداقل یک کلید کاندید و داشتن ستون‌های واقعی و نه مجازی حفظ می‌گردد و همچنین تناظر سطر به سطر مابین دید و جدول پایه حفظ می‌گردد.

فرض کنید کاربر با اجرای دستور زیر بخواهد یک سطر جدید را در دید Report1 درج نماید:

```
Insert into Report1
Values ('S4', 'Sn4')
```

از آنجا که دید Report1 استقلال وجودی ندارد و به جدول پایه S متصل است، عملیات درج فوق

در واقع روی جدول پایه S انجام می‌گردد.

در واقع دستور فوق به هنگام اجرا، به دستور زیر تبدیل می‌گردد:

```
Insert into S
Values ('S4' , 'Sn4' , NULL)
```

در هنگام ذخیره‌سازی عملی بر روی دید Report1 قوانین جامعیت بانک رعایت می‌شود. پس اجرای دستور ذخیره‌سازی بر روی دید Report1 به صورت عملی از سوی DBMS پذیرفته می‌شود. البته به شرطی که ستون City در هنگام تعریف جدول S به صورت NOT NULL تعریف نشده باشد.

مقادیر کنونی جدول S پس از درج رکورد مذکور به صورت زیر است:

S#	Sname	City
S1	**Sn1**	**C1**
S2	**Sn2**	**C2**
S3	**Sn3**	**C3**
S4	**Sn4**	**NULL**

توجه: می‌توان ستون‌های جدول مجازی را با نام‌های جدیدی نام‌گذاری کرد. برای مثال در دید بالا S# به SID و Sname به SN می‌تولند نام‌گذاری مجدد شود، برای این منظور دید Report1 به صورت زیر مجدد بازنویسی می‌شود:

```
Create View Report1 (SID , SN)
AS Select S# , Sname
    From S
```

بدنه دید Report1 پس از نام‌گذاری مجدد به صورت زیر است:

SID	SN
S1	Sn1
S2	Sn2
S3	Sn3
S4	Sn4

توجه: جداول مجازی به جداول پایه مرتبط به خود متصل هستند، بنابراین در پرس‌وجوها می‌توان از دیدها نیز استفاده کرد، بنابراین هنگامی که در یک پرس‌وجو از جدول مجازی استفاده می‌شود، ابتدا تعریف جدول مجازی مربوطه در آن پرس‌وجو جاگذاری می‌شود و سپس پرس‌وجو اجرا می‌شود.

مثال: پرس‌وجوی زیر را روی دید Report1 در نظر بگیرید:

```
Select *
From Report1
Where Sname <> 'Sn1'
```

در واقع DBMS دستور فوق را به دستور زیر تبدیل می‌کند:

```
Select S# , Sname
From S
Where Sname <> 'Sn1'
```

خروجی پرس‌وجو به صورت زیر است:

S#	Sname
S2	Sn2
S3	Sn3
S4	Sn4

مثال: دید زیر را در نظر بگیرید:

```
Create view Report2
AS Select S# , P#
    From SP
    Where QTY>10
```

در تعریف دید Report2 از یک جدول پایه استفاده شده است، بدنه دید Report2 به صورت زیر است:

S#	P#
S_1	P_2
S_2	P_1

آنچه واضح است، این است که ساختار دید Report2 از نظر تئوری پذیرای ذخیره‌سازی است.

فرض کنید کاربر با اجرای دستور زیر بخواهد یک سطر جدید را در دید Report2 درج نماید:

```
Insert into Report2
Values ('S3', 'P1')
```

از آنجا که دید Report2 استقلال وجودی ندارد، عملیات درج فوق در واقع روی جدول پایه SP انجام می‌گردد.

در واقع دستور فوق به هنگام اجرا، به دستور زیر تبدیل می‌گردد:

```
Insert into SP
Values ('S3', 'P1', NULL)
```

در هنگام ذخیره‌سازی عملی بر روی دید Report2 قوانین جامعیت بانک رعایت می‌شود. پس اجرای دستور ذخیره‌سازی بر روی دید Report2 به صورت عملی از سوی DBMS پذیرفته می‌شود. البته به شرطی که ستون QTY در هنگام تعریف جدول SP به صورت NOT NULL تعریف نشده باشد. قانون جامعیت ارجاعی هم نقض نمی‌شود.

مقادیر کنونی جدول SP پس از درج رکورد مذکور به صورت زیر است:

S#	P#	QTY
S1	P1	10
S1	P2	20
S2	P1	30
S3	P1	NULL

پذیرا از نظر تئوری و غیرپذیرا از نظر عملی

در این حالت دید از نظر تئوری پذیرای ذخیره‌سازی است، یعنی قوانین مدل رابطه‌ای همچون داشتن حداقل یک کلید کاندید و داشتن ستون‌های واقعی و نه مجازی حفظ می‌گردد و همچنین تناظر سطر به سطر مابین دید و جدول پایه حفظ می‌گردد. اما از نظر عملی پذیرای ذخیره‌سازی نیست به دلیل عدم رعایت قوانین جامعیت بانک در مدل رابطه‌ای در حین اجرا.

موارد زیر پذیرا بودن ذخیره‌سازی بر روی دید را از نظر تئوری تضمین می‌کند:

- که در بخش قبل بررسی شد.

مورد زیر پذیرا بودن ذخیره‌سازی بر روی دید را از نظر عملی تضمین می‌کند:

- که در بخش قبل بررسی شد.

سه جدول S، P و SP در سیستم تولیدکنندگان و قطعات با مقادیر زیر را در نظر بگیرید:

S#	Sname	City
S1	Sn1	C1
S2	Sn2	C2
S3	Sn3	C3

جدول S

S#	P#	QTY
S1	P1	10
S1	P2	20
S2	P1	30

جدول SP

P#	Pname	Color
P1	Pn1	Red
P2	Pn2	Blue

جدول P

مثال: دید زیر را در نظر بگیرید:

```
Create View Report3
AS Select S# , City
    From S
```

در تعریف دید Report3 از یک جدول پایه استفاده شده است، بدنه دید Report3 به صورت زیر است:

S#	City
S1	C1
S2	C2
S3	C3

آنچه واضح است، این است که ساختار دید Report3 از نظر تئوری پذیرای ذخیره‌سازی است.

فرض کنید کاربر با اجرای دستور زیر بخواهد یک سطر جدید را در دید Report3 درج نماید:

```
Insert into Report3
Values ('S4', 'C4')
```

از آنجا که دید Report3 اســـتقلال وجودی ندارد، عملیات درج فوق در واقع روی جدول پایه S انجام می‌گردد. اما جدول S دارای سه ستون است، درحالیکه، فقط دو ستون در دستور فوق مربوط به جدول S مقداردهی شده‌است. بنابراین برای ستون Sname مقدار NULL در نظر گرفته می‌شود. فرض کنید در هنگام تعریف جدول S برای ســتون Sname خاصــیت NOT NULL در نظر گرفته شـده باشـد، یعنی حتما می‌بایسـت سـتون Sname دارای مقدار باشـد و مقدار NULL برای آن قابل پذیرش نیست. بنابراین مطابق قانون جامعیت بانک نباید ستون Sname با NULL مقداردهی شود. درواقع دستور فوق به هنگام اجرا، به دستور زیر تبدیل می‌گردد:

```
Insert into S
Values ('S4', NULL , 'C4')
```

در هنگام ذخیره‌ســـازی عملی بر روی دید Report3 مطابق قانون جامعیت بانک در جداول نمی‌توان در ستون Sname از جدول S که خاصیت NOT NULL برای آن در نظر گرفته شده‌است، مقدار NULL را درج کرد، پس اجرای دستور ذخیره‌سازی بر روی دید Report3 به صورت عملی از سوی DBMS پذیرفته نمی‌شود و رد می‌شود.

سه جدول S، P و SP در سیستم تولیدکنندگان و قطعات با مقادیر زیر را در نظر بگیرید:

S#	Sname	City
S1	Sn1	C1
S2	Sn2	C2
S3	Sn3	C3

جدول S

S#	P#	QTY
S1	P1	10
S1	P2	20
S2	P1	30

جدول SP

P#	Pname	Color
P1	Pn1	Red
P2	Pn2	Blue

جدول P

مثال: دید زیر را در نظر بگیرید:

```
Create view Report4
AS Select S# , P# , QTY
    From SP
    Where QTY>10
```

در تعریف دید Report4 از یک جدول پایه استفاده شده است، بدنه دید Report4 به صورت زیر است:

S#	P#	QTY
S1	P2	20
S2	P1	30

آنچه واضح است، این است که ساختار دید Report4 از نظر تئوری پذیرای ذخیره‌سازی است.

فرض کنید کاربر با اجرای دستور زیر بخواهد یک سطر جدید را در دید Report4 درج نماید:

```
Insert into Report4
Values ('S10', 'P10' , 100)
```

از آنجا که دید Report4 اســتقلال وجودی ندارد، عملیات درج فوق در واقع روی جدول پایه SP انجام می‌گردد.

درواقع دستور فوق به هنگام اجرا، به دستور زیر تبدیل می‌گردد:

```
Insert into SP
Values ('S10', 'P10' , 100)
```

در هنگام ذخیره‌سـازی عملی بر روی دید Report4 هر چند هر سـه سـتون جدول SP به واسـطه دستور درج، مقداردهی شده‌اند، اما این‌بار **قانون جامعیت ارجاعی** نقض شده است، زیرا ستون‌های S# و P# به عنوان کلیدهای خارجی در جدول SP تعریف شـده‌اند و مطابق تعریف محتوایی کلید خـارجی و قـانون جـامعیـت ارجـاعی، مجموعـه مقـادیر کلیـد خـارجی بـایـد همواره زیر مجموعـه، مجموعه مقادیر کلید کاندید معادل خود باشـد تا ارجاع NULL ایجاد نگردد، یعنی مجموعه مقادیر کلید خارجی S# در جدول SP باید همواره زیر مجموعه، مجموعه مقادیر کلید کاندید معادل خود یعنی S# در جدول S باشـــد و مجموعه مقادیر کلید خارجی P# در جدول SP باید همواره زیر مجموعه، مجموعه مقادیر کلید کاندید معادل خود یعنی P# در جدول P باشـد، که درج مقادیر S10 و P10 در جدول SP این قاعده را نقض می‌کند، پس اجرای دســتور ذخیره‌ســازی بر روی دید Report4 به صورت عملی از سوی DBMS پذیرفته نمی‌شود و رد می‌شود.

همچنین فرض کنید کاربر با اجرای دسـتور زیر بخواهد یک سـطر جدید را در دید Report4 درج نماید:

```
Insert into Report4
Values ('S1', 'P1', 90)
```

از آنجا که دید Report4 اســتقلال وجودی ندارد، عملیات درج فوق در واقع روی جدول پایه SP انجام می‌گردد.

درواقع دستور فوق به هنگام اجرا، به دستور زیر تبدیل می‌گردد:

```
Insert into SP
Values ('S1', 'P1', 90)
```

در هنگام ذخیره‌سـازی عملی بر روی دید Report4 هر چند هر سـه سـتون جدول SP به واسـطه دسـتور درج، مقداردهی شـده‌اند، اما این‌بار **قانون جامعیت درون رابطه‌ای** نقض شـده اسـت، زیرا سـتون‌های S# و P# باهم به عنوان کلید کلنـدید در جدول SP تعریف شـــده‌لند و مطابق تعریف محتوایی کلید کاندید، مقادیر کلید کاندید باید یکتا باشـــد، که درج **مجدد** مقادیر S1 و P1 به طور همزمان در قالـب یـک سـطر در جـدول SP این قـاعـده را نقض می‌کنـد، پس اجرای دســتور ذخیره‌ســازی بر روی دید Report4 به صــورت عملی از ســوی DBMS پذیرفته نمی‌شــود و رد

می‌شود.

سه جدول S، P و SP در سیستم تولیدکنندگان و قطعات با مقادیر زیر را در نظر بگیرید:

S#	Sname	City
S1	Sn1	C1
S2	Sn2	C2
S3	Sn3	C3

جدول S

S#	P#	QTY
S1	P1	10
S1	P2	20
S2	P1	30

جدول SP

P#	Pname	Color
P1	Pn1	Red
P2	Pn2	Blue

جدول P

مثال: دید زیر را در نظر بگیرید:

```
Create View Report5
    AS Select S.S#, Sname, P#
    From S JOIN SP
```

در تعریف دید Report5 از الحاق طبیعی میان دو جدول S و SP استفاده شده است، بدنه دید Report5 به صورت زیر است:

S.S#	Sname	P#
S1	Sn1	P1
S1	Sn1	P2
S2	Sn2	P1

آنچه واضح است، این است که ساختار دید Report5 از نظر تئوری پذیرای ذخیره‌سازی است، یعنی قوانین مدل رابطه‌ای همچون داشتن حداقل یک کلید کاندید و داشتن ستون‌های واقعی و نه مجازی حفظ می‌گردد و همچنین تناظر سطر به سطر مابین دید و جدول پایه حفظ می‌گردد. کلید کلندید جدول حاصل از الحاق طبیعی همواره برابر کلید کلندید جدولی است که در آن کلید خارجی تعریف شده است، کلید کاندید جدول SP که در آن کلید خارجی تعریف شده است برابر (S#,P#) می‌باشد. بنابراین کلید کاندید جدول حاصل از الحاق طبیعی هم (S#,P#) است.

توجه مهم: شرط «**در تعریف دید فقط از یک جدول پایه با حداقل یک کلید کاندید استفاده شده باشد.**» برای پذیرا بودن ذخیره‌سازی بر روی دید را از نظر تئوری در الحاق طبیعی برقرار است، چون در نهایت خروجی الحاق طبیعی دو جدول یک جدول است که کلید کاندید آن نیز واضح و مشخص است.

فرض کنید کاربر با اجرای دستور زیر بخواهد یک سطر جدید را در دید Report5 درج نماید:

```
Insert into Report5
Values ('S4', 'Sn4', 'P3')
```

از آنجا که دید Report5 استقلال وجودی ندارد، عملیات درج فوق در واقع روی جداول پایه S و SP و P انجام می‌گردد.

درواقع دستور فوق به هنگام اجرا، به دستورات زیر می‌بایست تبدیل گردد:

```
Insert into S
Values ('S4', 'Sn4', NULL)

Insert into SP
Values ('S4', 'P3', NULL)

Insert into P
Values ('P3', NUUL, NULL)
```

در هنگام ذخیره‌سازی عملی بر روی دید Report5، برای مثال برای حفظ قوانین جامعیت بلنک می‌بایست در چند جای مختلف از بانک درج‌هایی براساس سه دستور فوق انجام گردد، که SQL در عمل توانایی انجام چنین کاری را ندارد، شناخت و اجرای دستورات فوق برای SQL امکان‌پذیر نیست، از آنجاکه عدم اجرای هر یک از دستورات فوق جامعیت بانک را برهم می‌زند، و SQL هم توانایی شناخت و اجرای همه دستورات فوق را ندارد، پس اجرای دستور فوق یعنی ذخیره‌سازی بر روی دید Report5 به صورت عملی از سوی DBMS پذیرفته نمی‌شود و رد می‌شود.

غیر پذیرا از نظر تئوری و عملی

در این حالت دید از نظر تئوری پذیرای ذخیره‌سازی نیست، یعنی قوانین مدل رابطه‌ای همچون داشتن حداقل یک کلید کاندید و داشتن ستون‌های واقعی و نه مجازی حفظ نمی‌گردد و همچنین تناظر سطر به سطر مابین دید و جدول پایه حفظ نمی‌گردد. همچنین از نظر عملی هم پذیرای ذخیره‌سازی نیست به دلیل عدم پذیرا بودن ذخیره‌سازی در دید از نظر تئوری.

موارد زیر پذیرا بودن ذخیره‌سازی بر روی دید را از نظر تئوری تضمین می‌کند:

- که در بخش قبل بررسی شد.

مورد زیر پذیرا بودن ذخیره‌سازی بر روی دید را از نظر عملی تضمین می‌کند:

- که در بخش قبل بررسی شد.

سه جدول S، P و SP در سیستم تولیدکنندگان و قطعات با مقادیر زیر را در نظر بگیرید:

S#	Sname	City
S1	Sn1	C1
S2	Sn2	C2
S3	Sn3	C3

جدول S

S#	P#	QTY
S1	P1	10
S1	P2	20
S2	P1	30

جدول SP

P#	Pname	Color
P1	Pn1	Red
P2	Pn2	Blue

جدول P

مثال: دید زیر را در نظر بگیرید:

```
Create view Report6
AS Select S# , QTY
    From SP
```

Where QTY>10

در تعریف دید Report6 از یک جدول پایه استفاده شده است، بدنه دید Report6 به صورت زیر است:

S#	QTY
S1	20
S2	30

آنچه واضح است، این است که ساختار دید Report6 از نظر تئوری پذیرای ذخیره‌سازی نیست، یعنی قوانین مدل رابطه‌ای همچون داشتن حداقل یک کلید کاندید حفظ نشده است و بخشی از کلید اصلی (S#,P#) از جدول SP حذف شده است.

