



Crogioli, alambicchi e beute

Dove mettere i vostri dati



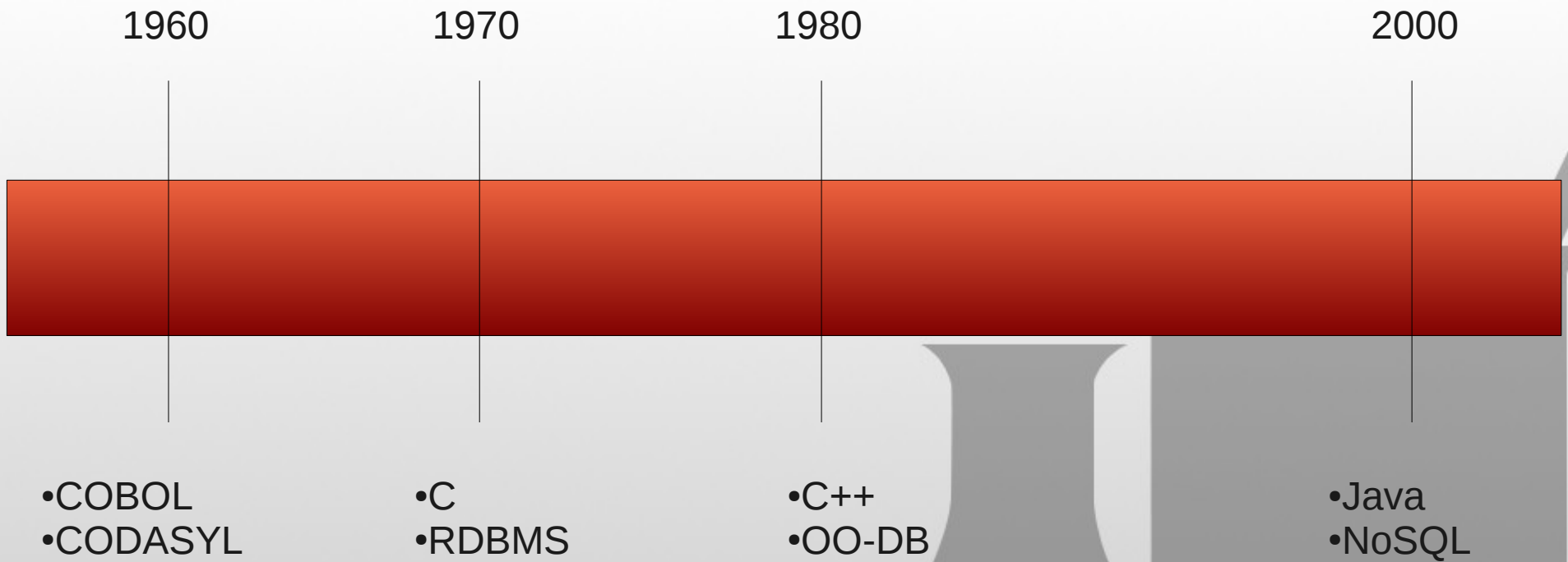
Simone Deponi (simone.deponi@abstract.it)

