SQLAlchemy

The Python SQL Toolkit and
Object Relational Mapper

PDXPUG
May 15, 2008

SQLAlchemy

- Open Source, MIT License
- Project started in 2005 by Mike Bayer
- Now up to 20 committers, plus patch contributors, plus third-party plugins and products (including a SA driver from IBM!)
- April 2008: 372 commits, ~800 messages on all mailing lists, ~100 on #sqlalchemy

Python

Python

```
if something.method() == 123:
    user = User('new user')
```

Python

```
if something.method() == 123:
    user = User('new user')
```

- Now you know Python

Features

- PostgreSQL, MySQL, SQLite, Firebird, Oracle, MSSQL, Sybase, DB2, Informix, SAPDB, MSAccess
- Runs anywhere Python runs
- CPython 2.3+, Jython soon!

And Yes,

- Composite keys
- Substitute hand-written SQL anywhere, with no hacks

- Very flexible: a toolkit
- *Slightly* opinionated

- Very flexible: a toolkit
- *Slightly* opinionated
    - The truth is fungible

- Very flexible: a toolkit
- *Slightly* opinionated
    - The truth is fungible
    - We like SQL and don't try to hide it

- Very flexible: a toolkit
- *Slightly* opinionated
    - The truth is fungible
    - We like SQL and don't try to hide it
    - SQLAlchemy never generates names

- Very flexible: a toolkit
- *Slightly* opinionated
    - The truth is fungible
    - We like SQL and don't try to hide it
    - SQLAlchemy never generates names
    - Generated SQL always uses bind parameters

What You Get

- Database Connection Management
- Schema Management and Data Types
- SQL Dialects
- SQL Expression Language
- Object Relational Mapper

Connection Management

- Connection pooling
- Unified interface for connect() parameters
- Auto-commit, transactions, two-phase transactions, SAVEPOINTs
- Multi-database

Schema Management & Data Types

- Table structure is represented in Python
- Tables can be specified in code, reflected from a live database, or a combination
- "Migration" support
- Supports user defined types- Python-side and server-side

A Join Table from RT

A Join Table from RT

```
CREATE TABLE groupmembers (
  id INTEGER DEFAULT nextval('groupmembers_id_seq'),
  groupid INTEGER NOT NULL DEFAULT 0,
  memberid INTEGER NOT NULL DEFAULT 0,
  PRIMARY KEY (id))
```

A Join Table from RT

```
CREATE TABLE groupmembers (
  id INTEGER DEFAULT nextval('groupmembers_id_seq'),
  groupid INTEGER NOT NULL DEFAULT 0,
  memberid INTEGER NOT NULL DEFAULT 0,
  PRIMARY KEY (id))
```

```
Table('groupmembers', metadata,
      Column('id', Integer, primary_key=True,
             autoincrement=True),
      Column('groupid', Integer, nullable=False,
             default=0),
      Column('memberid', Integer, nullable=False,
             default=0))
```

Wouldn’t It Be Nice If...

Wouldn't It Be Nice If...

```
CREATE TABLE groupmembers (
groupid INTEGER NOT NULL
        REFERENCES groups (id),
memberid INTEGER NOT NULL
         REFERENCES principals (id),
PRIMARY KEY (groupid, memberid))
```

Fungible!

Fungible!

```
CREATE TABLE groupmembers (
  id INTEGER DEFAULT nextval('groupmembers_id_seq'),
  groupid INTEGER NOT NULL DEFAULT 0,
  memberid INTEGER NOT NULL DEFAULT 0,
  PRIMARY KEY (id))
```

Fungible!

```
CREATE TABLE groupmembers (
   id INTEGER DEFAULT nextval('groupmembers_id_seq'),
   groupid INTEGER NOT NULL DEFAULT 0,
   memberid INTEGER NOT NULL DEFAULT 0,
   PRIMARY KEY (id))
```

```
Table('groupmembers', metadata,
      Column('groupid', Integer,
             ForeignKey('groups.id'),
             nullable=False, primary_key=True)
      Column('memberid', Integer,
             ForeignKey('principals.id'),
             nullable=False, primary_key=True))
```

SQL Dialects

SQL Dialects

- SQLAlchemy drivers?

SQL Dialects

- SQLAlchemy drivers?
- RETURNING

SQL Expression Language

- Defined tables form the most common basis for expressions and selectables

```
>>> principals.c.id
Column('id', Integer(), table=<principals>, ...)


>>> print principals.select()
SELECT principals.id, principals.principaltype,
principals.objectid, principals.disabled FROM principals
```

```
>>> s = select([principals.c.id]).limit(1)
>>> print s
SELECT principals.id FROM principals LIMIT 1


>>> s.execute().fetchall()
[(1,)]


>>> s2 = s.where(principals.c.principaltype == 'Group')
>>> print s2
SELECT principals.id FROM principals
WHERE principals.principletype = ? LIMIT 1
>>> s2.execute().fetchall()
[(2,)]
```

```
>>> select([principals.c.instanceid, groups.c.name]).
      where(groups.c.description.ilike('foo%bar')).
      select_from(principals.join(groups))
```