فرض کنید کاربر با اجرای دستور زیر بخواهد یک سطر جدید را در دید Report6 درج نماید:

```
Insert into Report6
Values ('S3', 50)
```

از آنجا که دید Report6 استقلال وجودی ندارد، عملیات درج فوق در واقع روی جدول پایه SP انجام می‌گردد. اما جدول SP دارای سه ستون است، درحالیکه، فقط دو ستون در دستور فوق مربوط به جدول SP مقداردهی شده‌است. بنابراین برای ستون P# مقدار NULL در نظر گرفته می‌شود. در هنگام تعریف جدول SP ترکیب ستون‌های (S#,P#) باهم کلید اصلی معرفی شده‌است. که به طور پیش فرض ستون‌های کلید اصلی NOT NULL در نظر گرفته می‌شود، یعنی حتما می‌بایست ستون‌های کلید اصلی (S#,P#) دارای مقدار باشد و مقدار NULL برای آن قابل پذیرش نیست. بنابراین مطابق **قانون جامعیت موجودیت** بلنک نباید ستون S# و P# در جدول SP با NULL مقداردهی شود.

درواقع دستور فوق به هنگام اجرا، به دستور زیر تبدیل می‌گردد:

```
Insert into SP
Values ('S3', NULL , 50)
```

در هنگام ذخیره‌سازی عملی بر روی دید Report6 مطابق قانون جامعیت موجودیت بلنک در جداول نمی‌توان در ستون‌های (S#,P#) به عنوان کلید اصلی که به طور پیش فرض خاصیت NOT NULL برای آن در نظر گرفته شده است، مقدار NULL را درج کرد، پس اجرای دستور ذخیره‌سازی بر روی دید Report6 به صورت عملی از سوی DBMS پذیرفته نمی‌شود و رد می‌شود. بنابراین ذخیره‌سازی بر روی دید Report6 به صورت عملی قابل انجام نیست به دلیل عدم پذیرا بودن ذخیره‌سازی در دید Report6 از نظر تئوری. دقت کنید که وقتی تئوری ذخیره‌سازی روی دید امکان‌پذیر نباشد به تبع قطعا ذخیره‌سازی عملی آن هم امکان‌پذیر نیست.

مثال: دید زیر را در نظر بگیرید:

```
Create View Report7
    AS Select S#, AVG (QTY)
From SP
Group by S#
```

در تعریف دید Report7 از یک جدول پایه، دستور group by و تابع آماری AVG استفاده شده است، بدنه دید Report7 به صورت زیر است:

S#	no column name
S1	15
S2	30

آنچه واضح است، این است که دید Report7 از نظر تئوری پذیرای ذخیره‌سازی نیست، یعنی قوانین مدل رابطه‌ای همچون داشتن حداقل یک کلید کاندید و داشتن ستون‌های واقعی و نه مجازی حفظ نمی‌گردد و همچنین تناظر سطر به سطر مابین دید و جدول پایه حفظ نمی‌گردد.

فرض کنید کاربر با اجرای دستور زیر بخواهد یک سطر جدید را در دید Report7 درج نماید:

```
Insert into Report7
Values ('S3', 12)
```

در هنگام ذخیره‌سازی عملی بر روی دید Report7، برای مثال، مقصد مقدار 12 مشخص نیست، زیرا مقصد آن یک ستون مجازی است و نه یک ستون واقعی، اگر به فرض محال هم درج شود میانگین تعداد قطعات S3 برابر 12 است حال آنکه #P آن مشخص نیست، پس اجرای دستور ذخیره‌سازی بر روی دید Report7 به صورت عملی از سوی DBMS پذیرفته نمی‌شود و رد می‌شود. بنابراین ذخیره‌سازی بر روی دید Report7 به صورت عملی قابل انجام نیست به دلیل عدم پذیرا بودن ذخیره‌سازی در دید Report7 از نظر تئوری. ضمن اینکه در تعریف دید Report7 همانند Report6 بازهم بخشی از کلید اصلی حذف شده‌است و قانون جامعیت موجودیت نیز نقض شده است.

تست‌های فصل هفتم: SQL دستورات DDL

۱۱۰- روش Cascade در قاعده تمامیت ارجاعی در پایگاه داده‌ها، چه کارکردی دارد؟

(مهندسی کامپیوتر-دولتی۱۴۰۳)

۱) فقط برای به‌روزرسانی استفاده می‌شود و هیچ تأثیری بر حذف رکوردها ندارد.

۲) هنگامی که یک رکورد در جدول مرجع حذف یا به‌روزرسانی می‌شود، فقط تغییرات حذف در جدول‌های مرتبط اعمال می‌شود.

۳) فقط درصورت تغییر مقدار ستون‌های غیرکلید در یک جدول، تغییرات را در جدول‌های دیگر اعمال می‌کند.

۴) هنگامی که یک رکورد در جدول مرجع حذف یا به‌روزرسانی می‌شود، تغییرات به‌صورت خودکار در جدول‌های مرتبط با کلیدهای خارجی اعمال می‌شود.

استاد ارسطو خلیلی فر
مولف کتاب پایگاه داده انتشارات بابان

پاسخ تست‌های هفتم: SQL دستورات DDL

۱۱۰- گزینه (۴) صحیح است.

پاسخ سریع: در زبان SQL هنگام تعریف کلید خارجی در جدول فرعی، می‌توان از دستورات on delete cascade و on update cascade استفاده کرد. بنابراین وقتی رکوردی از جدول اصلی (مرجع) حذف یا به‌روزرسانی شود، رکوردهای مرتبط در جدول فرعی به طور خودکار، حذف delete یا بروزرسانی update می‌شوند.

پاسخ تشریحی:

توجه: برای رفتار ستون کلید خارجی در یک جدول فرعی، در قبال تغییرات کلید کاندید از یک جدول اصلی گزینه‌های زیر وجود دارد:

```
Create Table name
        (attr domain [NOT NULL],
         .
         .
         .
        attr …,
        Primary key (…),
        [Unique (…)],
        [Foreign key (…) References …(…)]
              [on delete option]
              [on update option],
        [Check (…)]
       )
```

فیلد option می‌تواند یکی از موارد زیر باشد:

(restrict) no action

گزینه پیش فرض است و هیچ عملی انجام نمی‌شود. اگر بر اثر عملیات **حذف** یا **بروزرسانی** در جدول اصلی، قانون جامعیت ارجاعی در جدول فرعی نقض گردد، آنگاه این اعمال انجام نمی‌گردد. در واقع زمانی عملیات حذف یا بروزرسانی در جدول اصلی انجام می‌گردد که قانون جامعیت ارجاعی در جدول فرعی نقض نگردد. به عبارت دیگر زمانی عملیات حذف یا بروزرسانی در سطری از جدول اصلی انجام می‌گردد که سطری از جدول فرعی به آن ارجاع نکرده باشد. برای مثال مقدار S3 در جدول S می‌تواند به S33 بروزرسانی شود و یا از جدول S حذف شود، زیرا

هیچ ارجاعی روی مقدار S3 از جدول SP به جدول S وجود ندارد. اما مقدار S1 در جدول S مطابق رابطه no action نمی‌تواند به S11 بروزرسانی شود و یا از جدول S حذف شود، زیرا ارجاعی روی مقدار S1 از جدول SP به جدول S وجود دارد.

Cascade

اگر سطرهای جدول اصلی **حذف** یا **بروزرسانی** شود، آنگاه سطرهای جدول فرعی که توسط کلید خارجی به آن ارجاع کرده است نیز حذف یا بروزرسانی می‌شود. برای مثال اگر مقدار S1 در جدول S به S11 بروزرسانی شود و این مقدار در جدول SP نیز موجود باشد، آنگاه مطابق رابطه Cascade مقدار S1 در جدول SP به مقدار S11 نیز بروزرسانی می‌شود. برای مثالی دیگر اگر سطر S1 از جدول S حذف شود و این مقدار در جدول SP نیز موجود باشد، آنگاه مطابق رابطه Cascade سطر S1 از جدول SP نیز حذف می‌شود.

Set NULL (NULLIFY)

اگر سطرهای جدول اصلی **حذف** یا **بروزرسانی** شود، آنگاه ستون‌های جدول فرعی که توسط کلید خارجی به آن ارجاع کرده است با مقدار NULL پُر می‌شود. برای مثال اگر مقدار S1 در جدول S به S11 بروزرسانی شود و این مقدار در جدول SP نیز موجود باشد، آنگاه مطابق رابطه Set NULL مقدار S1 در جدول SP به مقدار NULL تغییر می‌کند. برای مثالی دیگر اگر مقدار S1 از جدول S حذف شود و این مقدار در جدول SP نیز موجود باشد، آنگاه مطابق رابطه Set NULL مقدار S1 در جدول SP به مقدار NULL تغییر می‌کند.

کارکرد قطعه کد زیر از کد تعریف جدول SP به صورت زیر است:

```
Foreing key (S#) References S(S#)
on delete cascade
on update casecade
```

این قطعه کد، سبب می‌گردد تا به طور خودکار هرگونه تغییری اعم از حذف یا بروزرسانی در سطرهای ستون S# در جدول S به سطرهای ستون S# در جدول SP نیز اعمال گردد. بنابراین **جامعیت داخلی** از نوع جامعیت ارجاعی نقض نمی‌گردد.

یا به طور مشابه، کارکرد قطعه کد زیر از کد تعریف جدول SP به صورت زیر است:

```
Foreing key (P#) References P(P#)
on delete cascade
on update casecade
```

این قطعه کد، سبب می‌گردد تا به طور خودکار هرگونه تغییری اعم از حذف یا بروزرسانی در سطرهای ستون P# در جدول P به سطرهای ستون P# در جدول SP نیز اعمال گردد. بنابراین **جامعیت داخلی** از نوع جامعیت ارجاعی نقض نمی‌گردد.

تست‌های فصل اول: مفاهیم پایه

۱۱۱- در مدیریت پایگاه داده‌ها، Schema Evolution چه چالش‌هایی را به‌همراه دارد؟

(مهندسی کامپیوتر-دولتی۱۴۰۳)

۱) تغییر Schema بدون اختلال در دسترسی کاربران به پایگاه داده و بدون از دست دادن یا آسیب به داده‌های موجود، یک چالش است.

۲) تغییر Schema به‌طور معمول نیازمند بازنویسی کل برنامه‌های کاربردی است که با پایگاه داده در ارتباط هستند.

۳) Schema Evolution فقط در پایگاه داده‌های شیءگرا امکان‌پذیر است و در سایر انواع پایگاه داده‌ها امکان‌پذیر نیست.

۴) Schema Evolution به معنای تغییر داده‌های ذخیره شده در پایگاه داده است و به همین دلیل اغلب منجر به از دست رفتن داده‌ها می‌شود.

استاد ارسطو خلیلی فر

مولف کتاب پایگاه داده انتشارات بابان

پاسخ تست‌های اول: مفاهیم پایه

۱۱۱- گزینه (۱) صحیح است.

تکامل شمای داده (Schema Evolution): طراحی پایگاه داده به روش ساخت‌یافته یا شی‌گرا یک فعالیت یک‌باره و برای همیشه ماندگار نیست. تغییر و توسعه، جزء جدایی ناپذیر محصولات نرم‌افزاری است. نیازهای یک سازمان به‌طور مداوم تکامل می‌یابد و داده‌هایی که نیاز به ذخیره‌سازی آن‌ها دارد نیز به‌طور متناظر تغییر می‌کنند و به تبع منجر به تغییر ساختار جداول می‌شود و در نهایت ممکن است منجر به تغییرات برنامه‌کاربردی شود. بنابراین تکامل شمای (ساختار یا اسکیما) داده در طول زمان بسته به تغییر نیازهای کسب‌وکار امری طبیعی است. تکامل شمای داده فقط مختص روش ساخت‌یافته (تابع‌گرا) یا شی‌گرا (کلاس‌گرا) نیست، بلکه پایگاه داده و برنامه کاربردی مرتبط به آن چه به روش ساخت‌یافته و با ابزارهای مدل ER ایجاد شود و چه به روش شی‌گرا و با ابزارهای مدل UML ایجاد شود قطعا در طول زمان نیاز به توسعه و تکامل دارد. اما آنچه اهمیت دارد این است که شمای داده اولیه چه به روش ساخت‌یافته و چه به روش شی‌گرا طوری نرمال‌سازی و طراحی شود که امکان توسعه‌های بعدی را داشته باشد. بدیهی است که تغییرات ساختار پایگاه داده منجر به تغییرات برنامه‌کاربردی خواهد شد. اما در یک طراحی پایگاه داده نرمال، تلاش این است که تغییرات در ساختار پایگاه داده منجر به حداقل تغییرات در برنامه کاربردی شود. و این همان معنای استقلال داده‌ای است. یک طراحی پایگاه داده نرمال، نیازهای آینده یک سازمان را پیش‌بینی می‌کند و اطمینان می‌دهد که شمای داده حداقل تغییرات را در هنگام تکامل نیازها نیاز دارد.

توجه: تغییر و توسعه Schema بدون اختلال، در دسترسی کاربران به پایگاه داده و بدون از دست دادن یا آسیب به داده‌های موجود، یک چالش است و این چالش در اغلب موارد قابل حل و کنترل است. اما این چالش آنقدرها بزرگ نیست که منجر به تغییر کل ساختار پایگاه داده شود تا به تبع منجر به تغییر کل برنامه‌کاربردی شود. تکامل شمای داده به تغییر و تکامل ساختار پایگاه داده مرتبط است و این موضع ارتباطی به محتوای داخل جداول پایگاه داده به معنی تغییر داده‌های ذخیره شده در پایگاه داده ندارد.

جمله تاکیدی: هرچه میزان استقلال منطقی ساختار پایگاه داده بیشتر باشد، آنگاه تکامل شمای داده‌ها آسان‌تر و راحت‌تر و با چالش کمتری همراه خواهد بود.

تست‌های فصل سوم: مدل رابطه‌ای

۱۱۲- عبارت «is a type of» در مدل‌سازی داده‌ها، به چه منظوری استفاده می‌شود؟

(مهندسی کامپیوتر-دولتی۱۴۰۳)

۱) نشان‌دادن رابطه یک به یک بین دو جدول در پایگاه داده

۲) نشان‌دادن روابط چند به چند بین جداول در پایگاه داده

۳) مشخص‌کردن انواع داده‌هایی که در یک ستون خاص می‌توانند ذخیره شوند.

۴) تعریف یک سلسله مراتب وراثت بین کلاس‌ها یا جداول، جایی که یک کلاس یا جدول خاص، زیرمجموعه‌ای از کلاس یا جدول دیگری است.

استاد ارسطو خلیلی فر
مولف کتاب پایگاه داده انتشارات بابان

پاسخ تست‌های فصل سوم: مدل رابطه‌ای

۱۱۲- گزینه (۴) صحیح است.

پاسخ سریع: به سلسله مراتب وراثت بین کلاس‌ها یا جداول، ارث‌بری گفته می‌شود. رابطه وراثت به عنوان رابطه «**is a**» یا «**is a kind of**» یا «**is a type of**» نیز شناخته می‌شود.

پاسخ تشریحی: (بعلاوه تدریس کامل سرفصل‌های جدید وزارت علوم مصوب سال ۱۴۰۳)

وراثت (Inheritance) در ساخت‌یافته (مدل رابطه‌ای)

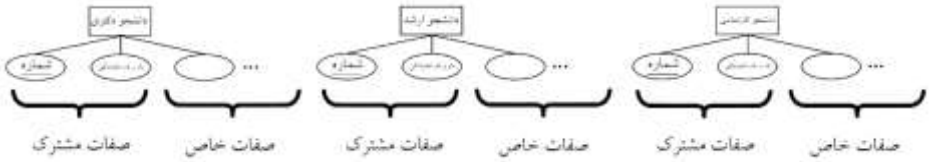
وراثت در ساخت‌یافته (مدل رابطه‌ای) در ادامه توسط تعمیم و تخصیص مورد تحلیل و بررسی کامل قرار می‌گیرد.

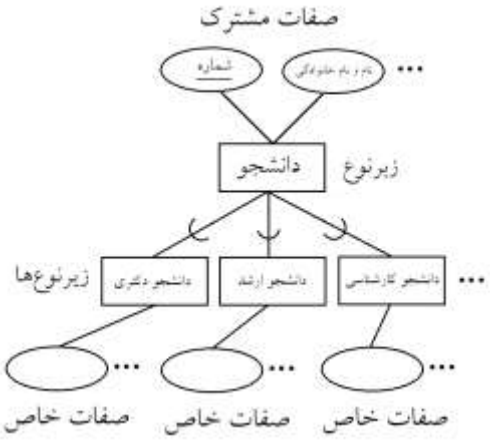
تعمیم (Generalization) و تخصیص (Specification)

به فرآیند انتقال **صفات و شناسه مشترک** بین موجودیت‌های هم خانواده به سمت بالا **تعمیم** گفته می‌شود و به فرآیند انتقال **صفات متفاوت** بین موجودیت‌های هم خانواده به سمت پایین **تخصیص** گفته می‌شود. این مدل‌سازی به این صورت تفسیر معنایی می‌شود که، اطلاعات یک رکورد اطلاعاتی به دو تکه **اطلاعات عمومی** و **اطلاعات اختصاصی** تقسیم می‌شود. اطلاعات عمومی در بخش تعمیم (پدر) نگهداری می‌شود و اطلاعات اختصاصی در بخش تخصیص (فرزند) نگهداری می‌شود. در نهایت دو سطر درج شده در دو جدول پدر و فرزند بهم مرتبط می‌شوند. تعمیم و تخصیص را با یک درخت به نام درخت تعمیم و تخصیص نمایش می‌دهند. اگر d تعداد سطوح این درخت باشد، داریم: $d \geq 2$. تعمیم و تخصیص به دو نوع کامل و ناقص به شکل رابطه پوشا (Overlap) و رابطه غیرپوشا (Disjoint) است که در ادامه مورد بررسی قرار می‌گیرد.

