



دانشگاه صنعتی امیر کبیر  
(پلی تکنیک تهران)

به نام خدا

آزمایشگاه پایگاه داده

جلسه ششم

تراکنش‌ها، اتصال پایتون به پایگاه داده

## تراکنش

▶ تراکنش یک واحد کاری پایگاه داده، متشکل از مجموعه‌ای از فعالیت‌ها است که یا به طور کامل انجام می‌شود و یا اصلاً انجام نمی‌شود و به همین دلیل به آن یک واحد کاری می‌گویند.

▶ تراکنش‌ها دارای ۴ ویژگی هستند که به ویژگی‌های ACID معروف هستند.

▶ Atomicity, consistency , Isolation, Dorability

## یکپارچگی (Atomicity)

- ▶ این خاصیت تضمین می‌کند که یا تمامی عملیات مربوط به تراکنش با موفقیت انجام می‌شود و یا در صورت وجود هرگونه مشکل در اجرای قسمتی از تراکنش، تمامی تراکنش دچار شکست می‌شود کلیه تغییرات انجام شده لغو شده و پایگاه داده به حالت قبل از اجرای تراکنش باز می‌گردد.
- ▶ برای مثال تراکنشی را در نظر بگیرید که در آن مقداری پول از حساب  $A$  کم شده و به حساب  $B$  اضافه می‌شود. در این صورت تراکنش یا به طور کامل انجام می‌شود و یا این که به کلی لغو می‌شود (این امکان وجود ندارد که پول از حساب  $A$  کم شده و به حساب  $B$  اضافه نشود)!

## سازگاری (Consistency)

- ▶ این خاصیت تضمین می‌کند که هر تراکنش، خواه موفق خواه ناموفق، پایگاه داده را در یک وضعیت سازگار قرار می‌دهد یعنی جامعیت داده‌ها حفظ می‌شود و پایگاه داده همواره سازگار با قیودی است که برای آن تعریف شده.
- ▶ به عنوان مثال در بعضی از جداول مقادیر قابل استفاده برای یک خانه از جدول با استفاده از قید **check** تعریف شده‌اند. در این صورت برای تغییر این خانه از جدول باید مطابق با قیودی که از قبل تعریف شده است عمل کنیم.

## انزوا (Isolation)

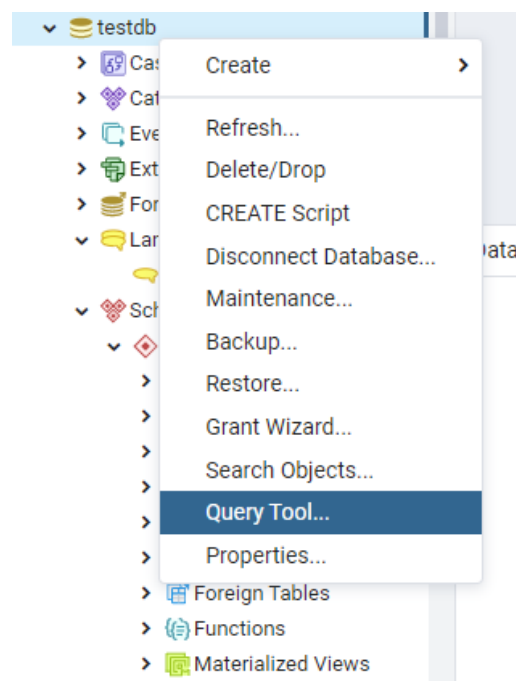
- ▶ هر تراکنش باید کاملاً مستقل و مجزا از سایر تراکنش‌ها عمل کند و مستقل از اینکه سایر تراکنش‌ها چه کارهایی انجام می‌دهند، کار خود را انجام دهد.
- ▶ به بیان دیگر نتیجه‌ی اجرای چند تراکنش بصورت همزمان باید با نتیجه‌ی اجرای پشت سر هم همان تراکنش‌ها برابر باشد
- ▶ به عنوان مثال حالتی را در نظر بگیرید که حساب A دارای مقدار ۵۰۰ دلار موجودی می‌باشد. فرض کنید قرار است مقدار ۴۰۰ دلار از حساب A به حساب B منتقل شود و همچنین ۳۰۰ دلار هم از حساب A به حساب C منتقل شود.
- ▶ در این صورت اگر هر دو تراکنش با هم موجودی حساب A را ۵۰۰ دلار بخوانند شروع به عملیات انتقال کرده و در پایان برای پایگاه داده مشکل پیش خواهد آمد. در صورتی که اگر به صورت متوالی این کار را انجام دهند عملیات انتقال برای تراکنش دوم به علت کمبود موجودی انجام نمی‌شود
- ▶ پایگاه داده از بروز چنین مشکلاتی جلوگیری می‌کند که در ادامه بیشتر با آن آشنا خواهید شد.

