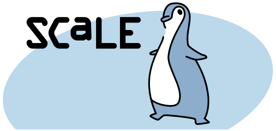
ORMs and ERDs, OMG! Forward and Reverse Engineering Databases ERDs vs Django's ORM and Migration Tools

Abe Kazemzadeh

SCALE23x, Pasadena, CA
abe.kazemzadeh AT stthomas DOT edu

2026-03-0x



UNIVERSITY OF St.Thomas | All for the Common Good™

Outline

Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
  - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
  - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
  - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
  - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
    - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
    - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

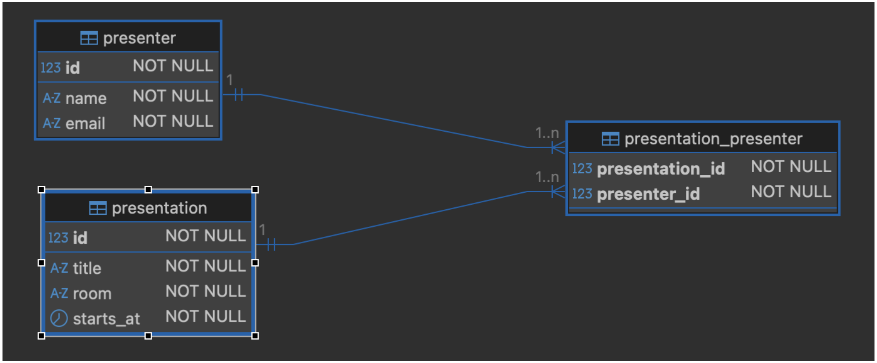
Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
    - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
- ERDs are more than just visualization tools:

Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
  - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
  - A foreign key in one table points to or "references" another table
  - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:

DBeaver ER Diagram Example



Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
  - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
  - A foreign key in one table points to or "references" another table
  - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many[1] (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:

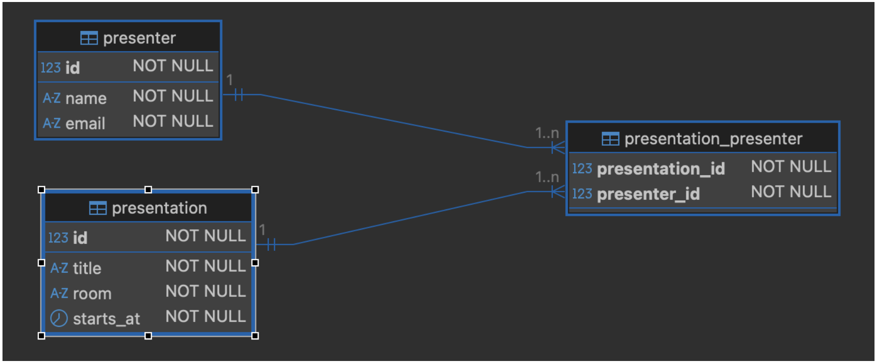
Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
    - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
    - A foreign key in one table points to or "references" another table
    - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:

Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
  - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
  - A foreign key in one table points to or "references" another table
  - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:

DBeaver ER Diagram Example



Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
  - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
  - A foreign key in one table points to or "references" another table
  - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:

Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
  - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
  - A foreign key in one table points to or "references" another table
  - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:
  - We can generate SQL code from an ERD (forward engineering), or
    We can generate an ERD from SQL code (reverse engineering)

Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
  - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
  - A foreign key in one table points to or "references" another table
  - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:
  - We can generate SQL code from an ERD (forward engineering), or
  - We can generate an ERD from SQL code (reverse engineering)

Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
  - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
  - A foreign key in one table points to or "references" another table
  - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:
  - We can generate SQL code from an ERD (forward engineering), or
  - We can generate an ERD from SQL code (reverse engineering)

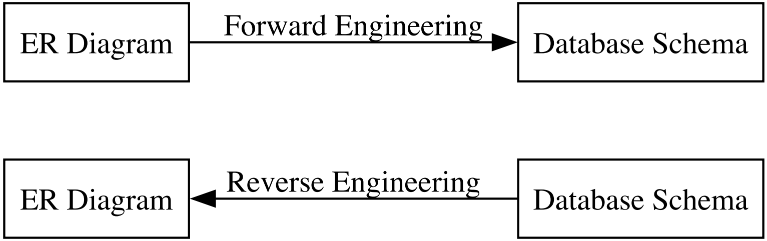
Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
  - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
  - A foreign key in one table points to or "references" another table
  - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:
  - We can generate SQL code from an ERD (forward engineering), or
  - We can generate an ERD from SQL code (reverse engineering)