توجه: لازمه تعمیم و تخصیص این است که حداقل دو نوع موجودیت از پیش دیده شده داشته باشیم که حداقل در صفت شناسه، اشتراک دارند.

مثال: نوع موجودیت‌های دانشجوی کارشناسی، دانشجوی ارشد و دانشجوی دکتری را در نظر می‌گیریم. این نوع موجودیت‌ها، مجموعه‌ای از صفات مشترک دارند، مثل شماره دانشجو، نام و نام خانوادگی، سال ورود و ... و در عین حال، هریک از این انواع می‌توانند صفات خاص خود را هم داشته باشد. یک زبرنوع (ابرنوع) به نام نوع موجودیت دانشجو در نظر می‌گیریم و مجموعه صفات مشترک بین آن نوع موجودیت‌ها را به این زبرنوع می‌دهیم. به این ترتیب عمل تعمیم انجام داده‌ایم. آن نوع موجودیت‌ها اینک هریک زیرنوع این زبرنوع می‌شوند، با صفات خاص خود. به این ترتیب عمل تخصیص انجام داده‌ایم.





توجه: به سلسله مراتب فوق **ارث‌بری** یا **وراثت** گفته می‌شود. رابطه وراثت به عنوان رابطه «**is a**» یا « **is a kind of**» یا «**is a type of**» نیز شناخته می‌شود.

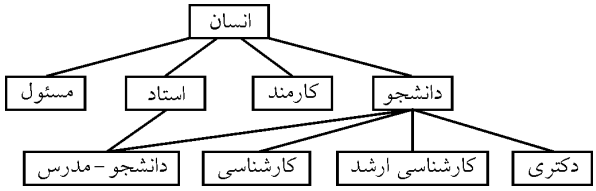
وراثت (Inheritance) در شی‌گرایی

وراثت فرآیندی است که به وسیله آن یک کلاس فرزند می‌تواند صفات و متدهای عمومی کلاس پدر را کسب کند. به عبارت کلی‌تر یک کلاس فرزند ضمن به ارث بردن مجموعه‌ای از صفات و متدهای عمومی کلاس پدر، می‌تواند ویژگی‌های خاص و مختص خود را نیز به آنها اضافه کند. به این ترتیب، سلسله مراتبی از کلاس‌ها **(Class Hierarchy)** ایجاد می‌شود که قدرت انعطاف‌پذیری زیادی به برنامه‌ساز می‌بخشد. سلسله مراتب کلاس‌ها به صورت درخت است. به عبارت دیگر مدل ریاضی وراثت، درخت است. در این درخت، کلاس‌ها به صورت «ابرکلاس» **(Super Class)** و «زیرکلاس» **(Sub Class)** در رده‌های مختلف قرار می‌گیرند.

در بالاترین سطح، کلاسی قرار دارد که خصوصیات مشترک همه کلاس‌های موردنظر را در خود جای می‌دهد. این کلاس از هیچ کلاس دیگری مشتق نمی‌شود (ارث‌بری ندارد) برگ‌های این درخت را کلاس‌هایی تشکیل می‌دهند که هیچ کلاسی از آنها مشتق نمی‌شود. در سطح‌های میانی، کلاس‌هایی قرار می‌گیرند که هم‌ارث می‌برند و هم کلاس‌های دیگر از آنها مشتق می‌شوند.

هر زیرکلاس، داده‌ها و متدهای کلاس‌های رده بالاتر را به ارث می‌برد و نیازهای ویژه خود را به آن می‌افزاید. کلاس ممکن است بعضی از صفات و متدهای کلاس پدر را به ارث نبرد (صفات و متدهای خصوصی) و یا آنها را تغییر دهد(دوباره‌نویسی کند). شکل زیر سلسله مراتب ساده‌ای از کلاس‌های «دانشگاهیان» را نشان می‌دهد. خصوصیات کلی همه انسان‌ها که در این کاربرد خاص مطرح می‌شود. در

کلاس انسان می‌آید و ویژگی‌های هر کدام (مثل نمره دانشجو، تخصص استاد، نوع و محل مسئولیت برای مسئول و اضافه کاری برای کارمند) در کلاس مربوط به آنها ذکر می‌شود.
کلاس (دانشجو- مدرس) حالت خاصی دارد. این کلاس مربوط به افرادی است که هم درس می‌خوانند و هم درس می‌دهند. این گونه افراد هم از کلاس دانشجو ارث می‌برند، هم از کلاس استاد. این نوع ارث‌بری را «ارث‌بری چندگانه» **(Multiple Inheritance)** می‌نامند.



سلسله مراتب ساده‌ای از کلاس‌های «دانشگاهیان»

زمانی که نیاز به ایجاد یک کلاس جدید باشد مهندس نرم‌افزار چند انتخاب دارد:

۱) کلاس جدید بدون استفاده از وراثت از ابتدا طراحی و ساخته شود.
۲) سلسله مراتب کلاس‌ها بررسی شود تا یک کلاس بالاتر که حاوی اکثر صفات و متدهای موردنیاز است پیدا شود. سپس کلاس جدید از کلاس بالاتر ارث برده و باقی صفات و متدهای موردنظر به آن اضافه گردد.

مدل‌سازی رابطه وراثت بین کلاس‌ها در نمودار کلاس در شی‌گرایی

به طورکلی هرگاه یک کلاس (ب)، از نوع یک کلاس (الف) باشد، گوییم بین دو کلاس مذکور رابطه وراثت برقرار است. در این صورت کلاس (ب) فرزند و کلاس (الف) پدر یا والد نامیده می‌شود. به عبارت دیگر همانطور که پیش از این نیز گفتیم، وراثت فرآیندی است که به وسیله آن یک کلاس (فرزند) می‌تواند صفات و متدهای کلاس دیگری (پدر) را کسب کند. به عبارت کلی‌تر یک کلاس فرزند ضمن به ارث بردن مجموعه‌ای از صفات و متدهای عمومی کلاس پدر، می‌تولند ویژگی‌های خاص و مختص خود را نیز به آنها اضافه کند. در رابطه ورلثت هر تغییر در کلاس پدر بر کلاس فرزند نیز اثر می‌گذارد اما عکس این مطلب برقرار نیست. برای مثال، بین سیب و میوه، رابطه ارث‌بری برقرار است، سیب (فرزند) یک نوع میوه (پدر) است. بنابراین یک سیب دارای تمام ویژگی‌های یک میوه است. بنابراین از دیدگاه وراثت، سیب نوعی میوه است، زیرا تمام ویژگی‌های میوه را به ارث می‌برد و البته دارای یک سری ویژگی‌های مختص به خودش نیز هست. از دیدگاهی دیگر، میوه تعمیم یافته سیب است، زیرا ویژگی‌های عمومی سیب که در میوه‌های دیگر نیز مشترک است در میوه گردآوری شده است. ابرکلاس، همان کلاس پدر است که ویژگی‌ها و عملیات خود را در اختیار کلاس‌های دیگر (فرزندان) قرار می‌دهد. برای مثال، «سیب» یک ابرکلاس است، زیرا ویژگی‌های خود را در اختیار کلاس «سیب قرمز» قرار می‌دهد. زیرکلاس نیز همان کلاس فرزند است که برخی از صفات و عملیاتش متعلق به یک ابرکلاس است. کلاس «سیب قرمز» نمونه‌ای از یک زیرکلاس است.

دو رویکرد برای ایجاد سلسله مراتب وراثت وجود دارد:

۱- رویکرد «بالا به پایین» یا **تخصیص** (**Specialization**) که در آن، ابتدا کلاس‌های پدر، تعریف و سپس کلاس‌های فرزند ایجاد می‌شوند.

۲- رویکرد «پایین به بالا» یا **تعمیم** (**Generalization**) که در آن، از طریق تعمیم کلاس‌های فرزند به کلاس‌های پدر می‌رسیم.

توجه: رابطه وراثت به عنوان رابطه «**is a**» یا «**is a kind of**» یا «**is a type of**» نیز شناخته می‌شود. زیرا در این رابطه می‌گوییم: «**A red apple is an apple**» و یا «**An apple is a fruit**».

مراحل لازم جهت مدل‌سازی یک رابطه ارث‌بری عبارتند از:

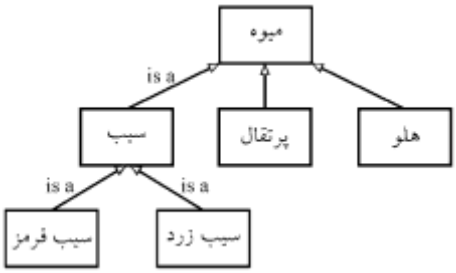
۱- ترسیم یک خط ممتد بین کلاس فرزند و کلاس پدر

۲- درج یک مثلث تو خالی در انتهایی از خط ممتد که کلاس پدر قرار دارد.

توجه: ذکر «**is a**» بر روی خط ممتد رابطه وراثت اختیاری است.

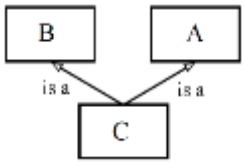
توجه: در تعریف رابطه وراثت، مشخصاتی نظیر تعدد و محدودیت وجود ندارد.

شکل مقابل نمونه‌ای از یک سلسله مراتب ارث‌بری میان «میوه»، «سیب» و «سیب قرمز»، را نشان می‌دهد.



توجه: هرگاه یک کلاس برخی از صفات و عملیات را از یک کلاس و برخی دیگر را از یک کلاس دیگر به ارث ببرد، در این حالت وراثت چندگانه رخ داده‌است.

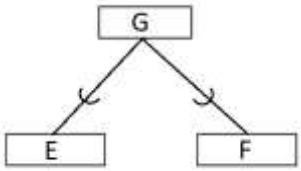
شکل مقابل نمونه‌ای از مدل‌سازی «**ارث‌بری چندگانه**» (**Multiple Inheritance**) را نشان می‌دهد.



در شکل فوق کلاس **C** به صورت وراثت چندگانه از کلاس **A** و کلاس **B** ارث‌بری کرده است.

روش عمومی نگاشت رابطه ISA یا وراثت به مدل رابطه‌ای

اگر در مسائل وراثت، Disjoint و Overlap روی مدل لحاظ نشود، نگاشت به مدل رابطه‌ای از **روش عمومی** انجام می‌شود.



در این حالت به n+1 جدول نیاز داریم. اگر ابرنوع n زیرنوع داشته باشد یک جدول برای ابرنوع در نظر می‌گیریم و برای هر یک از زیرنوع‌ها هم یک جدول با صفت‌های خاص هر زیرنوع در نظر گرفته می‌شود. همچنین کلید کلندید جدول پدر(ابرنوع یا زبرنوع) در جداول فرزند (زیرنوع) به عنوان کلید خارجی تعریف می‌گردد و همزمان در جداول فرزند کلید کاندید هم می‌باشد. و یک طراحی به شکل مدل n+1 جدولی اما با تحمل سربار حاصل از عمل الحاق مابین جدول زبرنوع و زیرنوع ایجاد می‌گردد.

مثال: نوع موجودیت دانشجو و گونه‌های خاص آن: دانشجوی «کارشناسی» ، دانشجوی «ارشد» و دانشجوی «دکتری» را در نظر می‌گیریم. چهار رابطه طراحی می‌کنیم به صورت زیر:

STUD (STID , STNAME , ...)
C.K.

صفات مشترک در همه نمونه‌های نوع موجودیت دانشجو در عنوان رابطه STUD می‌آیند.

BSSTUD (STID , A , B , ...)
C.K. , F.K.

MSSTUD (STID , M , N , ...)
C.K. , F.K.

DOCSTUD (STID , X , Y , Z , ...)
C.K. , F.K.

می‌بینیم که در این روش هرگاه زبرنوع E دارای n زیرنوع E_i باشد، n+1 جدول طراحی می‌شود.
در هریک از سه جدول آخر صفات خاص دانشجویان دوره کارشناسی، ارشد و دکتری در نظر گرفته می‌شوند و دیگر لزومی ندارد که صفات مشترک بین این گونه‌ها، که در همان جدول STUD آورده می‌شوند، در جداول دیگر تکرار شوند، چون گونه‌های دانشجو این صفات را از نوع موجودیت دانشجو به ارث می‌برند. البته کلید کاندید STUD باید در سه جدول دیگر آورده شود.
این مثال **روش عمومی** نمایش مفاهیم زبرنوع (Supertype) و زیرنوع (Subtype) در مدل رابطه‌ای را نشان می‌دهد.
توجه: دقت داشته باشیم که در جداول نشان‌دهنده هر زیرنوع، کلید خارجی جدول، کلید کاندید آن هم هست.

سوال: نوع موجودیت‌های F و G مفروضند. می‌خواهیم با استفاده از تکنیک **تعمیم**، نوع موجودیت E را بر اساس دو نوع موجودیت F و G ، در مدل‌سازی منظور کنیم. در اینصورت نوع موجودیت‌های F و G باید:

الف) حداقل یک صفت هیچ‌مقدار ناپذیر مشترک داشته باشند.

ب)حداقل شناسه مشترک داشته باشند.

ج) در دو صفت، که یکی از آنها شناسه باشد، مشترک باشند.

د) در صفت شناسه و یک صفت چندمقداری، مشترک باشند.

سوال: اگر N تعداد صفات مشترک بین n نوع موجودیت باشد، در تعمیم این n نوع موجودیت، یک شرط لازم کدام است؟

الف) $N \geq 2$

ب) $N \geq 1$

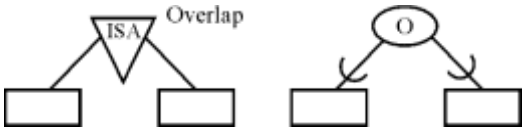
ج) $N > 1$

د) $N > 2$

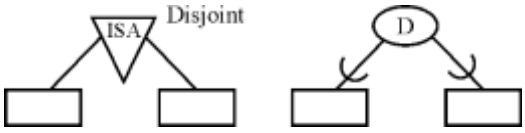
نگاشت رابطه ISA یا وراثت به مدل رابطه‌ای

در رابطه ISA رابطه پدر با فرزندان به دو صورت رابطه اختیاری یا ناقص یا جزئی (Partial) با نماد خط عمودی و رابطه اجباری یا کامل یا کلی (Total) با نماد خط مضاعف عمودی است و رابطه فرزندان با پدر به دو صورت رابطه پوشا یا تخصیص غیرمجزا (Overlap) و رابطه غیرپوشا یا تخصیص مجزا (Disjoint) می‌باشد.

رابطه پوشا یا تخصیص غیرمجزا (Overlap) مابین فرزندان و پدر به دو شیوه زیر نشان داده می‌شود:



رابطه غیرپوشا یا تخصیص مجزا (Disjoint) مابین فرزندان و پدر به دو شیوه زیر نشان داده می‌شود:



توجه: نگاشت رابطه ISA یا وراثت به مدل رابطه‌ای به سه مدل انجام می‌شود که در ادامه بررسی می‌شود:

مدل تک جدولی

در این مدل همه ستون‌های موجودیت زبرنوع و زیرنوع در یک جدول قرار می‌گیرد.

مدل دو جدولی

در این مدل کل صفات موجودیت زبرنوع در جداول زیرنوع قرار داده می‌شود و یک طراحی به شکل مدل دو جدولی ایجاد می‌گردد. کلید کلندید هر جدول زیرنوع همان کلید کلندید زبرنوع است.

مدل سه جدولی

در این مدل به n+1 جدول نیاز داریم. اگر زبرنوع n زیرنوع داشته باشد یک جدول برای زبرنوع در نظر می‌گیریم و برای هر یک از زیرنوع‌ها هم یک جدول با صفت‌های خاص هر زیرنوع در نظر گرفته می‌شود. همچنین کلید کلندید جدول پدر(ابرنوع یا زبرنوع) در جداول فرزند (زیرنوع) به عنوان کلید خارجی تعریف می‌گردد و همزمان در جداول فرزند کلید کاندید هم می‌باشد.

توجه: در بررسی سربار جداول وجود مقادیر NULL غیرقابل تحمل‌ترین نوع سربار، سپس وجود سربار افزونگی طبیعی و نه افزونگی تکنیکی در جایگاه بعدی و در نهایت سربار الحاق طبیعی در جایگاه آخر به عنوان قابل تحمل‌ترین نوع سربار است.

توجه بسیار مهم: جهت ساده به خاطر سپردن مدل‌های مختلف نگاشت موارد زیر را در نظر بگیرید:

مورد اول: در بحث نگاشت وراثت به مدل رابطه‌ای افزایش مقادیر NULL از **فرزند نداشتن پدر** حاصل می‌شود و این حالت فقط و فقط زمانی ایجاد می‌شود که مدل تک جدولی و اختیاری باشد یا مدل دو جدولی و اختیاری باشد.