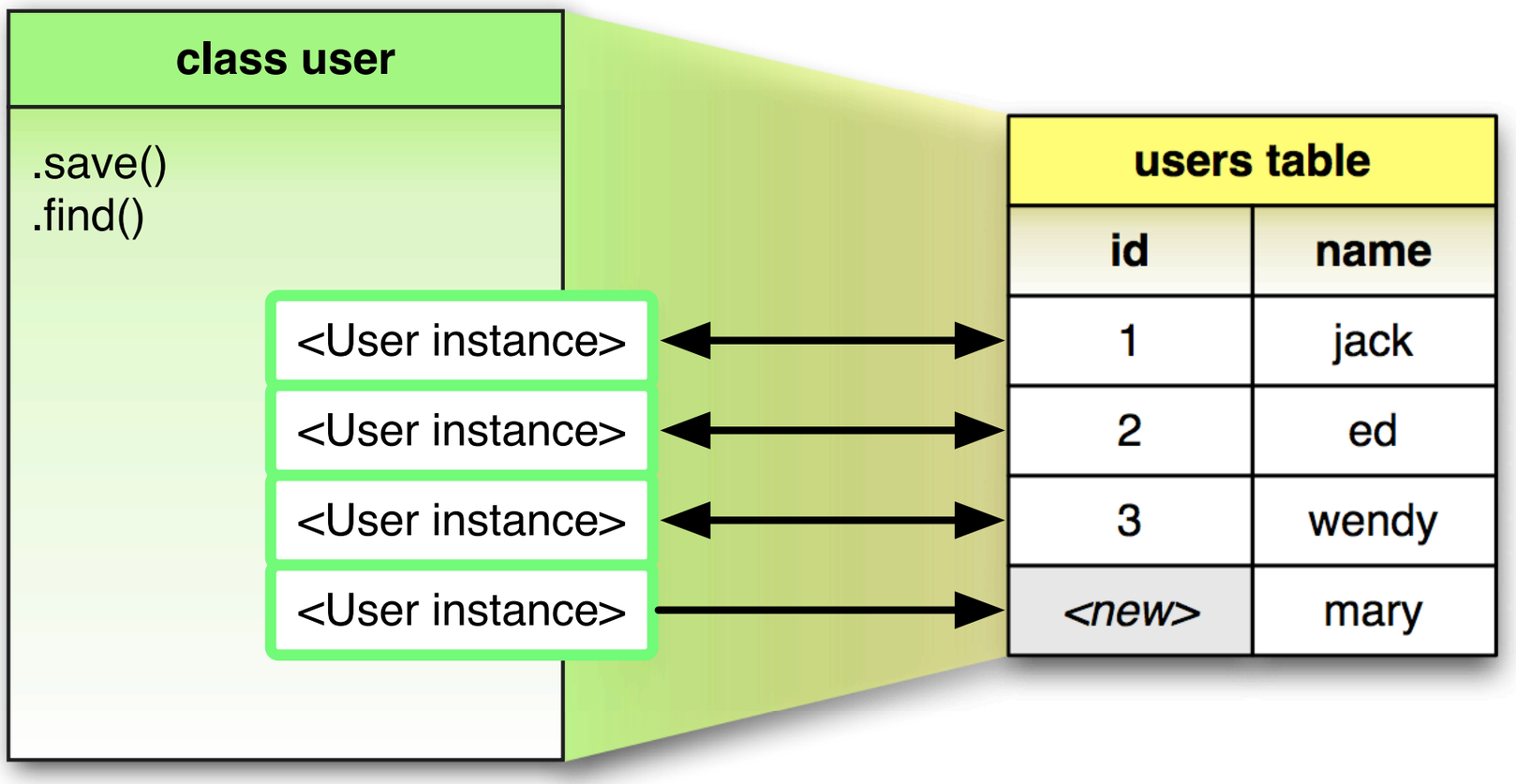
- WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, OFFSET, ...

- Assorted JOINs, UNIONs, INTERSECT, EXCEPT, subqueries with correlation control, IN, EXISTS, ...