Entity Relationship Diagrams

- Entity-Relationship Diagrams (ERDs) are diagrams that help to visualize a database's structure or "schema"
- Nodes represent tables/entities
    - In code, tables/entities roughly correspond to classes and rows of tables correspond to instances/objects of that class
- Edges represent relationships between tables
    - A foreign key in one table points to or "references" another table
    - Edge arrow head and tail shapes denote relationship patterns or "cardinality": one-to-one, one-to-many, or many-to-many* (many-to-many relationships are only present with logical/conceptual modeling features)
- ERDs are more than just visualization tools:
    - We can generate SQL code from an ERD (forward engineering), or
    - We can generate an ERD from SQL code (reverse engineering)

Forward and Reverse Engineering



Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
  - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
  - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

Object Relational Mapper (ORM)

- An object relational mapper (ORM) is a software library that helps store and retrieve ("map") programming language objects to and from a relational database
  - ORMs often hide the SQL details from programmers
  - Objects can have all sorts of data in them, but *relational* databases like to have tables that are dedicated to just one specific type of information ("normalization")
  - The difference in behavior between objects in a programming language and data in a relational database is called "object-relational impedance mismatch"
- I'll be using Django as an example
- Django's ORM also has a two-way transformation process

Object Relational Mapper (ORM)

- An object relational mapper (ORM) is a software library that helps store and retrieve (“map”) programming language objects to and from a relational database
  - ORMs often hide the SQL details from programmers
  - Objects can have all sorts of data in them, but *relational* databases like to have tables that are dedicated to just one specific type of information (“normalization”)
  - The difference in behavior between objects in a programming language and data in a relational database is called “object-relational impedance mismatch”
- I’ll be using Django as an example
- Django’s ORM also has a two-way transformation process

Object Relational Mapper (ORM)

- An object relational mapper (ORM) is a software library that helps store and retrieve (“map”) programming language objects to and from a relational database
  - ORMs often hide the SQL details from programmers
  - Objects can have all sorts of data in them, but *relational* databases like to have tables that are dedicated to just one specific type of information (“normalization”)
  - The difference in behavior between objects in a programming language and data in a relational database is called “object-relational impedance mismatch”
- I’ll be using Django as an example
- Django’s ORM also has a two-way transformation process

Object Relational Mapper (ORM)

- An object relational mapper (ORM) is a software library that helps store and retrieve (“map”) programming language objects to and from a relational database
    - ORMs often hide the SQL details from programmers
    - Objects can have all sorts of data in them, but *relational* databases like to have tables that are dedicated to just one specific type of information (“normalization”)
    - The difference in behavior between objects in a programming language and data in a relational database is called “object-relational impedance mismatch”
- I’ll be using Django as an example
- Django’s ORM also has a two-way transformation process
    - Take programming language code (class definitions) and generate SQL table definitions (migrate), or
    - Take database code (SQL table definitions) and generate Python class definitions (inspectdb)

Object Relational Mapper (ORM)

- An object relational mapper (ORM) is a software library that helps store and retrieve (“map”) programming language objects to and from a relational database
    - ORMs often hide the SQL details from programmers
    - Objects can have all sorts of data in them, but *relational* databases like to have tables that are dedicated to just one specific type of information (“normalization”)
    - The difference in behavior between objects in a programming language and data in a relational database is called “object-relational impedance mismatch”
- I’ll be using Django as an example
- Django’s ORM also has a two-way transformation process
    - Take programming language code (class definitions) and generate SQL table definitions (migrate), or
    - Take database code (SQL table definitions) and generate Python class definitions (inspectdb)

Object Relational Mapper (ORM)

- An object relational mapper (ORM) is a software library that helps store and retrieve (“map”) programming language objects to and from a relational database
  - ORMs often hide the SQL details from programmers
  - Objects can have all sorts of data in them, but *relational* databases like to have tables that are dedicated to just one specific type of information (“normalization”)
  - The difference in behavior between objects in a programming language and data in a relational database is called “object-relational impedance mismatch”
- I’ll be using Django as an example
- Django’s ORM also has a two-way transformation process
  - Take programming language code (class definitions) and generate SQL table definitions (migrate), or
  - Take database code (SQL table definitions) and generate Python class definitions (inspectdb)

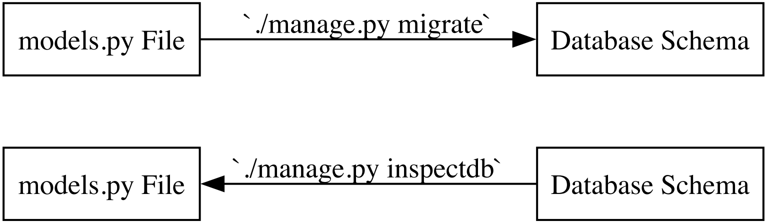
Object Relational Mapper (ORM)