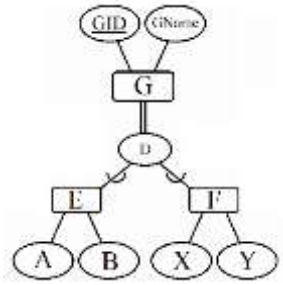
مورد دوم: در بحث نگاشت وراثت به مدل رابطه‌ای افزایش افزونگی از **تکرار رکورد پدر** حاصل می‌شود و این حالت فقط و فقط زمانی ایجاد می‌شود که مدل دو جدولی با حالت Overlap باشد.

مورد سوم: در بحث نگاشت وراثت به مدل رابطه‌ای سربار الحاق طبیعی از مدل سه جدولی حاصل می‌شود.

توجه: در یک عبارت ساده اختیاری بودن می‌تواند NULL ساز شود و Overlap بودن می‌تواند افزونگی‌ساز باشد.

الف) مدل تحلیل کامل (اجباری) و Disjoint

مدل تحلیل (نمودار ISA)



مدل طراحی (مدل رابطه‌ای)

مدل تک جدولی بهینه در شرایط خاص: دقت کنید که در شرایط خاص یعنی زمانی‌که (1) تعداد صفات زیرنوع زیاد، (2) تعداد زیرنوع‌ها کم و (3) تعداد صفات زیرنوع‌ها کم باشد مدل تحلیل کامل (اجباری) و Disjoint امکان پیاده‌سازی تک جدولی دارد و بهینه هم هست.

سربار افزایش مقادیر NULL ندارد: چون ارتباط زیرنوع با زیرنوع کامل (اجباری) است، پس رکوردهای زیرنوع حداقل یک فرزند دارد و به تبع مقادیر NULL افزایشی نمی‌شود. همچنین تعداد صفات زیرنوع‌ها هم جهت NULL سازی کم است.

سربار افزایش افزونگی ندارد: چون در مدل تک جدولی یک رکورد خاص زیرنوع فقط در یک جدول درج می‌شود.

سربار الحاق طبیعی ندارد: چون در مدل تک جدولی الحاق میان جدول زیرنوع و زیرنوع وجود ندارد.

صفات مشترک		صفات خاص E		صفات خاص F	
GID	GName	A	B	X	Y
GID1	**GN1**	**A1**	**B1**	NULL	NULL
GID2	**GN2**	**A2**	**B2**	NULL	NULL
GID3	**GN3**	NULL	NULL	**X3**	**Y3**

جدول GEF

توجه: دقت کنید چون رابطه پدر با فرزندان از نوع کامل (اجباری) است، پس هر رکورد از پدر حداقل باید یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 حداقل یک فرزند مثل A1,B1 از موجودیت E دارد.

توجه: دقت کنید چون رابطه فرزندان با پدر از نوع Disjoint است، پس هر رکورد از پدر فقط و فقط می‌تواند یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 فقط و فقط می‌تواند یک فرزند مثل A1,B1 از موجودیت E یا از موجودیت F داشته باشد.

مدل دو جدولی بهینه: مدل تحلیل کامل (اجباری) و Disjoint امکان پیاده‌سازی دو جدولی دارد و بهینه هم هست.

سربار افزایش مقادیر NULL ندارد: چون ارتباط زیرنوع با زیرنوع کامل (اجباری) است، پس رکوردهای زیرنوع حداقل یک فرزند دارد و به تبع مقادیر NULL افزایشی نمی‌شود.

سربار افزایش افزونگی ندارد: چون ارتباط زیرنوع با زیرنوع Disjoint است، پس یک رکورد خاص زیرنوع G در دو جدول E و F تکرار نمی‌شود.

سربار الحاق طبیعی ندارد: چون در مدل دو جدولی الحاق میان جدول زبرنوع و زیرنوع وجود ندارد.

GID	GName	A	B
GID1	**GN1**	A1	B1
GID2	**GN2**	A2	B2

جدول E

GID	GName	X	Y
GID3	**GN3**	X3	Y3

جدول F

مدل سه جدولی غیربهینه به دلیل سربار الحاق طبیعی: مدل تحلیل کامل (اجباری) و Disjoint امکان پیاده‌سازی سه جدولی دارد اما بهینه نیست.

سربار افزایش مقادیر NULL ندارد: چون در مدل سه جدولی جداول زبرنوع و زیرنوع ازهم جدا هستن.

سربار افزایش افزونگی ندارد: چون در مدل سه جدولی جداول زبرنوع و زیرنوع ازهم جدا هستن.

سربار الحاق طبیعی دارد: چون در مدل سه جدولی الحاق میان جدول زبرنوع و زیرنوع وجود دارد.

GID	GName
GID1	GN1
GID2	GN2
GID3	GN3

جدول G

GID	A	B
GID1	A1	B1
GID2	A2	B2

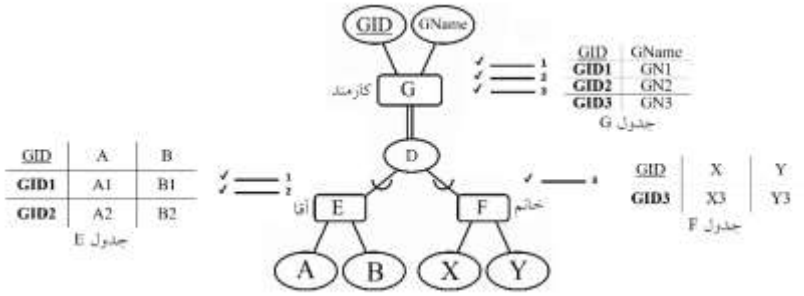
جدول E

GID	X	Y
GID3	X3	Y3

جدول F

توجه: دقت کنید که در رابطه‌های نشان دهنده هر زیرنوع، کلید خارجی جدول زیرنوع، کلید کاندید آن هم هست.

مدل شماتیک مدل تحلیل کامل (اجباری) و Disjoint به صورت زیر است:

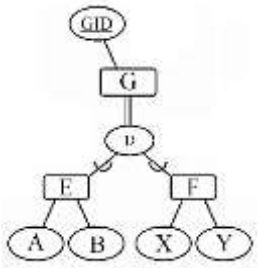


توجه: دقت کنید چون رابطه پدر با فرزندان از نوع کامل (اجباری) اســـت، پس هر رکورد از پدر حداقل باید یک فرزند داشـته باشـد. برای مثال رکورد GID1,GN1 حداقل یک فرزند مثل A1,B1 از موجودیت E دارد.

توجه: دقت کنید چون رابطه فرزندان با پدر از نوع Disjoint اســـت، پس هر رکورد از پدر فقط و فقط می‌تواند یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 فقط و فقط می‌تواند یک فرزند مثل A1,B1 از موجودیت E یا از موجودیت F داشته باشد.

توجه: اگر e مجموعه نمونه‌های E ، f مجموعه نمونه‌های F و g مجموعه نمونه‌های G باشد، آنگاه $card(e)+card(f)=card(g)$ است. یعنی $2+1=3$

ســـوال: در نمودار زیر، نوع موجودیت G فقط یک صـــفت شـــناســـه دارد. طراحی با چند رابطه (جدول) بهتر است انجام شود تا کمترین هیچ‌مقدار (NULL) و کمترین افزونگی پدید آید؟



الف) یک رابطه

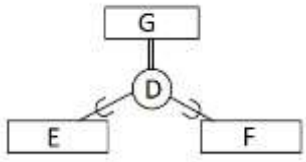
ب) دو رابطه

ج) سه رابطه

د) طراحی با یک یا دو رابطه باهم فرقی ندارند ولی از سه رابطه بهتر است.

پاسخ گزینه «ب» صحیح است.

سوال: با توجه به نمودار زیر :



اگر e مجموعه نمونه‌های E ، f مجموعه نمونه‌های F و g مجموعه نمونه‌های G باشد:

الف) $card(e) + card(f) \le card(g)$

ب) $card(e) + card(f) = card(g)$

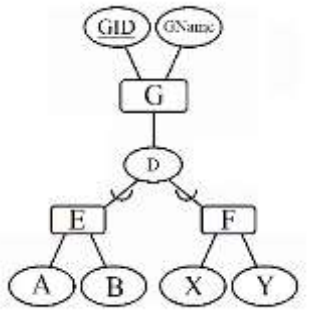
ج) $card(e) + card(f) \ne card(g)$

د) $card(e) + card(f) \ge card(g)$

پاسخ گزینه «ب» صحیح است.

ب) مدل تحلیل ناقص (اختیاری) و Disjoint

مدل تحلیل (نمودار ISA)



مدل طراحی (مدل رابطه‌ای)

مدل تک جدولی غیربهینه به دلیل افزایش مقدار NULL: دقت کنید که در شـــرایط خاص یعنی زمانی‌که (1) تعداد صـفات زیرنوع زیاد، (2) تعداد زیرنوع‌ها کم و (3) تعداد صـفات زیرنوع‌ها کم باشد مدل تحلیل ناقص (اختیاری) و Disjoint امکان پیاده‌سازی تک جدولی دارد اما بهینه نیست.

ســـربار افزایش مقدار NULL دارد: چون ارتباط زیرنوع با زیرنوع ناقص (اختیاری) اســـت، پس ممکن است رکوردهای زیرنوع، فرزند نداشته باشد و به تبع مقادیر NULL افزایشی می‌شود. مانند سطر 2. هرچند تعداد صفات زیرنوع‌ها هم جهت NULL سازی کم است.

ســـربار افزایش افزونگی ندارد: چون در مدل تک جدولی یک رکورد خاص زیرنوع فقط در یک جدول درج می‌شود.

ســـربار الحاق طبیعی ندارد: چون در مدل تک جدولی الحاق میان جدول زیرنوع و زیرنوع وجود ندارد.

صفات مشترک		صفات خاص E		صفات خاص F	
GID	GName	A	B	X	Y
GID1	**GN1**	**A1**	**A1**	NULL	NULL
GID2	**GN2**	NULL	NULL	NULL	NULL
GID3	**GN3**	NULL	NULL	**X3**	**Y3**

جدول GEF

توجه: دقت کنید چون رابطه پدر با فرزندان از نوع ناقص (اختیاری) است، پس ممکن است رکوردهای پدر، فرزند نداشته باشد. برای مثال رکورد GID2,GN2 هیچ فرزندی ندارد.

توجه: دقت کنید چون رابطه فرزندان با پدر از نوع Disjoint است، پس هر رکورد از پدر فقط و فقط می‌تواند یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 فقط و فقط می‌تواند یک فرزند مثل A1,B1 از موجودیت E یا از موجودیت F داشته باشد.

مدل دو جدولی غیربهینه به دلیل افزایش مقدار NULL: مدل تحلیل ناقص (اختیاری) و Disjoint امکان پیاده‌سازی دو جدولی دارد اما بهینه نیست.

سربار افزایش مقدار NULL دارد: چون ارتباط زیرنوع با زیرنوع ناقص (اختیاری) است، پس ممکن است زیرنوع، فرزند نداشته باشد و به تبع مقادیر NULL افزایشی می‌شود. مانند سطر 2 جدول E.

سربار افزایش افزونگی ندارد: چون ارتباط زیرنوع با زیرنوع Disjoint است، پس یک رکورد خاص زیرنوع G در دو جدول E و F تکرار نمی‌شود.

سربار الحاق طبیعی ندارد: چون در مدل دو جدولی الحاق میان جدول زیرنوع و زیرنوع وجود ندارد.

GID	GName	A	B
GID1	**GN1**	A1	B1
GID2	**GN2**	NULL	NULL

جدول E

GID	GName	X	Y
GID3	**GN3**	X3	Y3

جدول F

مدل سه جدولی بهینه با تحمل سربار الحاق طبیعی: مدل تحلیل ناقص (اختیاری) و Disjoint امکان پیاده‌سازی سه جدولی دارد و بهینه هم هست.

سربار افزایش مقدار NULL ندارد: چون در مدل سه جدولی جداول زیرنوع و زیرنوع ازهم جدا هستن.

سربار افزایش افزونگی ندارد: چون در مدل سه جدولی جداول زیرنوع و زیرنوع ازهم جدا هستن.

سربار الحاق طبیعی دارد: چون در مدل سه جدولی الحاق میان جدول زبرنوع و زیرنوع وجود دارد.

GID	GName
GID1	GN1
GID2	GN2
GID3	GN3

جدول G

GID	A	B
GID1	A1	B1

جدول E

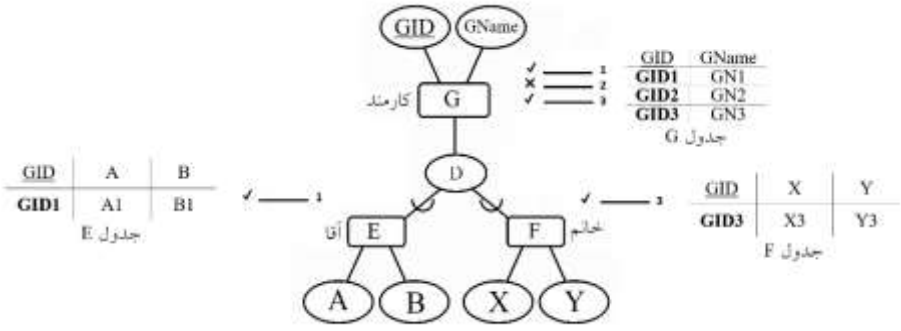
GID	X	Y
GID3	X3	Y3

جدول F

توجه: دقت کنید که مقدار GID2 در جدول E یا F درج نشده است چون فرض شده است مقادیر صفات خاص آن برابر مقدار NULL است. بنابراین مدل سه جدولی سربار افزایش مقادیر NULL را ندارد.

توجه: دقت کنید که در رابطه‌های نشان دهنده هر زیرنوع، کلید خارجی جدول زیرنوع، کلید کاندید آن هم هست.

مدل شماتیک مدل تحلیل ناقص (اختیاری) و Disjoint به صورت زیر است:



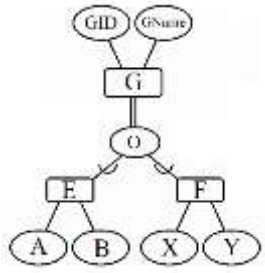
توجه: دقت کنید چون رابطه پدر با فرزندان از نوع ناقص (اختیاری) است، پس ممکن است رکوردهای پدر، فرزند نداشته باشد. برای مثال رکورد GID2,GN2 هیچ فرزندی ندارد.

توجه: دقت کنید چون رابطه فرزندان با پدر از نوع Disjoint است، پس هر رکورد از پدر فقط و فقط می‌تواند یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 فقط و فقط می‌تواند یک فرزند مثل A1,B1 از موجودیت E یا از موجودیت F داشته باشد.

توجه: اگر e مجموعه نمونه‌های E ، f مجموعه نمونه‌های F و g مجموعه نمونه‌های G باشد، آنگاه $card(e)+card(f)\le card(g)$ است. یعنی $1+1\le 3$

ج) مدل تحلیل کامل (اجباری) و Overlap

مدل تحلیل (نمودار ISA)



مدل طراحی (مدل رابطه‌ای)

مدل تک جدولی بهینه در شرایط خاص: دقت کنید که در شرایط خاص یعنی زمانی‌که (1) تعداد صفات زبرنوع زیاد، (2) تعداد زیرنوع‌ها کم و (3) تعداد صفات زیرنوع‌ها کم باشد مدل تحلیل کامل (اجباری) و Overlap امکان پیاده‌سازی تک جدولی دارد و بهینه هم هست.

سربار افزایش مقدار NULL ندارد: چون ارتباط زبرنوع با زیرنوع کامل (اجباری) است، پس رکوردهای زبرنوع حداقل یک فرزند دارد و به تبع مقادیر NULL افزایشی نمی‌شود. همچنین تعداد صفات زیرنوع‌ها هم جهت NULL سازی کم است.

سربار افزایش افزونگی ندارد: چون در مدل تک جدولی یک رکورد خاص زبرنوع فقط در یک جدول درج می‌شود.

سربار الحاق طبیعی ندارد: چون در مدل تک جدولی الحاق میان جدول زبرنوع و زیرنوع وجود ندارد.

صفات مشترک		صفات خاص E		صفات خاص F	
GID	GName	A	B	X	Y
GID1	**GN1**	**A1**	**B1**	**X1**	**Y1**
GID2	**GN2**	**A2**	**B2**	NULL	NULL
GID3	**GN3**	**A3**	**B3**	**X3**	**Y3**

جدول GEF

توجه: دقت کنید چون رابطه پدر با فرزندان از نوع کامل (اجباری) است، پس هر رکورد از پدر حداقل باید یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 حداقل یک فرزند مثل A1,B1 از موجودیت E دارد.

توجه: دقت کنید چون رابطه فرزندان با پدر از نوع Overlap است، پس هر رکورد از پدر ممکن است بیش از یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 می‌تواند دو فرزند A1,B1 از موجودیت E و X1,Y1 از موجودیت F داشته باشد.

مدل دو جدولی غیربهینه‌به دلیل افزایش افزونگی: مدل تحلیل کامل (اجباری) و Ovelap امکان پیاده‌سازی دو جدولی دارد اما بهینه نیست.

سربار افزایش مقدار NULL ندارد: چون ارتباط زیرنوع با زیرنوع کامل (اجباری) است، پس رکوردهای زیرنوع حداقل یک فرزند دارد و به تبع مقادیر NULL افزایشی نمی‌شود.

