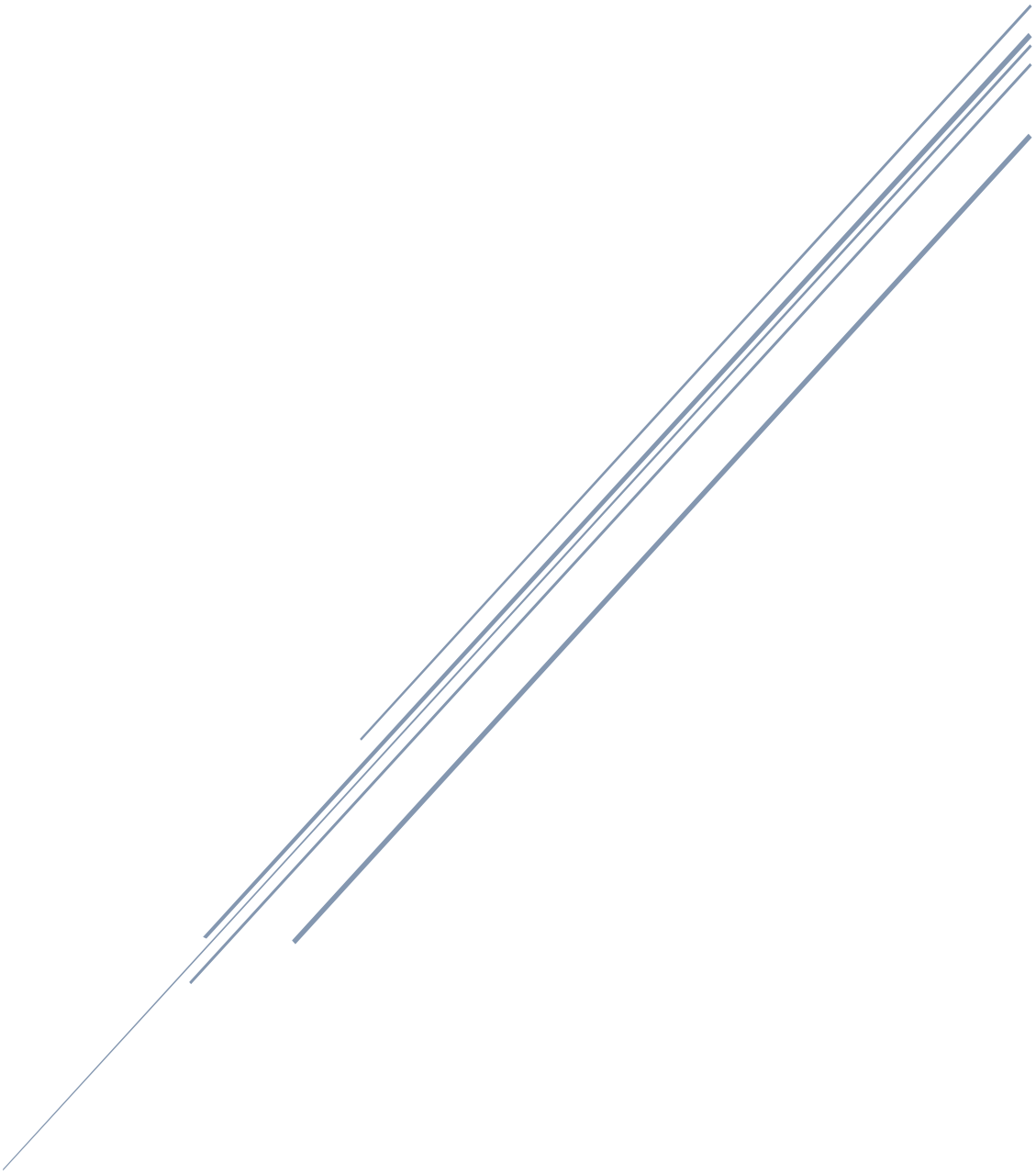
PRİVATE SECURİTY

Python ile SQLite3 Veri Tabanı Programlama



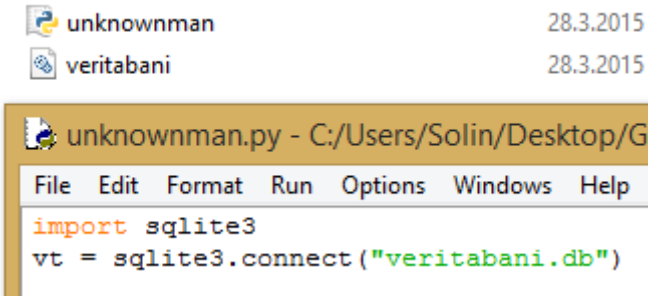
Kara Ayaz
www.privatesec.org

Bölüm 1 – Temel Kullanım ve Veri Tabanı Oluşturma

SQLite, diğer modüller gibi programa "*import*" komutu ile dâhil edilir. "*import sqlite3*" dedikten sonra versiyonu öğrenmek için "*sqlite3.version*" demeniz yeterlidir.

```
Type "copyright", "credits" or
>>> import sqlite3
>>> sqlite3.version
'2.6.0'
>>>
```

Bu çıktı bizim "2.6" sürümünü kullandığımızı gösteriyor. Veri tabanı oluşturmak için *connect* fonksiyonuna ihtiyacımız var. Kullanımı şu şekildedir:

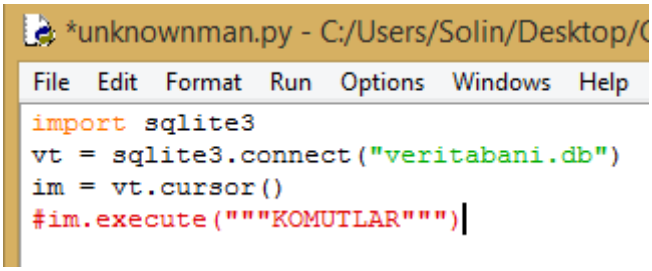


Bu kodu çalıştırdığımızda, programın kaydedildiği dizinde "*veritabani.db*" diye bir dosya oluşacaktır. Veri tabanı bir değişkene aktarılmalıdır.

- Connect fonksiyonuna tırnak "*C:\\privsec*" yazarsanız, veri tabanı C diskinde ki pivsec klasöründe açılacaktır.
- Eğer RAM üzerinde geçici bir veri tabanı oluşturulmak istenirse tırnak içerisinde "*:memory:*" yazılmalıdır. RAM üzerinde ki veri tabanları program kapandığı an silinir. Deneme yapılacaksa bu db üzerinde çalışması önerilir.
- Eğer belirtilen isimde bir veri tabanı mevcut ise o veri tabanına bağlanılır. Yeni bir veri tabanı oluşturulmaz.
- Verilen ismin sonuna muhakkak *.db* eklenmelidir.

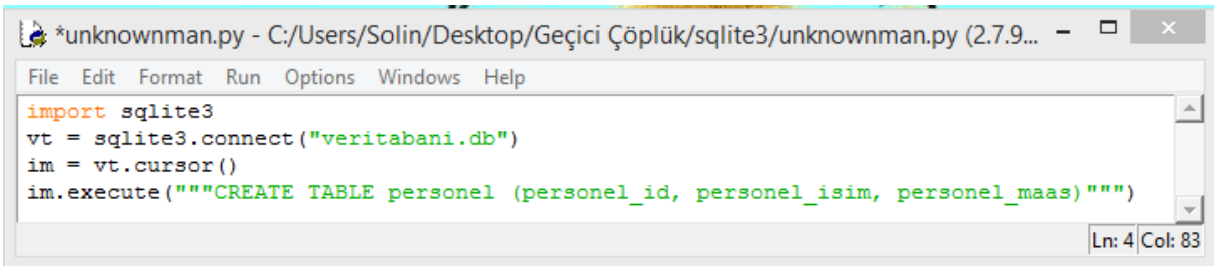
Bölüm 2 – İmleç, Tablo ve Kolon Oluşturma

SQLite üzerinde işlem yapabilmek için bir imlece (*cursor*) ihtiyacınız vardır. Execute fonksiyonu dâhil olmak üzere hemen hemen her işlemde imlecimizi kullanıyoruz.