- An object relational mapper (ORM) is a software library that helps store and retrieve (“map”) programming language objects to and from a relational database
  - ORMs often hide the SQL details from programmers
  - Objects can have all sorts of data in them, but *relational* databases like to have tables that are dedicated to just one specific type of information (“normalization”)
  - The difference in behavior between objects in a programming language and data in a relational database is called “object-relational impedance mismatch”
- I’ll be using Django as an example
- Django’s ORM also has a two-way transformation process
  - Take programming language code (class definitions) and generate SQL table definitions (migrate), or
  - Take database code (SQL table definitions) and generate Python class definitions (inspectdb)

Object Relational Mapper (ORM)

- An object relational mapper (ORM) is a software library that helps store and retrieve ("map") programming language objects to and from a relational database
  - ORMs often hide the SQL details from programmers
  - Objects can have all sorts of data in them, but *relational* databases like to have tables that are dedicated to just one specific type of information ("normalization")
  - The difference in behavior between objects in a programming language and data in a relational database is called "object-relational impedance mismatch"
- I'll be using Django as an example
- Django's ORM also has a two-way transformation process
  - Take programming language code (class definitions) and generate SQL table definitions (migrate), or
  - Take database code (SQL table definitions) and generate Python class definitions (inspectdb)

Django InspectDB and Migration



Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
    - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
    - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

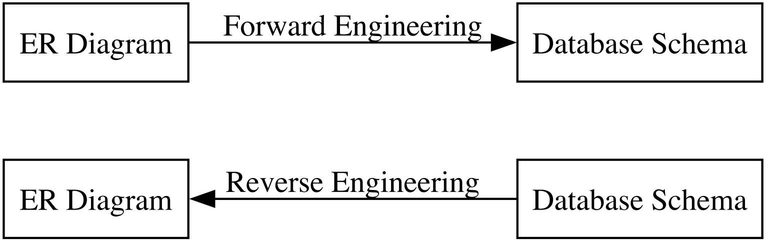
Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
    - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
    - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

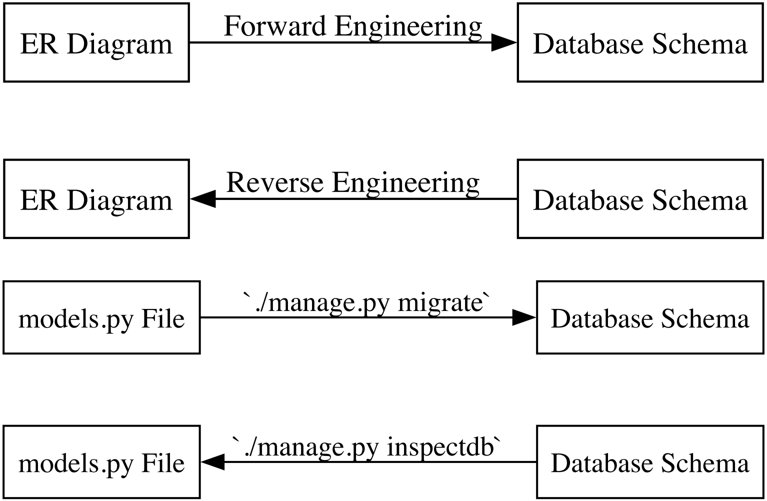
Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
  - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
  - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

Forward-Reverse Engineering and Migrate/inspectdb



Forward-Reverse Engineering and Migrate/inspectdb



Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
  - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
  - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
  - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
  - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

Motivation

- I teach a database class that uses mainly proprietary software (Oracle), but I like to use open source tools
- I saw parallels between forward and reverse engineering features in entity relationship diagram (ERD) tools and Django's object relational mapper (ORM)
  - ERD software examples: Oracle Data Modeler, MySQL Workbench, DBeaver, Erwin, PGModeler
  - ORM example: makemigrations/migrate and inspectdb in Django; other examples: Rails/ActiveRecord (Ruby), SQLAlchemy, Hibernate (Java), Entity Framework (C#)
- These ERD and ORM features have different names and aren't usually presented together but I find it interesting to do so
- It is a connection that I wish I realized earlier so hopefully it may help others
- It also helps understand other tools, like UML/PyReverse