سربار افزایش افزونگی دارد: چون ارتباط زیرنوع با زیرنوع Overlap است، پس ممکن است یک رکورد خاص زیرنوع G در دو جدول E و F تکرار شود.

سربار الحاق طبیعی‌ندارد: چون در مدل دو جدولی الحاق میان جدول زیرنوع و زیرنوع وجود ندارد.

GID	GName	A	B
GID1	**GN1**	A1	B1
GID2	**GN2**	A2	B2
GID3	**GN3**	A3	B3

جدول E

GID	GName	X	Y
GID1	**GN1**	X1	Y1
GID3	**GN3**	X3	Y3

جدول F

مدل سه جدولی بهینه با تحمل سربار الحاق طبیعی: مدل تحلیل کامل (اجباری) و Overlap امکان پیاده‌سازی سه جدولی دارد و بهینه هم هست.

سربار افزایش مقدار NULL ندارد: چون در مدل سه جدولی جداول زیرنوع و زیرنوع ازهم جدا هستن.

سربار افزایش افزونگی‌ندارد: چون در مدل سه جدولی جداول زیرنوع و زیرنوع ازهم جدا هستن.

سربار الحاق طبیعی دارد: چون در مدل سه جدولی الحاق میان جدول زیرنوع و زیرنوع وجود دارد.

GID	GName
GID1	GN1
GID2	GN2
GID3	GN3

جدول G

GID	A	B
GID1	A1	B1
GID2	A2	B2
GID3	A3	B3

جدول E

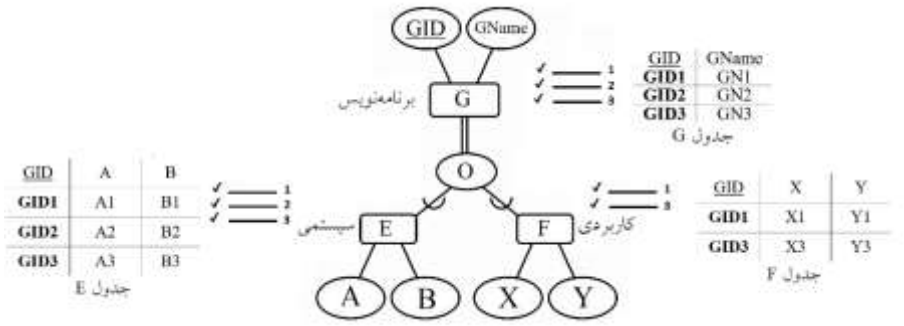
GID	X	Y
GID1	X1	Y1
GID3	X3	Y3

جدول F

توجه: دقت کنید که در رابطه‌های نشـــان دهنده هر زیرنوع، کلید خارجی جدول زیرنوع، کلید کاندید آن هم هست.

توجه: دقت کنید که تکرار GID1 یا GID3 در دو جدول E و F افزونگی طبیعی محسوب نمی‌شود بلکه افزونگی تکنیکی به دلیل تعریف کلید خارجی محسوب می‌شود.

مدل شماتیک مدل تحلیل کامل (اجباری) و Overlap به صورت زیر است:



توجه: دقت کنید چون رابطه پدر با فرزندان از نوع کامل (اجباری) اســـت، پس هر رکورد از پدر حداقل باید یک فرزند داشـته باشـد. برای مثال رکورد GID1,GN1 حداقل یک فرزند مثل A1,B1 از موجودیت E دارد.

توجه: دقت کنید چون رابطه فرزندان با پدر از نوع Overlap اســـت، پس هر رکورد از پدر ممکن اسـت بیش از یک فرزند داشـته باشـد. برای مثال رکورد GID1,GN1 می‌تواند دو فرزند A1,B1 از موجودیت E و X1,Y1 از موجودیت F داشته باشد.

توجه: اگر e مجموعه نمونه‌های E ، f مجموعه نمونه‌های F و g مجموعه نمونه‌های G باشد، آنگاه $card(e)+card(f)\geq card(g)$ است. یعنی $3+2\geq 3$

سوال: بر اساس نمودار فوق، طراحی پایگاه داده رابطه‌ای با چند رابطه (جدول) باید انجام شـود تا هیچ‌مقدار (NULL) پدید نیاید؟

الف) یک

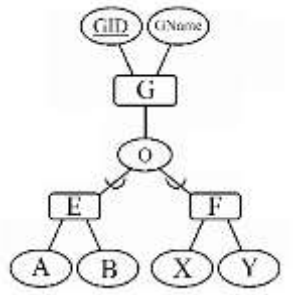
ب) دو

ج) سه

د) دو و سه

د) مدل تحلیل ناقص (اختیاری) و Overlap

مدل تحلیل (نمودار ISA)



مدل تک جدولی غیربهینه به دلیل افزایش مقدار NULL: دقت کنید که در شـــرایط خاص یعنی زمانی‌که (1) تعداد صـفات زبرنوع زیاد، (2) تعداد زیرنوع‌ها کم و (3) تعداد صـفات زیرنوع‌ها کم باشد مدل تحلیل ناقص (اختیاری) و Overlap امکان پیاده‌سازی تک جدولی دارد اما بهینه نیست.

ســـربار افزایش مقدار NULL دارد: چون ارتباط زبرنوع با زیرنوع ناقص (اختیاری) اســـت، پس ممکن است رکوردهای زبرنوع، فرزند نداشته باشد و به تبع مقادیر NULL افزایشی می‌شود. مانند سطر 2. هرچند تعداد صفات زیرنوع‌ها هم جهت NULL سازی کم است.

ســـربار افزایش افزونگی ندارد: چون در مدل تک جدولی یک رکورد خاص زبرنوع فقط در یک جدول درج می‌شود.

ســـربار الحاق طبیعی ندارد: چون در مدل تک جدولی الحاق میان جدول زبرنوع و زیرنوع وجود ندارد.

صفات مشترک		صفات خاص E		صفات خاص F	
GID	GName	A	B	X	Y
GID1	**GN1**	**A1**	**B1**	**X1**	**Y1**
GID2	**Gn2**	NULL	NULL	NULL	NULL
GID3	**Gn3**	**A3**	**B3**	NULL	NULL

جدول GEF

توجه: دقت کنید چون رابطه پدر با فرزندان از نوع ناقص (اختیاری) اســـت، پس ممکن اســـت رکوردهای پدر، فرزند نداشته باشد. برای مثال رکورد GID2,GN2 هیچ فرزندی ندارد.

توجه: دقت کنید چون رابطه فرزندان با پدر از نوع Overlap اســـت، پس هر رکورد از پدر ممکن است بیش از یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 می‌تواند دو فرزند A1,B1 از موجودیت E و X1,Y1 از موجودیت F داشته باشد.

مدل دو جدولی غیربهینه به دلیل افزایش افزونگی و افزایش مقادیر NULL: مدل تحلیل کامل (اجباری) و Ovelap امکان پیاده‌سازی دو جدولی دارد اما بهینه نیست.

سـربار افزایش مقادیر NULL دارد: چون ارتباط زبرنوع با زیرنوع ناقص (اختیاری) اسـت، پس ممکن است زبرنوع، فرزند نداشته باشد و به تبع مقادیر NULL افزایشی می‌شود. مانند سطر 2 در جدول E و F.

سربار افزایش افزونگی دارد: چون ارتباط زیرنوع با زبرنوع Overlap است، پس ممکن است یک رکورد خاص زبرنوع G در دو جدول E و F تکرار شود.

سـربار الحاق طبیعی ندارد: چون در مدل تک جدولی الحاق میان جدول زبرنوع و زیرنوع وجود ندارد.

GID	GName	A	B
GID1	**GN1**	A1	B1
GID2	**GN2**	NULL	NULL
GID3	**GN3**	A3	B3

جدول E

GID	GName	X	Y
GID1	**GN1**	X1	Y1

جدول F

مدل سـه جدولی بهینه با تحمل سـربار الحاق طبیعی: مدل تحلیل ناقص (اختیاری) و Overlap امکان پیاده‌سازی سه جدولی دارد و بهینه هم هست.

سربار افزایش مقدار NULL ندارد: چون در مدل سه جدولی جداول زبرنوع و زیرنوع ازهم جدا هستن.

سـربار افزایش افزونگی‌ندارد: چون در مدل سـه جدولی جداول زبرنوع و زیرنوع ازهم جدا هستن.

سـربار الحاق طبیعی دارد: چون در مدل سـه جدولی الحاق میان جدول زبرنوع و زیرنوع وجود دارد.

GID	GName
GID1	GN1
GID2	GN2
GID3	GN3

جدول G

GID	A	B
GID1	A1	B1
GID3	A3	B3

جدول E

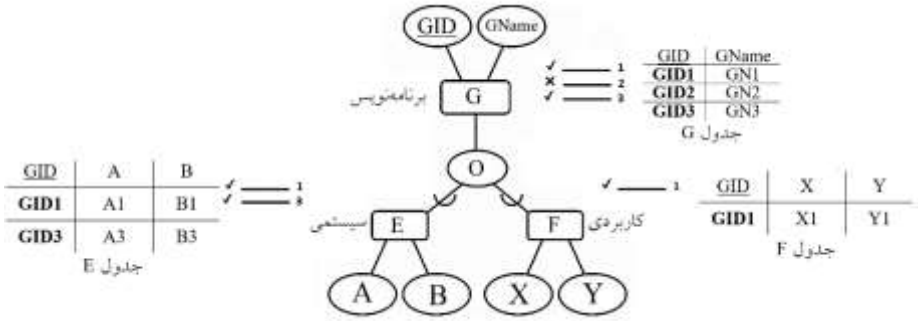
GID	X	Y
GID1	X1	Y1

جدول F

توجه: دقت کنید که مقدار GID2 در جدول E یا F درج نشده است چون فرض شده است مقادیر صفات خاص آن برابر مقدار NULL است. بنابراین مدل سه جدولی سربار افزایش مقادیر NULL را ندارد.

توجه: دقت کنید که در رابطه‌های نشـــان دهنده هر زیرنوع، کلید خارجی جدول زیرنوع، کلید کاندید آن هم هست.

مدل شماتیک مدل تحلیل ناقص (اختیاری) و Overlap به صورت زیر است:



توجه: دقت کنید چون رابطه پدر با فرزندان از نوع ناقص (اختیاری) اســـت، پس ممکن اســـت رکوردهای پدر، فرزند نداشته باشد. برای مثال رکورد GID2,GN2 هیچ فرزندی ندارد.

توجه: دقت کنید چون رابطه فرزندان با پدر از نوع Overlap اســـت، پس هر رکورد از پدر ممکن است بیش از یک فرزند داشته باشد. برای مثال رکورد GID1,GN1 می‌تواند دو فرزند A1,B1 از موجودیت E و X1,Y1 از موجودیت F داشته باشد.

توجه: اگر e مجموعه نمونه‌های E ، f مجموعه نمونه‌های F و g مجموعه نمونه‌های G باشد، آنگاه $card(e)+card(f)\geq card(g)$ است. یعنی $2+1\geq 3$

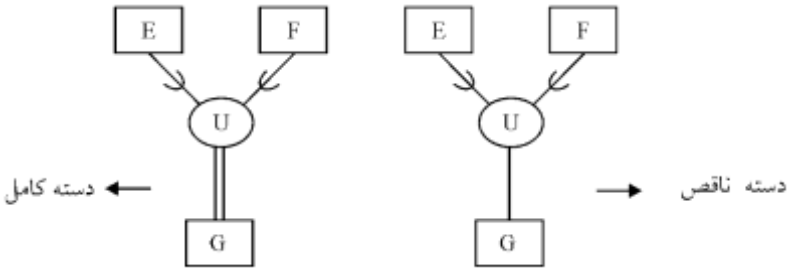
زیرنوع اجتماع (Union subtype: U-Type)

زیرنوع موجودیت G زیرنوع اجتماع نوع موجودیت‌های E ، F و... است هرگاه مجموعه نمونه‌های G ، اجتماع مجموعه نمونه‌هایی (بعضی نمونه‌ها یا همه) از E ، F و... باشد.

بنابراین زیرنوع اجتماع، گونه‌ای زیرنوع اســـت که اولا بیش از یک زیرنوع دارد و ثانیا مجموعه نمونه‌هایش، اجتماع بعضی از (یا همه) نمونه‌های زیرنوع‌هایش است.

توجه: گاه به این گونه زیرنوع ، **دســـته (Category)** یا طبقه گفته می‌شـــود. **زیرنوع اجتماع** را با حرف **U** (حرف اول کلمه Union) نمایش می‌دهیم.

توجه: دسته ممکن است **کامل** یا **ناقص** باشد. دسته وقتی کامل است که در مجموعه نمونه‌های زیرنوع اجتماع، **همه نمونه‌های همه زیرنوع‌ها** وجود داشته باشند، و در غیر اینصورت، دسته را ناقص گوییم.



توجه: در اینجا G یک دسته از E و F است.

توجه: یک نمونه از G یا نمونه‌ای از E و یا نمونه‌ای از F و ... است. مجموعه نمونه‌های G اجتماع نمونه‌هایی از E و نمونه‌هایی از F و ... است.

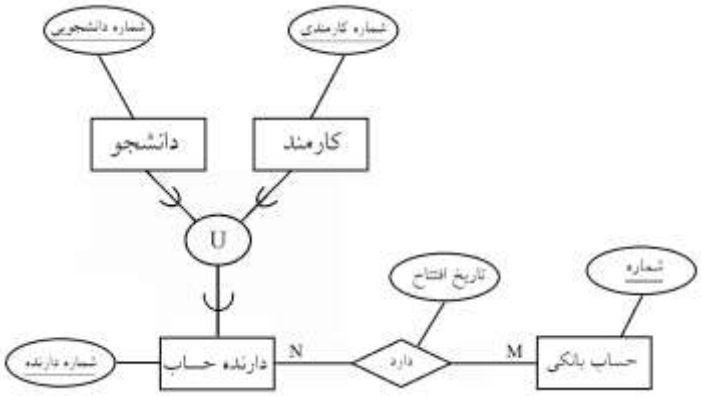
توجه: یک نمونه دسته (زیرنوع اجتماع)، بسته به اینکه از نوع کدام زیرنوع باشد، صفات همان زیرنوع را به ارث می‌برد. بر این اساس به زیرنوع اجتماع **وراثت انتخابی** نیز گفته می‌شود.

توجه: شناسه زیرنوع‌های یک **زیرنوع اجتماع** می‌تواند از دو نوع میدان (دامنه) باشد:

شناسه زیرنوع از میدان متفاوت: پس شناسه **زیرنوع** شناسه‌ای است که طراح باید در نظر بگیرد.

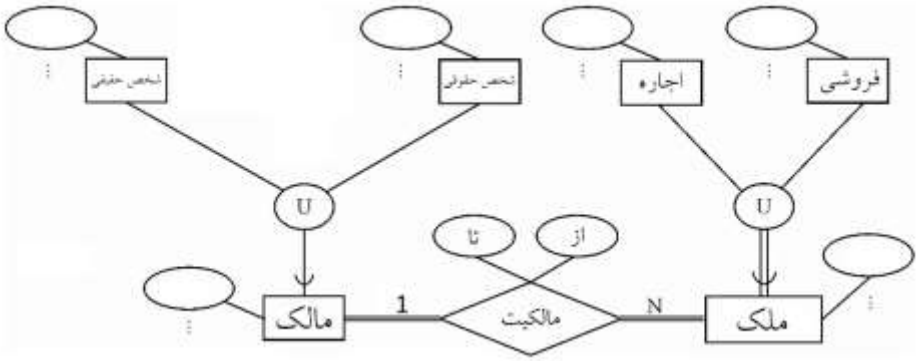
شناسه زیرنوع از میدان یکسان: پس شناسه **زیرنوع** همان شناسه زیرنوع‌ها است.

مثال: نوع موجودیت‌های دانشجو و کارمند را در نظر بگیرید. این دو نوع موجودیت می‌توانند حساب بانکی داشته باشند. در این مثال نوع موجودیت دارنده حساب می‌تواند دانشجو یا کارمند باشد (در مجموعه نمونه‌های آن، هم نمونه‌هایی از دانشجو وجود دارد و هم نمونه‌هایی از کارمند.) همانطور که گفتیم شناسه زیرنوع‌های یک زیرنوع اجتماع ممکن است از یک میدان باشند یا از میدان‌های متفاوت. در این مثال اگر به جای صفت شماره کارمندی و صفت شماره دانشجویی، یک صفت دیگر برای‌مثال کد ملی را شناسه بگیریم، شناسه دو زیرنوع دانشجو و کارمند، از یک میدان می‌شود. وقتی که شناسه زیرنوع‌ها از یک میدان نباشد، طراح باید برای زیرنوع «**دسته**»، یک شناسه در نظر بگیرد، مثل صفت شماره دارنده حساب در شکل زیر. اگر شناسه زیرنوع‌ها یکسان باشد، همان شناسه برای زیرنوع «**دسته**» منظور می‌شود.



توجه: اگر N تعداد زیرنوع‌های زیرنوع «دسته» باشد، تعداد جداول لازم N+1 است.

