



اصول طراحی پایگاه داده

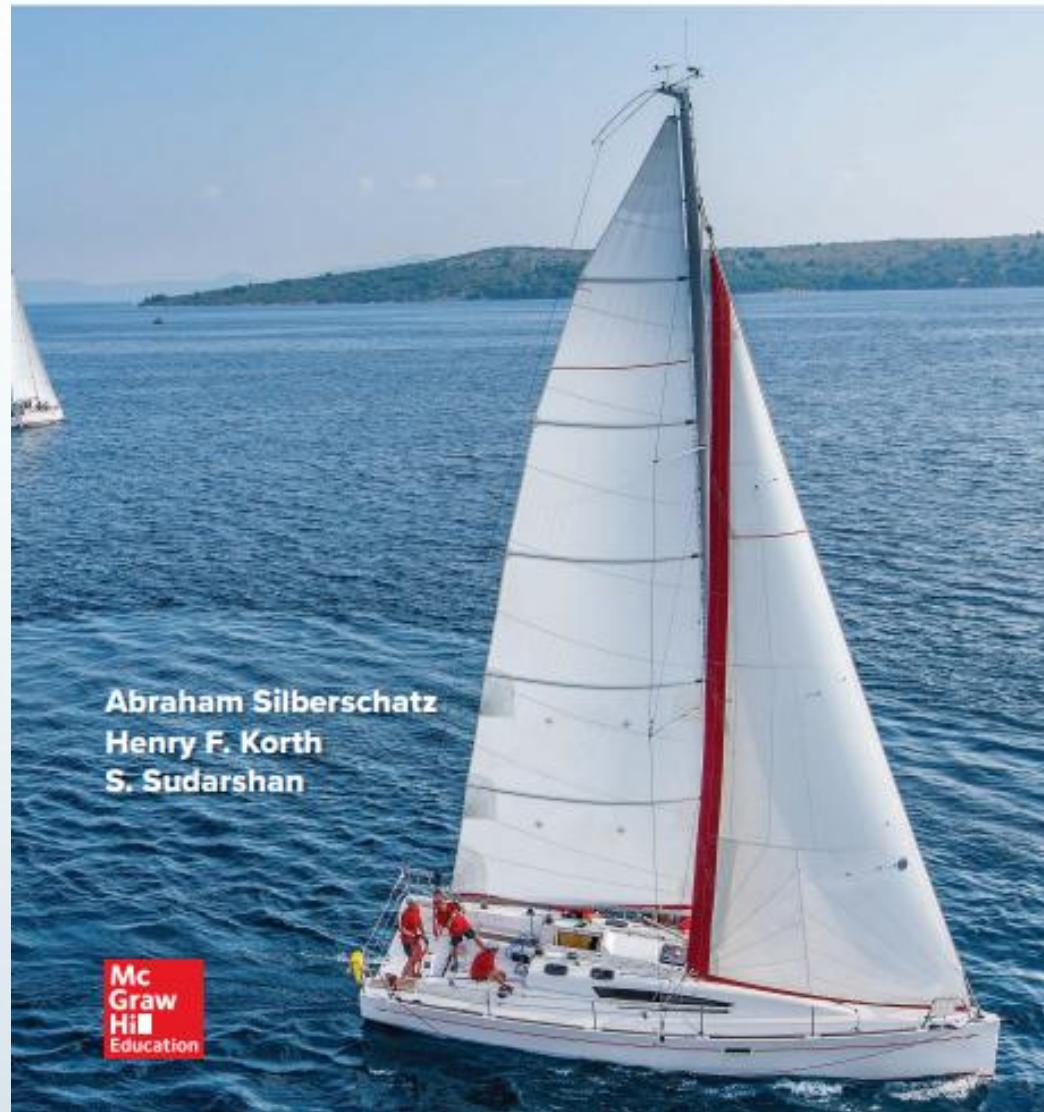
By Dr. Taghinezhad

Mail:

a0taghinezhad@gmail.com

SEVENTH EDITION

Database System Concepts





Chapter 4 : Intermediate SQL

Database System Concepts, 7th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



Outline

- عبارات پیوند (Join Expressions)
- نماها (Views)
- تراکنش‌ها (Transactions)
- قیود یکپارچگی (Integrity Constraints)
- نوع داده و طرح‌ها (SQL Data Types and Schemas)
- تعریف شاخص (Index Definition in SQL)
- مجوزدهی (Authorization)



Joined Relations (پیوندهای رابطه‌ای)

- عملگرهای پیوند دو رابطه را به عنوان ورودی دریافت کرده و یک رابطه دیگر را به عنوان نتیجه بازمی‌گردانند.
- یک عملگر پیوند در واقع یک حاصل ضرب دکارتی (Cartesian product) است که مستلزم آن است که تاپل در دو رابطه، تحت یک شرط مشخص، با یکدیگر تطابق داشته باشند. همچنین این عملگر، صفاتی را که در نتیجه نهایی پیوند حضور خواهند داشت، مشخص می‌کند.
- عملگرهای پیوند معمولاً به عنوان عبارت‌های زیرپرس وجو در بند **FROM** استفاده می‌شوند.
- سه نوع اصلی پیوند وجود دارد:
 - Natural join (پیوند طبیعی)
 - Inner join (پیوند درونی)
 - Outer join (پیوند بیرونی)



(پیوند طبیعی) Natural Join

- پیوند طبیعی تاپل‌هایی را که دارای مقادیر یکسان در تمام صفات مشترک هستند، با هم تطبیق می‌دهد و تنها یک نسخه از هر ستون مشترک را در خروجی حفظ می‌کند.