Goals

- Describe and compare/contrast forward and reverse engineering in DBeaver to Django's migrations and inspectdb commands
- Compare and contrast ER diagramming tools
- Give a tutorial example with DBeaver and Django
- Show another parallel technique from Python's UML generation tool, `pyreverse` (from pylint package)
- Two-way communication is ideal: please ask questions and share your experiences

Goals

- Describe and compare/contrast forward and reverse engineering in DBeaver to Django's migrations and inspectdb commands
- Compare and contrast ER diagramming tools
- Give a tutorial example with DBeaver and Django
- Show another parallel technique from Python's UML generation tool, `pyreverse` (from pylint package)
- Two-way communication is ideal: please ask questions and share your experiences

Goals

- Describe and compare/contrast forward and reverse engineering in DBeaver to Django's migrations and inspectdb commands
- Compare and contrast ER diagramming tools
- Give a tutorial example with DBeaver and Django
- Show another parallel technique from Python's UML generation tool, `pyreverse` (from pylint package)
- Two-way communication is ideal: please ask questions and share your experiences

Goals

- Describe and compare/contrast forward and reverse engineering in DBeaver to Django's migrations and inspectdb commands
- Compare and contrast ER diagramming tools
- Give a tutorial example with DBeaver and Django
- Show another parallel technique from Python's UML generation tool, `pyreverse` (from pylint package)
- Two-way communication is ideal: please ask questions and share your experiences

Goals

- Describe and compare/contrast forward and reverse engineering in DBeaver to Django's migrations and inspectdb commands
- Compare and contrast ER diagramming tools
- Give a tutorial example with DBeaver and Django
- Show another parallel technique from Python's UML generation tool, `pyreverse` (from pylint package)
- Two-way communication is ideal: please ask questions and share your experiences

Overview

Outline

Terminology

- SQL: Structured Query Language, how to talk to databases
  - DDL: data definition language (creating table structure)
  - DML: data manpulation language (select/insert/update/delete)
- ER diagram: visualize databases, node ≡ table a.k.a. entity, edge≡foreign key relationship between tables
- Logical design vs relational design: is a many-to-many relationship shown as an edge or as an intersection/association table
- Forward engineering: convert/export an ER diagram into a database schema (CREATE TABLE statements), a.k.a. database modeler, SQL generation (DDL)
- Reverse engineering: convert/import a database schema (existing database) to an ER diagram. A.k.a. db metadata analysis, after-the-fact visualization

Terminology

- SQL: Structured Query Language, how to talk to databases
  - DDL: data definition language (creating table structure)
  - DML: data manpulation language (select/insert/update/delete)
- ER diagram: visualize databases, node ≡ table a.k.a. entity, edge≡foreign key relationship between tables
- Logical design vs relational design: is a many-to-many relationship shown as an edge or as an intersection/association table
- Forward engineering: convert/export an ER diagram into a database schema (CREATE TABLE statements), a.k.a. database modeler, SQL generation (DDL)
- Reverse engineering: convert/import a database schema (existing database) to an ER diagram. A.k.a. db metadata analysis, after-the-fact visualization

Terminology

- SQL: Structured Query Language, how to talk to databases
  - DDL: data definition language (creating table structure)
  - DML: data manpulation language (select/insert/update/delete)
- ER diagram: visualize databases, node ≡ table a.k.a. entity, edge≡foreign key relationship between tables
- Logical design vs relational design: is a many-to-many relationship shown as an edge or as an intersection/association table
- Forward engineering: convert/export an ER diagram into a database schema (CREATE TABLE statements), a.k.a. database modeler, SQL generation (DDL)
- Reverse engineering: convert/import a database schema (existing database) to an ER diagram. A.k.a. db metadata analysis, after-the-fact visualization

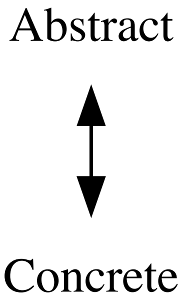
Terminology

- SQL: Structured Query Language, how to talk to databases
  - DDL: data definition language (creating table structure)
  - DML: data manpulation language (select/insert/update/delete)
- ER diagram: visualize databases, node ≡ table a.k.a. entity, edge≡foreign key relationship between tables
- Logical design vs relational design: is a many-to-many relationship shown as an edge or as an intersection/association table
- Forward engineering: convert/export an ER diagram into a database schema (CREATE TABLE statements), a.k.a. database modeler, SQL generation (DDL)
- Reverse engineering: convert/import a database schema (existing database) to an ER diagram. A.k.a. db metadata analysis, after-the-fact visualization