## ماندگاری (Durability)

- ▶ نتایج حاصل از اجرای موفق یک تراکنش، باید در سیستم باقی بماند حتی در صورت وقوع خطای سیستم.
- ▶ اغلب سیستم‌های مدیریت بانک اطلاعاتی رابطه‌ای، از طریق ثبت تمام فعالیت‌های تغییر دهنده‌ی داده‌ها در بانک اطلاعاتی، ماندگاری را تضمین می‌کنند.
- ▶ در صورت خرابی، سیستم قادر است آخرین بهنگام سازی موفق را هنگام راه اندازی مجدد، بازیابی کند.

## مثال

- ▶ به عنوان مثال می‌خواهیم انتقال ۱۰۰۰ واحد پول را از حسابی به حساب دیگر با استفاده از تراکنش‌ها انجام دهیم. در واقع هدف این است که از ویژگی‌های گفته شده برای اطمینان از درست انجام شدن فعالیت‌ها استفاده کنیم.
- ▶ برای مشخص شدن نحوه کار تراکنش‌ها لازم است که دو نشست (session) بر روی پایگاه داده داشته باشیم. برای ایجاد یک نشست کافی است بر روی پایگاه داده کلیک راست کرده و گزینه Query Tool را انتخاب کنید.



# مثال

فرض شود جدول accounts بصورت زیر وجود دارد.

Data Output		Explain	Messages	Notifications
	id [PK] integer		name character varying (100)	balance numeric (15,2)
1		1	Bob	10000.00
2		2	Alice	000.00

Data Output		Explain	Messages	Notifications
	id [PK] integer		name character varying (100)	balance numeric (15,2)
1		1	Bob	10000.00
2		2	Alice	10000.00



## مثال

### BEGIN

در نشست اول با استفاده از کلید واژه **BEGIN** یا **BEGIN TRANSACTION** یا **BEGIN WORK** یک تراکنش را شروع می‌کنیم و پس از آن پرس و جویی می‌نویسیم که مقدار موجودی Bob را ۱۰۰۰ واحد کم کند. ►

**Begin work;**

**UPDATE** accounts

**SET** balance = balance - 1000

**WHERE** id = 1;

## مثال

► پس از آن می‌توان در همین نشست پرس و جویی را اجرا کرد که به موجب آن بتوانیم کل جدول accounts را بررسی کنیم. از آن جایی که این پرس و جو در ادامه تراکنش فعلی اجرا شده، می‌توان مشاهده کرد که تغییراتی که انجام داده ایم اعمال شده است.

	Data Output	Explain	Messages	Notifications
	id [PK] integer		name character varying (100)	balance numeric (15,2)
1		2	Alice	10000.00
2		1	Bob	9000.00

► حال بدون پایان دادن این تراکنش در این نشست، به سراغ نشست دیگر می‌رویم و با اجرای پرس و جوی قبلی تغییرات جدول را بررسی می‌کنیم. همانطور که مشخص است هیچ تغییری هنوز صورت نگرفته است.