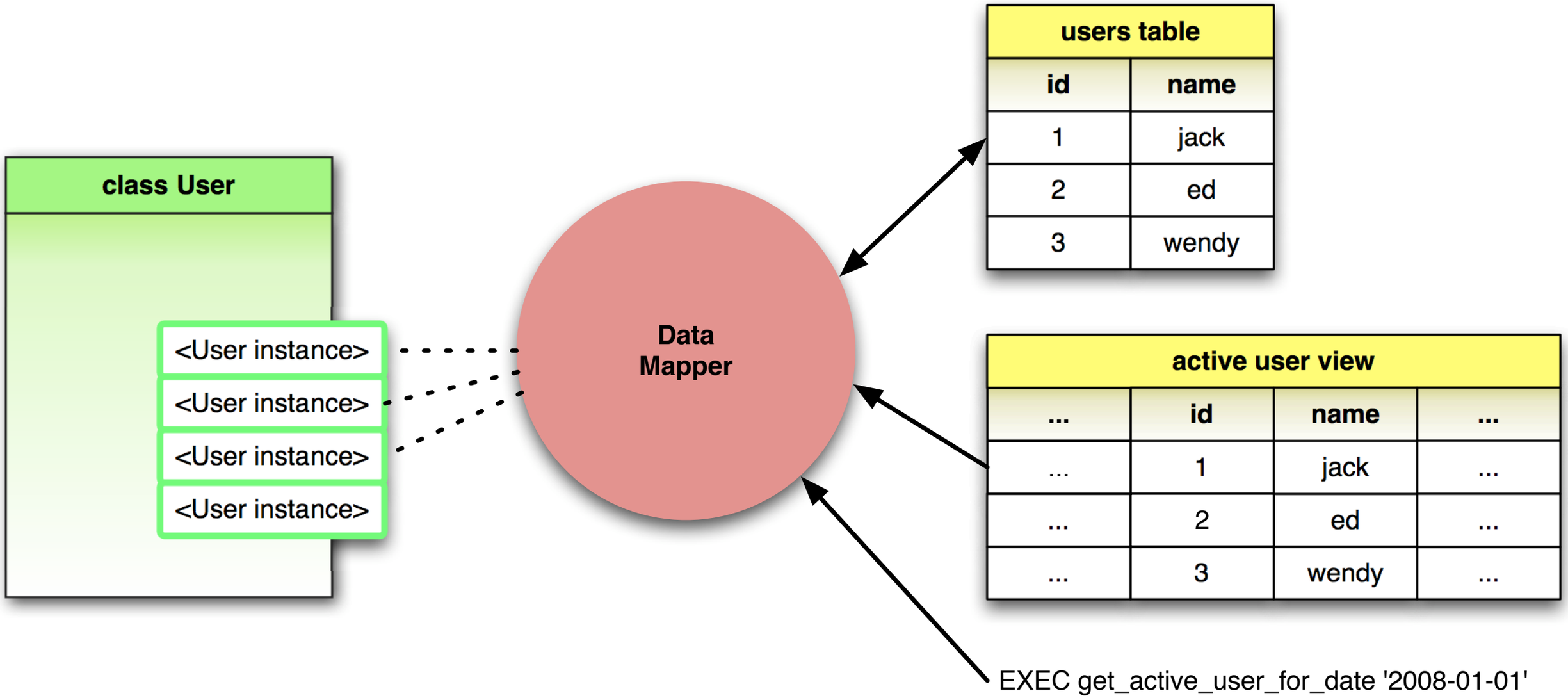
- INSERT, UPDATE and DELETE too

Object Relational Mapper

“Active Record”



“Data Mapper”


class User
<User instance>
<User instance>
<User instance>
<User instance>
Data Mapper
users table
id
name
1
jack
2
ed
3
wendy
active user view
...
id
name
...
...
1
jack
...
...
2
ed
...
...
3
wendy
...
EXEC get_active_user_for_date '2008-01-01'

Some ORM Features

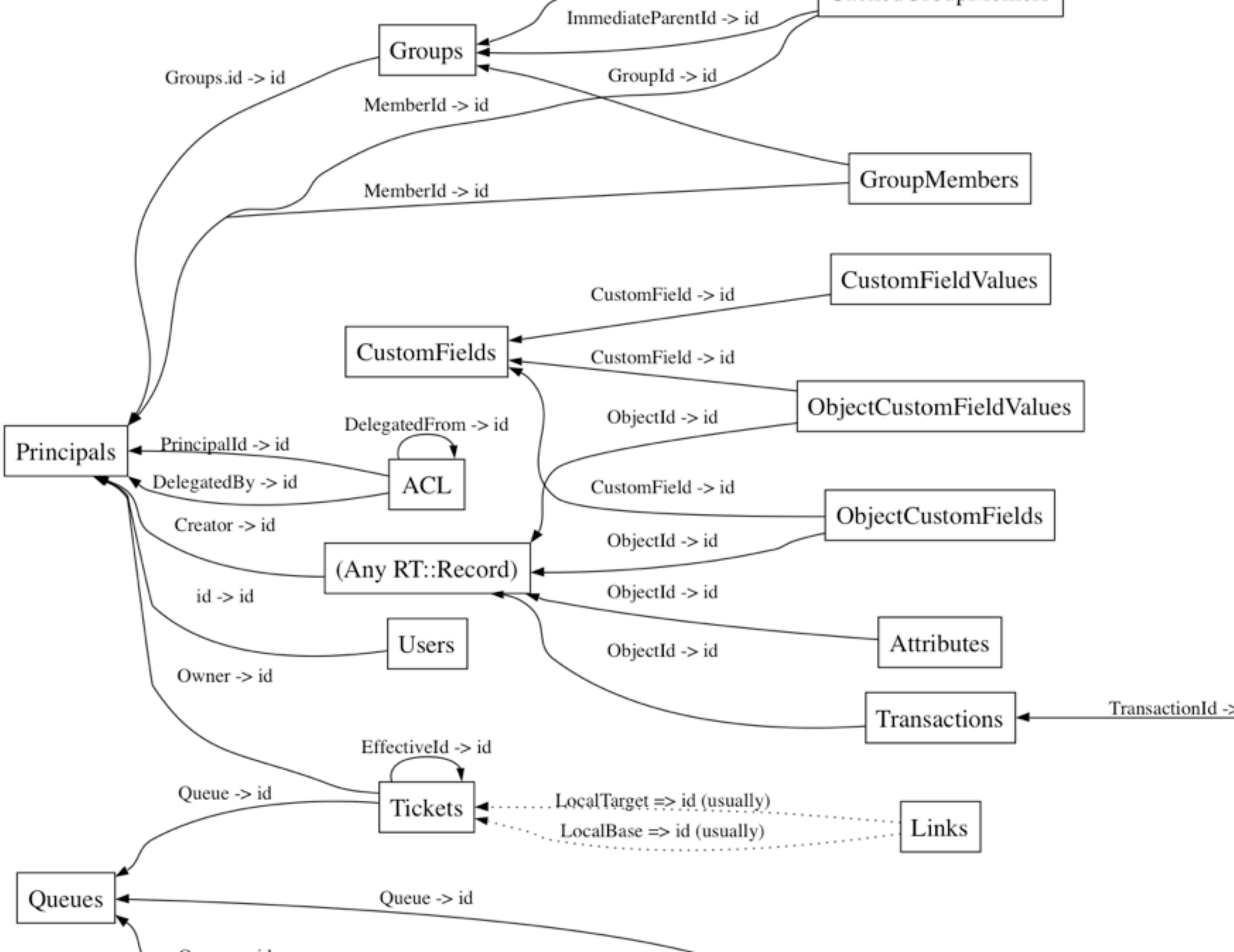
- Data Mapper + Any Selectable
- Unit of Work
- Eager / Lazy Loading
- Deferred Properties
- Inheritance Mapping
- Self Referential Mapping
- Sharding