Terminology

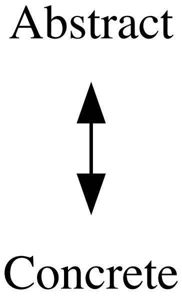
- SQL: Structured Query Language, how to talk to databases
  - DDL: data definition language (creating table structure)
  - DML: data manpulation language (select/insert/update/delete)
- ER diagram: visualize databases, node $\equiv$ table a.k.a. entity, edge$\equiv$foreign key relationship between tables
- Logical design vs relational design: is a many-to-many relationship shown as an edge or as an intersection/association table
- Forward engineering: convert/export an ER diagram into a database schema (CREATE TABLE statements), a.k.a. database modeler, SQL generation (DDL)
- Reverse engineering: convert/import a database schema (existing database) to an ER diagram. A.k.a. db metadata analysis, after-the-fact visualization

Levels of Abstraction



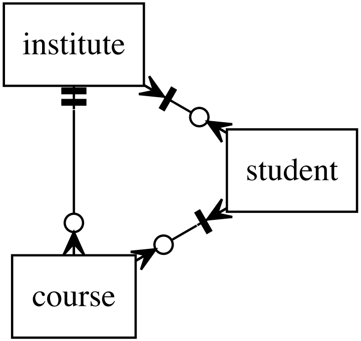
- Conceptual design/schema:
- Logical design/schema:
- Physical design/schema:

Levels of Abstraction



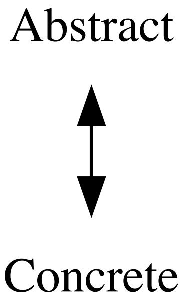
- Conceptual design/schema: only entities and relations
- Logical design/schema:
- Physical design/schema:

Crow's Foot: Conceptual



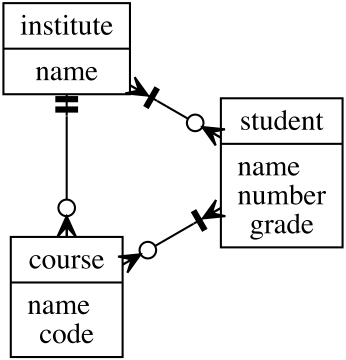
Crow's Foot Diagram
drawn by GraphViz/Circo

Levels of Abstraction



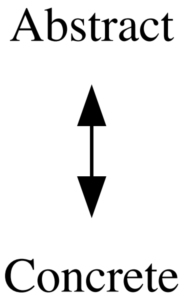
- Conceptual design/schema: only entities and relations
- Logical design/schema: entities/relations plus attributes
- Physical design/schema:

ER Logical



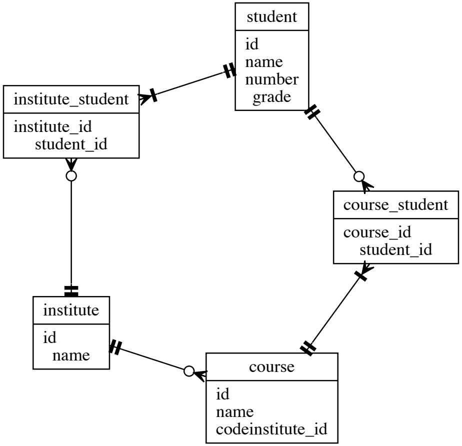
Crow's Foot Diagram
drawn by GraphViz/Circo

Levels of Abstraction



- Conceptual design/schema: only entities and relations
- Logical design/schema: entities/relations plus attributes
- Physical design/schema: entities/relations/attributes plus keys, types, and many-to-many "join" tables (a.k.a. associative entities)

ER Physical



Crow's Foot Diagram
drawn by GraphViz/Circo

Terminology

- SQL: Structured Query Language, how to talk to databases
  - DDL: data definition language (creating table structure)
  - DML: data manpulation language (select/insert/update/delete)
- ER diagram: visualize databases, node ≡ table a.k.a. entity, edge≡foreign key relationship between tables
- Logical design vs relational design: is a many-to-many relationship shown as an edge or as an intersection/association table
- Forward engineering: convert/export an ER diagram into a database schema (CREATE TABLE statements), a.k.a. database modeler, SQL generation (DDL)
- Reverse engineering: convert/import a database schema (existing database) to an ER diagram. A.k.a. db metadata analysis, after-the-fact visualization