Il problema della persistenza

1. Problema fondamentale dell'informatica
2. Soluzione semplice: *file*
3. Soluzione complessa: *sistemi di storage*



Cenni storici



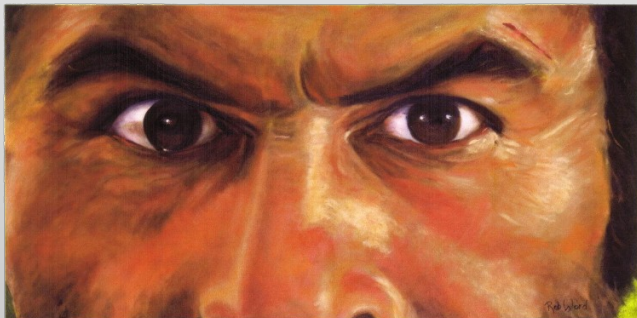
Gli sfidanti



SQLAlchemy



BigTable



ZODB

SQLAlchemy: ORM e oltre

1.ORM: semplificano l'uso di SQL

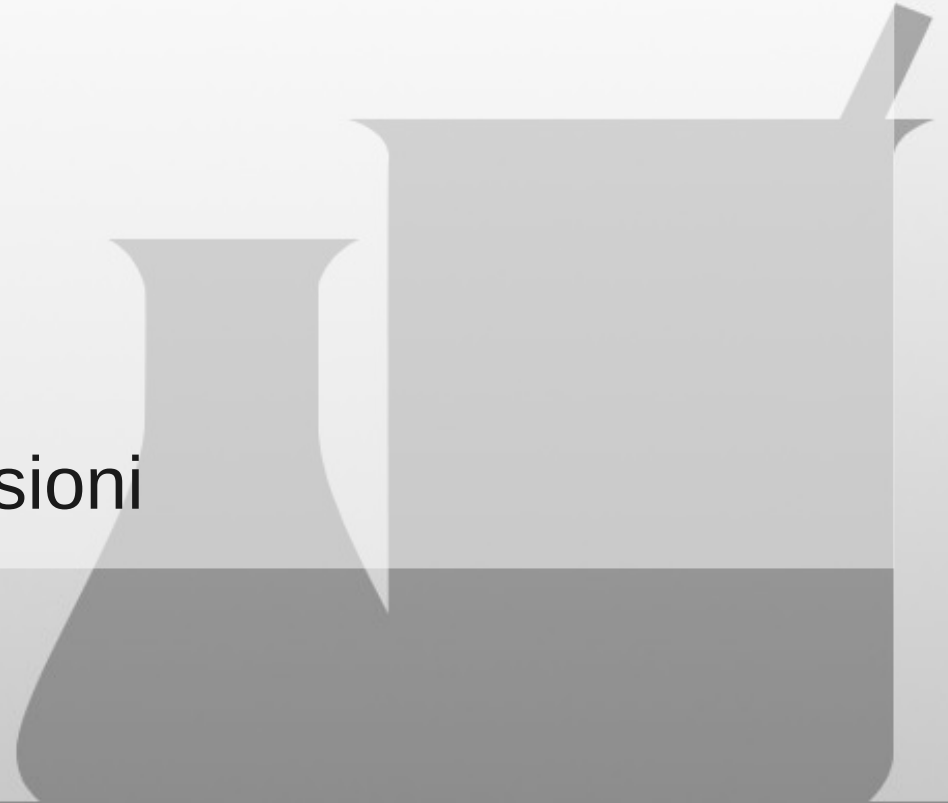
- No problemi compatibilità
- Più sicurezza

2.Astrazione RDBMS

- Compensazione feature

3.Non solo tabelle

- Mappa oggetti su espressioni



SQLAlchemy: in azione

```
1 # import and initialize stuff...
2
3 class User(Base):
4     __tablename__ = 'users'
5
6     id = Column(Integer, primary_key=True)
7     name = Column(String)
8     fullname = Column(String)
9     password = Column(String)
10
11     def __init__(self, name, fullname, password):
12         self.name = name
13         self.fullname = fullname
14         self.password = password
15
16 # initialize session or get it
17
18 ed_user = User('ed', 'Ed Jones', 'edspassword')
19 session.add(ed_user)
20 our_user = session.query(User).filter_by(name='ed').first()
```