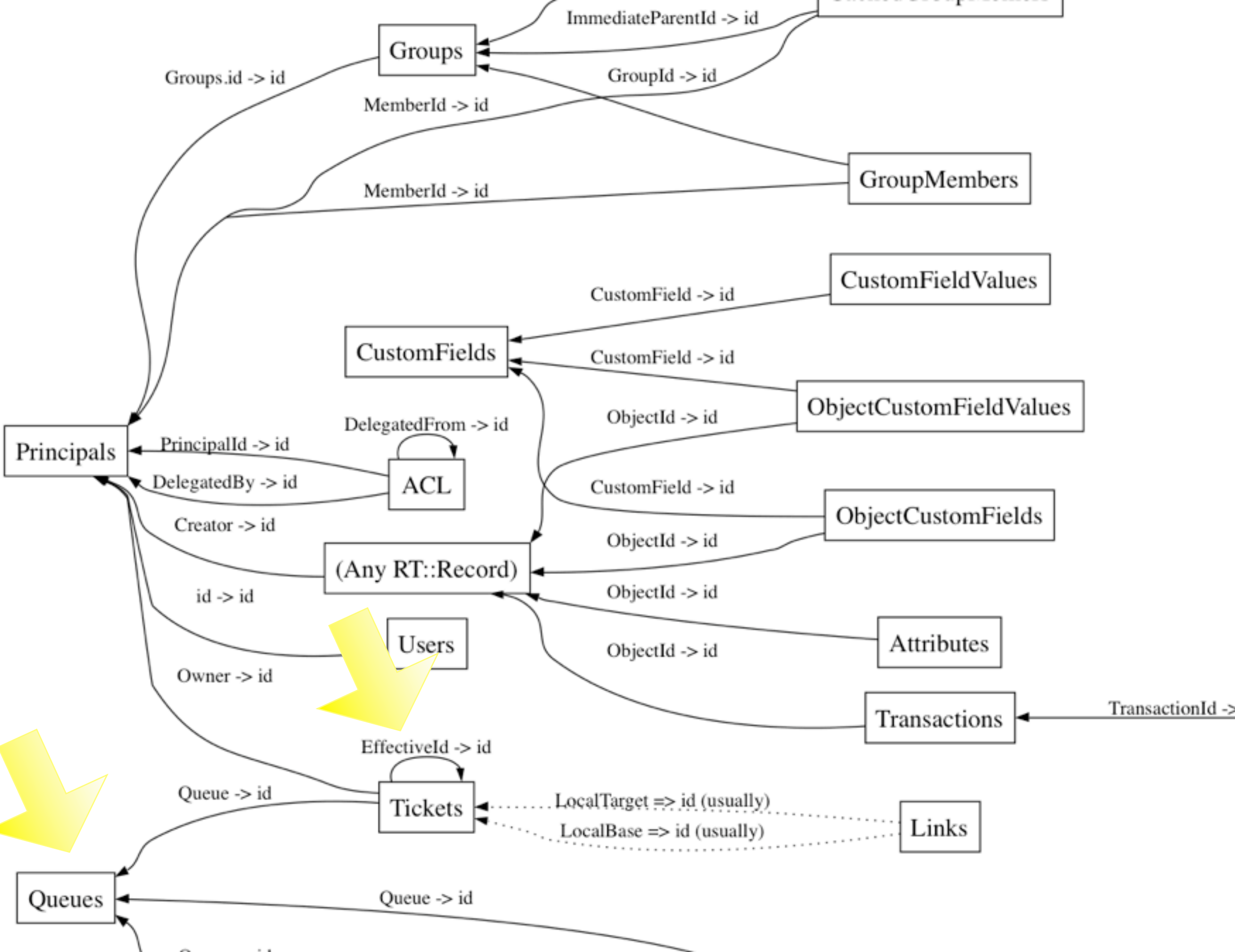
```
# option 1: purity

principals = Table('principals', metadata,
   Column('id', Integer, primary_key=True),
   Column('objectid', Integer))

class Principal(object):
    pass

mapper(Principal, principals)

# option 2: practicality

class Principal(Base):
    id = Column(Integer, primary_key=True)
    objectid = Column(Integer)
```

```
>>> session = create_session(autocommit=False)
>>> q1 = session.query(Queue).filter_by(name='Support').one()
>>> u1 = session.query(User).filter_by(nickname='jek').one()
```

```
>>> session = create_session(autocommit=False)
>>> q1 = session.query(Queue).filter_by(name='Support').one()
>>> u1 = session.query(User).filter_by(nickname='jek').one()


>>> t1 = Ticket(queue=q1, owner=u1, creator=u1)
>>> session.add(t1)
>>> session.commit()
-] SELECT nextval('"tickets_id_seq"')
-] INSERT INTO tickets (id, effectiveid, queue, type,
issuestatement, resolution, owner, ...) VALUES (%(id)s, %
(effective_id)s, %(queue_id)s, %(type)s, %(issue_statement)s,
%(resolution)s, %(owner_id)s, ...)
-] {'id': 332L, 'queue_id': 2, 'issue_statement': 0,
'owner_id': 1, 'resolution': 0, ...}
-] COMMIT
```

```
>>> t.subject = ‘fix this!’
>>> s.commit()
-] UPDATE tickets SET subject=%(subject)s WHERE tickets.id =
%(tickets_id)s
-] {'tickets_id': 332L, 'subject': 'fix this!'}
```