Bu andan itibaren execute (çalıştır/icra et) diyerek sql komutlarımızı girmeye başlayabiliriz. Ancak dikkat ettiyseniz 3 tırnak kullandım. Komutları 3 tırnak içerisine yazarsanız sizin için daha rahat olacaktır, veri girerken karakter dizilerini (*string*) tırnak içerisinde girmemiz gerekiyor çünkü.

Bize şu anda lazım olan bir tablo. Tablomuzu oluşturmak için "*CREATE TABLE tablo_ismi*" dememiz yeterli.

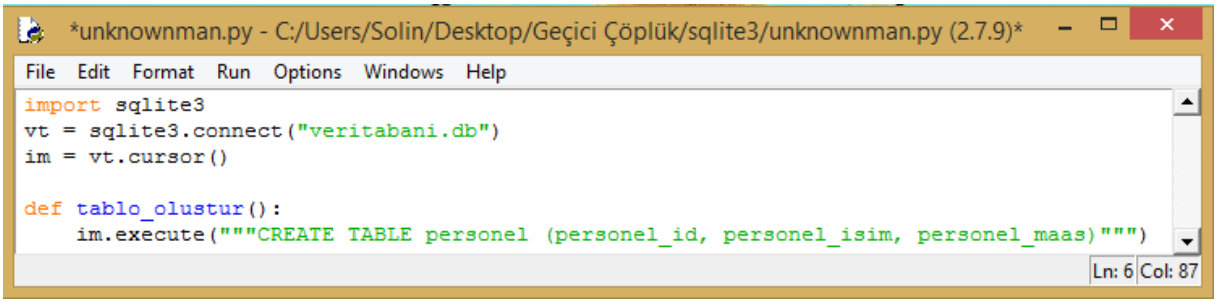


Bu komutu çalıştırdığımızda konsol bir alt satıra inecektir. Biz burada tablo_ismi (personel) dedikten sonra parantez içerisinde kolon yani sütun isimlerini yazdık. Böylece kolonlarımızı da oluşturmuş olduk.

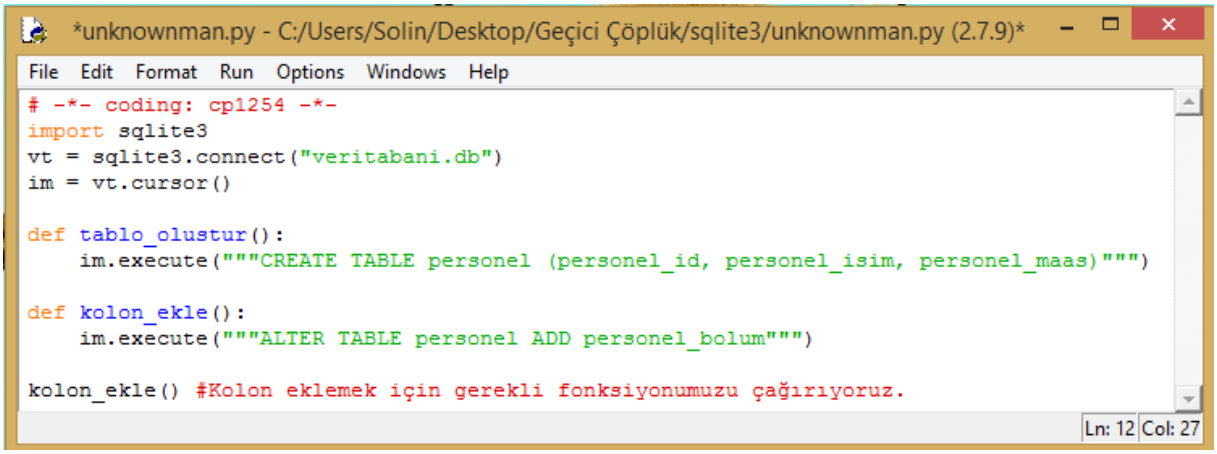
Ancak 2. Çalıştırdığımızda bir hata alacağız.

```
Traceback (most recent call last):
  File "C:/Users/Solin/Desktop/Geçici Çöplük/sqlite3/unknownman.py", line 4, in
<module>
    im.execute("""CREATE TABLE personel (personel_id, personel_isim, personel_ma
as)""")
OperationalError: table personel already exists
```

Burada diyor ki "**Zaten personel diye bir tablo var, neden ikinci kez oluşturmaya çalışıyorsun?**". Bu yüzden bu satırı fonksiyon içerisine almak en mantıklısı olacaktır. Aslında daha rahat çalışmamız için birçok komutumuzu fonksiyonlar içine alacağız.



Evet, artık hata almadan tablolarımızı ve kolonlarımızı ekleyebiliyoruz. Ancak bir problem var. Biz id, isim, maaş ve bölüm kolonları olsun istemiştik ama bölüm kolonunu unuttuk? Yeni bir kolon eklemek için "*ALTER TABLE*" komutuna ihtiyacımız var.

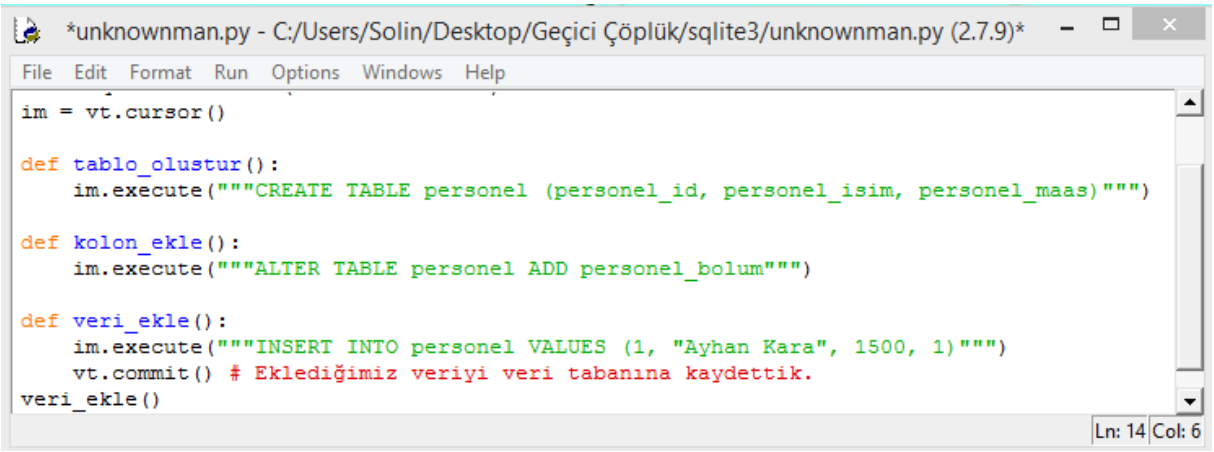


"*ALTER TABLE tablo_ismi ADD kolon ismi*" komutunu bir fonksiyona kaydediyoruz ve fonksiyonumuzu çağırıyoruz. Programı 1 kez çalıştırdıktan sonra 12. Satırda ki kolon_ekle() kısmını silebilirsiniz.