- لیست نام مدرسین به همراه شناسه درس (Course ID) دروسی که تدریس کرده‌اند:

- ```
select name, course_id
 from students, takes
 where student.ID = takes.ID;
```

- همین پرس‌وجو با ساختار NATURAL JOIN در SQL :

- ```
select name, course_id
      from student natural join takes;
```



Natural Join (پیوند طبیعی)

- بند FROM می‌تواند چندین رابطه (Relation) را با استفاده از پیوند طبیعی (Natural Join) ترکیب کند.

```
select A1, A2, ... An  
from r1 natural join r2 natural join .. natural join rn  
where P;
```



TAKES , STUDENT جداول

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

ID	course_id	sec_id	semester	year	grad
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	nu



student natural join takes

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>tot_cred</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>grade</i>
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	<i>null</i>



خطرات پیوند طبیعی

- مراقب صفات نامرتبه (Unrelated Attributes) با نام یکسان باشید که به اشتباہ توسط عملگر پیوند طبیعی با هم معادل‌سازی (equated) می‌شوند.
- مثال: لیست نام دانشجویان به همراه عنوان دروسی که اخذ کرده‌اند.

• نسخه صحیح:

```
select name, title  
from student natural join takes, course  
where takes.course_id = course.course_id;
```

• نسخه نادرست:

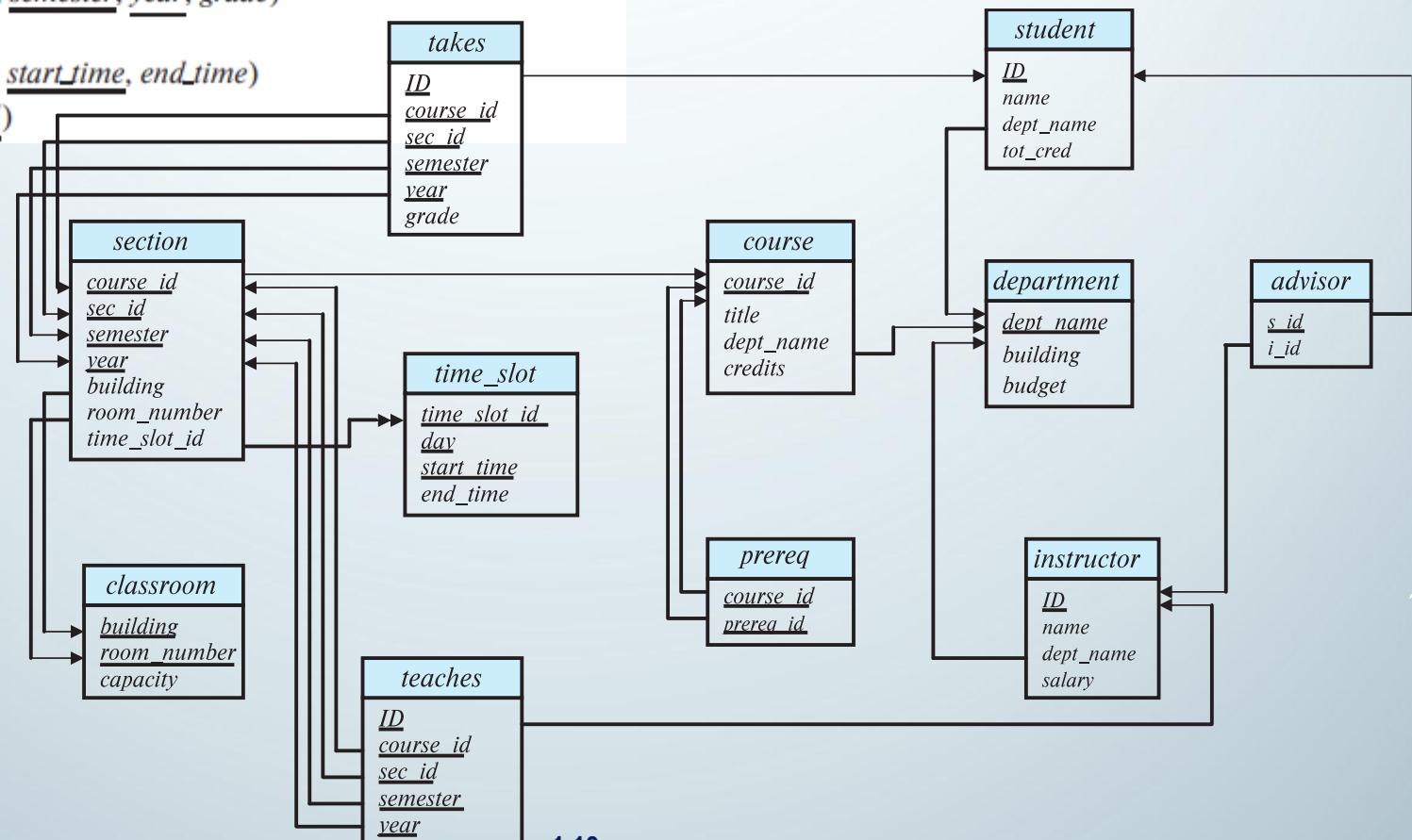
```
select name, title  
from student natural join takes natural join course;
```

- این پرس‌وجو تمامی جفت‌های (نام دانشجو، عنوان درس) را که در آن‌ها دانشجو درسی را از دپارتمانی غیر از دپارتمان خودش اخذ کرده است، حذف می‌کند
- نسخه صحیح، تاپل‌هایی را که در آن‌ها دپارتمان دانشجو و درس یکسان نیست، به درستی خروجی می‌دهد.



طرح و دیاگرام پایگاهداده دانشگاه

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)





پیوند طبیعی با استفاده از using

- برای پرهیز از معادل‌سازی اشتباه صفات، می‌توانیم از ساختار "using" استفاده کنیم که به ما اجازه می‌دهد دقیقاً مشخص کنیم کدام ستون‌ها باید معادل‌سازی شوند.
- مثال:

```
select name, title  
  from (student natural join takes) join course using  
(course_id)
```



شرط پیوند

شرط ON امکان تعریف یک گزاره کلی (General Predicate) بر روی رابطه‌هایی که در حال پیوند هستند را فراهم می‌آورد.