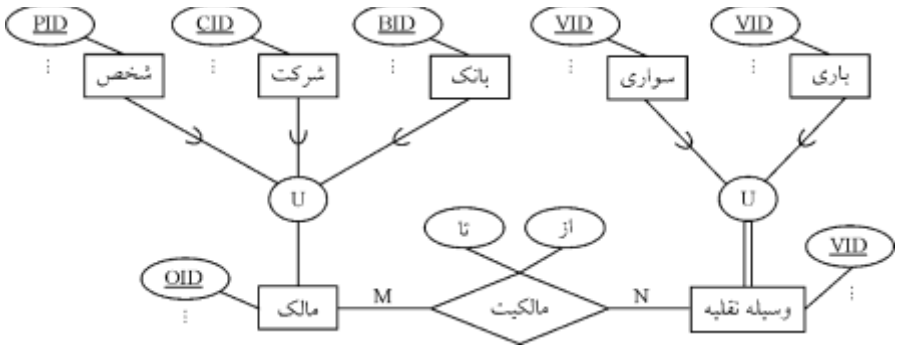
مثال: نمودار EER که دسته‌های املاک را نشان می‌دهد:



در زیرنوع اجتماع ملک، هر ملک یا فروشی است یا اجاره‌ای. پس نمونه‌های زیرنوع اجتماع جدول ملک، اجتماع نمونه املاک فروشی و نمونه املاک اجاره‌ای است. رکورد اطلاعاتی همه نمونه املاک فروشی یا اجاره‌ای الزاما به عنوان ملک در جدول ملک ثبت می‌شوند، چون یک ملک فروشی یا اجاره‌ای قطعا ملک است. پس زیرنوع اجتماع ملک یک **دسته کامل** است.

در زیرنوع اجتماع مالک، هر مالک یا شخص حقوقی است یا شخص حقیقی. پس نمونه‌های زیرنوع اجتماع جدول مالک، اجتماع نمونه اشخاص حقوقی و نمونه اشخاص حقیقی است. رکورد اطلاعاتی همه نمونه اشخاص حقوقی یا اشخاص حقیقی الزاما به عنوان مالک در جدول مالک ثبت نمی‌شوند، چون ممکن است برخی نمونه‌های شخص حقوقی یا شخص حقیقی مالک نباشند. پس زیرنوع اجتماع مالک یک **دسته ناقص** است.

مثال- مدل رابطه‌ای متناظر با نمودار **EER** زیر به چه صورتی است؟



به طور کلی در مدل رابطه‌ای، هر موجودیت شناسایی شده در نمودار ER (مدل تحلیل) هنگام نگاشت به مدل رابطه‌ای (مدل طراحی) به یک جدول تبدیل می‌شود. همچنین صفت‌های موجودیت پس از نگاشت آن در مدل رابطه‌ای به صورت ستون‌های جدول بیان می‌شوند. همچنین ارتباط بین جداول از طریق کلید خارجی برقرار می‌گردد.

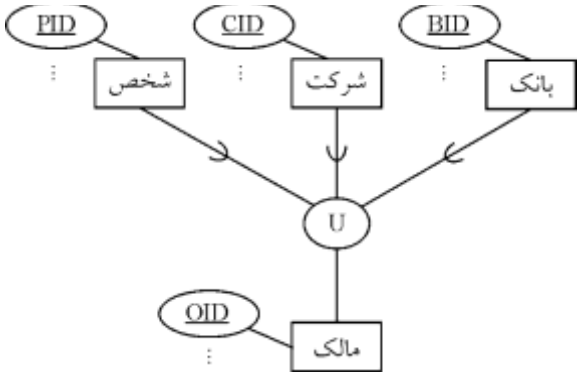
در زیرنوع اجتماع وسیله نقلیه، هر وسیله نقلیه یا باری است یا سواری. پس نمونه‌های زیرنوع اجتماع جدول وسیله نقلیه، اجتماع نمونه وسایل نقلیه باری و سواری است. رکورد اطلاعاتی همه نمونه وسایل نقلیه باری یا سواری الزاما به عنوان وسیله نقلیه در جدول وسیله نقلیه ثبت می‌شوند، چون یک وسیله نقلیه باری یا سواری قطعا وسیله نقلیه است. پس زیرنوع اجتماع وسیله نقلیه یک **دسته کامل** است.

در زیرنوع اجتماع مالک، هر مالک یا بانک است یا شرکت یا شخص. پس نمونه‌های زیرنوع اجتماع جدول مالک، اجتماع نمونه‌های بانک و شرکت و شخص است. رکورد اطلاعاتی همه نمونه‌های بانک یا شرکت یا شخص الزاما به عنوان مالک در جدول مالک ثبت نمی‌شوند، چون ممکن است برخی نمونه‌های بانک یا شرکت یا شخص مالک نباشند. پس زیرنوع اجتماع مالک یک **دسته ناقص** است.

نگاشت رابطه زیرنوع‌ها از میدان متفاوت و زیرنوع اجتماع U-Type به مدل رابطه‌ای

هر موجودیت به یک جدول تبدیل می‌گردد، در شکل زیر شناسه زیرنوع‌ها از **میدان متفاوت** (PID,CID,BID) است، پس شناسه OID توسط طراح بانک برای زیرنوع در نظر گرفته می‌شود. همچنین کلید کاندید موجودیت زیرنوع (همان شناسه انتخاب شده توسط طراح بانک) به عنوان کلیدخارجی در موجودیت زیرنوع‌ها تعریف می‌شود.

مدل تحلیل:



مدل طراحی:

PID	…	OID
PID1		OID1
PID2		OID2

جدول Pers

CID	…	OID
CID1		OID3
CID2		NULL

جدول Comp

BID	…	OID
BID1		OID4
BID2		NULL

جدول Bank

OID	…
OID1	
OID2	
OID3	
OID4	

جدول Owner

توجه: ستون OID در جدول‌های Pers، Comp و Bank به عنوان کلید خارجی تعریف می‌گردد که به جدول Owner ارجاع می‌کند.

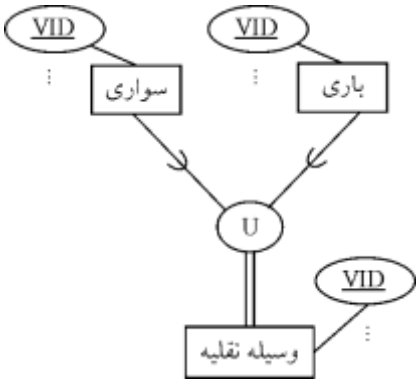
PERS (PID ,…, OID)
COMP (CID ,…, OID)
BANK (BID ,…, OID)
OWNER (OID , …)

توجه: مثال فوق از نوع دسته ناقص است و این موضوع مستقل از نوع میدان است.

نگاشت رابطه زیرنوع‌ها از میدان یکسان و زیرنوع اجتماع U-Type به مدل رابطه‌ای

هر موجودیت به یک جدول تبدیل می‌گردد، اگر شناسه زیرنوع‌ها از **میدان یکسان** باشد، کلید کاندید موجودیت زیرنوع، همان کلید کاندید موجودیت زیرنوع‌ها است.

مدل تحلیل:



مدل طراحی:

VID	N	…
VID1		
VID2		

جدول Savary

VID	T	…
VID3		
VID4		

جدول Bary

VID	…
VID1	
VID2	
VID3	
VID4	

جدول Vehic

SAVARY (VID ,…)
BARY (VID ,…)
VEHIC (VID , …)

توجه: مثال فوق از نوع دسته کامل است و این موضوع مستقل از نوع میدان است.

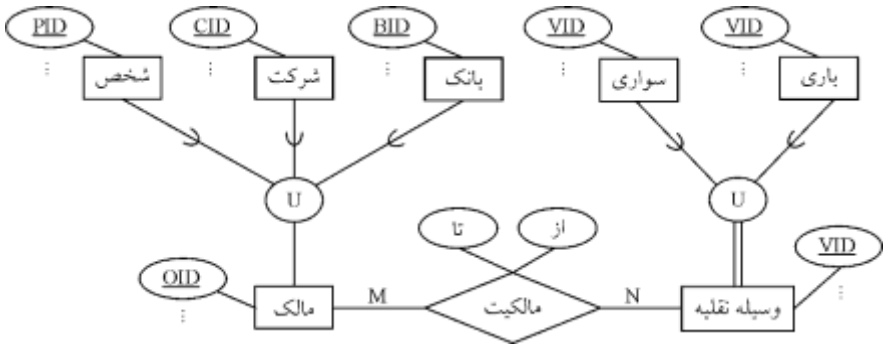
نگاشت رابطه چند به چند بین دو موجودیت به مدل رابطه‌ای

مستقل از اختیاری یا اجباری بودن موجودیت‌ها، هر موجودیت به یک جدول تبدیل می‌گردد و یک جدول پُل (Bridge) نیز به عنوان ارتباط دهنده دو جدول مورد استفاده قرار می‌گیرد. همچنین

کلید کاندید جدول پُل از ترکیب کلید کاندید دو جدول دیگر ایجاد می‌گردد. همچنین صفات متصل به **رابطه**، درون جدول پُل **مستتر** می‌شود.

روال کلی نگاشت در این حالت به صورت زیر است:

مدل تحلیل:



توجه: در شکل فوق صفت OID کلید موجودیت Owner و صفت VID کلید موجودیت Vehic است.

مدل طراحی:

OID	…

جدول Owner

OID	VID	F	T

جدول Owns

VID	…

جدول Vehic

توجه: ستون OID در جدول Owns به عنوان کلید خارجی تعریف می‌گردد که به جدول Owner ارجاع می‌کند. همچنین ستون VID در جدول Owns به عنوان کلید خارجی تعریف می‌گردد که به جدول Vehic ارجاع می‌کند.

توجه: همچنین صفات متصل (مالکیت) به **رابطه**، درون جدول پُل یعنی جدول Owns **مستتر** می‌شود.

توجه: همچنین کلید کلنید جدول پُل از ترکیب کلید کلنید دو جدول دیگر ایجاد می‌گردد. یعنی کلید کاندید جدول Owns برابر (OID,VID) است.

توجه: کلید کاندید جدول چند چپ یعنی **موجودیت Owner** برابر همان **کلید کاندید سابق** در جدول موجودیت Owner است. یعنی کلید کاندید جدول Owner برابر (OID) است.

توجه: کلید کاندید جدول چند راست یعنی **موجودیت Vehic** برابر همان **کلید کاندید سابق** در جدول موجودیت Vehic است. یعنی کلید کاندید جدول Vehic برابر (VID) است.

توجه: جدول Owner در نگاشت مرحله قبل ایجاد شده است که در اینجا دقیقاً به همان شکل و همان مشخصات، استفاده شده است.

توجه: جدول Vehic در نگاشت مرحله قبل ایجاد شده است که در اینجا دقیقاً به همان شکل و همان مشخصات، استفاده شده است.

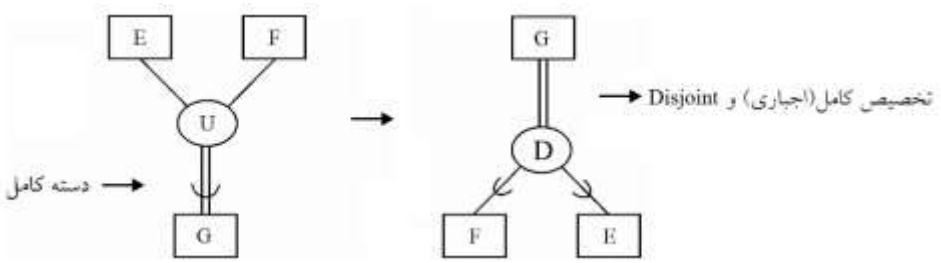
معادل‌سازی U-Type و Disjoint

زیرنوع (ابرنوع) در U-Type معادل زیرنوع در Disjoint است، البته اگر شرایط معادل بودن برقرار باشد.

توجه: در U-Type زیرنوع (ابرنوع) می‌تواند اجباری (کامل) یا اختیاری (ناقص) باشد. اما در Disjoint زیرنوع همواره بدون قید و شرط اجباری است. البته زیرنوع (ابرنوع) در Disjoint می‌تولند اجباری (کامل) یا اختیاری (ناقص) باشد. دقت کنید که زیرنوع در U-Type به خودی خود مفهوم اجباری و اختیاری را ندارد، بلکه اجباری و اختیاری بودن زیرنوع (ابرنوع) در زیرنوع نمایان می‌شود. که به آن «**دسته کامل**» یا «**دسته ناقص**» نیز گفته می‌شود.

نتیجه مهم: در U-Type اگر زیرنوع (ابرنوع) اجباری باشد که به تبع آن زیرنوع هم دسته کامل است، آنگاه در این شرایط U-Type معادل با زیرنوع Disjoint و زیرنوع (ابرنوع) اجباری (کامل) است.

توجه: هرگاه دسته **کامل** باشد، می‌توان به جای مدل «**دسته**»، از مدل تخصیص کامل(اجباری) و Disjoint استفاده کرد.

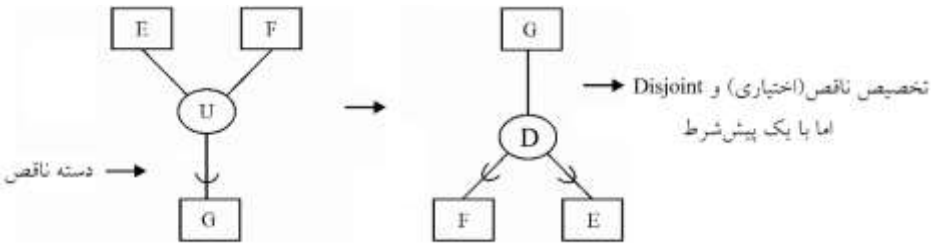


توجه: اینکه کدام مدل‌سازی انتخاب شود، بسته به نظر طراح بانک است.

توجه: هرگاه دسته **ناقص** باشد، می‌توان به جای مدل «**دسته**» ، از مدل تخصیص ناقص(اختیاری) و Disjoint اما با یک پیش‌شرط استفاده کرد.

پیش‌شرط: $C_G = C_E + C_F$

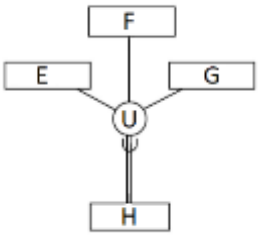
فرض کنید: C_X : کاردینالیتی مجموعه نمونه‌های نوع موجودیت X است.



توجه: اینکه کدام مدل‌سازی انتخاب شود، بسته به نظر طراح بانک است.

توجه: در U-Type، دسته کامل یا ناقص تحت هیچ شرایطی امکان مدل‌سازی توسط Overlap را ندارد. چون ماهیت Overlap همپوشانی است که با ذات زیرنوع اجتماع (U-Type) کاملا متفاوت است. همانطور که گفتیم یک نمونه دسته (زیرنوع اجتماع)، بسته به اینکه از نوع کدام زبرنوع باشد، صفات همان زبرنوع را به ارث می‌برد. و این به معنی مستقل بودن رکوردهای زبرنوع‌های مستقل ازهم در مفهوم U-Type است و تداعی‌کننده همان مفهوم Disjoint است.

سوال: با توجه به نمودار، نوع موجودیت H :



الف) همه صفات نوع موجودیت‌های E ، F و G را به ارث می‌برد.

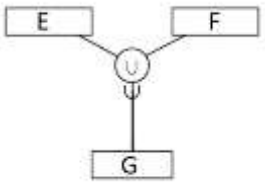
ب) همه صفات حداقل یکی از سه نوع موجودیت E ، F و G را به ارث می‌برد.

ج) حداقل شناسه سه نوع موجودیت E ، F و G را به ارث می‌برد.

د) صفات یکی از سه نوع موجودیت E یا F یا G را به ارث می‌برد.

پاسخ گزینه «د» صحیح است.

سوال: با توجه به نمودار زیر :



اگر e مجموعه نمونه‌های E ، f مجموعه نمونه‌های F و g مجموعه نمونه‌های G باشد:

الف) $g = e \cup f$

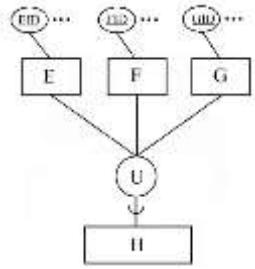
ب) $g \subset e \cup f$

ج) $g \subseteq e \cup f$

د) $g \supseteq e \cup f$

پاسخ گزینه «ج» صحیح است.

سوال: با توجه به نمودار، شناسه نوع موجودیت H چیست؟



فرض: $EID \neq FID \neq GID$

الف) $HID \in \{EID, FID, GID\}$

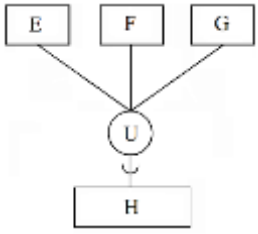
ب) $HID \in \{(\underline{EID, GID}), (\underline{EID, FID}), (\underline{GID, FID})\}$

ج) $HID \in \{\underline{EID, FID, GID}\}$

د) $HID \neq I \mid I \in \{EID, FID, GID\}$

پاسخ گزینه «د» صحیح است.

سوال: با توجه به نمودار، نوع موجودیت H چند شناسه دارد؟



الف) یک شناسه

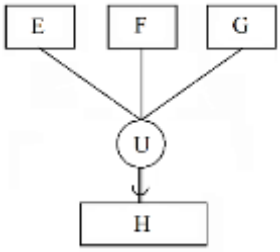
ب) یک شناسه حاصل ترکیب شناسه‌های E و F و G

ج) سه شناسه، هر یک حاصل ترکیب شناسه‌های هربار دو نوع موجودیت

د) سه شناسه

پاسخ گزینه «الف» صحیح است.