Groups
ImmediateParentId -> id
Groups.id -> id
GroupId -> id
MemberId -> id
GroupMembers
MemberId -> id
CustomFieldValues
CustomField -> id
CustomFields
CustomField -> id
ObjectCustomFieldValues
ObjectId -> id
DelegatedFrom -> id
Principals
PrincipalId -> id
ACL
DelegatedBy -> id
CustomField -> id
ObjectCustomFields
Creator -> id
ObjectId -> id
(Any RT::Record)
id -> id
ObjectId -> id
Users
ObjectId -> id
Attributes
Owner -> id
Transactions
TransactionId ->
EffectiveId -> id
Queue -> id
Tickets
LocalTarget => id (usually)
LocalBase => id (usually)
Links
Queues
Queue -> id

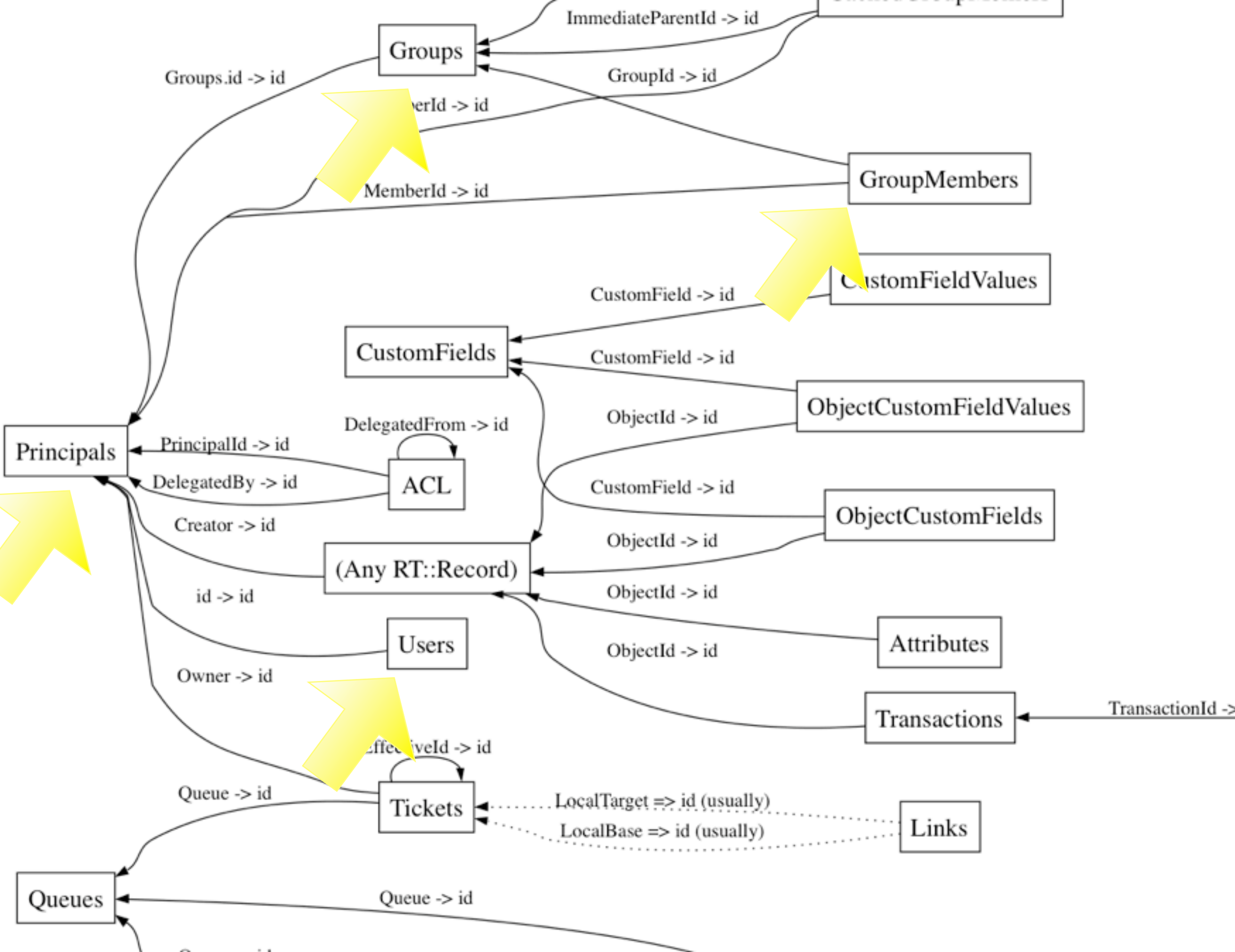

Groups
ImmediateParentId -> id
Groups.id -> id
GroupId -> id
MemberId -> id
GroupMembers
MemberId -> id
CustomFieldValues
CustomField -> id
CustomFields
CustomField -> id
ObjectCustomFieldValues
DelegatedFrom -> id
ObjectId -> id
Principals
PrincipalId -> id
ACL
DelegatedBy -> id
CustomField -> id
ObjectCustomFields
Creator -> id
ObjectId -> id
(Any RT::Record)
id -> id
ObjectId -> id
Users
ObjectId -> id
Attributes
Owner -> id
Transactions
TransactionId ->
EffectiveId -> id
Queue -> id
Tickets
LocalTarget => id (usually)
LocalBase => id (usually)
Links
Queues
Queue -> id

```
mapper(Queue, t_queues, properties={
   'tickets': relation(Ticket, backref='queue',
                       cascade='all, delete, delete-orphan')}

mapper(Ticket, t_tickets)
```

```
>>> t = session.query(Ticket).first()
>>> t
<Ticket #22>
>>> t.queue
<Queue #4 name='Stuff Not Work Am Bad'>
>>> t.queue.tickets
[<Ticket #22>, <Ticket #24>, <Ticket #7>, <Ticket
#23>, <Ticket #8>, <Ticket #12>, <Ticket #9>,
<Ticket #13>, <Ticket #18>, <Ticket #19>, <Ticket
#11>, <Ticket #10>, <Ticket #20>, <Ticket #21>,
<Ticket #25>, <Ticket #26>, <Ticket #27>]
```

```
>>> t = session.query(Ticket).first()
>>> t
<Ticket #22>
>>> t.queue
<Queue #4 name='Stuff Not Work Am Bad'>
>>> t.queue.tickets
[<Ticket #22>, <Ticket #24>, <Ticket #7>, <Ticket
#23>, <Ticket #8>, <Ticket #12>, <Ticket #9>,
<Ticket #13>, <Ticket #18>, <Ticket #19>, <Ticket
#11>, <Ticket #10>, <Ticket #20>, <Ticket #21>,
<Ticket #25>, <Ticket #26>, <Ticket #27>]


>>> t.queue.tickets.pop()
>>> session.flush()
-] DELETE FROM tickets WHERE tickets.id = %(id)s
-] {'id': 27}
```

```
>>> t = session.query(Ticket).first()
>>> t
<Ticket #22>
>>> t.queue
<Queue #4 name='Stuff Not Work Am Bad'>