	Data Output	Explain	Messages	Notifications
	id [PK] integer		name character varying (100)	balance numeric (15,2)
1		1	Bob	10000.00
2		2	Alice	10000.00

## مثال

### COMMIT

- ▶ درواقع تغییراتی که در تراکنش ایجاد شده است، هنوز ثبت نشده و کاربران دیگری که طبیعتاً در نشست های دیگری در حال استفاده از پایگاه داده هستند، نمی توانند این تغییرات را مشاهده کنند.
- ▶ برای ثبت این تغییرات باید از کلید واژه COMMIT استفاده کرد. اما قبل از آن نیاز است که به حساب دیگر (Alice) مقدار ۱۰۰۰ واحد اضافه شود.
- ▶ برای این کار نیز پرس و جوی لازم را در تراکنش فعلی اجرا می کنیم پس از آن هنوز نشست های دیگر تغییرات را مشاهده نمی کنند. برای این کار همانطور که گفته شد از COMMIT استفاده می کنیم تا تغییرات ثبت شود. پرس و جوی زیر یک تراکنش کامل از این انتقال پول است.

## مثال

### COMMIT

```
-- start a transaction
BEGIN;

-- deduct 1000 from account 1
UPDATE accounts
SET balance = balance - 1000
WHERE id = 1;

-- add 1000 to account 2
UPDATE accounts
SET balance = balance + 1000
WHERE id = 2;

-- select the data from accounts
SELECT id, name, balance
FROM accounts;

-- commit the transaction
COMMIT;
```

▶ درواقع پس از اجرای COMMIT پایگاه داده تضمین می‌کند که تغییرات در پایگاه-ذخیره می‌شود. این تضمین بیان کننده خاصیت ماندگاری است.

## مثال

### ROLLBACK

▶ حال فرض شود که هنگام اجرای پرس و جو ها در تراکنش خطایی رخ دهد. به عنوان مثال بجای Alice برای فرد دیگری پول واریز شود. برای برگشت به حالت قبلی می توان از دستور **ROLLBACK** استفاده کرد. به کمک این دستور میتوان در هنگام وقوع خطایی در اجرای تراکنش، به حالت قبل از اجرای تراکنش بازگشت

▶ در واقع این دستور باعث میشود کلیه ی کارهای انجام شده توسط تراکنش از دستور **TRANSACTION BEGIN** فراموش شود (مگر اینکه در بدنه ی تراکنش **POINT SAVE** تعریف شده باشد که بعدتر به آن خواهیم پرداخت) تراکنش زیر نمونه از استفاده از این دستور می باشد.

مثال

ROLLBACK

```
-- begin the transaction
BEGIN;

-- deduct the amount from the account 1
UPDATE accounts
SET balance = balance - 1500
WHERE id = 1;

-- add the amount from the account 3 (instead
  of 2)
UPDATE accounts
SET balance = balance + 1500
WHERE id = 3;

-- roll back the transaction
ROLLBACK;
```

## مثال

### SAVEPOINT

با استفاده از این دستورات میتوان نقاط بازگشتی را تعریف کرد که با استفاده از دستور **ROLLBACK** به جای بازگشت به ابتدای تراکنش، به این نقاط برگشت. ساختار این دستور بصورت زیر است.

► **SAVEPOINT savepoint\_name**

و برای بازگشت به هر نقطه بازگشتی با استفاده از **ROLLBACK** بصورت زیر می توان عمل کرد.

► **ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint\_name**

نکته: پس از اجرای دستور **ROLLBACK** همه نقاط بازگشتی که بعد از نقطه ای که قرار است به آن بازگشت کند، حذف می شود و باید دوباره آن نقاط را ایجاد کرد.