سوال: با توجه به نمودار، نوع موجودیت H چند شناسه دارد؟



الف) سه شناسه

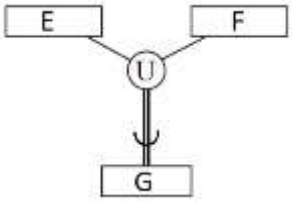
ب) حداکثر سه شناسه

ج) یک شناسه

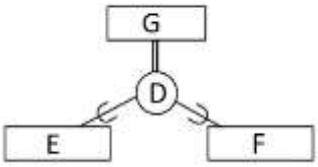
د) حداقل یک شناسه

پاسخ گزینه «ج» صحیح است.

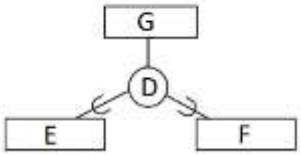
سوال: فرض کنید یک محیط را به صورت زیر مدل‌سازی کرده باشیم، به چه طرز دیگری می‌توان این محیط را مدل کرد؟



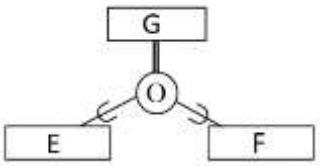
الف)



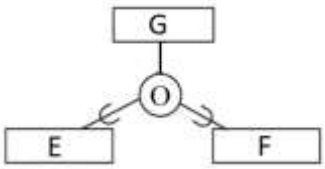
ب)



ج)

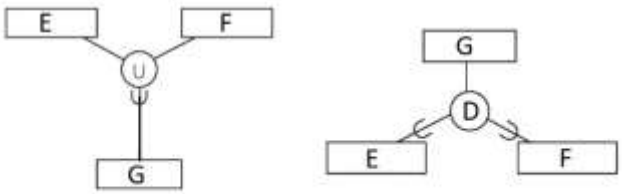


د)



پاسخ گزینه «الف» صحیح است.

سوال: مدل‌سازی I را با چه شرطی می‌توان به صورت II هم انجام داد؟



I II

فرض کنید: C_X : کاردینالیتی مجموعه نمونه‌های نوع موجودیت X است.

الف) $C_G \leq C_E + C_F$

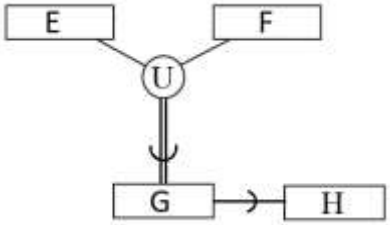
ب) $C_G > C_E + C_F$

ج) $C_G = C_E + C_F$

د) $C_G \geq C_E + C_F$

پاسخ گزینه «ج» صحیح است.

سوال: در نمودار زیر، شناسه نوع موجودیت H چه می‌تواند باشد؟



I : شناسه G ، یکسان با شناسه E و شناسه F

II : شناسه G ، یکسان با شناسه E یا شناسه F و نه هردو

III : شناسه G ، متفاوت با شناسه E و شناسه F

الف) I یا III

ب) II

ج) III

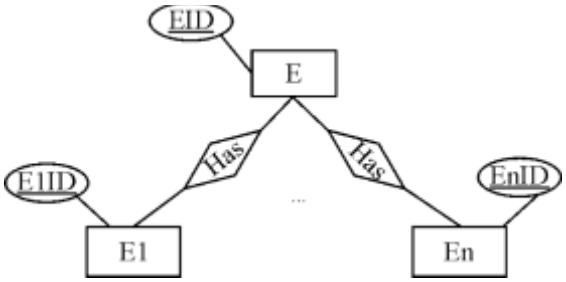
د) I یا II

پاسخ گزینه «الف» صحیح است.

نگاشت رابطه IS-A-PART-OF به مدل رابطه‌ای

هر موجودیت به یک جدول تبدیل می‌گردد و کلید کاندید موجودیت زبرنوع در موجودیت‌های زیرنوع به عنوان کلید خارجی تعریف می‌گردد.

مدل تحلیل:



مدل طراحی:

EID	…

جدول E

E1ID	EID	…

جدول E1

EnID	EID	…

جدول En

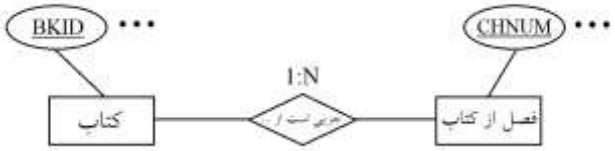
توجه: موجودیت کل (E) به طور مستقل برای خود کلید کلنید دارد و همچنین موجودیت‌های جزء (E1 و En) به طور مستقل برای خود کلید کاندید دارند.

توجه: کلید کاندید جدول E1 ترکیبی و به صورت E1ID,EID است.

توجه: ستون EID در جدول‌های E1 و Enk به عنوان کلید خارجی تعریف می‌گردد که به جدول E ارجاع می‌کند.

توجه: ستون EID در جدول E1 علاوه بر اینکه کلید خارجی است جزء کلید کاندید جدول E1 نیز است.

مثال: هر فصل کتاب جزیی از کتاب است.



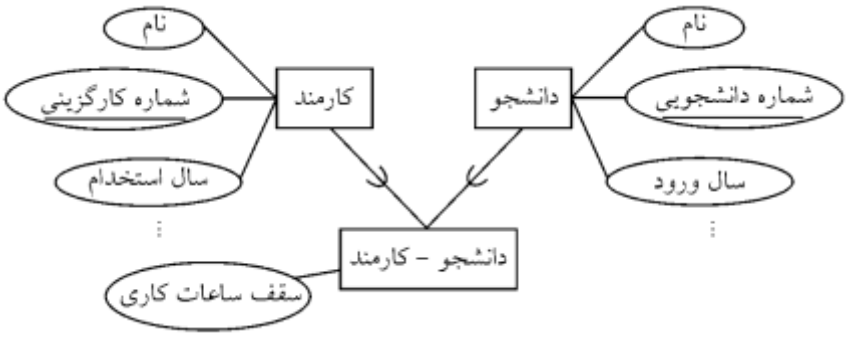
BOOK (BKID , BKTITLE , ...)
C.K.

CHAP (BKID, CHNUM , CHAPTITLE , NO – OF – PAGE , ...)
C.K.

نگاشت رابطه ارث‌بری چندگانه به مدل رابطه‌ای

هر موجودیت به یک جدول تبدیل می‌گردد و کلید کلندید موجودیت‌های زیرنوع در موجودیت زیرنوع به عنوان کلید خارجی تعریف می‌گردد. در جدول زیرنوع کلید خارجی همزمان کلید کاندید نیز می‌باشد.

مدل تحلیل:



مدل طراحی:

EID	Ename	…

STID	STname	..

جدول Empl *جدول Stud*

EID	STID	MAXW

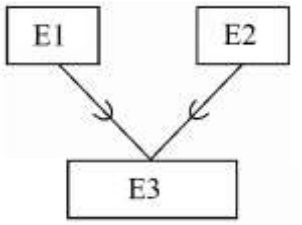
جدول Stem

EMPL (EID ,…)
STUD (STID ,…)
STEM (EID , … , STID , … , MAXW)

توجه: زیرنوع ارث‌بری چندگانه با اتصال به زبرنوع‌های خود توسط کلید خارجی، تمام صفات زبرنوع‌های خود را به ارث می‌برد. به عبارت دیگر مجموعه صفات جدول زیرنوع، **اجتماع** مجموعه صفات جداول زبرنوع‌ها است. بدین معنی که در اجتماع مجموعه صفات جداول زبرنوع‌ها، صفات تکراری یکبار در جدول زیرنوع قرار می‌گیرند و صفات تکراری حذف می‌شود.

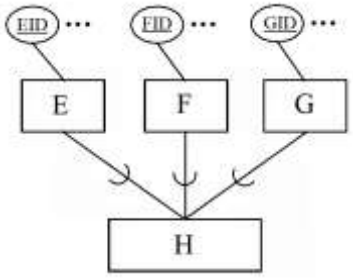
توجه: اگر زیرنوع تعداد N زبرنوع داشته باشد، جدول زیرنوع **حداقل** N کلید کاندید **مجزا** دارد. حداقل چون ممکن است هر زبرنوع خودش حداقل دو کلید کاندید مجزا داشته باشد.

سوال: در نمودار زیر، نوع موجودیت E3:



الف) دو شناسه دارد.
ب)حداقل یک شناسه مرکب دارد.
ج)حداقل دو شناسه دارد.
د) حداقل دو شناسه مرکب دارد.
پاسخ گزینه «ج» صحیح است.

سوال: با توجه به نمودار، شناسه نوع موجودیت H چیست؟



الف) $HID \in \{EID, FID, GID\}$

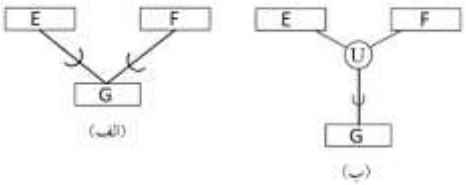
ب) $HID \in \{(EID, GID), (EID, FID), (GID, FID)\}$

ج) $HID = \{EID, FID, GID\}$

د) هیچکدام

پاسخ گزینه «الف» صحیح است.

سوال: با توجه به نمودار (الف) و (ب) و گزاره‌های I و II ، مورد درست کدام است؟



گزاره I : G یا صفات E را به ارث می‌برد یا صفات F را.

گزاره II : G هم صفات E را به ارث می‌برد و هم صفات F را.

الف) گزاره II برای (ب) و گزاره I برای (الف)

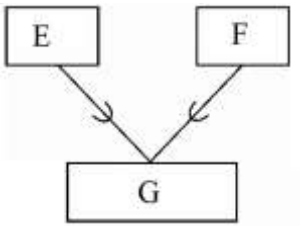
ب) گزاره II برای (الف) و گزاره I برای (ب)

ج) گزاره I برای (الف) و گزاره I برای (ب)

د) گزاره II برای (الف) و گزاره II برای (ب)

پاسخ گزینه «ب» صحیح است.

سوال: با توجه به نمودار زیر، اگر A_X مجموعه صفات نوع موجودیت X باشد، در اینصورت:



الف) $A_G = A_E \cup A_F$

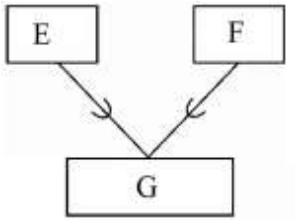
ب) $A_G \supseteq A_E \cup A_F$

ج) $A_G = A_E \cap A_F$

د) $A_G = (A_E - A_F) \cup (A_F - A_E)$

پاسخ گزینه «ب» صحیح است.

سوال: با توجه به نمودار زیر، اگر $Card[A_X]$ کاردینالیتی مجموع صفات نوع موجودیت X باشد، در این صورت :



الف) $Card[A_G] = Card[A_E] + Card[A_F]$

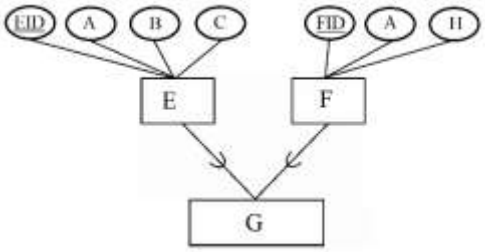
ب) $Card[A_G] \leq Card[A_E] + Card[A_F]$

ج) $Card[A_G] < Card[A_E \cup A_F]$

د) $Card[A_G] \geq Card[A_E \cup A_F]$

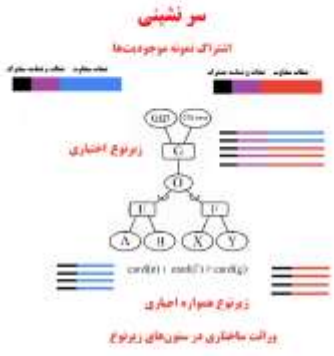
پاسخ گزینه «د» صحیح است. چون موجودیت G ممکن است صفات خاصه خودش را نیز اضافه کند.

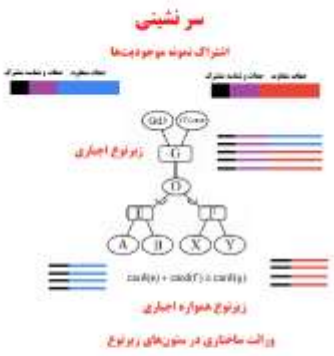
سوال: با توجه به نمودار زیر، اگر در طراحی پایگاه داده رابطه‌ای برای نوع موجودیت G ، یک رابطه طراحی کنیم، درجه این رابطه چند است؟ (منظور از درجه ارتباط رابطه همان تعداد صفات G است.)

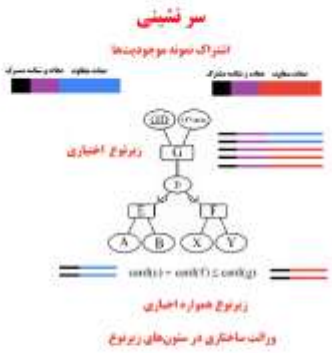


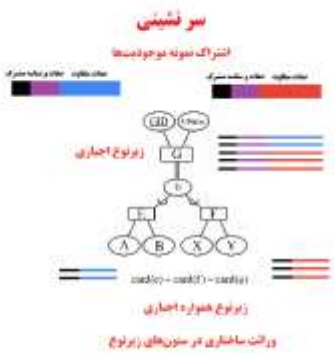
الف) پنج
ب) حداقل شش
ج) هفت
د) شش

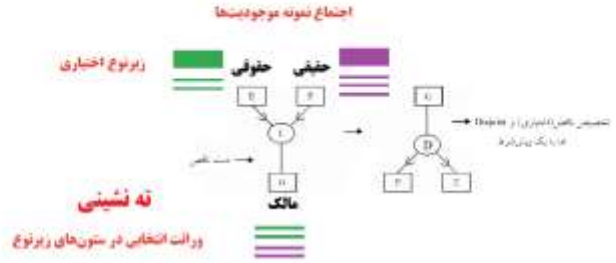
پاسخ گزینه «د» صحیح است.











تست‌های فصل ششم: SQL دستورات DML

۱۱۳- فرض کنید دو جدول داریم:

- جدول Employees با ستون‌های Salary , Age , DepartmentID , EmployeeID , Name
- جدول Departments با ستون‌های Budget , DepartmentName , DepartmentID

(ستون DepartmentID در جدول Employees به عنوان کلید خارجی است که به ستون DepartmentID در جدول Departments ارتباط دارد.)

کدام‌یک از پرس‌وجوهای زیر به درستی نام هر کارمند را همراه با نام دپارتمان مربوطه نشان می‌دهد؟

(مهندسی کامپیوتر-دولتی۱۴۰۳)

۱)
```
SELECT Employees.Name , Employees.DepatmentID
FROM Employees
```

۲)
```
SELECT Employees.Name , Departments.DepartmentName
FROM Employees  RIGHT OUTER JOIN Departments
WHERE Employees.DepartmentID=Departments.DepartmentID
```

۳)
```
SELECT Employees.Name , Departments.DepartmentName
FROM Employees  INNER JOIN Departments
ON  Employees.DepartmentID=Departments.DepartmentID
```

۴)
```
SELECT Employees.Name , Departments.DepartmentName
FROM Employees  LEFT OUTER JOIN Departments
ON Employees.EmployeeID=Departments.DepartmentID
```

استاد ارسطو خلیلی فر
مولف کتاب پایگاه داده انتشارات بابان

پاسخ تست‌های ششم: SQL دستورات DML

۱۱۳- گزینه (۳) صحیح است.

در صورت سوال گفته شده است ستون DepartmentID در جدول Employees به عنوان کلید خارجی است که به ستون DepartmentID در جدول Departments ارتباط دارد. مطابق تعریف کلید خارجی یعنی مقادیر ستون DepartmentID در جدول Employees همواره باید زیرمجموعه مقادیر ستون DepartmentID در جدول Departments باشد. البته کلید خارجی اگر قانون بالادستی نداشته باشد می‌تواند مقدار NULL داشته‌باشد.

جداول زیر را در نظر بگیرید:

EmployeeID	DepartmentID	Name	…
Emp1	Dep1	EmpName1	…
Emp2	Dep2	EmpName2	…
Emp3	Dep2	EmpName3	…
Emp4	NULL	EmpName4	…

Employees

DepartmentID	DepartmentName	…
Dep1	DepName1	
Dep2	DepName2	
Dep3	DepName3	

Departments

مطابق پرس‌وجوی مطرح شده در گزینه اول، داریم:

```
SELECT Employees.Name , Employees.DepatmentID
FROM Employees
```

خروجی به صورت زیر است:

Employees.Name	Employees.DepartmentID
EmpName1	Dep1
EmpName2	Dep2
EmpName3	Dep2
EmpName4	NULL

فقط **نام کارمند** (Employees.Name) و **شناسه دپارتمان** (Employees.DepartmentID) را نمایش می‌دهد و اصلاً از جدول Departments و به تبع نام دپارتمان (DepartmentName) استفاده نمی‌کند.