این گزاره مانند محمول بند WHERE نوشته می‌شود، با این تفاوت که از کلمه کلیدی استفاده می‌کند.

مثال:

```
select *  
from student join takes on student_ID = takes_ID
```

شرط ON بالا مشخص می‌کند که یک تاپل از رابطه student با یک تاپل از رابطه takes تطابق دارد، اگر مقادیر ID آنها برابر باشد.

معادل است با:

```
select *  
from student , takes  
where student_ID = takes_ID
```



Outer Join (پیوند بیرونی)

- تعمیمی (Extension) از عملگر پیوند است که از از دست رفتن اطلاعات جلوگیری می‌کند.
- این عملگر ابتدا پیوند را محاسبه می‌کند و سپس تاپل‌هایی (Tuples) را از یک رابطه که با تاپل‌های رابطه دیگر تطابق ندارند، به نتیجه پیوند اضافه می‌کند.
- از مقادیر نال (Null) استفاده می‌کند.
- سه نوع پیوند بیرونی وجود دارد:
 - left outer join (پیوند بیرونی چپ)
 - right outer join (پیوند بیرونی راست)
 - full outer join (پیوند بیرونی کامل)



مثال‌های Outer Join

جدول *course*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

جدول *prereq*

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

دقت کنید که:

اطلاعات CS-347 در جدول *course* گم شده است.

اطلاعات CS-315 در جدول *prereq* گم شده است.



Left Outer Join

- *course natural left outer join prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>

در جبر رابطه‌ای: *course* \bowtie *prereq*





Right Outer Join

- *course natural right outer join prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

در جبر رابطه‌ای: *course* \bowtie *prereq* ▪



Full Outer Join

- *course natural full outer join prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	null
CS-347	null	null	null	CS-101

در جبر رابطه‌ای: *course* \bowtie *prereq*





انواع و شروط پیوند

- عملگرهای پیوند، دو رابطه را به عنوان ورودی دریافت کرده و یک رابطه دیگر را به عنوان نتیجه باز می‌گردانند.
- این عملگرهای تکمیلی معمولاً به عنوان عبارت‌های زیرپرس‌ساز (subquery) استفاده می‌شوند.
- شرط پیوند – تعریف می‌کند که کدام تاپل‌ها در دو رابطه با یکدیگر تطابق دارند.
- نوع پیوند – تعریف می‌کند که با تاپل‌هایی در هر رابطه که (بر اساس شرط پیوند) با هیچ تاپلی در رابطه دیگر تطابق ندارند، چگونه رفتار می‌شود.

<i>Join types</i>
inner join
left outer join
right outer join
full outer join

<i>Join conditions</i>
natural
on < predicate >
using (A₁, A₂, ..., A_n)



مثال‌ها

- *course natural right outer join prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	null	null	null	CS-101

- *course full outer join prereq using (course_id)*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	null
CS-347	null	null	null	CS-101

course_id	title	dept_name	credits	course_id	prereq_id
BIO-301	Genetics	Biology	4	BIO-301	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-190	CS-101
CS-315	Robotics	Comp. Sci.	3	CS-347	CS-101



مثال‌ها

- **course inner join prereq on**
 $course.course_id = prereq.course_id$

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

تفاوت پیوند بالا با پیوند طبیعی چیست؟

- **course left outer join prereq on**
 $course.course_id = prereq.course_id$

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	null	null

course_id	title	dept_name	credits	course_id	prereq_id
BIO-301	Genetics	Biology	4	BIO-301	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-190	CS-101
CS-315	Robotics	Comp. Sci.	3	CS-347	CS-101



مثال‌ها

- ***course natural right outer join prereq***

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	null	null	null	CS-101

- ***course full outer join prereq using (course_id)***

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	null
CS-347	null	null	null	CS-101

course_id	title	dept_name	credits	course_id	prereq_id
BIO-301	Genetics	Biology	4	BIO-301	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-190	CS-101
CS-315	Robotics	Comp. Sci.	3	CS-347	CS-101



Views (نماها)

- در برخی موارد، مطلوب نیست که تمام کاربران، کل مدل منطقی (یعنی تمام رابطه‌های واقعی ذخیره شده در پایگاه داده) را ببینند.
- فردی را در نظر بگیرید که نیاز به دانستن نام و دپارتمان یک مدرس دارد، اما نیازی به اطلاع از حقوق او نیست. این فرد باید رابطه‌ای را ببیند که در SQL به صورت زیر تعریف شده است:

```
select ID, name, dept_name  
from instructor
```

- یک view، سازوکاری را برای پنهان‌سازی داده‌های مشخص از دید کاربران خاص فراهم می‌کند.
- هر رابطه‌ای که بخشی از مدل مفهومی (Conceptual Model) نباشد، اما به عنوان یک "رابطه مجازی" برای کاربر قابل مشاهده باشد، نما (view) نامیده می‌شود.



تعريف View

■ يك نما با استفاده از دستور CREATE VIEW تعريف مىشود که دارای ساختار زير است

create view *v* as <query expression >

■ در اين ساختار، <query expression> هر عبارت معتبر SQL است. نام نما با *v* نمايش داده مىشود.

■ به محض تعريف يك نما، مىتوان از نام آن برای ارجاع دادن به رابطه مجازی (Virtual Relation) که آن نما ايجاد مىکند، استفاده کرد.

■ تعريف نما با ايجاد يك رابطه جديد از طريق ارزیابی عبارت پرس و جو، متفاوت است.