## قفل گذاری و همزمانی

- ▶ همزمانی یکی از مسائل مهم در بحث پایگاه داده‌ها است و بیانگر این مفهوم است که دو یا چند کاربر (یا تراکنش) می‌خواهند با یک موجودیت تعامل کنند.
- ▶ البته ماهیت تعامل‌ها می‌تواند متفاوت باشد (read, delete, update insert, ) و انتخاب نوع قفل گذاری و برقراری همزمانی هم به ماهیت تعامل کاربران (یا تراکنش‌ها) بستگی دارد.
- ▶ قفل گذاری هم از مسائل مهم در بحث پایگاه داده‌ای رابطه‌ای است و با جلوگیری از وقوع عملیات همزمان روی داده، موجب تضمین جامعیت داده‌ها می‌شود.
- ▶ قفل‌ها معمولاً بصورت پویا و توسط manager lock (بخشی از موتور پایگاه‌داده) کنترل و مدیریت می‌شوند.



# قفل گذاری و همزمانی

▶ بطور کلی به کمک قفل ها میتوان از بروز مشکلات زیر جلوگیری کرد:

## Dirty Read ▶

▶ این مشکل زمانی به وقوع می پیوندد که یک تراکنش دادهای را بخواند که تراکنش دیگری که هنوز `commit` نشده، آن داده را تغییر داده باشد. در این صورت اگر تراکنشی که داده را تغییر داده در ادامه مجددا داده را تغییر دهد یا به جای آنکه `commit` شود، `rollback` شود، تراکنش اول داده را اشتباه میخواند.

## Non-repeatable reads ▶

▶ این مشکل زمانی به وقوع می پیوندد که یک تراکنش دادهای را دو مرتبه بخواند و در بین این دو مرتبه یک تراکنش دیگر داده را تغییر دهد در این صورت خروجی تراکنشی که دو مرتبه یک داده را با دو مقدار متفاوت خوانده، نامعتبر خواهد بود.

# قفل گذاری و همزمانی

▶ بطور کلی به کمک قفل‌ها میتوان از بروز مشکلات زیر جلوگیری کرد:

## Phantoms ▶

▶ Phantom به رکوردهایی گفته میشود که به طور غیرمنتظره ای در نتایج ظاهر می‌شوند و این غیرمنتظره بودن ناشی از آن است که مثلاً برخلاف انتظار ما تحت تأثیر یک update یا delete قرار نگرفته‌اند. مثال حالتی را در نظر بگیرید که یک تراکنش همه ی رکوردهای یک جدول را پاک(delete) کند و در همین حین تراکنش دیگری چند رکورد به آن اضافه کند. در این صورت انتظار ما حذف همه ی رکوردها بوده در صورتی که هنوز چندین رکورد در جدول باقی مانده است.

## قفل گذاری و همزمانی

► با توجه به مشکلاتی که مطرح شد می توان راه حل های مختلفی برای این مشکلات بیان کرد. یکی از راه حل ها استفاده از قفل ها می باشد. قفل ها را می توان بر حسب نیاز روی منابع مختلف (پایگاه داده، جدول، سطر و ...) قرار داد و همچنین میتوان از انواع قفل ها استفاده کرد که مهمترین آنها را در ادامه معرفی می شود.

### ► قفل اشتراکی

► از این قفل ها هنگام انجام عملیات خواندن استفاده می کنیم تا حین انجام عملیات خواندن، داده ها تغییر نکنند. قفل های اشتراکی با یکدیگر سازگار هستند و بنابراین میتوان از چندین قفل اشتراکی بصورت همزمان استفاده کرد.

# قفل گذاری و همزمانی

## ▶ قفل انحصاری

▶ همانطور که از اسمشان مشخص است این قفل‌ها وقتی روی یک منبع قرار می‌گیرند، آن منبع را منحصراً در اختیار یک کاربر (تراکنش) قرار می‌دهند و با هیچ نوع قفل دیگری سازگاری ندارند و نمی‌توانند همزمان با قفل دیگری فعال شوند. با وجود این قفل‌ها هیچ دو کاربری نمی‌توانند همزمان عملیات update, delete یا هر عملیات دیگری را روی منبعی که این قفل را دارد انجام دهند.