Bölüm 3 – Veri Tabanına Veri Ekleme ve Okuma

Veri Ekleme

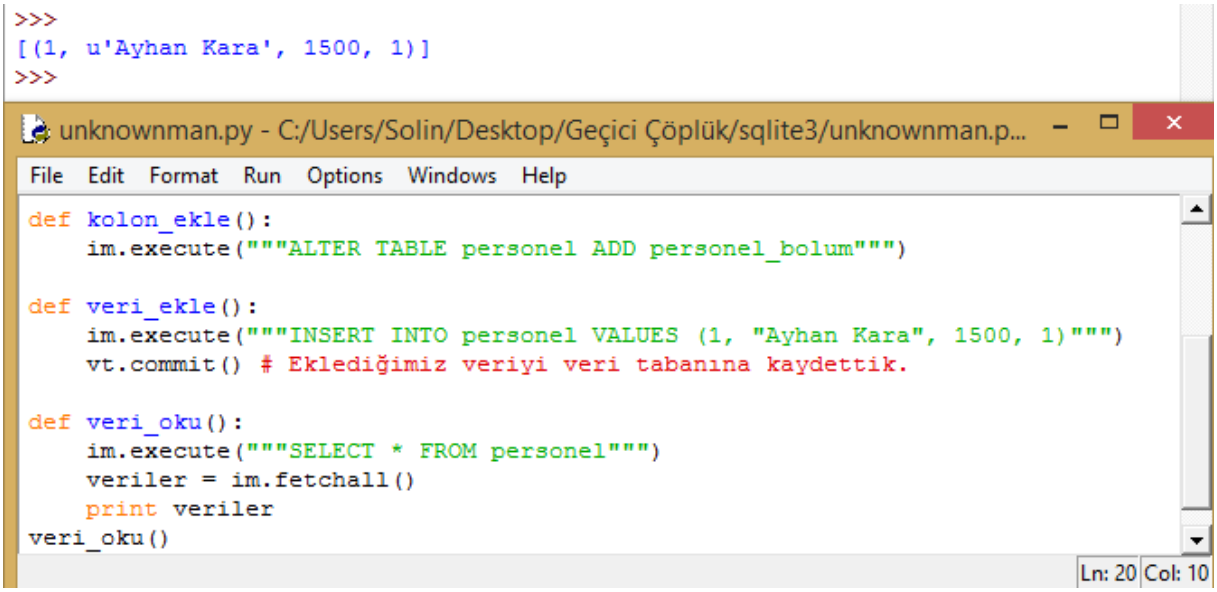
Diğer SQL dillerinde olduğu gibi veri tabanına veri ekleyebilmek *INSERT INTO / VALUES* komutunu kullanıyoruz.



Dikkat edilecek bazı hususlar var, sayıları tırnak içerisine almadık. Şayet onları daha sonra işlemler yapmak için kullanacağız. Bununla beraber ismi tırnak içerisine aldık. Neden tek tırnak kullandığımızı anlamışsınızdır. Bununla beraber *vt.commit()* fonksiyonu da gözümüzden kaçmamış olsa gerek. Bu fonksiyon da *execute* ile aldığı veriyi veri tabanına işliyor.

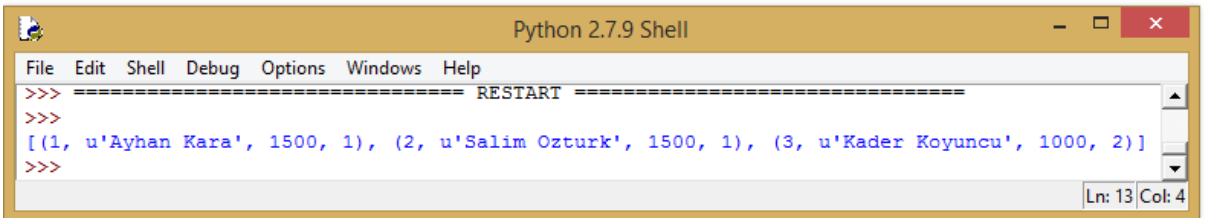
Veri Okuma

Gelin şimdi veri tabanından veri okuyalım. Okuma işlemi için "*SELECT * FROM tablo*" dememiz yeterli. Burada ki yıldız (*) tüm kolonları al demek. Yani siz komutu personel tablosuna göre "*SELECT personel_id, personel_isim, personel_maas, personel_bolum FROM personel*" olarak ta yazabilirsiniz. Yada sadece isim ve maaş bilgilerini okumak için "*SELECT personel_isim, personel_maas FROM personel*" demeniz de yeterlidir.



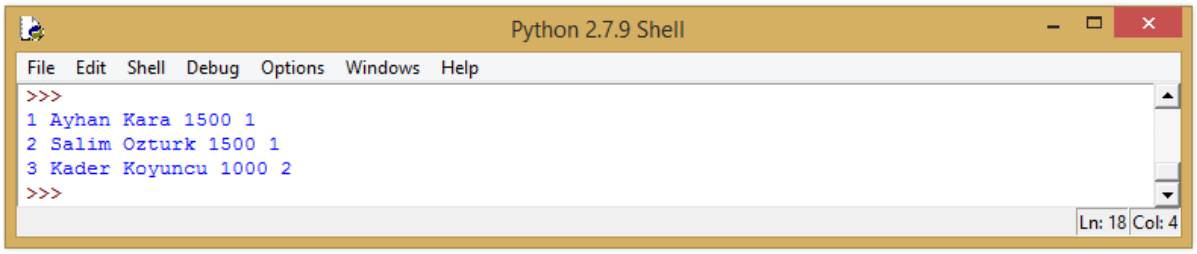
Gördüğünüz gibi kodu çalıştırdığımızda bize bir listenin içerisinde demek halinde sundu listeyi. Bununla birlikte belirtmeliyim ki tek başına **select * from** yeterli olmayacaktır. *İm.fetchall()* fonksiyonunu kullanarak verileri ön belleğe alıp daha sonra **print** komutu ile ekrana bastırmalısınız. Bir isim daha ekleyelim (veri_ekle fonksiyonunu 2 kere farklı isim ve bilgiler ile çalıştıracağım.) Daha sonra da gelen verileri düzenli bir biçimde sizlere sunacağım.

```
def veri_ekle():
    im.execute("""INSERT INTO personel VALUES (2, "Salim Ozturk", 1500, 1)""")
    im.execute("""INSERT INTO personel VALUES (3, "Kader Koyuncu", 1000, 2)""")
    vt.commit()

def veri_oku():
    im.execute("""SELECT * FROM personel""")
    veriler = im.fetchall()
    print veriler
veri_ekle()
veri_oku()
```



Gördüğünüz gibi elimizde bir liste var ve bu liste demetler halinde bize sunulmuş. Bunları düzenli bir biçimde almak için for döngüsüne ihtiyaç duymaktayız.

```
def veri_oku():
    im.execute("""SELECT * FROM personel""")
    veriler = im.fetchall()
    for veri in veriler:
        p_id, p_isim, p_maas, p_bolum = veri
        print p_id, p_isim, p_maas, p_bolum

veri_oku()
```



Gördüğünüz gibi öncelikle for döngüsü ile listeden çıkarttım daha sonra demet içerisinde ki bilgileri değişkenlere aktararak kullandım.

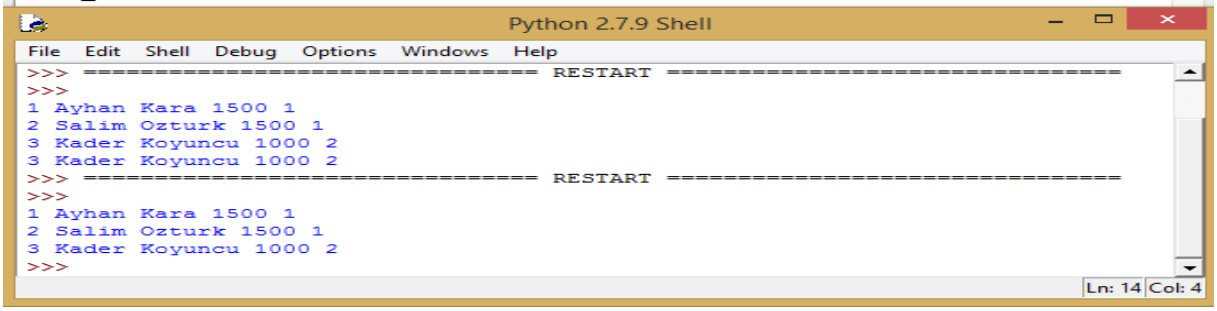
Bölüm 4 – Listeleme Seçenekleri

Tekrarlı Satırları Ortadan Kaldırma (DISTINCT)

Yazdığınız veri tabanında çoğu satırlar tekrar unsuru oluşturabilir. Örneğin bir kullanıcıyı 2 kez kaydettiniz, onu silmek yerine tekrar eden satırı gizlemek istiyorsunuz. İşte tam bu sırada DISTINC sözcüğü komuta eklenir. Kullanımı "*SELECT DISTINCT * FROM tablo_adi*" şeklindedir.