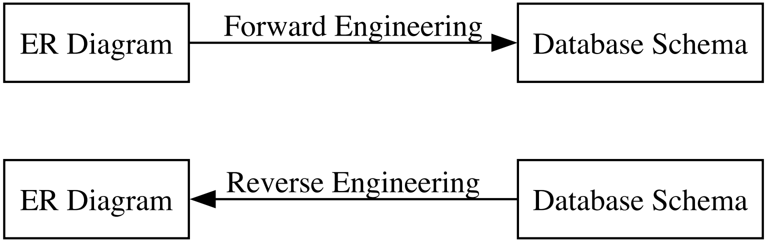
Terminology

- SQL: Structured Query Language, how to talk to databases
  - DDL: data definition language (creating table structure)
  - DML: data manpulation language (select/insert/update/delete)
- ER diagram: visualize databases, node ≡ table a.k.a. entity, edge≡foreign key relationship between tables
- Logical design vs relational design: is a many-to-many relationship shown as an edge or as an intersection/association table
- Forward engineering: convert/export an ER diagram into a database schema (CREATE TABLE statements), a.k.a. database modeler, SQL generation (DDL)
- Reverse engineering: convert/import a database schema (existing database) to an ER diagram. A.k.a. db metadata analysis, after-the-fact visualization

Terminology

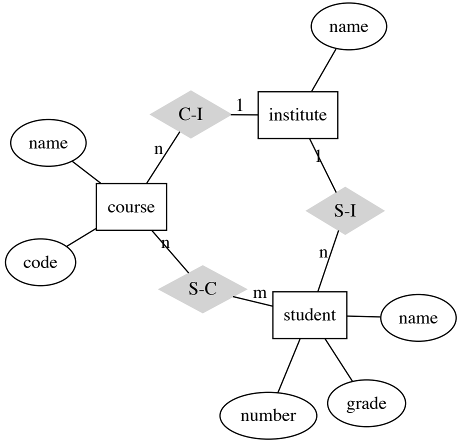
- SQL: Structured Query Language, how to talk to databases
  - DDL: data definition language (creating table structure)
  - DML: data manpulation language (select/insert/update/delete)
- ER diagram: visualize databases, node $\equiv$ table a.k.a. entity, edge$\equiv$foreign key relationship between tables
- Logical design vs relational design: is a many-to-many relationship shown as an edge or as an intersection/association table
- Forward engineering: convert/export an ER diagram into a database schema (CREATE TABLE statements), a.k.a. database modeler, SQL generation (DDL)
- Reverse engineering: convert/import a database schema (existing database) to an ER diagram. A.k.a. db metadata analysis, after-the-fact visualization

Forward and Reverse Engineering



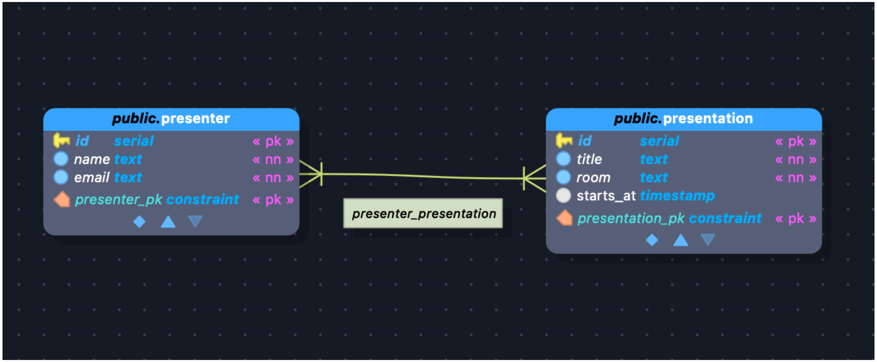
$$\begin{tikzcd}
\boxed{\text{ER Diagram}} \arrow[r, "\text{Forward Engineering}"] & \boxed{\text{Database Schema}} \\
\boxed{\text{ER Diagram}} & \boxed{\text{Database Schema}} \arrow[l, "\text{Reverse Engineering}"']
\end{tikzcd}$$

Older Peter Chen-Style ER Diagram

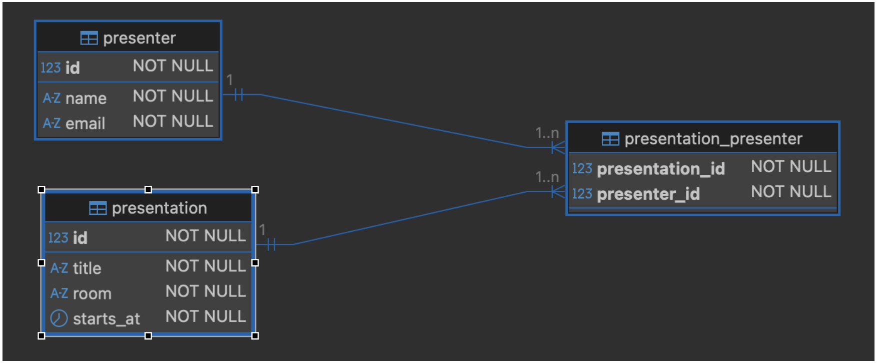


Entity Relation Diagram
drawn by GraphViz/NEATO

pgModeler ER Diagram (Many-to-Many/Conceptual Design)