## ▶ قفل به روزرسانی

▶ میتوان عملیات به‌روزرسانی را متشکل از دو مرحله دانست. مرحله اول، جستجو توسط عبارت **where** برای یافتن داده‌هایی که باید به روز شود. و مرحله دوم، انجام عملیات به روزرسانی روی داده یافته شده در مرحله اول. میتوان گفت قفل-های به روزرسانی در مرحله اول به روزرسانی از اشتراکی هستند و در مرحله دوم به روزرسانی به قفل انحصاری تبدیل میشوند. و این از بروز بن‌بست جلوگیری می‌کند.

## بن بست

- ▶ بن بست زمانی رخ می دهد که دو تراکنش هرکدام یک منبع در اختیار دارند و آن را قفل کرده اند و هر یک منتظر دیگری هستند تا منبعی که در دست آن تراکنش است را بگیرند و قفل کنند بنابراین این دو تراکنش باید تا بی نهایت منتظر یکدیگر باشند و هیچ یک نمی توانند کار خود را به اتمام برسانند.
- ▶ به طور معمول در پایگاه داده تراکنش ها مدت تعریف شده ای برای آزاد شدن یک منبع و استفاده از آن منتظر می مانند مگر آنکه پایگاه داده تشخیص وقوع بن بست دهد. در این صورت بر اساس تخمینی از هزینه ی rollback کردن هر تراکنش، یکی از تراکنش ها را به عنوان قربانی انتخاب کرده و آن را rollback میکند تا تراکنش دیگر بتواند کار خود را ادامه دهد.

## سطوح انزوا

- ▶ در قسمت قبل توضیح داده شد که در نبود قفل ها چه اتفاقی رخ می دهد و همچنین قفل ها را به عنوان یک راه حل توانستیم در نظر بگیریم.
- ▶ با این حال در پایگاه داده هنگام اجرای یک تراکنش بر روی یک جدول، تراکنش دیگری نمی تواند به آن داده ها دسترسی داشته باشد، لذا می توان گفت در شرایطی وقوع بن بست اجتناب ناپذیر است. اما با استفاده از سطوح انزوا می توان این موضوع را تا حدی کنترل کرد.
- ▶ برخی از سطوح دسترسی در اینجا بحث می شود.

### **Read Uncommitted** ▶

- ▶ این سطح انزوا اجازه ی خواندن رکوردهایی که توسط یک تراکنش Commit نشده، تغییر کرده اند را می دهد. بنابراین درگیری تراکنش ها را بسیار کم می کند اما ممکن است `repeatable-non, phantoms, read dirty reads` رخ دهند.

# سطوح انزوا

## Read Committed ►

► این سطح انزوا، سطح پیشفرض برای تمام تراکنش‌ها در پایگاه داده است و اجازه‌ی خواندن داده‌هایی که در تراکنش دیگری تغییر کرده‌اند را فقط به شرطی می‌دهد که آن تراکنش commit شده باشد. بنابراین read dirty رخ نخواهد داد اما ممکن است reads repeatable-non, phantoms رخ دهند.

## Repeatable Read ►

► این سطح انزوا تضمین می‌کند که هر رکوردی که در تراکنش خوانده شود، بعداً هم مجدداً می‌تواند خوانده شود و اجازه‌ی تغییر یا حذف رکوردهای خوانده شده را نمی‌دهد. بنابراین قفل‌های اشتراکی را تا پایان تراکنش نگه می‌دارد.

# سطوح انزوا

## Serializable ►

► این سطح از انزوا که بیشترین درگیری را بین تراکنش ها بوجود می آورد اجازه ی تغییر در داده ای را که توسط یک تراکنش Commit نشده، خوانده شده است، نمی دهد تا زمانی که آن تراکنش خاتمه پذیرد و مثل آن است که تراکنش های وابسته به یک منبع مشترک کاملاً پشت سرهم (و نه بصورت همزمان) اجرا می شوند.