Aşağıda ki şekilde ilk sıralamada DISTINCT kullanılmadan yapılan listeleme var. İkinci sıralamada ise DISTINCT kullanılmış listeleme var. Görüldüğü gibi tekrarlı satırlar ortadan kaldırılmış.

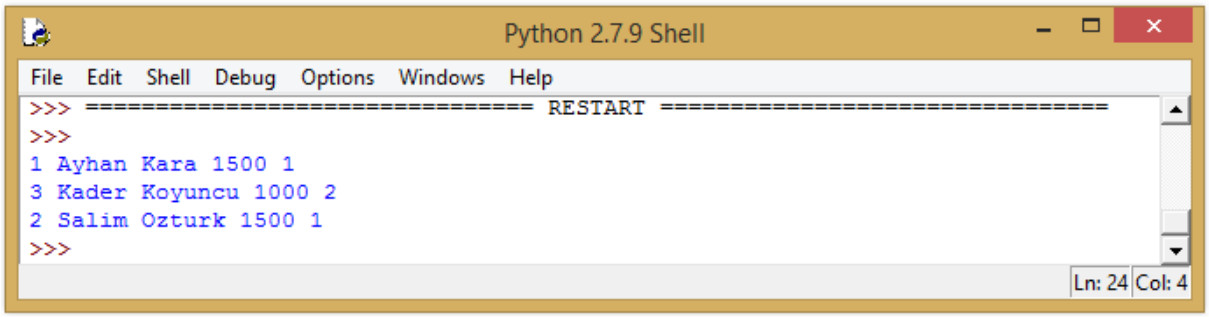
```
def veri_oku():
    im.execute("""SELECT DISTINCT * FROM personel""")
    veriler = im.fetchall()
    for veri in veriler:
        p_id, p_isim, p_maas, p_bolum = veri
        print p_id, p_isim, p_maas, p_bolum
veri_oku()
```



ORDER BY İle Listeleme Türü

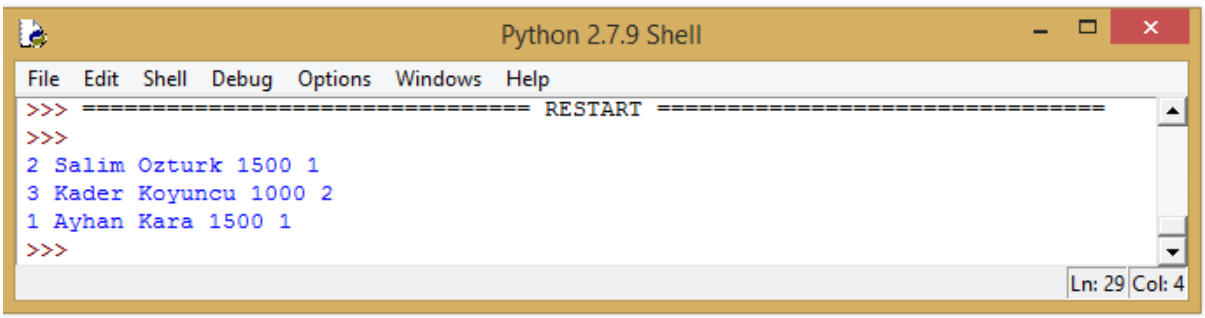
SQLite de temel olarak 2 listeleme türü var. Biri Azdan çoğa [A-Z/0-9(ASC)] diğeri ise çoktan aza [Z-A/9-0(DESC)]. Örnekler üzerinden bunları açıklamak daha mantıklı olacaktır.

```
def veri_oku():
    im.execute("""SELECT DISTINCT * FROM personel ORDER BY personel_isim ASC""")
    veriler = im.fetchall()
    for veri in veriler:
        p_id, p_isim, p_maas, p_bolum = veri
        print p_id, p_isim, p_maas, p_bolum
veri_oku()
```



Mantık basit, "*SELECT * FROM tablo_ismi ORDER BY esas_kolon ASC/DESC*" komutunu vererek istediğimiz şekilde listelememizi yapıyoruz. Yukarıda ki örnekte personel tablosunu personel ismine göre sıralayarak (*personel_isim*) isimleri alfabetik olarak dizdik. Altta ki örnek üzerinde ise tam tersi bir listeleme mevcut.

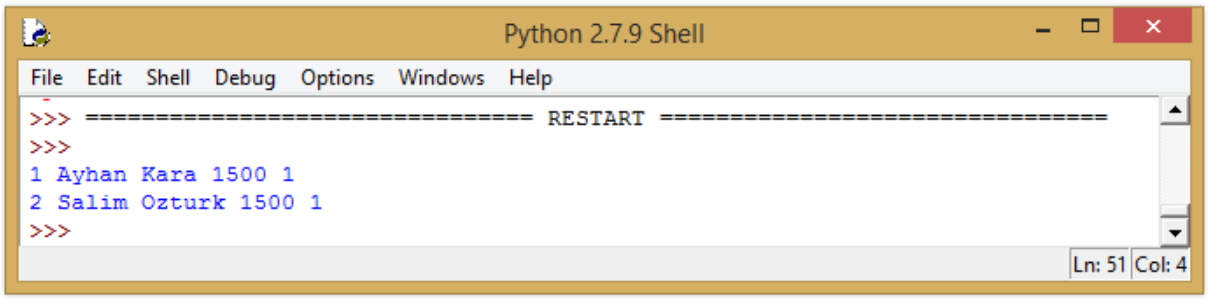
```
def veri_oku():
    im.execute("""SELECT DISTINCT * FROM personel ORDER BY personel_isim DESC""")
    veriler = im.fetchall()
    for veri in veriler:
        p_id, p_isim, p_maas, p_bolum = veri
        print p_id, p_isim, p_maas, p_bolum
veri_oku()
```



Koşula Bağlı Listeleme (WHERE)

Diyelim ki maaşı 1000 TL nin üzerinde olan kullanıcıları görüntülemek istiyorsunuz. Bu defa devreye koşullar girmekte.

```
def veri_oku():
    im.execute("""SELECT DISTINCT * FROM personel WHERE personel_maas > 1000""")
    veriler = im.fetchall()
    for veri in veriler:
        p_id, p_isim, p_maas, p_bolum = veri
        print p_id, p_isim, p_maas, p_bolum
veri_oku()
```



"personel_maas > 1000" diyerek "*personel tablosunda maaşı 1000 den büyük kullanıcıları göster*" dedik. Tüm tabloyu şu şekilde ifade edebiliriz.

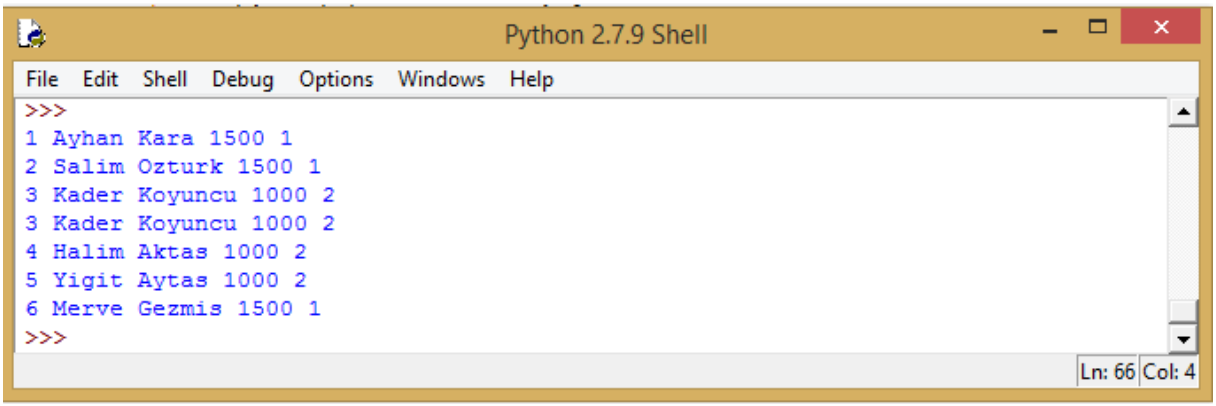
Eşittir	==	Verilen değere eşit değerler
Büyüktür	>	Verilen değerden büyük değerler (Verilen değer hariç)
Küçüktür	<	Verilen değerden küçük değerler (Verilen değer hariç)
Eşit Değildir	!=	Verilen değer dışında ki tüm değerler
Büyük Eşittir	>=	Verilen değerden büyük değerler (Verilen değer dâhil)
Küçük Eşittir	<=	Verilen değerden küçük değerler (Verilen değeler dâhil)