>>> t.queue.tickets
[<Ticket #22>, <Ticket #24>, <Ticket #7>, <Ticket
#23>, <Ticket #8>, <Ticket #12>, <Ticket #9>,
<Ticket #13>, <Ticket #18>, <Ticket #19>, <Ticket
#11>, <Ticket #10>, <Ticket #20>, <Ticket #21>,
<Ticket #25>, <Ticket #26>, <Ticket #27>]


>>> t.queue.tickets.pop()
>>> session.flush()
-] DELETE FROM tickets WHERE tickets.id = %(id)s
-] {'id': 27}
```


Groups
ImmediateParentId -> id
Groups.id -> id
GroupId -> id
erId -> id
GroupMembers
MemberId -> id
CustomFieldValues
CustomField -> id
CustomFields
CustomField -> id
ObjectCustomFieldValues
ObjectId -> id
DelegatedFrom -> id
Principals
PrincipalId -> id
ACL
DelegatedBy -> id
CustomField -> id
ObjectCustomFields
Creator -> id
ObjectId -> id
(Any RT::Record)
id -> id
ObjectId -> id
Users
ObjectId -> id
Attributes
Owner -> id
Transactions
TransactionId ->
veId -> id
Queue -> id
Tickets
LocalTarget => id (usually)
LocalBase => id (usually)
Links
Queues
Queue -> id

```
class Principal:
    pass


class Group(Principal):
    pass

class User(Principal):
    pass
```

```
mapper(Principal, t_principals,
       polymorphic_on=t_principals.c.principal_type,
       properties={
         'parents': relation(Group,
            secondary=t_groupmembers, backref='members')})


mapper(User, t_users, inherits=Principal,
       inherit_condition=t_principals.c.id == t_users.c.id,
       polymorphic_identity='User')


mapper(Group, t_groups,
       inherits=Principal,
       polymorphic_identity='Group')
```

```
>>> p = session.query(Principal).first()
-] SELECT principals.id AS principals_id,
principals.principaltype AS principals_principaltype,
principals.objectid AS principals_objectid,
principals.disabled AS principals_disabled
FROM principals LIMIT 1 OFFSET 0
```

```
>>> p = session.query(Principal).first()
-] SELECT principals.id AS principals_id,
principals.principaltype AS principals_principaltype,
principals.objectid AS principals_objectid,
principals.disabled AS principals_disabled
FROM principals LIMIT 1 OFFSET 0


>>> type(p)
<class '__main__.User'>
>>> p.name
-] SELECT users.id AS users_id, users.name AS users_name,
users.password AS users_password, users.comments AS
users_comments, users.signature AS users_signature,
users.emailaddress AS users_emailaddress, ...
FROM users
WHERE %(param_1)s = users.id
-] {'param_1': 1}
'jek'
```