DBeaver ER Diagram Example (Many-to-Many Relationship Converted to an Intersection/Association/Bridge/Junction/Join Table)



Examples

- DBeaver: supporter of SCaLE, supports many different DBs, paid version includes forward engineering, DB-centric (not as many modeling features
- MySQL Workbench: specfic to MySQL
- pgModeler: specific to Postgres, modeling-centric (not bundled with an SQL IDE)
- Oracle Data Modeler (free but not open source), modeling-centric
- SchemaSpy: just reverse engineering
- Erwin (not free or open source)
- DBSchema (not free or opensource, SaaS)

Examples

- DBeaver: supporter of SCaLE, supports many different DBs, paid version includes forward engineering, DB-centric (not as many modeling features
- MySQL Workbench: specfic to MySQL
- pgModeler: specific to Postgres, modeling-centric (not bundled with an SQL IDE)
- Oracle Data Modeler (free but not open source), modeling-centric
- SchemaSpy: just reverse engineering
- Erwin (not free or open source)
- DBSchema (not free or opensource, SaaS)

Examples

- DBeaver: supporter of SCaLE, supports many different DBs, paid version includes forward engineering, DB-centric (not as many modeling features
- MySQL Workbench: specfic to MySQL
- pgModeler: specific to Postgres, modeling-centric (not bundled with an SQL IDE)
- Oracle Data Modeler (free but not open source), modeling-centric
- SchemaSpy: just reverse engineering
- Erwin (not free or open source)
- DBSchema (not free or opensource, SaaS)

Examples

- DBeaver: supporter of SCaLE, supports many different DBs, paid version includes forward engineering, DB-centric (not as many modeling features
- MySQL Workbench: specfic to MySQL
- pgModeler: specific to Postgres, modeling-centric (not bundled with an SQL IDE)
- Oracle Data Modeler (free but not open source), modeling-centric
- SchemaSpy: just reverse engineering
- Erwin (not free or open source)
- DBSchema (not free or opensource, SaaS)

Examples

- DBeaver: supporter of SCaLE, supports many different DBs, paid version includes forward engineering, DB-centric (not as many modeling features
- MySQL Workbench: specfic to MySQL
- pgModeler: specific to Postgres, modeling-centric (not bundled with an SQL IDE)
- Oracle Data Modeler (free but not open source), modeling-centric
- SchemaSpy: just reverse engineering
- Erwin (not free or open source)
- DBSchema (not free or opensource, SaaS)

Examples

- DBeaver: supporter of SCaLE, supports many different DBs, paid version includes forward engineering, DB-centric (not as many modeling features
- MySQL Workbench: specfic to MySQL
- pgModeler: specific to Postgres, modeling-centric (not bundled with an SQL IDE)
- Oracle Data Modeler (free but not open source), modeling-centric
- SchemaSpy: just reverse engineering
- Erwin (not free or open source)
- DBSchema (not free or opensource, SaaS)

Examples

- DBeaver: supporter of SCaLE, supports many different DBs, paid version includes forward engineering, DB-centric (not as many modeling features
- MySQL Workbench: specfic to MySQL
- pgModeler: specific to Postgres, modeling-centric (not bundled with an SQL IDE)
- Oracle Data Modeler (free but not open source), modeling-centric
- SchemaSpy: just reverse engineering
- Erwin (not free or open source)
- DBSchema (not free or opensource, SaaS)

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
- Fwd/rev engineering features not always available
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
- Fwd/rev engineering features not always available
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
- Fwd/rev engineering features not always available
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
- Fwd/rev engineering features not always available
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
  - E.g. MySQL workbench
- Fwd/rev engineering features not always available
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
  - E.g. MySQL workbench
- Fwd/rev engineering features not always available
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
  - E.g. MySQL workbench
- Fwd/rev engineering features not always available
  - sometimes only available on "enterprise" editions
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
  - E.g. MySQL workbench
- Fwd/rev engineering features not always available
  - sometimes only available on "enterprise" editions
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
  - E.g. MySQL workbench
- Fwd/rev engineering features not always available
  - sometimes only available on "enterprise" editions
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
  - E.g. MySQL workbench
- Fwd/rev engineering features not always available
  - sometimes only available on “enterprise” editions
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
  - E.g. MySQL workbench
- Fwd/rev engineering features not always available
  - sometimes only available on “enterprise” editions
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Pros and Cons of ER Diagram Fwd/Rev Engineering