=> SELECT * FROM personel ORDER BY personel_id DESC / ID ye göre büyükten küçüğe doğru sıralar.
=> SELECT * FROM personel ORDER BY personel_id ASC / ID ye göre küçükten büyüğe doğru sıralar
=> SELECT * FROM personel WHERE personel_id < 3 ORDER BY personel_id DESC / id si 3 ten küçük kullanıcıları büyükten küçüğe doğru sıralar.
=> SELECT * FROM personel ORDER BY personel_isim DESC / Personel ismine Z den A ya doğru sıralar

Bölüm 5 – Veri Güncelleme ve Silme

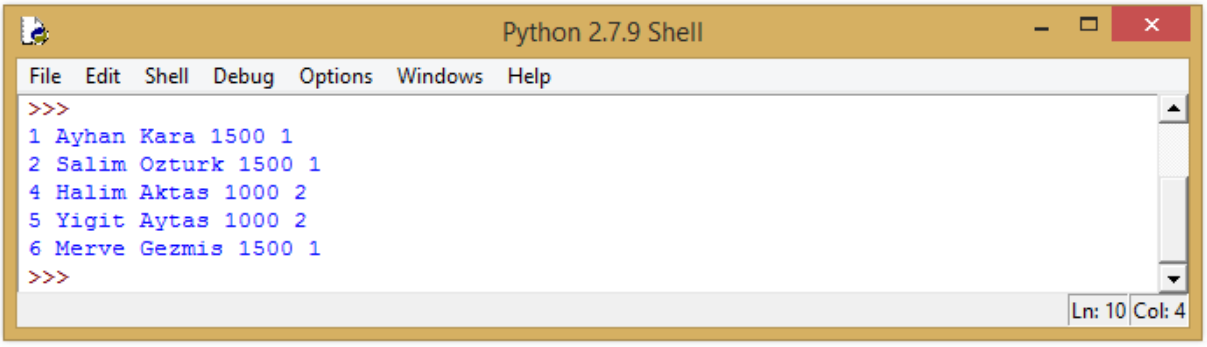
Veri Silme (DELETE FROM)

Eğer hatırlarsanız Kader Koyuncu adlı arkadaşı 2 kez eklemiştik. Artık onunla işimiz bittiğine göre kullanıcıyı veri tabanımızdan silebiliriz. Tablomuzun son hali bu şekilde:



Şimdi kader koyuncuyu yani ID numarası 3 olan kullanıcıları silebiliriz.

```
def veri_sil():
    im.execute("""DELETE FROM personel WHERE personel_id == 3""")
    vt.commit()
veri_sil()
veri_oku()
```



Gördüğünüz gibi "*DELETE FROM tablo_ismi WHERE koşul*" diyerek koşulu sağlayan verileri sildik. Aşağıda ki örnekleri inceleyin.

DELETE FROM personel WHERE personel_bolum == 2 // Bölüm no 2 olan kullanıcıları sil
DELETE FROM personel WHERE personel_maas < 1000 // Personel maaşı 1000den küçükleri sil.
DELETE FROM personel WHERE personel_id >= 5 // Personel no 5 ve büyük olanları sil.
DELETE FROM personel // Personel tablosunda ki tüm verileri sil.

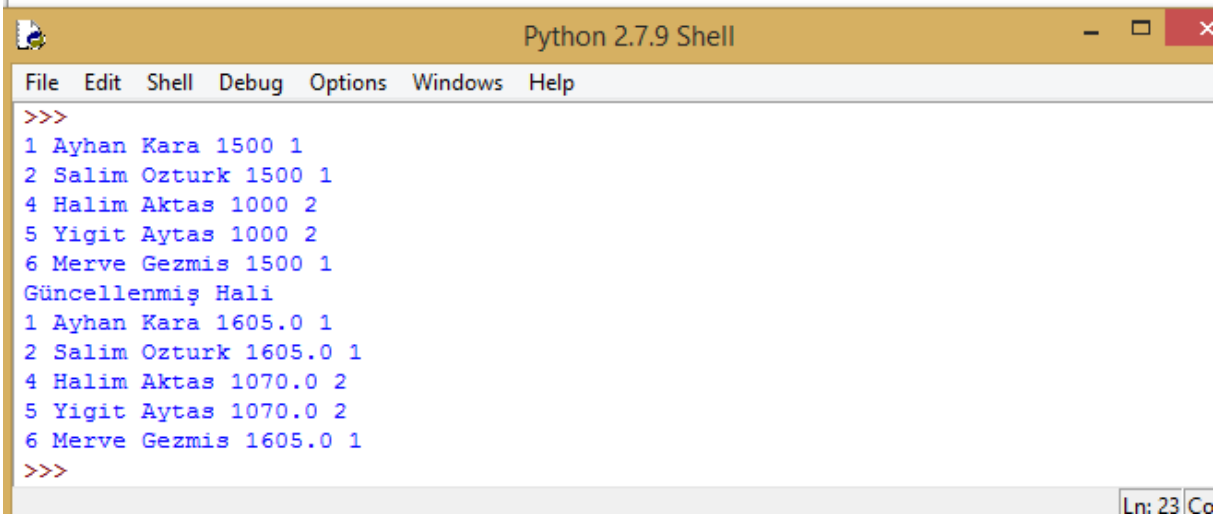
VERİ Güncelleme (UPDATE SET)

Evet patronun iyi gününde, tüm personelin maaşına %7 zam yaptı. Komutumuz;

UPDATE tablo_adi SET kolon=yeni_deger

Burada tablo adı yazan yere tablomuzun adını, kolon yazan yere güncelleme yapılacak kolonu ve yeni_deger yazan yere de yeni güncellememizi yazıyoruz.

```
def veri_guncelle():
    im.execute("""UPDATE personel SET personel_maas = personel_maas*1.07""")
veri_oku()
print "Güncellenmiş Hali"
veri_guncelle()
veri_oku()
```



Görüldüğü gibi tüm kullanıcıların maaşına %7 lik bir zam yapılmış oldu. Aşağıda ki kodları inceleyin. Tabi ki vt.commit() yazmayı unutmuyoruz.

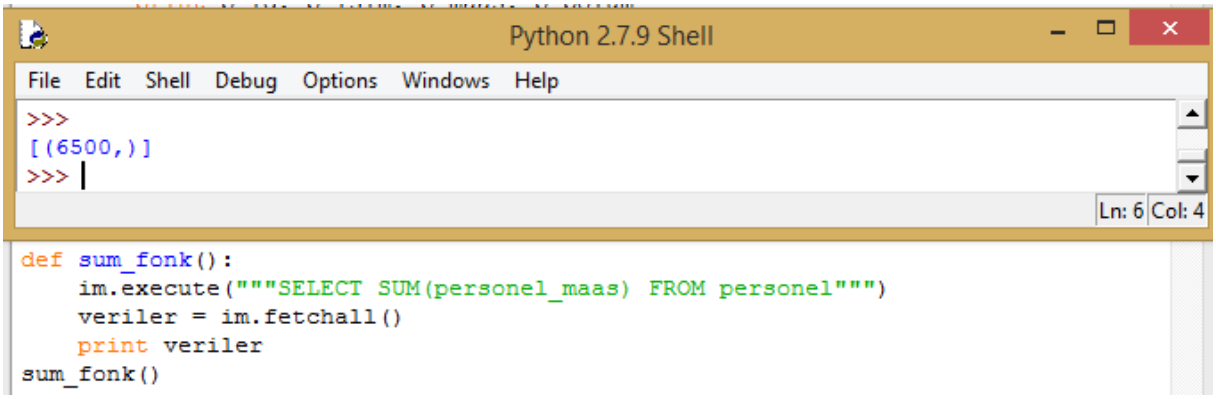
```
UPDATE personel SET personel_maas = personel_maas*1.32 WHERE personel_bolum == 1
//1. Bölümde ki tüm personelin maaşına %32 zam yap
UPDATE personel SET personel_bolum = 1 WHERE personel_id = 8
//ID numarası 8 olan kullanıcının bölüm nosunu 1 olarak güncelle.
```

Bölüm 6 – Kümeleme Fonksiyonları

SQL, tablo içerisinde çeşitli matematiksel işlemler yapan bir takım fonksiyonlar mevcuttur. Bir kaçı şu şekildedir:

SUM Fonksiyonu (Toplama İşlemi)

Verilen kolonlar içerisinde toplama işlemi gerçekleştirir. Kullanımı "*SELECT SUM(kolon_ismi) FROM tablo_ismi*" şeklindedir. Örneğin;



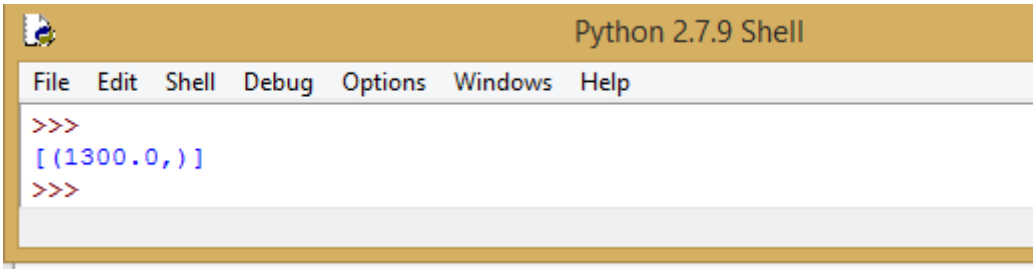
Yukarıda ki şekilde **personel** tablosunda ***personel maaşların***ı hesapladık. Şirketin personele ödediği toplam maaş 6 milyar 500 lira imiş. Ancak dikkat edilmesi gereken husus bu hesaplamaların integer (int/sayı) türünde ki verileri hesapladığıdır. Bu yüzden maaşları girerken tırnak içerisinde girdik verileri.

⇨ SELECT SUM(personel_maas) FROM personel WHERE personel_bolum IN (1,2)

Personel tablosunda ki bölüm nosu 1 ve 2 olan personelin maaşları toplamı nedir sorusu için yazılan sorgu.

AVG Fonksiyonu (Aritmetiksel Ortalama)

Aritmetiksel ortalamayı bulmak için kullanılan bir fonksiyondur. Kullanım şekli "*SELECT AVG(kolon_ismi) FROM tablo_ismi*" şeklindedir.



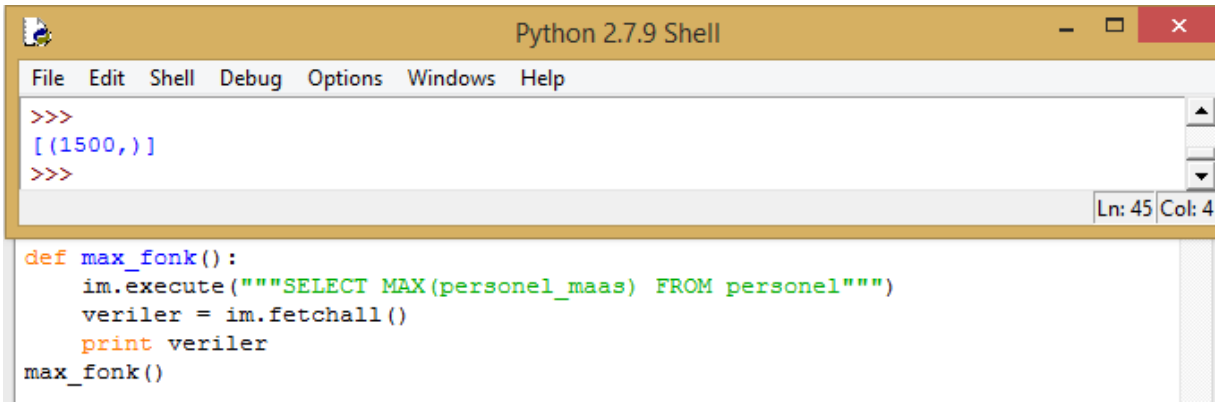

```
def avg_fonk():
    im.execute("""SELECT AVG(personel_maas) FROM personel""")
    veriler = im.fetchall()
    print veriler
avg_fonk()
```

Bu sorgu ile personel tablosunda ki personel maaşlarının ortalamasını öğrenmiş olduk. Üstte olduğu gibi bunu sadece belirli kısıtlamalar ile de yapabilirsiniz (WHERE).

MAX ve MİN Fonksiyonu (Kolonda ki En Büyük ve En Küçük Değer)

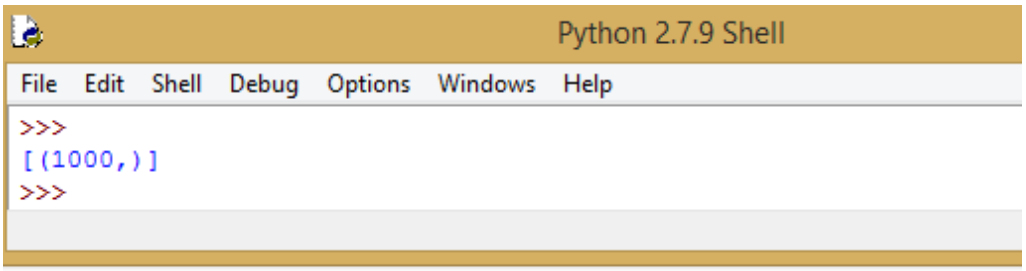
Tablo içerisinde belirtilen kolonda ki en büyük ve en küçük değeri gösterir. Örnek kullanımları;

"*SELECT MAX(kolon_ismi) FROM tablo_ismi*" veya "*SELECT MİN(kolon_ismi) FROM tablo_ismi*"




```
def max_fonk():
    im.execute("""SELECT MAX(personel_maas) FROM personel""")
    veriler = im.fetchall()
    print veriler
max_fonk()
```

Görüldüğü gibi en yüksek maaş 1500 lira imiş.




```
def min_fonk():
    im.execute("""SELECT MIN(personel_maas) FROM personel""")
    veriler = im.fetchall()
    print veriler
min_fonk()
```

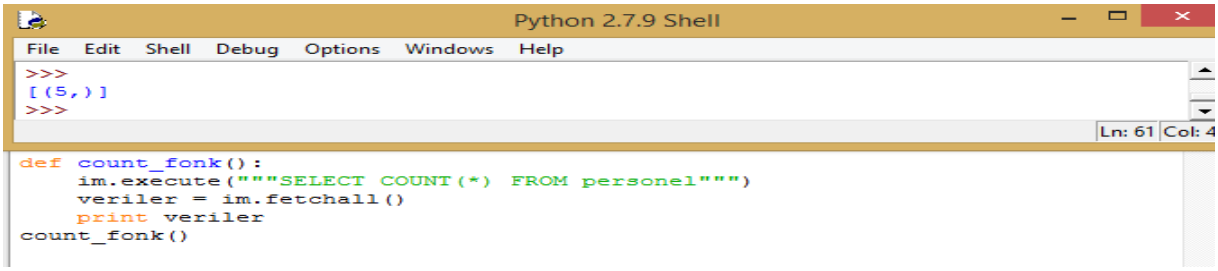
En düşük maaş da 1000 lira imiş.

COUNT Fonksiyonu (Saydırma İşlemi)

Tablo içerisinde sayma işlemi yaptırmak için kullanılır. Misal şirkette kaç personel var? Veya

Şirkette 1500 lira üstü maaş alan kaç personel var? Yada Şirkette bölüm numarası 2 olan kaç kişi var? Bu soruları cevaplandırmak için sorgulara eklenmiştir.

Örnek Sorgu: "*SELECT COUNT(*) FROM tablo_ismi*"

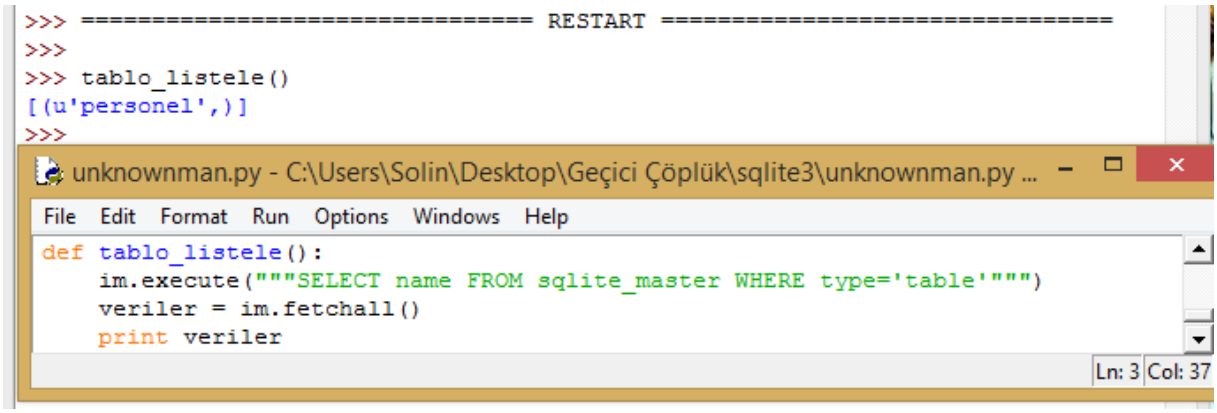


Görüldüğü gibi toplam personel sayımız 5 idi, bize 5 sonucunu verdi.

Bölüm 7 – Tablo İşlemleri

Veri Tabanında ki Tabloları Listeleme

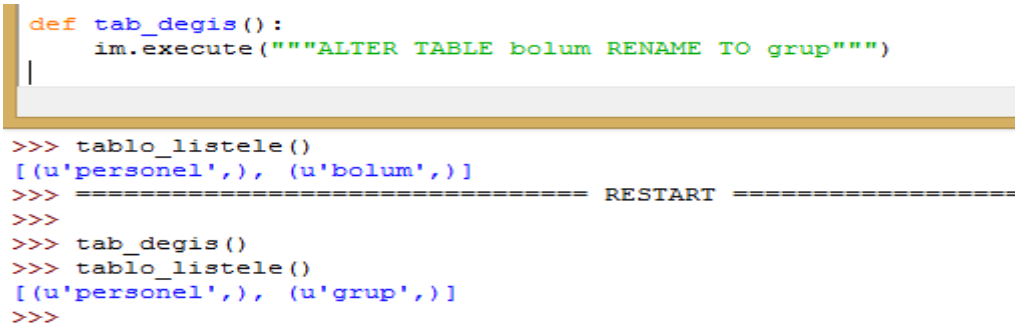
Bazı durumlarda DB üzerinde ki veri tabanlarını öğrenmek isteyebilirsiniz. Bunun için gerekli komutumuz "*SELECT name FROM sqlite_master WHERE type = 'table'*". Burada sqlite_master tablosundan (standart bir tablodur) name kolonunda ki tablolara ait isimleri çekmesini söyledik.



Görüldüğü gibi tek tablo olan "personel" tablosunu bize gösterdi.

Tablo İsmi Değiştirme

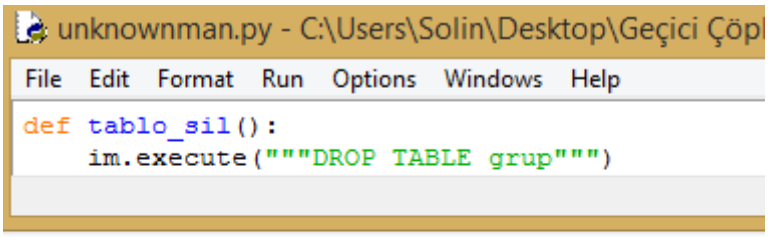
Veri tabanına eklediğimiz bir tablo olan "***bolum***" tablosunun ismini "***grup***" olarak değiştirmek istiyoruz. Bunun için gerekli sorgumuz ise "*ALTER TABLE eski_isim RENAME TO yeni_isim*" şeklindedir.



Gördüğünüz gibi "bolum" tablosu "grup" olarak değiştir. İçeriği ise tamamen aynı.

Tablo Silme

DB üzerinde tablo silmek isterseniz "*DROM TABLE tablo_ismi*" demeniz yeterli. Aşağıda ki örnek yeterince açık.




```
>>> tablo_listele()
[(u'personel',), (u'grup',)]
>>> tablo_sil()
>>> tablo_listele()
[(u'personel',)]
>>>
```

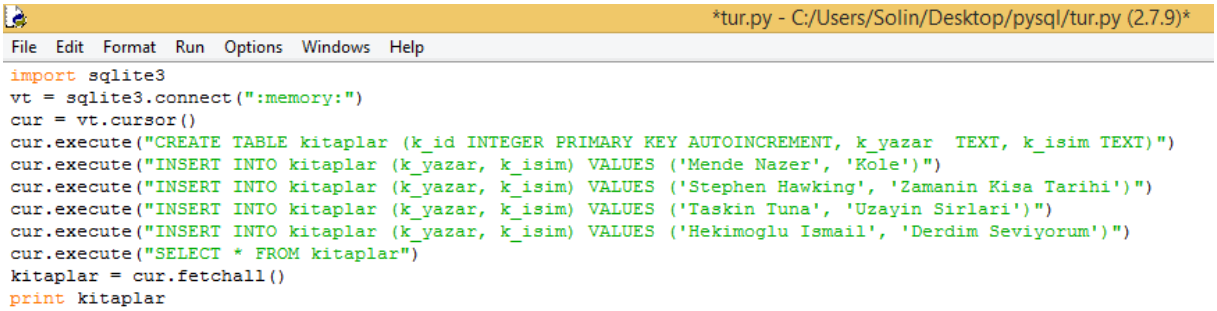
Öncelikle tabloları listeledik ve 2 tablomuz olduğunu gördük. İkinci etapta ise tablo silme fonksiyonunu çağırdık ve tekrar okuyunca tek tablomuz olduğunu gördük.

Bölüm 8 – AUTO INCREMENT ve PRIMARY KEY

Bazen kolonlar üzerinde id değerlerinin otomatik artmasını ve satıra özgün bir değer almasını isteyebilirsiniz. Bu sadece id için değil, örneğin kullanıcı adının değerinin kullanıcının satırına özgün olmasını, başka kullanıcının aynı değeri kullanmasını istiyor olabilirsiniz. Bu durumlarda kolonlara özel değerler vermeniz gerekiyor..

```
def kutuphane():
    im.execute("""CREATE TABLE kutuphane (kitap_id INTEGER PRIMARY KEY   AUTOINCREMENT, kitap_yazar, kitap_isim)""")
```