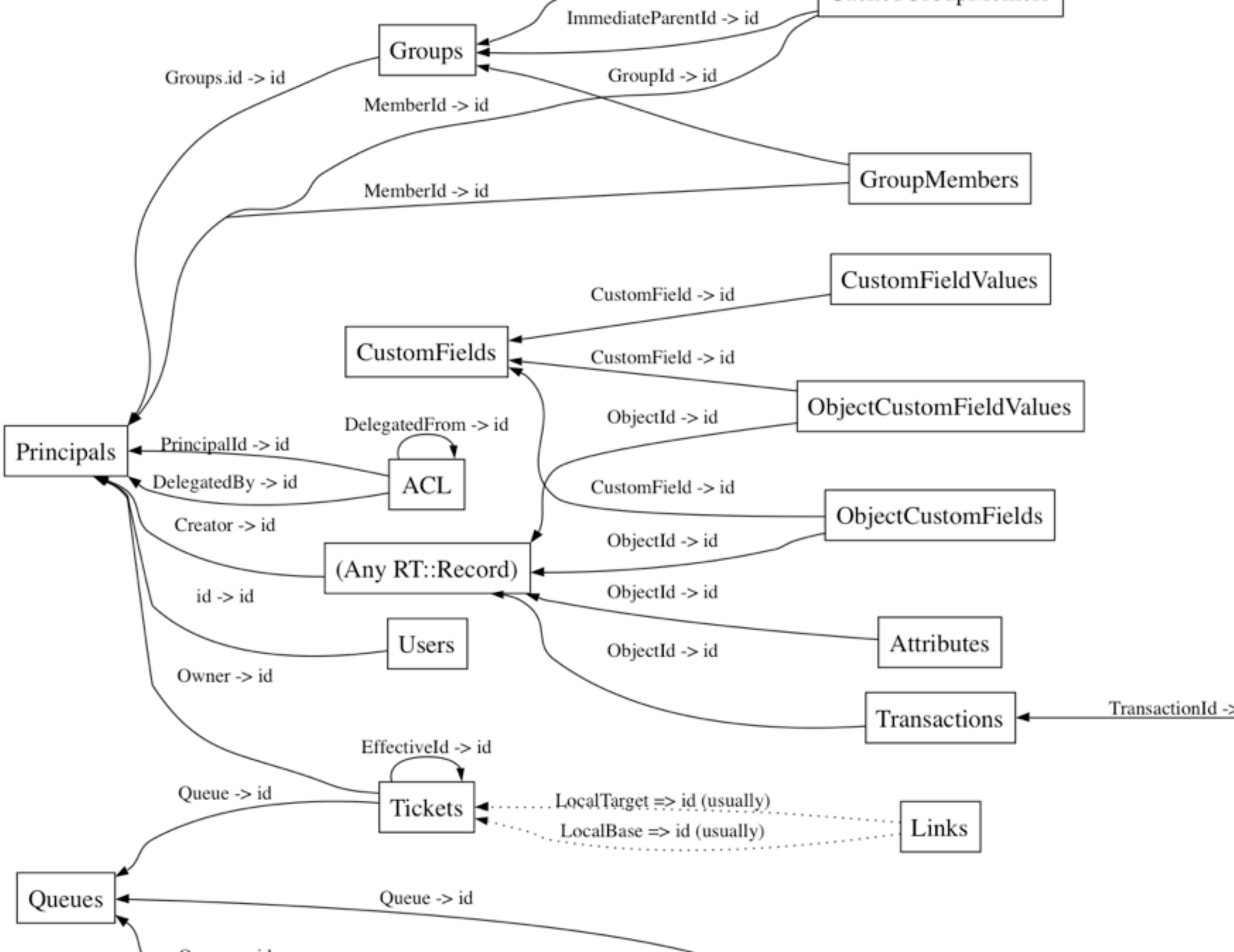
```
>>> p = session.query(Principal).first()
-] SELECT principals.id AS principals_id,
principals.principaltype AS principals_principaltype,
principals.objectid AS principals_objectid,
principals.disabled AS principals_disabled
FROM principals LIMIT 1 OFFSET 0


>>> type(p)
<class '__main__.User'>
>>> p.name
-] SELECT users.id AS users_id, users.name AS users_name,
users.password AS users_password, users.comments AS
users_comments, users.signature AS users_signature,
users.emailaddress AS users_emailaddress, ...
FROM users
WHERE %(param_1)s = users.id
-] {'param_1': 1}
'jek'
```

```
>>> p = session.query(Principal).
        with_polymorphic('*').first()
-] SELECT principals.id AS principals_id,
principals.principaltype AS principals_principaltype, ...,
users.id AS users_id, users.name AS users_name,
users.password AS users_password, users.comments AS
users_comments, users.signature AS users_signature,
users.emailaddress AS users_emailaddress, ... groups.id AS
groups_id, groups.name AS groups_name, groups.description AS
groups_description, ...
FROM principals
  LEFT OUTER JOIN users ON principals.id = users.id
  LEFT OUTER JOIN groups ON principals.id = groups.id
LIMIT 1 OFFSET 0
>>> p.name
'jek'
```

```
>>> session.query(Ticket).
    options(eagerload('owner')).first()
SELECT anon_1.tickets_owner AS anon_1_tickets_owner, anon_1.tickets_id AS anon_1_tickets_id,
anon_1.tickets_effectiveid AS anon_1_tickets_effectiveid, anon_1.tickets_queue AS anon_1_tickets_queue,
anon_1.tickets_type AS anon_1_tickets_type, anon_1.tickets_issuestatement AS
anon_1_tickets_issuestatement, anon_1.tickets_resolution AS anon_1_tickets_resolution,
anon_1.tickets_subject AS anon_1_tickets_subject, anon_1.tickets_initialpriority AS
anon_1_tickets_initialpriority, anon_1.tickets_finalpriority AS anon_1_tickets_finalpriority,
anon_1.tickets_priority AS anon_1_tickets_priority, anon_1.tickets_timeestimated AS
anon_1_tickets_timeestimated, anon_1.tickets_timeworked AS anon_1_tickets_timeworked,
anon_1.tickets_status AS anon_1_tickets_status, anon_1.tickets_timeleft AS anon_1_tickets_timeleft,
anon_1.tickets_told AS anon_1_tickets_told, anon_1.tickets_starts AS anon_1_tickets_starts,
anon_1.tickets_started AS anon_1_tickets_started, anon_1.tickets_due AS anon_1_tickets_due,
anon_1.tickets_resolved AS anon_1_tickets_resolved, anon_1.tickets_lastupdatedby AS
anon_1_tickets_lastupdatedby, anon_1.tickets_lastupdated AS anon_1_tickets_lastupdated,
anon_1.tickets_creator AS anon_1_tickets_creator, anon_1.tickets_created AS anon_1_tickets_created,
anon_1.tickets_disabled AS anon_1_tickets_disabled, principals_1.id AS principals_1_id,
principals_1.principaltype AS principals_1_principaltype, principals_1.objectid AS principals_1_objectid,
principals_1.disabled AS principals_1_disabled
FROM (SELECT tickets.owner AS tickets_owner, tickets.id AS tickets_id, tickets.effectiveid AS
tickets_effectiveid, tickets.queue AS tickets_queue, tickets.type AS tickets_type, tickets.issuestatement
AS tickets_issuestatement, tickets.resolution AS tickets_resolution, tickets.subject AS tickets_subject,
tickets.initialpriority AS tickets_initialpriority, tickets.finalpriority AS tickets_finalpriority,
tickets.priority AS tickets_priority, tickets.timeestimated AS tickets_timeestimated, tickets.timeworked
AS tickets_timeworked, tickets.status AS tickets_status, tickets.timeleft AS tickets_timeleft,
tickets.told AS tickets_told, tickets.starts AS tickets_starts, tickets.started AS tickets_started,
tickets.due AS tickets_due, tickets.resolved AS tickets_resolved, tickets.lastupdatedby AS
tickets_lastupdatedby, tickets.lastupdated AS tickets_lastupdated, tickets.creator AS tickets_creator,
tickets.created AS tickets_created, tickets.disabled AS tickets_disabled
FROM tickets LIMIT 1 OFFSET 0) AS anon_1 LEFT OUTER JOIN principals AS principals_1 ON
anon_1.tickets_owner = principals_1.id
```

```
>>> session.query(Ticket, User).
            join(Ticket.owner).filter(User.id == 1)
[(<Ticket #70>, <User #1 name='RT_System'>),
 (<Ticket #65>, <User #1 name='RT_System'>),
 (<Ticket #66>, <User #1 name='RT_System'>),
 (<Ticket #67>, <User #1 name='RT_System'>)]
```

```
>>> session.query(Ticket).
    filter(Ticket.owner.has(User.name.like('%jek%').all()
-] SELECT tickets.id AS tickets_id, tickets.effectiveid AS
tickets_effectiveid, tickets.queue AS tickets_queue,
tickets.type AS tickets_type, ... tickets.disabled AS
tickets_disabled
FROM tickets
WHERE EXISTS (SELECT 1
FROM principals JOIN users ON principals.id = users.id
WHERE tickets.owner = principals.id AND users.name LIKE %
(name_1)s)
```


Groups
ImmediateParentId -> id
Groups.id -> id
GroupId -> id
MemberId -> id
GroupMembers
MemberId -> id
CustomFieldValues
CustomField -> id
CustomFields
CustomField -> id
ObjectCustomFieldValues
ObjectId -> id
DelegatedFrom -> id
Principals
PrincipalId -> id
ACL
DelegatedBy -> id
CustomField -> id
ObjectCustomFields
Creator -> id
ObjectId -> id
(Any RT::Record)
id -> id
ObjectId -> id
Users
ObjectId -> id
Attributes
Owner -> id
Transactions
TransactionId ->
EffectiveId -> id
Queue -> id
Tickets
LocalTarget => id (usually)
LocalBase => id (usually)
Links
Queues
Queue -> id

- LOTS more Query features
- Object, session, transaction lifecycle hooks
- Redefine or customize most aspects of row <-> object mapping
- ...