■ در عوض، تعريف نما باعث ذخیره شدن يك عبارت مىشود؛ اين عبارت در پرس و جوهایی که از نما استفاده مىکنند، جایگزین میگردد.



تعريف و استفاده View

■ نمای مدرسین بدون حقوق آن‌ها:

```
create view faculty as  
    select ID, name, dept_name  
        from instructor
```

■ یافتن تمامی مدرسین در دپارتمان زیست‌شناسی:

```
select name  
from faculty  
where dept_name = 'Biology'
```

■ ایجاد نمای مجموع حقوق دپارتمان‌ها:

```
create view departments_total_salary(dept_name, total_salary) as  
    select dept_name, sum (salary)  
        from instructor  
    group by dept_name;
```



تعریف نماها با استفاده از نماهای دیگر

- می‌توان از یک نما در عبارت تعریف‌کننده نمای دیگر استفاده کرد.
- گفته می‌شود که یک رابطه نما ($v1$) دارای وابستگی مستقیم به یک رابطه نما ($v2$) است، اگر $v2$ در عبارت تعریف‌کننده $v1$ استفاده شود.
- گفته می‌شود که یک رابطه نما ($v1$) دارای وابستگی به رابطه نما ($v2$) است، اگر یا $v1$ دارای وابستگی مستقیم به $v2$ باشد یا یک مسیر از وابستگی‌ها از $v1$ به $v2$ وجود داشته باشد.
- گفته می‌شود که یک رابطه نما V بازگشتی (Recursive) است، اگر به خودش وابسته باشد.



تعریف نماها با استفاده از نماهای دیگر

- **create view *physics_fall_2017* as**
select *course.course_id, sec_id, building,*
room_number
from *course, section*
where *course.course_id = section.course_id*
and *course.dept_name = 'Physics'*
and *section.semester = 'Fall'*
and *section.year = '2017';*
- **create view *physics_fall_2017_watson* as**
select *course_id, room_number*
from *physics_fall_2017*
where *building= 'Watson';*



View بسط

: view بسط

```
create view physics_fall_2017_watson as  
select course_id, room_number  
from physics_fall_2017  
where building= 'Watson'
```

: ۴

```
create view physics_fall_2017_watson as  
select course_id, room_number  
from (select course.course_id, building, room_number  
      from course, section  
     where course.course_id= section.course_id  
       and course.dept_name = 'Physics'  
       and section.semester= 'Fall'  
       and section.year= '2017')  
  where building= 'Watson';
```



بسط View

- روشی برای تعریف مفهوم نماهایی است که بر اساس نماهای دیگر تعریف شده‌اند.
- فرض کنید نمای v_1 با عبارت e_1 تعریف شده باشد که ممکن است خود حاوی استفاده از رابطه‌های نما باشد.
- بسط view در یک عبارت، حلقه جایگزینی زیر را تکرار می‌کند:

repeat

Find any view relation v_i in e_1

Replace the view relation v_i by the expression
defining v_i

until no more view relations are present in e_1

- تا زمانی که تعاریف نماها بازگشتی (Recursive) نباشند، این حلقه خاتمه خواهد یافت.



بسط View

▪ تعریف view

- CREATE VIEW v1 AS SELECT a, b FROM t1;
- CREATE VIEW v2 AS SELECT v1.a, t2.c FROM v1, t2 WHERE v1.b = t2.b;
 - فرآیند بسط view
 - شناسایی: v1 یک رابطه نما است.
 - جایگزینی v1 با تعریف آن:
 - SELECT t1.a, t2.c FROM (SELECT a, b FROM t1) AS v1, t2 WHERE v1.b = t2.b;
 - عبارت نهايی:
 - عبارت نمای بسط يافته دیگر حاوی هیچ رابطه نمایی نیست.
 - نکات کلیدی:
 - تعاریف غیربازگشته: این فرآیند تا زمانی کار می‌کند که تعاریف نما بازگشته نباشند. تعاریف بازگشته باعث ایجاد حلقه بی‌نهایت می‌شوند.
 - کارایی: این بسط به درک و بهینه‌سازی پرس‌وجوها کمک کرده و اجرای صحیح و کارآمد را تضمین می‌کند.

▪ با استفاده از بسط نما، تعاریف نمای تودرتوی پیچیده به مؤلفه‌های بنیادی آن‌ها تجزیه می‌شود و پرس‌وجوی کلی را شفاف‌تر و قابل مدیریت‌تر می‌سازد



Materialized Views

- برخی از سیستم‌های پایگاه داده اجازه می‌دهند که رابطه‌های نما به صورت فیزیکی ذخیره شوند.
 - نسخه فیزیکی هنگام تعریف نما ایجاد می‌شود.
 - این نماها، **Materialized View** نامیده می‌شوند.

چالش: اگر رابطه‌های زیرین مورد استفاده در پرس‌وجو به روزرسانی شوند، نتیجه **Materialized View** قدیمی می‌شود.

نیاز است با به روزرسانی نما در هر زمان که رابطه‌های زیرین به روزرسانی می‌شوند، نما حفظ و نگهداری شود.



بروزرسانی View

- افزودن تاپل جدید به نمای `:faculty`

`insert into faculty`

`values ('30765', 'Green', 'Music');`

- این درج باید توسط یک درج در رابطه زیرین `instructor` نمایش داده شود.
 - باید دارای مقداری برای صفت حقوق (salary attribute) در جدول باشد.

دو رویکرد:

- Reject the insert
- Insert the tuple

`('30765', 'Green', 'Music', null)`

`into the instructor relation`



برخی از به روزرسانی ها را نمی توان به طور یکتا ترجمه کرد

- **create view instructor_info as**
select *ID, name, building*
from *instructor, department*
where *instructor.dept_name = department.dept_name;*
- **insert into** *instructor_info*
values ('69987', 'White', 'Taylor');

■ سوال:

- کدام دپارتمان، اگر چندین دپارتمان در Taylor وجود داشته باشد؟
- چه می شود اگر هیچ دپارتمانی در Taylor وجود نداشته باشد؟



و برعی دیگر اصلاً امکان پذیر نیستند

- **create view *history_instructors* as**
select *
from *instructor*
where *dept_name*= 'History';

چه اتفاقی می‌افتد اگر تاپل زیر را در **history_instructors** درج کنیم؟

('25566', 'Brown', 'Biology', 100000)



به روزرسانی نماها در SQL

اغلب پیاده‌سازی‌های SQL، به روزرسانی‌ها را تنها بر روی نماهای ساده مجاز می‌دانند:

- بند **FROM** تنها یک رابطه پایگاه داده داشته باشد.
- بند **SELECT** تنها حاوی نام صفات رابطه باشد و فاقد هرگونه عبارت، توابع تجمعی (aggregates) باشد.
- هر صفتی که در بند **SELECT** لیست نشده باشد، بتواند بر روی **NULL** تنظیم شود.
- پرس‌وجو فاقد بندهای **HAVING** یا **GROUP BY** باشد.



تراکنش‌ها (Transactions)

- یک تراکنش شامل دنباله‌ای از دستورات پرس‌وجو و/یا به‌روزرسانی است و یک "واحد" کار محسوب می‌شود.
- استاندارد SQL مشخص می‌کند که یک تراکنش به‌طور ضمنی (implicitly) هنگامی که یک دستور SQL اجرا می‌شود، آغاز می‌گردد.
- تراکنش باید با یکی از دستورات زیر خاتمه یابد:
- به‌روزرسانی‌های انجام شده توسط تراکنش در پایگاه داده دائمی می‌شوند. : **COMMIT WORK**
- تمام به‌روزرسانی‌های انجام شده توسط دستورات SQL در تراکنش، خنثی (undone) می‌شوند. : **ROLLBACK WORK**
- تراکنش : **Atomic**
 - یا به‌طور کامل اجرا می‌شود یا به‌گونه‌ای بازگردانی می‌شود که گویی هرگز رخ نداده است.
 - جداسازی (Isolation) از تراکنش‌های همزمان



قیود(محدودیت‌های) یکپارچگی

- قیود یکپارچگی با تضمین این‌که تغییرات مجاز در پایگاه داده منجر به از دست رفتن سازگاری داده‌ها نشود، در برابر آسیب‌های تصادفی از پایگاه داده محافظت می‌کند.
- موجودی یک حساب جاری باید بیشتر از ۱۰,۰۰۰.۰۰ دلار باشد.
- حقوق یک کارمند بانک باید حداقل ۴۰۰ دلار در ساعت باشد.
- یک مشتری باید دارای شماره تلفن (غیر null) باشد.



قیود بر روی یک رابطه

- **not null**
- **primary key**
- **unique**
- **check (P)**, where P is a predicate



Not Null قیود

not null ▪

- مشخص کردن نام و بودجه به صورت **not null**

name varchar(20) not null
budget numeric(12,2) not null



قيود Unique

■ **unique** (A_1, A_2, \dots, A_m)

- مشخصه UNIQUE بیان می‌کند که صفات A_1, A_2, \dots, A_m یک کلید کاندید (Candidate Key) را تشکیل می‌دهند.
- کلیدهای کاندید (برخلاف کلیدهای اصلی) مجاز به null بودن هستند.



بند check

- بند CHECK (P) یک گزاره P را مشخص می‌کند که باید توسط هر تاپل در یک رابطه برآورده شود.
- تضمین این‌که ترم semester یکی از مقادیر پاییز، زمستان، بهار یا تابستان باشد:

create table section

```
(course_id varchar (8),  
sec_id varchar (8),  
semester varchar (6),  
year numeric (4,0),  
building varchar (15),  
room_number varchar (7),  
time_slot_id varchar (4),  
primary key (course_id, sec_id, semester, year),  
check (semester in ('Fall', 'Winter', 'Spring', 'Summer')))
```



جامعیت ارجاعی (Referential Integrity)

- جامعیت ارجاعی تضمین می‌کند که برای یک مجموعه صفت مشخص در یک رابطه ظاهر می‌شود، برای مجموعه صفت مشخصی در رابطه دیگر نیز وجود داشته باشد.
 - مثال: اگر "Biology" نام دپارتمانی باشد که در یکی از تاپل‌های رابطه ظاهر می‌شود، آنگاه باید یک تاپل برای "Biology" در رابطه instructor نیز وجود داشته باشد.
- کلید خارجی (Foreign Key): فرض کنید A مجموعه‌ای از صفات باشد. فرض کنید R و S دو رابطه‌ای باشند که صفات A را دربردارند و A کلید اصلی S باشد.
 - گفته می‌شود که A کلید خارجی R است، اگر به ازای هر مقداری از A که در R ظاهر می‌شود، این مقادیر در S نیز ظاهر شوند.



جامعیت ارجاعی (Referential Integrity)

- کلیدهای خارجی می‌توانند به عنوان بخشی از دستور CREATE TABLE در SQL مشخص شوند

foreign key (dept_name) references department

- به طور پیش‌فرض، یک کلید خارجی به صفات کلید اصلی رابطه مرجع، ارجاع می‌دهد.

sql اجازه می‌دهد لیستی از صفات رابطه مرجع به‌طور صریح (explicitly) مشخص شود:

***foreign key (dept_name) references department
(dept_name)***



اقدامات آبشاری (Cascading) در جامعیت ارجاعی

- هنگامی که یک قید جامعیت ارجاعی نقض می‌شود، روال معمول این است که عملی را که باعث نقض شده است، رد (Reject) کنیم.
- یک گزینه جایگزین، در مورد عملیات حذف (Delete) یا بهروزرسانی (Update)، استفاده از Cascade است

```
create table course (
  ...
  dept_name varchar(20),
  foreign key (dept_name) references department
    on delete cascade
    on update cascade,
  ...)
```

- به جای Cascade می‌توانیم از موارد زیر استفاده کنیم
 - set null,
 - set default



نقض قید یکپارچگی در طول تراکنش‌ها

در نظر بگیرید:

```
create table person (
    /D char(10),
    name char(40),
    mother char(10),
    father char(10),
    primary key /D,
    foreign key father references person,
    foreign key mother references person)
```

چگونه یک tuple را بدون ایجاد نقض قید یکپارچگی درج کنیم؟

- راه حل‌ها:
 - پدر و مادر یک شخص را قبل از درج شخص، درج کنید.
 - یا، پدر و مادر را در ابتدا **Null** تنظیم کنید، و پس از درج تمام اشخاص، به روزرسانی کنید (در صورتی که صفات پدر و مادر به صورت **not null** اعلام نشده باشند، این روش امکان‌پذیر نیست).
 - یا، بررسی قید را به تعویق اندازید.



شروط CHECK پیچیده

- گزاره (Predicate) در بند CHECK می‌تواند یک گزاره دلخواه باشد که می‌تواند شامل یک زیرپرس و جو (Subquery) باشد.

```
check (time_slot_id in (select time_slot_id from  
time_slot))
```

- شرط CHECK بیان می‌کند که شناسه بازه زمانی (time_slot_id) در هر تاپل از رابطه section در واقع شناسه یک بازه زمانی در رابطه time_slot است.

- این شرط نه تنها زمانی که تاپلی در section درج یا اصلاح می‌شود، بلکه هنگامی که رابطه time_slot تغییر می‌کند نیز باید بررسی شود.



Assertions

- یک **Assertion**، یک گزاره (**Predicate**) است که بیانگر شرطی است که می‌خواهیم پایگاه داده همیشه آن را برآورده سازد.
- قیود زیر، می‌توانند با استفاده از **Assertion** بیان شوند:
- به ازای هر تاپل در رابطه **student**، مقدار صفت مجموع واحدها (**tot_cred**) باید برابر با مجموع واحدهای دروسی باشد که دانشجو با موفقیت گذرانده است.
- یک مدرس نمی‌تواند در یک ترم، در بازه زمانی یکسان در دو کلاس درس متفاوت تدریس کند.
- یک **SQL** در **Assertion** به صورت زیر است:

```
create assertion <assertion-name> check (<predicate>);
```



مقایسه محدودیت و ادعا

Constraint محدودیت

- قانونی است که روی یک جدول اعمال می‌شود و صحت داده‌ها را به صورت محلی بررسی می‌کند.
- مثال:

CHECK (semester IN ('Fall','Spring'))

این محدودیت تضمین می‌کند مقدار **semester** فقط از مقادیر مجاز باشد.

Assertion ادعا

- قانونی سراسری است که می‌تواند چند جدول و شروط پیچیده را دربر بگیرد.
- مثال مفهومی:

تعداد دانشجویان هر درس نباید بیشتر از ۵۰ نفر باشد.

در عمل با **Trigger** پیاده‌سازی می‌شود، چون **ASSERTION** ها از DBMS پشتیبانی نمی‌کنند.

جمع‌بندی کوتاه

Constraint: ساده، محلی، پشتیبانی شده

Assertion: کلی، مفهومی، معمولاً با **Trigger** جایگزین می‌شود



مقایسه محدودیت و ادعا

در استاندارد SQL، **ASSERTION** برای بیان «قیود سراسری» تعریف شده است؛ اما در عمل هیچ DBMS تجاری مهمی دستور CREATE ASSERTION را پیاده‌سازی نکرده است. دلیل اصلی، هزینه محاسباتی بالا و پیچیدگی بررسی مداوم قیود بین جدولی است. به همین علت، سیستم‌های واقعی از جایگزین‌ها استفاده می‌کنند.

ASSERTION شبیه‌سازی Triggers

وقتی قانون چند جدول یا شرط کلی داشته باشد، از **Trigger** استفاده می‌شود.

```
CREATE OR REPLACE TRIGGER trg_max_enrollment  
AFTER INSERT OR UPDATE ON takes
```

```
FOR EACH ROW
```

```
DECLARE
```

```
    v_count NUMBER;
```

```
BEGIN
```

```
    SELECT COUNT(*)
```

```
    INTO v_count
```

```
    FROM takes
```

```
    WHERE course_id = :NEW.course_id;
```

```
    IF v_count > 50 THEN
```

```
        RAISE_APPLICATION_ERROR(
```

```
            -20001,
```

```
            'حداکثر ظرفیت درس پر شده است'
```

```
        );
```

```
    END IF;
```

```
END;
```

قانون (**Assertion** مفهومی)

تعداد دانشجویان ثبت‌نام شده در هر درس نباید بیشتر از ۵۰ نفر باشد.

پیاده‌سازی در Oracle با Trigger



Built-in Data Types in SQL

- **date:** تاریخ، حاوی سال (۴ رقمی)، ماه و روز.
 - Example: **date** '2005-7-27'
- **time:** زمان روز، بر حسب ساعت، دقیقه و ثانیه.
 - Example: **time** '09:00:30' **time** '09:00:30.75'
- **timestamp:** تاریخ به علاوه زمان روز.
 - Example: **timestamp** '2005-7-27 09:00:30.75'
- **interval:** یک دوره زمانی.
 - Example: **interval** '1' day
- تفریق یک مقدار تاریخ/زمان /بر چسب زمانی از دیگری، یک مقدار بازه زمانی به دست می‌دهد.
- مقادیر بازه زمانی را می‌توان به مقادیر تاریخ/زمان /بر چسب زمانی اضافه کرد.



نوع داده‌ای اشیاء بزرگ (Large-Object)

- اشیاء بزرگ (مانند عکس‌ها، ویدئوها، فایل‌های CAD و غیره) به صورت یک شیء بزرگ ذخیره می‌شوند:

- **Binary Large Object (BLOB)** - شیء بزرگ دودویی): شیء عبارت است از یک مجموعه بزرگ از داده‌های دودویی تفسیرنشده (که تفسیر آن به یک برنامه کاربردی خارج از سیستم پایگاه داده سپرده می‌شود).
- **Character Large Object (CLOB)** - شیء بزرگ کاراکتری): شیء عبارت است از یک مجموعه بزرگ از داده‌های کاراکتری.
- هنگامی که یک پرس‌وجو یک شیء بزرگ را باز می‌گرداند، به جای خود شیء بزرگ، یک اشاره‌گر (Pointer) بازگردانده می‌شود.



نوع داده‌ای تعریف شده توسط کاربر (User-Defined)

- ساختار SQL ، انواع تعریف شده توسط کاربر را ایجاد می‌کند

create type *Dollars* as numeric (12,2) final

- مثال:

```
create table department
(dept_name varchar (20),
building varchar (15),
budget Dollars);
```



دامنه‌ها - Domains

- ساختار CREATE DOMAIN در SQL-92، انواع دامنه تعریف شده توسط کاربر را ایجاد می‌کند

create domain person_name char(20) not null

- انواع و دامنه‌ها مشابه هستند. می‌توان برای دامنه‌ها، قیودی مانند null نبودن را بر روی آن‌ها مشخص کرد.
- مثال:

**create domain degree_level varchar(10)
constraint degree_level_test
check (value in ('Bachelors', 'Masters', 'Doctorate'));**



ایجاد شاخص (Index)

- بسیاری از پرس‌وجوها تنها به بخش کوچکی از رکوردهای یک جدول ارجاع می‌دهند.
- خواندن هر رکورد برای یافتن یک رکورد با مقدار مشخص، برای سیستم ناکارآمد است.
- **شاخص** بر روی یک صفت از یک رابطه، یک ساختار داده است که به سیستم پایگاه داده اجازه می‌دهد تاپل‌هایی را در آن رابطه که دارای یک مقدار مشخص برای آن صفت هستند، به‌طور کارآمد و بدون پیمایش تمامی تاپل‌های رابطه، بیابد.
- ما یک شاخص را با استفاده از دستور **CREATE INDEX** ایجاد می‌کنیم:

create index <name> on <relation-name> (attribute);



مثال ساخت شاخص

- **create table student**
*(ID varchar (5),
name varchar (20) not null,
dept_name varchar (20),
tot_cred numeric (3,0) default 0,
primary key (ID))*
- **create index studentID_index on student(ID)**

پرس و جو:

- **select *
from student
where ID ='12345'**

می‌تواند با استفاده از شاخص برای یافتن رکورد مورد نیاز، بدون بررسی تمام رکوردهای دانشجو، اجرا شود.



مجوزدهی (Authorization)

- ما می‌توانیم چندین نوع مجوزدهی را بر روی بخش‌هایی از پایگاه داده به یک کاربر تخصیص دهیم
 - خواندن (Read): اجازه خواندن را می‌دهد، اما اجازه اصلاح داده‌ها را نمی‌دهد.
 - درج (Insert): پ اجازه درج داده‌های جدید را می‌دهد، اما اجازه اصلاح داده‌های موجود را نمی‌دهد.
 - بهروزرسانی (Modification) : اجازه اصلاح (Modification) را می‌دهد، اما اجازه حذف داده‌ها را نمی‌دهد.
 - حذف (Delete) : اجازه حذف داده‌ها را می‌دهد.
- هر یک از این انواع مجوزدهی، یک امتیاز (Privilege) نامیده می‌شود. می‌توانیم تمامی، هیچ‌کدام یا ترکیبی از این امتیازات را بر روی بخش‌های مشخصی از پایگاه داده، مانند یک رابطه یا یک نما، به کاربر اعطا کنیم.



مجوزدهی (Authorization)

- اشکال مجوزدهی برای اصلاح طرح‌واره پایگاه داده
- شاخص (Index) : اجازه ایجاد و حذف شاخص‌ها را می‌دهد.
- منابع (Resources) : اجازه ایجاد رابطه‌های جدید را می‌دهد.
- تغییر (Alteration) : اجازه افزودن یا حذف صفات در یک رابطه را می‌دهد.
- حذف (Drop) : اجازه حذف رابطه‌ها را می‌دهد.



مشخص کردن مجوزدهی در SQL

دستور GRANT برای اعطای مجوزدهی استفاده می‌شود

grant <privilege list> on <relation or view > to <user list>

عبارت است از:

- یک شناسه کاربر
- که به تمام کاربران معتبر اجازه می‌دهد امتیاز اعطای شده را داشته باشند.
- یک نقش (Role)

مثال:

grant select on department to Akbar, Asghar

اعطای یک امتیاز بر روی یک نما به معنای اعطای هیچ امتیازی بر روی رابطه‌های زیرین آن نیست.

اعطا کننده امتیاز باید قبلًا امتیاز را بر روی آیتم مشخص شده دارا باشد (یا مدیر پایگاه داده باشد).



امتیازات در SQL

- **SELECT** (انتخاب): اجازه دسترسی خواندن به رابطه، یا توانایی پرس و جو با استفاده از نما.
- مثال: اعطای مجوز **SELECT** بر روی رابطه *instructor* به کاربران U_1 , U_2 و U_3 :
grant select on instructor to U_1 , U_2 , U_3
- **INSERT** (درج): توانایی درج تاپل‌ها.
- **UPDATE** (بهروزرسانی): توانایی بهروزرسانی با استفاده از دستور **UPDATE** در SQL
- **DELETE** (حذف): توانایی حذف تاپل‌ها.
- **ALL PRIVILEGES** (تمامی امتیازات): به عنوان شکل کوتاه برای تمامی امتیازات مجاز استفاده می‌شود.



لغو مجوزدهی در Sql

دستور REVOKE برای لغو مجوزدهی استفاده می‌شود:

revoke <privilege list> on <relation or view> from <user list>

مثال:

revoke select on student from U₁, U₂, U₃

می‌تواند ALL باشد تا تمامی امتیازاتی را که سلب شونده ممکن است دارا باشد، لغو کند.

اگر <revokkee-list> شامل PUBLIC باشد، تمامی کاربران امتیاز را از دست می‌دهند، به جز کسانی که به طور صریح (explicitly) آن را دریافت کرده‌اند.

اگر همان امتیاز دو بار توسط اعطا کنندگان (grantees) متفاوت به یک کاربر اعطا شده باشد، ممکن است کاربر پس از لغو، همچنان آن امتیاز را حفظ کند.

تمامی امتیازاتی که به امتیاز در حال لغو وابسته هستند، نیز لغو می‌شوند.



نقش‌ها

- نقش روشی است برای تمایز قائل شدن بین کاربران مختلف، از نظر این‌که این کاربران به چه چیزهایی در پایگاه داده می‌توانند دسترسی/به‌روزرسانی داشته باشند.
- برای ایجاد نقش از دستور زیر استفاده می‌کنیم:
 - **create role <name>**
 - مثال:
 - **create role instructor**
 - پس از ایجاد نقش، می‌توانیم "کاربران" را با استفاده از دستور زیر به آن نقش تخصیص دهیم:
 - **grant <role> to <users>**



Roles مثال

create role *instructor*;

grant *instructor* **to** Amit;

امتیازات می‌توانند به نقش‌ها اعطا شوند:

- **grant select on** *takes* **to** *instructor*,

نقش‌ها می‌توانند هم به کاربران و هم به نقش‌های دیگر اعطا شوند:

- **create role** *teaching_assistant*

- **grant** *teaching_assistant* **to** *instructor*,

تمامی امتیازات *teaching_assistant* را به ارث می‌برد.

زنجیره نقش‌ها:

- **create role** *dean*;

- **grant** *instructor* **to** *dean*;

- **grant** *dean* **to** Satoshi;



مجوزدهی بر روی نماها

- **create view geo_instructor as**
(select *
from instructor
where dept_name = 'Geology');
- **grant select on geo_instructor to geo_staff**
 - فرض کنید یک عضو geo_staff دستور زیر را اجرا می‌کند:
 - **select ***
from geo_instructor,
 - چه می‌شود اگر:
 - مجوزهایی بر روی instructor نداشته باشد؟
 - ایجاد‌کننده نما، مجوزهایی بر روی instructor نداشته باشد؟



ویژگی های دیگر مجوزها

■ مجوز References برای ایجاد کلید خارجی:

- **grant reference (*dept_name*) on *department* to Mariano;**
 - چرا نیاز داریم؟
- انتقال مجوزها:
- **grant select on *department* to Amit with grant option;**
- **revoke select on *department* from Amit, Satoshi cascade;**
- **revoke select on *department* from Amit, Satoshi restrict;**
 - و موارد دیگر...



سایر ویژگی‌های مجوزدهی

- **GRANT REFERENCES (dept_name) ON department TO Mariano;**

▪ مجوز **REFERENCES** به یک کاربر (در این حالت، Mariano) اجازه می‌دهد یک کلید خارجی ایجاد کند که به ستون مشخص شده **department** (در جدول **dept_name**) ارجاع می‌دهد.

- **REVOKE SELECT ON department FROM Amit, Satoshi CASCADE;**

▪ تضمین می‌کند که لغو امتیاز گسترش یابد و امتیازات را از تمامی دریافتکنندگان غیرمستقیم آن امتیاز حذف کند.

- **REVOKE SELECT ON department FROM Amit, Satoshi RESTRICT;**

▪ در صورتی که کاربرانی امتیاز را از Amit یا Satoshi دریافت کرده باشند، جلوی لغو را می‌گیرد. این امر سلسله مراتب امتیازات اعطا شده را حفظ می‌کند.

- **grant select on department to Amit with grant option;**

▪ به این معنی است که Amit می‌تواند امتیاز خود را به کاربران دیگر انتقال دهد.



مقایسه RESTRICT و CASCADE در لغو امتیاز

جنبه	CASCADE	RESTRICT
نحوه عملکرد	امتیاز را از کاربران مشخص شده و تمامی کسانی که امتیاز توسط آنها به آنها اعطا شده بود، لغو می‌کند.	امتیاز را تنها در صورتی لغو می‌کند که هیچ کاربر وابسته‌ای وجود نداشته باشد. در غیر این صورت، خطا ایجاد می‌کند.
مورد استفاده	زمانی استفاده می‌شود که بخواهید تمامی وابستگی‌ها را از بین ببرید.	زمانی استفاده می‌شود که بخواهید اطمینان حاصل کنید که هیچ کاربر دیگری تحت تأثیر قرار نگیرد.
مثال	اگر Amit امتیاز SELECT را به Bob اعطا کرده باشد، با لغو امتیاز Amit، امتیاز Bob نیز سلب می‌شود.	امتیاز Amit قابل لغو نخواهد بود اگر Bob امتیاز را از Amit دریافت کرده باشد



End of Chapter 4