SQLAlchemy

Forza

1. Semplice

- API “scala” bene

2. Veloce

- Come sviluppo
- Come prestazioni

3. Flessibile

Debolezze

1. Programmazione generica difficile

2. Difficile con

- Definizioni schemi labili
- Strutture gerarchiche

3. Scalabilità

- Normalizzazione
- Verticalizzazione

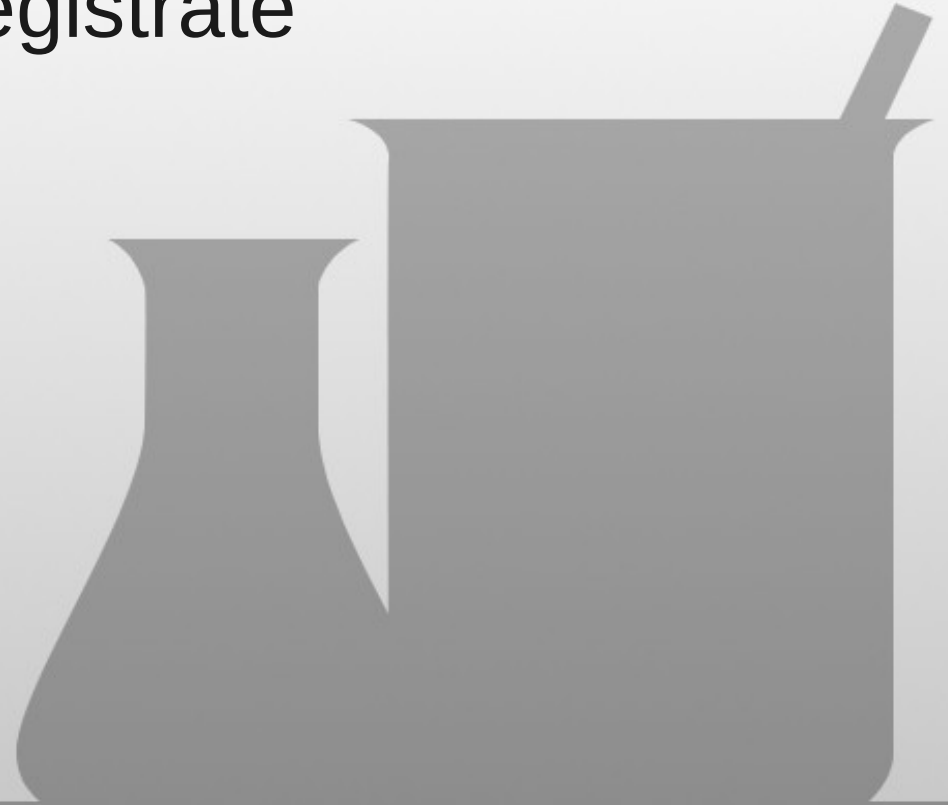
SQLAlchemy: per chi

A photograph of a laptop keyboard where the keys are replaced by various colored LEGO bricks. The keyboard is open, and the screen is visible in the background. The background is a light-colored surface with some green decorative lines.

- Potenza (e problemi) del relazionale
- Ottimale se:
 - Strutture dati definite
 - Collezioni omogenee
 - Forte tipizzazione del dato
- ~ 99% casi applicativi

ZODB: Object DataBase

1. Object database: il ritorno del filesystem
2. Inserimento oggetti nativi Python
3. Le referenze vengono registrate
4. Dinamico
5. Pickling



ZODB: in azione

```
1 # Import stuff, initialize connections, etc
2
3 class User(Persistent):
4
5     def __init__(self, name, fullname, password):
6         self.name = name
7         self.fullname = fullname
8         self.password = password
9
10 root[u'users'][u'ed'] = User(u'ed', u'Ed Jones', u'edspassword')
11 transaction.commit()
```

ZODB

Forza

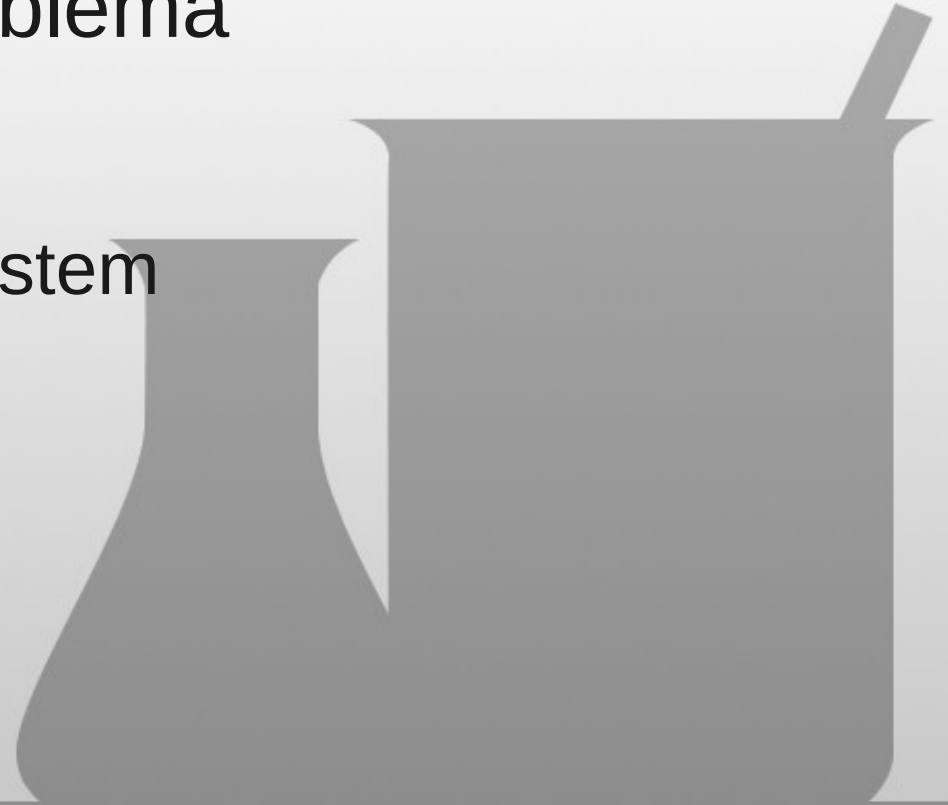
1. Dinamismo
2. Programmazione generica e introspezione
3. Collezioni eterogenee
4. API trasparente