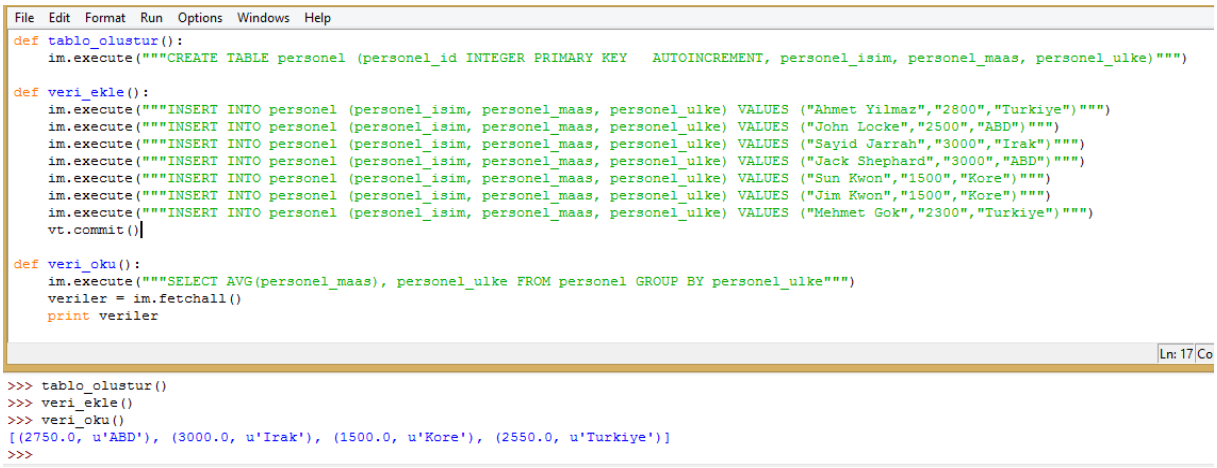
Yukarıda kitap_id kolonunun yanına virgül ile ayırmadan INTEGER PRIMARY KEY AUTO INCREMENT dedik. Bu kitap_id kolonu SAYI, BİRİNCİL ANAHTAR DEĞERİ OTOMATİK ARTACAK demek.




```
>>> ================================ RESTART ================================
>>>
+---+--------------------+--------------------+
+ 1 + Mende Nazer         Kole                 +
+---+--------------------+--------------------+
+ 2 + Stephen Hawking     Zamanin Kisa Tarihi  +
+---+--------------------+--------------------+
+ 3 + Taskin Tuna         Uzayin Sirlari       +
+---+--------------------+--------------------+
+ 4 + Hekimoglu Ismail    Derdim Seviyorum     +
+---+--------------------+--------------------+
>>>
```

Bölüm 9 – Gruplama Fonksiyonu (GROUP BY)

Bazı durumlarda SQL verilerini gruplamak gerekir. Mesela personel diye bir tablo oluşturup buraya bazı veriler ekleyelim. Daha sonra ülkeleri gruplayarak maaş ortalamasını hesaplayalım.



```
File  Edit  Format  Run  Options  Windows  Help
def tablo_olustur():
    im.execute("""CREATE TABLE personel (personel_id INTEGER PRIMARY KEY   AUTOINCREMENT, personel_isim, personel_maas, personel_ulke)""")

def veri_ekle():
    im.execute("""INSERT INTO personel (personel_isim, personel_maas, personel_ulke) VALUES ("Ahmet Yilmaz","2800","Turkiye")""")
    im.execute("""INSERT INTO personel (personel_isim, personel_maas, personel_ulke) VALUES ("John Locke","2500","ABD")""")
    im.execute("""INSERT INTO personel (personel_isim, personel_maas, personel_ulke) VALUES ("Sayid Jarrah","3000","Irak")""")
    im.execute("""INSERT INTO personel (personel_isim, personel_maas, personel_ulke) VALUES ("Jack Shephard","3000","ABD")""")
    im.execute("""INSERT INTO personel (personel_isim, personel_maas, personel_ulke) VALUES ("Sun Kwon","1500","Kore")""")
    im.execute("""INSERT INTO personel (personel_isim, personel_maas, personel_ulke) VALUES ("Jim Kwon","1500","Kore")""")
    im.execute("""INSERT INTO personel (personel_isim, personel_maas, personel_ulke) VALUES ("Mehmet Gok","2300","Turkiye")""")
    vt.commit()

def veri_oku():
    im.execute("""SELECT AVG(personel_maas), personel_ulke FROM personel GROUP BY personel_ulke""")
    veriler = im.fetchall()
    print veriler
                                                                                    Ln: 17 Co
>>> tablo_olustur()
>>> veri_ekle()
>>> veri_oku()
[(2750.0, u'ABD'), (3000.0, u'Irak'), (1500.0, u'Kore'), (2550.0, u'Turkiye')]
>>>
```

Görüldüğü gibi ülkelere göre göre maaş ortalaması bize sunuldu.

Sorgumuz ise "*SELECT kolon FROM tablo GROUP BY kolon_ismi*" şeklinde. Personel maaşını ve personel ülkesi kolonlarını seçtik, bunları ülkeye göre grupladık. Maaşı ise kümeleme fonksiyonlarından AVG (ortalama için) fonksiyonunu kullandık. Tabi ki birden fazla sıralamada yapılabilir. "*SELECT kolon_isimi, kolon_isimi FROM tablo_ismi GROUP BY kolon_ismi, kolon_isimi*" şeklinde de çalıştırılabilir.

Bölüm 10 – LIMIT Fonksiyonu

SELECT ile dönen kayıtların belirli sayıda olması isteniyorsa bu fonksiyon kullanılır. Sorgu kullanımı ise "*SELECT * FROM tablo_ismi LIMIT belirlenen_limit*" şeklindedir. Örneğin veri tabanında ki ilk 2 kaydı döndürmek istiyoruz:

```
def veri_oku():
    im.execute("""SELECT * FROM personel LIMIT 2""")
    veriler = im.fetchall()
    print veriler
```

```
>>> veri_oku()
[(1, u'Ahmet Yilmaz', u'2800', u'Turkiye'), (2, u'John Locke', u'2500', u'ABD')]
>>>
```

Görüldüğü gibi sadece ilk 2 kayıt döndü.

Peki, bu fonksiyonun ne kadar yararlı olduğunu görmek ister misiniz? Diyelim ki veri tabanı üzerinde binlerce kullanıcı var. Tabi ki her kullanıcının kendine özel kullanıcı adı ve şifresi var. Kullanıcı giriş yaparken şöyle bir sorgu oluşacak

*SELECT * FROM üyeler WHERE k_adi = "ayaz" AND sifre = "123654"*

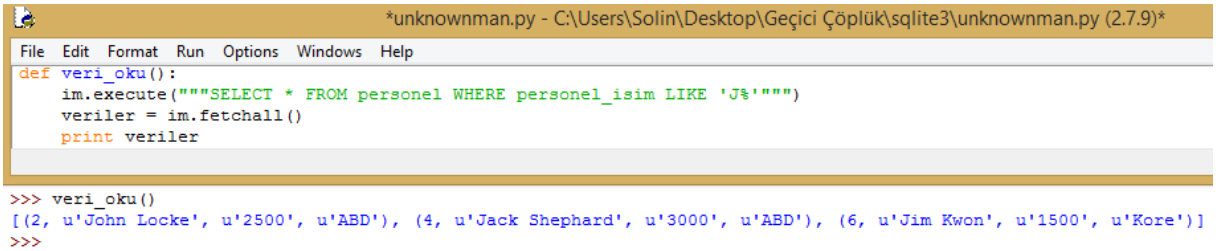
Var sayalım ki Ayaz kullanıcısı 40. Satırda. Program 40. Satıra gelip doğruladığı halde çalışmaya devam edecek ve binlerce kullanıcıyı tarayacak. Bu büyük bir yük ve zaman kaybıdır. Eğer kodu şu şekilde düzeltirsek

*SELECT * FROM üyeler WHERE k_adi = "ayaz" AND sifre = "123654" LIMIT 1*

İlk eşleşmede program duracaktır. Buda büyük bir zaman tasarrufu demektir. Sonuçta kullanıcı adı ve şifresi uyuşan tek üye olacaktır.

Bölüm 11 – LIKE Fonksiyonu

Yazdığınız program içerisinde ismi "j" ile başlayan kullanıcıları bulmak ister misiniz?



LIKE fonksiyonu kullanılarak bu tür problemler aşılabilir. Sorgu kullanım şekli ise

"*SELECT * FROM tablo_ismi WHERE kolon_ismi LIKE 'X%'*"

Burada X neye göre arayacağıdır. Siz oraya ***ist&*** yazarsanız belirttiğiniz konumda ***ist*** ile başlayan tüm veriler ekrana dizilecektir. % ise joker karakterdir.

x%	X ile başlayan tüm veriler
%x%	İçerisinde x olan tüm veriler
_xx%	2. ve 3. Karakteri x olan tüm veriler
%x	X ile biten tüm veriler.

SQLite Dersleri Burada Sona Ermiştir.
Tüm makale privatesec.org adına Kara Ayaz tarafından yazılmıştır.

You still want to trick me? Heh.