مطابق پرس‌وجوی مطرح شده در گزینه دوم، داریم:

```
SELECT Employees.Name , Departments.DepartmentName
FROM Employees  RIGHT OUTER JOIN  Departments
WHERE Employees.DepartmentID = Departments.DepartmentID
```

خروجی به صورت زیر است:

Employees.Name	Departments.DepartmentName
EmpName1	DepName1
EmpName2	DepName2
EmpName3	DepName2
NULL	**DepName3**

از یک RIGHT OUTER JOIN استفاده می‌کند و شرط پیوند به صورت زیر است:
Employees.DepartmentID = Departments.DepartmentID
تمامی دپارتمان‌ها نمایش داده می‌شوند، حتی اگر کارمندی نداشته باشند. البته گزینه دوم یک خطای نحوی هم دارد، زیرا به جای دستور WHERE باید عملگر ON استفاده شود.

مطابق پرس‌وجوی مطرح شده در گزینه سوم، داریم:

```
SELECT Employees.Name , Departments.DepartmentName
FROM Employees INNER JOIN Departments
ON Employees.DepartmentID=Departments.DepartmentID
```

خروجی به صورت زیر است:

Employees.Name	Departments.DepartmentName
EmpName1	DepName1
EmpName2	DepName2
EmpName3	DepName2

از یک INNER JOIN استفاده می‌کند و شرط پیوند به صورت زیر است:
Employees.DepartmentID = Departments.DepartmentID
فقط کارمندانی نمایش داده می‌شود که به دپارتمانی مرتبط هستند.

مطابق پرس‌وجوی مطرح شده در گزینه چهارم، داریم:

```
SELECT Employees.Name , Departments.DepartmentName
FROM Employees LEFT OUTER JOIN Departments
ON Employees.EmployeeID=Departments.DepartmentID
```

خروجی به صورت زیر است:

Employees.Name	Departments.DepartmentName
EmpName1	DepName1
EmpName2	DepName2
EmpName3	DepName2
EmpName4	**NULL**

از یک LEFT OUTER JOIN استفاده می‌کند و شرط پیوند به صورت زیر است که اشتباه است:
Employees.EmployeeID = Departments.DepartmentID
که اگر به صورت زیر اصلاح شود:
Employees.DepartmentID = Departments.DepartmentID
تمامی کارمندان نمایش داده می‌شوند، حتی اگر دپارتمان مربوطه‌ای نداشته باشند.

تست‌های فصل هفتم: SQL دستورات DDL

۱۱۴- فرض کنید دو جدول داریم:

- جدول Employees با ستون‌های Salary , Age , DepartmentID , EmployeeID , Name
- جدول Departments با ستون‌های Budget , DepartmentName , DepartmentID

(ستون DepartmentID در جدول Employees به عنوان کلید خارجی است که به ستون DepartmentID در جدول Departments ارتباط دارد.)

فرض کنید می‌خواهید اطمینان حاصل کنید که مجموع حقوق کارمندان در هر بخش از بودجه آن بخش تجاوز نمی‌کند. کدام‌یک از پرس‌وجوهای زیر به‌درستی یک Assertion را برای این قاعده تعریف می‌کند؟

(مهندسی کامپیوتر-دولتی۱۴۰۳)

۱)
```
ALTER TABLE Employees
ADD CONSTRAINT SalaryBudgetCheck
CHECK (SUM (Salary) <= (SELECT Budget
                        FROM Departments
                        WHERE DepartmentID = Employees.DepartmentID));
```

۲)
```
CREATE ASSERTION SalaryBudgetCheck
CHECK (NOT EXISTS (SELECT E.DepartmentID
                   FROM Employees E
                   WHERE SUM(E.Salary) > (SELECT D.Budget
                                          FROM Departments D
                                          WHERE D.DepartmentID = E.DepartmentID)));
```

۳)
```
CREATE ASSERTION SalaryBudgetCheck AS CHECK
(SELECT SUM(E.Salary)
 FROM Employees E
 GROUP BY E.DepartmentID <= SELECT D.Budget
                            FROM Departments D
                            WHERE D.DepartmentID = E.DepartmentID);
```

۴)
```
CREATE TRIGGER SalaryBudgetCheck
BEFORE INSERT OR UPDATE
ON Employees
FOR EACH ROW
EXECUTE PROCEDURE CheckSalaryBudget() ;
```

پاسخ تست‌های فصل هفتم: SQL

۱۱۴ – گزینه () صحیح است.

پاسخ این تست در کلید اولیه گزینه ۲ اعلام شد. اما در کلید نهایی به درستی سوال حذف شد، چون هیچکدام از گزینه‌ها پاسخ سوال نبودن.

جداول زیر را در نظر بگیرید:

EmployeeID	DepartmentID	Name	Salary	…
Emp1	Dep1	EmpN1	10	…
Emp2	Dep2	EmpN2	20	…
Emp3	Dep2	EmpN3	30	…
Emp4	NULL	EmpN4	NULL	

Employees

DepartmentID	DepartmentName	Budget
Dep1	DepName1	100
Dep2	DepName2	200
Dep3	DepName3	300

Departments

خواسته سوال از دو بخش «**مجموع حقوق کارمندان در هر دپارتمان**» و «**بودجه آن دپارتمان**» تشکیل شده است. نحوه محاسبه «**بودجه آن دپارتمان**» در گزینه‌های اول تا سوم یکسان است.

مطابق پرس‌وجوی مطرح شده در گزینه اول، داریم:

```
ALTER TABLE Employees
ADD CONSTRAINT SalaryBudgetCheck
CHECK (SUM (Salary) <= (SELECT Budget
                        FROM Departments
                        WHERE DepartmentID = Employees.DepartmentID));
```

دستور CHECK به تنهایی و بدون Assertion در SQL برای بررسی محدودیت‌ها در هر سطر (Row) استفاده می‌شود، و نمی‌توان جلوی آن تابع تجمعی (Aggregate Function) مثل SUM(Salary) و زیرپرس‌وجو (Subquery) قرار داد. بنابراین گزینه اول خطای نحوی دارد.

ساختار CHECK

دستور CHECK در SQL، یک محدودیت (Constraint) است که برای بررسی مقادیر ستون‌ها در هر سطر از جدول استفاده می‌شود.

مثال: پیاده‌سازی محدودیت برای اطمینان از اینکه مقدار Salary همیشه مثبت باشد، با استفاده از

CHECK به صورت زیر است:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    Salary DECIMAL(10,2) CHECK (Salary > 0) );
```

افزودن CHECK به جدول موجود

درج محدودیت CHECK بر روی جدول Employees که قبلا ایجاد شده‌است:

```
ALTER TABLE Employees
ADD CONSTRAINT PositiveSalary CHECK (Salary > 0);
```

ساختار Assertion

دستور Assertion در SQL ، یک محدودیت (Constraint) است که می‌تواند شرایطی را بر روی داده‌های چندین جدول اعمال کند.

```
CREATE ASSERTION assertion_name
CHECK (condition);
```

مثال: پیاده‌سازی محدودیت برای اطمینان از اینکه مقدار Salary همیشه مثبت باشد، با استفاده از Assertion به صورت زیر است:

```
CREATE ASSERTION PositiveSalary
CHECK (NOT EXISTS (
    SELECT 1
    FROM Employees
    WHERE Salary < 0 ));
```

مثال: پیاده‌سازی محدودیت برای اطمینان از اینکه مجموع حقوق همه کارمندان در جدول Employees نباید بیشتر از 1,000,000 باشد، با استفاده از Assertion به صورت زیر است:

```
CREATE ASSERTION TotalSalaryCheck
CHECK ((SELECT SUM(Salary)
        FROM Employees) <= 1000000);
```

توجه: در SQL استاندارد (SQL Standard) ، مفهوم ASSERTION تعریف شده است، اما اکثر سیستم‌های مدیریت پایگاه داده (DBMS) رایج مانند MySQL، PostgreSQL و SQL Server از آن پشتیبانی نمی‌کنند. به همین دلیل، برای اعمال چنین محدودیت‌هایی معمولا از روش‌های جایگزین مانند Triggers یا منطق برنامه‌نویسی در لایه‌های بالاتر استفاده می‌شود.

خواسته سوال را توسط Assertion به صورت زیر می‌توان بازنویسی کرد:

شروع با Departments و دسترسی مستقیم به Budget

```
CREATE ASSERTION SalaryBudgetCheck
CHECK (NOT EXISTS
   (SELECT 1
    FROM Departments d
    WHERE (SELECT SUM(e.Salary)
           FROM Employees e
           WHERE e.DepartmentID = d.DepartmentID) > d.Budget));
```

شروع با Employees و دسترسی غیرمستقیم به Budget

```
CREATE ASSERTION SalaryBudgetCheck
CHECK (NOT EXISTS
    (SELECT 1
     FROM Employees e
     WHERE (SELECT SUM(e2.Salary)
            FROM Employees e2
            WHERE e2.DepartmentID = e.DepartmentID)
          >
           (SELECT Budget
            FROM Departments
            WHERE DepartmentID = e.DepartmentID)));
```

مطابق پرس‌وجوی مطرح شده در گزینه دوم، داریم:

```
CREATE ASSERTION SalaryBudgetCheck
CHECK (NOT EXISTS (SELECT E.DepartmentID
                   FROM Employees E
                   WHERE SUM(E.Salary) > (SELECT  D.Budget
                                          FROM Departments D
                                          WHERE D.DepartmentID = E.DepartmentID)));
```

استفاده از تابع تجمعی (Aggregate Function) مانند SUM(E.Salary) به تنهایی در شرط WHERE باعث خطای نحوی (Syntax Error) می‌شود. شرط WHERE برای فیلترکردن سطرها در سطح

رکورد (Row Level) طراحی شده است، اما توابع تجمعی بر روی مجموعه‌ای از رکوردها کار می‌کنند. بنابراین، استفاده از SUM (یا سایر توابع تجمعی) در WHERE منجر به خطا می‌شود. برای استفاده از توابع تجمعی در شرط‌ها، باید از HAVING به جای WHERE استفاده کرد.

شروع با Employees و دسترسی غیرمستقیم به Budget

```
CREATE ASSERTION SalaryBudgetCheck
CHECK (NOT EXISTS
   (SELECT 1
    FROM Employees E
    GROUP BY E.DepartmentID
    HAVING SUM(E.Salary) > (SELECT D.Budget
                            FROM Departments D
                           WHERE D.DepartmentID = E.DepartmentID)));
```

شروع با Employees و دسترسی مستقیم به Budget

```
CREATE ASSERTION SalaryBudgetCheck
CHECK (NOT EXISTS
      (SELECT 1
      FROM Employees E JOIN Departments D
      ON E.DepartmentID = D.DepartmentID
      GROUP BY E.DepartmentID
      HAVING SUM(E.Salary) > D.Budget));
```

توجه: دستور HAVING بر روی گروه‌ها، اعمال می‌گردد.

مطابق پرس‌وجوی مطرح شده در گزینه سوم، داریم:

```
CREATE ASSERTION SalaryBudgetCheck AS CHECK
(SELECT SUM(E.Salary)
 FROM Employees E
 GROUP BY E.DepartmentID <= SELECT D.Budget
                            FROM Departments D
                            WHERE D.DepartmentID = E.DepartmentID)
```

عبارت AS CHECK در SQL معتبر نیست و از نظر نحوی اشتباه است.

نمی‌توان GROUP BY را به شکل شرطی با استفاده از => مستقیماً در کوئری استفاده کرد.

مطابق پرس‌وجوی مطرح شده در گزینه چهارم، داریم:

```
CREATE TRIGGER SalaryBudgetCheck
BEFORE INSERT OR UPDATE
ON Employees
FOR EACH ROW
EXECUTE PROCEDURE CheckSalaryBudget() ;
```

درستی این گزینه به این بستگی دارد که تابع CheckSalaryBudget() به درستی تعریف شده باشد که قطعه کد آن بیان نشده است.

ساختار کلی Trigger

ساختار **Trigger** به شکل زیر تعریف می‌شود:

```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER | INSTEAD OF }
{ INSERT | UPDATE | DELETE }
ON table_name
[ FOR EACH ROW | FOR EACH STATEMENT ]
[ WHEN (condition) ]
BEGIN
    -- SQL Statements
END;
```

استفاده از Triggerبرای بررسی شرط

اگر پایگاه داده از **Assertion** پشتیبانی نکند مثلاً درMySQL ، می‌توان از **Trigger** برای بررسی شرط استفاده کرد. (صفحه بعدی ادامه دارد...)

بررسی شرط قبل از (**BEFORE**) تغییرات جداول در **TRIGGER**

BEFORE : قبل از انجام عملیات (INSERT ، UPDATE یا DELETE) اجرا می‌شود.

حالت DELETE در این تریگر نیامده چون حذف رکورد باعث کاهش مجموع حقوق دپارتمان می‌شود.

```
CREATE TRIGGER SalaryBudgetCheck
BEFORE INSERT OR UPDATE
ON Employees
FOR EACH ROW
BEGIN
    -- متغیر برای ذخیره مجموع حقوق دپارتمان
    DECLARE total_salary DECIMAL(10, 2);

        -- متغیر برای ذخیره بودجه دپارتمان
    DECLARE department_budget DECIMAL(10, 2);

    -- محاسبه مجموع حقوق کارمندان دپارتمان مربوطه با در نظر گرفتن مقدار جدید
    SELECT SUM(Salary) + NEW.Salary
    INTO total_salary
    FROM Employees
    WHERE DepartmentID = NEW.DepartmentID
      AND EmployeeID != NEW.EmployeeID;

    -- دریافت بودجه دپارتمان
    SELECT Budget
    INTO department_budget
    FROM Departments
    WHERE DepartmentID = NEW.DepartmentID;

    -- بررسی تجاوز از بودجه
    IF total_salary > department_budget THEN
        SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Salary exceeds the department budget.';
    END IF;
END;
```

خط AND EmployeeID != NEW.EmployeeID اضافه شده تا هنگام محاسبه مجموع حقوق، رکورد جاری که در حال درج یا به‌روزرسانی است، دوباره در محاسبات لحاظ نشود.

- **در INSERT** : چون رکورد جدید هنوز در جدول نیست، این شرط اثری ندارد.

- **در UPDATE** : این شرط جلوی این را می‌گیرد که حقوق رکورد جاری هم به‌صورت قدیمی و هم جدید در مجموع حساب شود.

بررسی شرط قبل از (**AFTER**) تغییرات جداول در **TRIGGER**

AFTER : بعد از انجام عملیات (INSERT ، UPDATE یا DELETE) اجرا می‌شود.

```
CREATE TRIGGER SalaryBudgetCheck
AFTER INSERT OR UPDATE
ON Employees
FOR EACH ROW
BEGIN
    -- متغیر برای ذخیره مجموع حقوق دپارتمان
    DECLARE total_salary DECIMAL(10, 2);

    -- متغیر برای ذخیره بودجه دپارتمان
    DECLARE department_budget DECIMAL(10, 2);

    -- محاسبه مجموع حقوق کارمندان دپارتمان مربوطه
    SELECT SUM(Salary)
    INTO total_salary
    FROM Employees
    WHERE DepartmentID = NEW.DepartmentID;

    -- دریافت بودجه دپارتمان
    SELECT Budget
    INTO department_budget
    FROM Departments
    WHERE DepartmentID = NEW.DepartmentID;

    -- بررسی تجاوز از بودجه
    IF total_salary > department_budget THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Salary exceeds the department budget.';
    END IF;
END;
```

تست‌های فصل نهم: نرمال‌سازی

۱۱۵- چند مورد از عبارات زیر، نادرست است؟

(مهندسی کامپیوتر-دولتی۱۴۰۳)

- اگر یک مجموعه از جداول در سطح 3NF باشد، حتما 2NF هم هست.
- اگر یک مجموعه از جداول در سطح BCNF باشد، حتما 2NF هم هست.
- ممکن است یک مجموعه از جداول در سطح BCNF باشد، ولی 3NF نباشد.
- ممکن است یک مجموعه از جداول در سطح 4NF باشد ولی BCNF نباشد.

۱) یک
۲) دو
۳) سه
۴) چهار

استاد ارسطو خلیلی فر
مولف کتاب پایگاه داده انتشارات بابان

پاسخ تست‌های نهم: نرمال‌سازی

۱۱۵- گزینه (۲) صحیح است.

گزاره اول درست است. زیرا هر جدولی که در سطح 3NF باشد، حتما 1NF و 2NF هم هست. برای رسیدن به 3NF، ابتدا جدول باید 1NF و 2NF را پاس کند.

گزاره دوم درست است. زیرا هر جدولی که در سطح BCNF باشد، حتما 1NF و 2NF و 3NF هم هست. برای رسیدن به BCNF، ابتدا جدول باید 1NF و 2NF و 3NF را پاس کند.

گزاره سوم نادرست است. زیرا هر جدولی که در سطح BCNF باشد، حتما 1NF و 2NF و 3NF هم هست. برای رسیدن به BCNF، ابتدا جدول باید 1NF و 2NF و 3NF را پاس کند.

گزاره چهارم نادرست است. زیرا هر جدولی که در سطح 4NF باشد، حتما 1NF و 2NF و 3NF و BCNF هم هست. برای رسیدن به 4NF، ابتدا جدول باید 1NF و 2NF و 3NF و BCNF را پاس کند.

در سطوح نرمال‌سازی شکل زیر گویای مطلب است: