Chapter 4 : Intermediate SQL

Database System Concepts, 7th Ed.

Outline

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Index Definition in SQL
- Authorization

Joined Relations

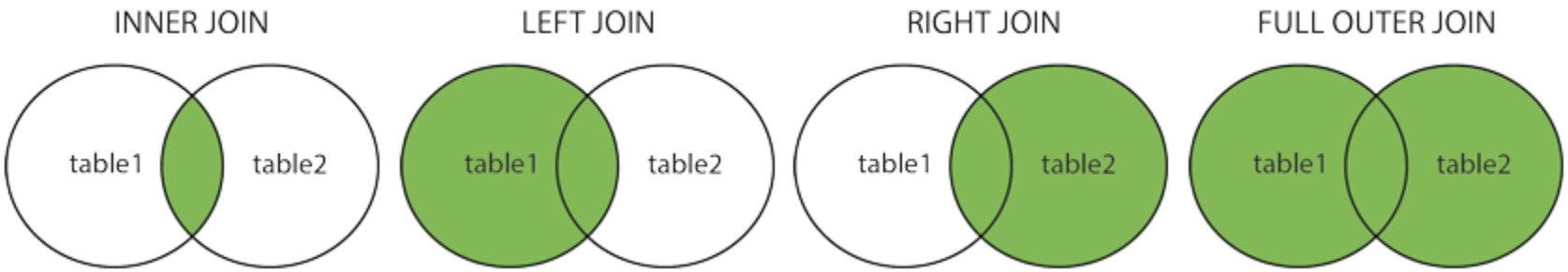
- **Join operations** take two relations and return as a result another relation.

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join

Important types of joins

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



INNER JOIN

- The INNER JOIN keyword selects records that have matching values in both tables:
- SELECT *column_name(s)*
  FROM *table1* INNER JOIN *table2*
  ON *table1.column_name = table2.column_name*;

- The **on** condition allows a general predicate over the relations being joined
- Query example

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

  - The **on** condition above specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal.
- Equivalent to:

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

این مفهومش با inner join یکی است

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.
- Three forms of outer join:
  - left outer join
  - right outer join
  - full outer join

Outer Join Examples

- Relation *course*

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

- Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

  course information is missing CS-437

  prereq information is missing CS-315

- *X*

Joined Relations – Examples

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

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

- *Select **

  from 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

- *Select **

  from 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*

```
SELECT * FROM [Customer_v] t;
select * from [Branch_v];
```

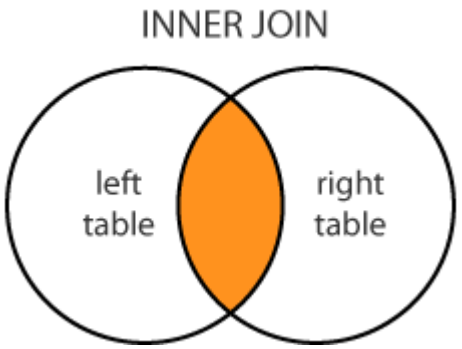
100 %

Results | Messages

	customer_number	name	L_name	Branch_Cod	salary
1	1	mohammad	mohammadi	20	1000
2	2	Saeed	mohammadi	60	2000
3	21	arash	mofidi	20	4000
4	16	ali	navidi	32	4000
5	18	ali	ahmadi	NULL	3000
6	20	ali	mohammadi	20	4000
7	3	Saeed	mofidi	20	6000
8	4	hamid	hamidi	20	5000

	Branch_Code	Branch_Name	Address
1	20	b20	Tehran
2	60	b60	Isf
3	120	b120	Tehran

```
SELECT *
FROM [Customer_v] t inner join [Branch_v]
on t.Branch_Cod = b.Branch_Code;
```

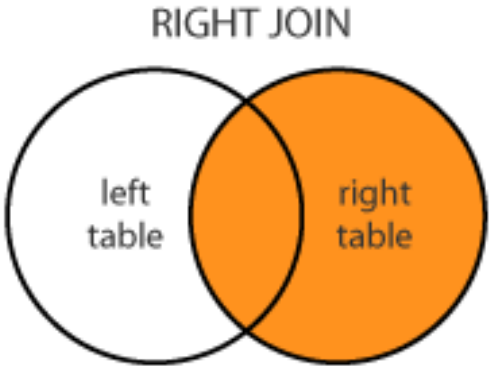


	customer_number	name	L_name	Branch_Cod	salary
1	1	mohammad	mohammadi	20	1000
2	2	Saeed	mohammadi	60	2000
3	21	arash	mofidi	20	4000
4	16	ali	navidi	32	4000
5	18	ali	ahmadi	NULL	3000
6	20	ali	mohammadi	20	4000
7	3	Saeed	mofidi	20	6000
8	4	hamid	hamidi	20	5000

	Branch_Code	Branch_Name	Address
1	20	b20	Tehran
2	60	b60	Isf
3	120	b120	Tehran

	customer_number	name	L_name	Branch_Cod	salary	Branch_Code	Branch_Name	Address
1	1	mohammad	mohammadi	20	1000	20	b20	Tehran
2	2	Saeed	mohammadi	60	2000	60	b60	Isf
3	21	arash	mofidi	20	4000	20	b20	Tehran
4	20	ali	mohammadi	20	4000	20	b20	Tehran
5	3	Saeed	mofidi	20	6000	20	b20	Tehran
6	4	hamid	hamidi	20	5000	20	b20	Tehran

```
SELECT *
 FROM [Customer_v] t right outer join [Branch_v] b
on t.Branch_Cod = b.Branch_Code;
```



	customer_number	name	L_name	Branch_Cod	salary
1	1	mohammad	mohammadi	20	1000
2	2	Saeed	mohammadi	60	2000
3	21	arash	mofidi	20	4000
4	16	ali	navidi	32	4000
5	18	ali	ahmadi	NULL	3000
6	20	ali	mohammadi	20	4000
7	3	Saeed	mofidi	20	6000
8	4	hamid	hamidi	20	5000

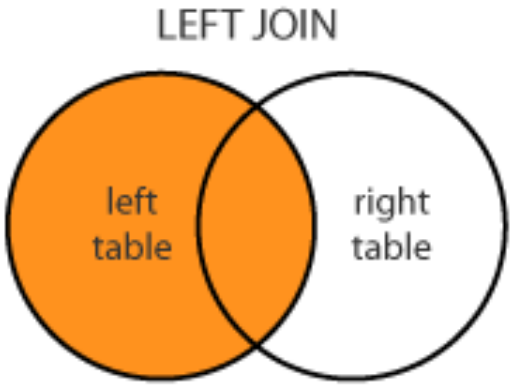
	Branch_Code	Branch_Name	Address
1	20	b20	Tehran
2	60	b60	Isf
3	120	b120	Tehran

	customer_number	name	L_name	Branch_Cod	salary	Branch_Code	Branch_Name	Address
1	1	mohammad	mohammadi	20	1000	20	b20	Tehran
2	21	arash	mofidi	20	4000	20	b20	Tehran
3	20	ali	mohammadi	20	4000	20	b20	Tehran
4	3	Saeed	mofidi	20	6000	20	b20	Tehran
5	4	hamid	hamidi	20	5000	20	b20	Tehran
6	2	Saeed	mohammadi	60	2000	60	b60	Isf
7	NULL	NULL	NULL	NULL	NULL	120	b120	Tehran

```
SELECT *
FROM [Customer_v] t left outer join [Branch_v] b
on t.Branch_Cod = b.Branch_Code;
```

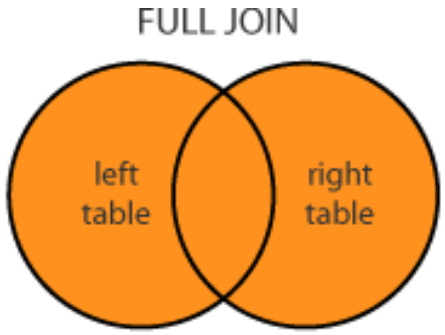
	customer_number	name	L_name	Branch_Cod	salary
1	1	mohammad	mohammadi	20	1000
2	2	Saeed	mohammadi	60	2000
3	21	arash	mofidi	20	4000
4	16	ali	navidi	32	4000
5	18	ali	ahmadi	NULL	3000
6	20	ali	mohammadi	20	4000
7	3	Saeed	mofidi	20	6000
8	4	hamid	hamidi	20	5000

	Branch_Code	Branch_Name	Address
1	20	b20	Tehran
2	60	b60	Isf
3	120	b120	Tehran



	customer_number	name	L_name	Branch_Cod	salary	Branch_Code	Branch_Name	Address
1	1	mohammad	mohammadi	20	1000	20	b20	Tehran
2	2	Saeed	mohammadi	60	2000	60	b60	Isf
3	21	arash	mofidi	20	4000	20	b20	Tehran
4	16	ali	navidi	32	4000	NULL	NULL	NULL
5	18	ali	ahmadi	NULL	3000	NULL	NULL	NULL
6	20	ali	mohammadi	20	4000	20	b20	Tehran
7	3	Saeed	mofidi	20	6000	20	b20	Tehran
8	4	hamid	hamidi	20	5000	20	b20	Tehran

```
SELECT *
FROM [Customer_v] t full outer join [Branch_v] b
on t.Branch_Cod = b.Branch_Code;
```



	customer_number	name	L_name	Branch_Cod	salary
1	1	mohammad	mohammadi	20	1000
2	2	Saeed	mohammadi	60	2000
3	21	arash	mofidi	20	4000
4	16	ali	navidi	32	4000
5	18	ali	ahmadi	NULL	3000
6	20	ali	mohammadi	20	4000
7	3	Saeed	mofidi	20	6000
8	4	hamid	hamidi	20	5000

	Branch_Code	Branch_Name	Address
1	20	b20	Tehran
2	60	b60	Isf
3	120	b120	Tehran

	customer_number	name	L_name	Branch_Cod	salary	Branch_Code	Branch_Name	Address
1	1	mohammad	mohammadi	20	1000	20	b20	Tehran
2	2	Saeed	mohammadi	60	2000	60	b60	Isf
3	21	arash	mofidi	20	4000	20	b20	Tehran
4	16	ali	navidi	32	4000	NULL	NULL	NULL
5	18	ali	ahmadi	NULL	3000	NULL	NULL	NULL
6	20	ali	mohammadi	20	4000	20	b20	Tehran
7	3	Saeed	mofidi	20	6000	20	b20	Tehran
8	4	hamid	hamidi	20	5000	20	b20	Tehran
9	NULL	NULL	NULL	NULL	NULL	120	b120	Tehran

Same Answers?

```
SELECT *
FROM [Customer_v] t left outer join [Branch_v] b
on t.Branch_Cod = b.Branch_Code;
```

```
SELECT *
FROM [Customer_v] t left outer join [Branch_v] b
 on t.Branch_Cod = b.Branch_Code
 where t.Branch_Cod = b.Branch_Code;
```

```
 SELECT *
FROM [Customer_v] t, [Branch_v] b
 where t.Branch_Cod = b.Branch_Code
```

Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

  select *ID*, *name*, *dept_name*
  from *instructor*

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a **view**.

-
میخوایم این select رو ذخیره کنیم ولی براش نمی خوایم جدولی ایجاد کنیم
دید مناسبی رو برای هر یوزری که از پایگاه داده استفاده میکنه براساس نیازش ایجاد بکنیم = view
مثال: نمیخوایم حقوق استادا رو همه ببینن
توی ویو بحث سطح دسترسی هم مطرح میشه
ینی از اون جدول ستون هایی که میخوایم کاربر ببینه رو انتخاب کن فقط
پس اون ستون هارو توی سلکت انتخاب می کنیم و به این یک اسم می دیم ینی به اون جدولی که درست کردیم به کاربر دسترسی میدیم که ببینه (همون اسمی که گذاشتیم روش) ولی به کاربر دسترسی نمیدیم که جدول اصلی رو ببینه
پس دید مناسب رو برای هر کاربر ایجاد میکنیم
نکته: به صورت دیفالت ویو داده های مارو ذخیره نمیکنه چیزی که ذخیره میشه همون خود اون کوئریس ینی توی این مثال این دو خط فقط ذخیره میشه ولی داده ها نمیشن پس ویو اون کوئری رو ذخیره میکنه نه داده ها بلکه داده ها رو از from... می گیره

View Definition

- A view is defined using the **create view** statement which has the form

  اسم واسه ویو

  create view *v* **as** < query expression >

  where <query expression> is any legal query expression. The view name is represented by *v.*

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- the view relation conceptually contains the tuples in the query result, but it is not precomputed and stored. Instead, the database system stores the query expression associated with the view relation. Whenever the view relation is accessed, its tuples are created by computing the query result. Thus, the view relation is created whenever needed, on demand.

View Definition and Use

- A view of instructors without their salary

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

create view ما فقط یک دستور select میگیره

- Find all instructors in the Biology department

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

- Create a view of department salary totals

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

این ینی ما یک ویو داریم به اسم departments_total_salary که اسم فیلد هاشو میخواد بذاره dept_name , total_salary

--

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to ***depend directly*** on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to ***depend on*** view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2

-
ما میتونیم توی یک ویو از یک یا چند ویوی دیگر هم استفاده بکنیم ینی ما اجازه داریم یک ویویی استفاده بکنیم که بعدش از from اش اسم یک ویوی دیگر امده باشه ینی بعد از from ما می تونستیم یک جدول داشته باشیم یا حالا یک ویو

Views Defined Using Other Views

این ستون course_id رو نشون میده

- **create view** ***physics_fall_2017*** **as**
  select *course.course_id*, *sec_id*, *building*, *room_number*
  from *course*, *section* جوین کرده
  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* این دوتا ستون های ویوی قبلی هستند
  from ***physics_fall_2017***
  where *building*= 'Watson';

اینجا از همون ویوی قبلی استفاده کرده این ینی از ویوی physics_fall_2017 بیا ستون های course_id , room_number رو برگردون با این شرط که building=watson باشه

View Expansion

- Expand the view :

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

- To:

```
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';
```

View Expansion (Cont.)

- A way to define the meaning of views defined in terms of other views.
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:

  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

- As long as the view definitions are not recursive, this loop will terminate

--

Update of a View

- Add a new tuple to *faculty* view which we defined earlier

  insert into *faculty*

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

- This insertion must be represented by the insertion into the *instructor* relation
  - Must have a value for salary.
- Two approaches
  - Reject the insert
  - Inset the tuple

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

    into the *instructor* relation

می خواد این سه مقدار رو توی faculty ذخیره کنه و از اونجایی که اینجا جدول نداریم این به معنی که میخواد بیاد توی جدول اصلی این اطلاعات رو وارد کنه در این حالت چون ما برای فیلد salary هیچ مقداری نداریم توی ویو پس دو حالت رخ میده یا اصلا نباید insert انجام بدیم یا باید توی جدول اصلی null رو بذاریم
که این کار اصلا منطقی نیست!!!

-
کلا سعی کنیم ویو رو اصلا اپدیت نکنیم ینی بصیری پیشنهاد نمی کنه که از اپدیت استفاده بکنیم
کار منطقی نیست بخوایم روی داده های ویو تغییری ایجاد کنیم چون ویو اصلا داده ای خودش نداره و شبیه پوینتره که اشاره میکنه به یک جدول دیگه

View Updates in SQL

- Most SQL implementations allow updates only on simple views
  - The **from** clause has only one database relation.
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
  - Any attribute not listed in the **select** clause can be set to null
  - The query does not have a **group** by or **having** clause.

-
توی شرایط خیلی خیلی خاص ممکنه بتونیم ویو رو اپدیت کنیم و یک insert انجام بدیم و اون شرایط به این صورت است که:

Transactions

- A **transaction** consists of a sequence of query and/or update statements and is a “unit” of work
- The transaction must end with one of the following statements:
  - **Commit work**. The updates performed by the transaction become permanent in the database.
  - **Rollback work**. All the updates performed by the SQL statements in the transaction are undone.
- Atomic transaction
  - either fully executed or rolled back as if it never occurred

اگر یه کاری نصفه انجام شد rollback work داریم ینی برمیگرده خونه اول

وقتی یه کاری کامل انجام شد ته اون کار از دستور commit استفاده میکنیم که بگیم این کار کامل انجام شده و با این کار تمام تغییرات توی پایگاه داده ثبت میشه

-
مهمه!!!
توی پایگاه داده زمانی که میایم کوئری هایی می نویسیم که هدفشون تغییر روی داده هاس مثلا اپدیت و دیلیت یا اضافه کردن
بعضی وقتا ما میخوایم چندتا اپدیت یا چندتا دیلیت و ... که همه اینا با هم دیگه یا انجام بشن به شکل کامل یا اگر یکی انجام شد و سیستم به هر دلیلی متوقف شد ینی نشد که دستور بعدی انجام بشه کار متوقف بمونه و برگردیم به حالت اولیه ینی عملیات اول هم نادیده گرفته بشه مثال همون قضیه بانک اینا ینی می خوایم از سپرده یک 50 تومن برداریم به سپرده 2 اضافه کنیم که اینجا دو تا insert متفاوت داریم ینی وقتی insert اول تمام شد سراغ insert دوم می رویم حالا اگر insert اول انجام بشه و بعدش سیستم کراش بکنه اتفاق بدی می افته پس برمیگردیم به حالت قبل ینی این insert اول رو دیگه نمی بینیم پس با مفهوم transactions این اتفاق رو کنترل میکنیم
ینی زمانی که کار با دیتا پیچیدگی خاصی داره از اینا استفاده میکنیم

Integrity Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  - A checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $4.00 an hour
  - A customer must have a (non-null) phone number

-
توی پایگاه داده معمولا می خوایم یه سری شرایطی رو روی جدول ها اعمال کنیم و روی بعضی از فیلدها کنترل داشته باشیم مثلا این فیلد null نباشه که با اینا میگن constraints ینی یه سری قید هایی که برای یکپارچه سازی داده ها میخوایم روی جداولمون اعمال بشه

Constraints on a Single Relation

روش های مختلفی برای این کار وجود داره:
که این کار قیدهای یکپارچه سازی است
نکته: این شرط هارو موقع ایجاد جدول می ذاریم

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

Not Null Constraints

- **not null**
  - Declare *name* and *budget* to be **not null**

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

اگر موقع ایجاد کردن یک جدول جلوی فیلدش کلمه not null رو بنویسیم
موقعی که جدول ایجاد شد نمیشه توی این جدول برای این فیلد مقداری وارد
نکرد ینی کاربر اجازه این کارو نداره

Unique Constraints

- **unique** (A_1, A_2, …, A_m)
  - The unique specification states that the attributes A_1, A_2, …, A_m form a candidate key.
  - Candidate keys are permitted to be null (in contrast to primary keys).

-
دستور unique به دنبال این است که بیاد یک فیلد یا چندتا فیلد رو با هم دیگه یه کاری کنیم
تکراری نباشه ینی مثلا شماره دانشجویی رکوردی وجود نداشته باشه که این شماره دانشجویی
دوباره تکرار شده باشه
نکته : اگر کلید اصلی نباشه این فیلد اجازه null شدن هم داره ینی ما می تونیم 5 تا رکورد داشته
باشیم که شماره دانشحویی همشون null باشه اینو دیگه جز تکرار حساب نمی کنیم
مثال چندتا فیلد: شماره دانشجویی + اسم فرد این باید یونیک باشه
پس یوینک باعث میشه که ما روی یک یا چندتا فیلد این قید رو بذاریم که این ترکیبه حتما باید توی
کل رکورد ها یونیک باشه

The check clause

- The **check** (P) clause specifies a predicate P that must be satisfied by every tuple in a relation.
- Example: ensure that semester is one of fall, winter, spring or summer

```
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')));
```

این ینی فیلد فصل حتما باید این چهار تا مقدارو بگیره ینی مقدار semester حداقل باید یکی ازاین چهار تا مقدارو به خودش بگیره ینی مقدار فصل نمی تونی چیزی غیر از اینا باشه ینی با توجه با check از اون به بعد اگر خواست چیزی رو توی فیلد semester قرار بده چیزی غیر ازاین چهارتا نمی تونه باشه ینی کاربر غیر از اینا نمی تونه چیزی وارد کنه

```
create table department
    (dept name varchar (20),
     building varchar (15),
     budget numeric (12,2) check (budget > 0),
     primary key (dept name));
```

اینجا یه فیلد گذاشته که مقدار بودجه حتما باید بزرگتر از صفر باشه

-
چک:
ینی ما اجازه داریم وقتی جدول رو می سازیم یه سری شرایط رو چک بکینم که حتما باید فیلدی که ما داریم براش اون چک رو انجام میدیم این شرایط رو داشته باشه

Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If “Biology” is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for “Biology”.
- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a **foreign key** of R if for any values of A appearing in R these values also appear in S.

-
رابطه بین جداول و کلیدها:
اگر بین جدول A و جدول B کلید خارجی تعریف بکنیم یکی از فیلدهای جدول B به عنوان کلید خارجی وصل باشه به کلید اصلی جدول A توی این شرایط اگر حالت اول مقداری میخواد توی جدول B وارد بکنیم حتما این مقدار اول باید توی جدول A وجود داشته باشه و حالت دومش هم این است که اگر بخوایم مقداری توی جدول A پاک بکنیم که داخل جدول B استفاده شده در این دو حالت دیتا بیس به ما اجازه نمیده

Cascading Actions in Referential Integrity

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.
- An alternative, in case of delete or update is to cascade

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

- Instead of cascade we can use :
  - **set null**,
  - **set default**

توی این مثال میگه اگر رکوردی از جدول department حذف بکنیم از جدول کورس هم اون رکورد حذف میشه و همین طور برای اپدیت هم به همین صورت است
غیر از این می تونیم یه کار دیگه هم بکنیم و اون این است که چه دیلیت شد و چه اپدیت شد برای ما مقدار null رو در نظر بگیر --> set null
و کار دیگه هم که می تونیم بکنیم این است که اگر دیلیت شد یا اپدیت شد ما مقدار دیفالتی براش در نظر بگیریم --> set default

-
چاره صفحه قبل چیه؟
ما گاهی اوقات می خوایم استثناهایی رو قائل بشیم و نخوایم قاعده صفحه قبل رو رعایت بکنیم پس پایگاه داده به دستور cascade این اجازه رو به ما میده

Built-in Data Types in SQL

- **date:** Dates, containing a (4 digit) year, month and date
  - Example: **date** '2005-7-27'
- **time:** Time of day, in hours, minutes and seconds.
  - Example: **time** '09:00:30' **time** '09:00:30.75'
- **timestamp:** date plus time of day
  - Example: **timestamp** '2005-7-27 09:00:30.75'

-
انواع دیتا تایپ توی sql:
سه تا تایپ داده ای اینجا گفته که به زمان برمیگرده
data : به صورت سال و ماه و روز ذخیره میشه
time : به صورت ساعت و دقیقه و ثانیه ذخیره مشه
timestamp : این هم date , time رو با هم داره

Index Creation

- Many queries reference only a small proportion of the records in a table.
- It is inefficient for the system to read every record to find a record with particular value
- An **index** on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation.
- We create an index with the **create index** command

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

  اسم واسه ایندکس اسم جدول

روی فیلدی هم که میخواد ایندکس گذاشته بشه توی پرانتز می نویسیم

نکته: روی ستون ایندکس می ذاریم

-
: index
وقتی که جدول رو طراحی کردیم شروع میکنیم به insert کردن داده و به مرور حجم داده داخلش زیاده میشه در نتیجه با این حجم از داده با کندی روبرو میشیم
پس راهکاری برای از بین بردن این کندی این است که:
یکی از مهمترین کارها استفاده از index است ینی با توجه به فیلدی که اومدیم روش ایندکس گذاشتیم پایگاه داده میاد اینارو به صورت درختی در نظر میگیره و برای مقادیر مختلفش از شاخه ها استفاده میکنه مثلا روی جدول دانشجو روی شماره دانشجویی ایندکس گذاشتیم توی این حالت وقتی ک بهش میگیم این اطلاعات دانشجو با این شماره دانشجو رو بهمون نشون بده میاد از ایندکس استفاده میکنه و خیلی سریع می ره سراغ اون رکورده و اطلاعات رو به ما میده
نکته: باز راهکارهای دیگه ای هم وجود داره که سرعت پایگاه داده رو زیاد کنه که اینجا یه مورد رو گفتیم

Index Creation Example

نکته: استفاده از index برای insert باعث میشه سرعت پایگاه داده کندتر بشه
index توی سرچ سرعت بیشتری دارد

- **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*)
- The query:

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

  can be executed by using the index to find the required record, without looking at all records of *student*

نکته: این کوئری خط به خط اجرا میشود واسه همین وقتی به id رسید می دونست که این id یک ایندکس است حالا اگر این id ایندکس نبود دیتا بیس می اومد کل جدول رو می گرفت و روش کار میکرد ولی زمانی که id ایندکس شد اومد این ستون id رو گرفت و روش کار کرد و بعد برگشت به اون جدول اصلی و اون تاپل هایی رو که میخواست برداشت و به ما داد

End of Chapter 4