Debolezze

1. "Listing cartella con 10000 files"
2. No index, no search
3. Scalabilità "differente"

ZODB: per chi

1. Definizioni schemi labili
2. Necessità di componenti generici
3. Indicizzare non è un problema
4. Esempio
 - Content Management System



BigTable: distribuito

1. Problema strutture fortemente distribuite
2. Simile a RDBMS
 - Concetto di record e colonna
3. Differente da RDBMS
 - Famiglie di colonne
4. CAP Theorem
 - Consistenza “a latere”



BigTable: cosa

1. Matrice sparsa

- Famiglie di colonne e colonne
- Righe, inserimento non atomico su intera riga

2. Matrice 3D

- (Id riga, Id colonna, timestamp) → valore

3. Matrice 4D

- Solo Cassandra

BigTable: con Python?

1. Hbase (BigTable classico) o Cassandra (BigTable + Dynamo)
2. Via Thrift
 - Generazione protocolli binari multilinguaggio
3. Molto diverso da concetti base presenti

BigTable: in azione

```
1 # Import stuff etc
2
3 class UserKey(Key):
4     """This is a key class which defaults to storage in the Users CF
5     in the UserData Keyspace."""
6     def __init__(self, key=None):
7         Key.__init__(self, "UserData", "Users", key)
8
9 # Subclass Record to create an object of the correct type.
10 class User(record.Record):
11     """A class representing a user in Cassandra."""
12
13     # Anything in here _must_ be set before the object is saved
14     _required = ('username',)
15
16     def __init__(self, *args, **kwargs):
17         """Initialize the record, along with a new key."""
18         record.Record.__init__(self, *args, **kwargs)
19         self.key = UserKey()
20
21 data = {'username': 'ieure', 'email': 'ian@digg.com'}
22 # Arguments to __init__ are passed to update()
23 u = User(data)
24 # key is auto-set
25 print u.key
26 # If you want to store records in a SuperColumn, set key.super_column:
27 superkey = u.key.clone(super_column="scol_a")
28 # Save to Cassandra
29 u.save()
30 # Load it in a new instance.
31 u_ = User().load(u.key.clone())
```

BigTable

Forza

1. Superscalabilità
2. Availability & Partition tolerance
3. Buon dinamismo

Debolezze

1. API non intuitiva
2. Delega consistenza
 - Fa del proprio meglio, non garantisce
3. Sforzo sviluppo applicativo
 - Tablet e località

BigTable: per chi

1. Google, Facebook, Yahoo
2. Data mining su grandi moli
 - Map/Reduce
3. Dati spezzati in microelementi
4. Necessità di distribuire aggressivamente

Conclusioni

1. Idee chiare: ORM + RDBMS
2. “robe”, “documenti”: OO-DB
3. Grandissime moli: BigTable



Per saperne di più

- <http://sqlalchemy.org/docs/>
- <http://www.zodb.org>
- <http://www.julianbrowne.com/article/viewer/browsers-cap-theorem>
- <http://wiki.apache.org/cassandra/DataModel>
- <http://github.com/digg/lazyboy>

Credits

1. Brueghel the Elder
2. Rob Word
3. <http://www.flickr.com/photos/gilest/1>