► برای اعمال هرکدام از سطوح انزوا بر روی یک تراکنش می توان از دستور زیر استفاده کرد.

► **SET SESSION CHARACTERISTICS AS TRANSACTION**  
**ISOLATION LEVEL** transaction\_mode



# اتصال به پایگاه داده از طریق Python

► برای اتصال به پایگاه داده می‌توان از پکیج‌های مختلفی استفاده کرد. تعدادی از این پکیج‌ها در زیر لیست شده‌اند. در این قسمت از psycopg2 استفاده می‌شود.

Psycopg2 ►

pg8000 ►

py-postgresql ►

PyGreSQL ►

ocpgdb ►

bpgsql ►

► برای مشاهده توضیحات کامل مستندات این پکیج می‌توان از لینک زیر استفاده کرد.

► <https://www.psycopg.org/>

# اتصال به پایگاه داده از طریق Python

► پس از نصب این پکیج می‌توان با استفاده از تابع `connect` به پایگاه داده مورد نظر متصل شد. برای اتصال نیاز است که آدرس پایگاه داده، نام پایگاه داده، نام کاربری و رمز ورود برای کاربر وجود داشته باشد.

```
conn = psycopg2.connect(  
    host="localhost",  
    database="databasename",  
    user="username",  
    password="pass")
```

## اتصال به پایگاه داده از طریق Python

► در مرحله بعدی با استفاده از کانکشن ایجاد شده می‌توان یک `cursor` ساخت و با استفاده از آن پرس و جو هارا با استفاده از `execute()` اجرا کرد. هر کانکشن به منزله یک نشست می‌باشد. و پس از اجرای پرس و جو ها حتما باید کانکشن را `commit()` کرد تا تغییرات اعمال شود. همچنین می‌توان `rollback` انجام داد.

► برای دریافت اطلاعات، بطور مثال پرسو جوی `select` باید پس از اجرای پرس و جو از تابع `fetchone()` استفاده کرد. کد زیر نمونه استفاده از این پکیج می‌باشد.

# اتصال به پایگاه داده از طریق Python

```
import psycopg2

# Connect to an existing database
conn = psycopg2.connect("dbname=test user=postgres")

# Open a cursor to perform database operations
cur = conn.cursor()

# Execute a command: this creates a new table
cur.execute(
    "CREATE TABLE test (id serial PRIMARY KEY, num integer, data varchar);")

# Pass data to fill a query placeholders and let Psycopg perform
# the correct conversion (no more SQL injections!)
cur.execute("INSERT INTO test (num, data) VALUES (%s, %s)",
            ...      (100, "abc'def"))

# Query the database and obtain data as Python objects
cur.execute("SELECT * FROM test;")
cur.fetchone()
(1, 100, "abc'def")

# Make the changes to the database persistent
conn.commit()

# Close communication with the database
cur.close()
conn.close()
```

## تمرین

۱. ▶ تراکنشی بنویسید که درون یک جدول دلخواه ۵ رکورد وارد کند. فرض شود رکورد های زوج با خطا مواجه شده است لذا نباید اضافه شود. این کار را باید با استفاده از rollback و savepoint ها انجام دهید. خروجی در نهایت مطابق شکل زیر باید باشد.

۱ ▶

۳ ▶

۵ ▶

۲. ▶ با یک مثال نشان دهید که قفل انحصاری با قفل اشتراکی سازگاری ندارد.

۳. ▶ دو تراکنش بنویسید که منجر به بن بست شود.

۴. ▶ یک سناریو برای Nonrepetable read طراحی کنید.

۵. ▶ با استفاده از زبان برنامه نویسی پایتون، کدی بنویسید که به پایگاه داده متصل شود و پس از ساختن یک جدول اطلاعاتی در آن اضافه کند و همان اطلاعات را در نشست یا جلسه ای دیگر بخواند و نمایش دهد.