Pros

- Visual representation
- GUI
- Management of detail
- Ease for beginners
- May or may not be integrated in a DB IDE
- Keeps relational DB theory (as opposed to object-oriented design)

Cons

- Tools often specfic to DBs
    - E.g. MySQL workbench
- Fwd/rev engineering features not always available
    - sometimes only available on “enterprise” editions
- Not designed to be integrated into applications (tools, not libraries)
- May or may not be integrated in a DB IDE

Outline

Django and ORM Overview

- Django is a Python server-side web framework
- We will be considering a part of Django, the ORM (object relational mapper)
- This component deals with mapping objects, defined in Python code, to a relational database
- Other ORMs:

Django and ORM Overview

- Django is a Python server-side web framework
- We will be considering a part of Django, the ORM (object relational mapper)
- This component deals with mapping objects, defined in Python code, to a relational database
- Other ORMs:
  - Rails/ActiveRecord (Ruby)
  - SQLAlchemy (Python),
  - Hibernate (Java)
  - Entity Framework (C#)

Django and ORM Overview

- Django is a Python server-side web framework
- We will be considering a part of Django, the ORM (object relational mapper)
- This component deals with mapping objects, defined in Python code, to a relational database
- Other ORMs:
  - Rails/ActiveRecord (Ruby)
  - SQLAlchemy (Python),
  - Hibernate (Java)
  - Entity Framework (C#)

Django and ORM Overview

- Django is a Python server-side web framework
- We will be considering a part of Django, the ORM (object relational mapper)
- This component deals with mapping objects, defined in Python code, to a relational database
- Other ORMs:
  - Rails/ActiveRecord (Ruby)
  - SQLAlchemy (Python),
  - Hibernate (Java)
  - Entity Framework (C#)

Django and ORM Overview

- Django is a Python server-side web framework
- We will be considering a part of Django, the ORM (object relational mapper)
- This component deals with mapping objects, defined in Python code, to a relational database
- Other ORMs:
  - Rails/ActiveRecord (Ruby)
  - SQLAlchemy (Python),
  - Hibernate (Java)
  - Entity Framework (C#)

Django and ORM Overview

- Django is a Python server-side web framework
- We will be considering a part of Django, the ORM (object relational mapper)
- This component deals with mapping objects, defined in Python code, to a relational database
- Other ORMs:
  - Rails/ActiveRecord (Ruby)
  - SQLAlchemy (Python),
  - Hibernate (Java)
  - Entity Framework (C#)

Django and ORM Overview

- Django is a Python server-side web framework
- We will be considering a part of Django, the ORM (object relational mapper)
- This component deals with mapping objects, defined in Python code, to a relational database
- Other ORMs:
  - Rails/ActiveRecord (Ruby)
  - SQLAlchemy (Python),
  - Hibernate (Java)
  - Entity Framework (C#)

Django and ORM Overview

- Django is a Python server-side web framework
- We will be considering a part of Django, the ORM (object relational mapper)
- This component deals with mapping objects, defined in Python code, to a relational database
- Other ORMs:
    - Rails/ActiveRecord (Ruby)
    - SQLAlchemy (Python),
    - Hibernate (Java)
    - Entity Framework (C#)

Django Migrations

- Roughly equivalent to forward engineering
    - But includes version info, data/DML in addition to schema/structure/DDL

Django Migrations

- Roughly equivalent to forward engineering
    - But includes version info, data/DML in addition to schema/structure/DDL

Django Inspect DB

- inspectdb is used when we have an existing database that we don't want to be managed by Django
  - The normal approach is that Django creates and manages the database completely, but sometimes you may want to integrate Django with a database that already exists
- Roughly equivalent to reverse engineering

Django Inspect DB

- inspectdb is used when we have an existing database that we don't want to be managed by Django
  - The normal approach is that Django creates and manages the database completely, but sometimes you may want to integrate Django with a database that already exists
- Roughly equivalent to reverse engineering

Django Inspect DB

- inspectdb is used when we have an existing database that we don't want to be managed by Django
    - The normal approach is that Django creates and manages the database completely, but sometimes you may want to integrate Django with a database that already exists
- Roughly equivalent to reverse engineering

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
- Prioritizes object-oriented design over strict relational database design (“leaky abstraction”)

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
  - Stores migrations in the DB and in Python code that can be checked into version control
    Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
  - Stores migrations in the DB and in Python code that can be checked into version control
  - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
  - Stores migrations in the DB and in Python code that can be checked into version control
  - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
    - Stores migrations in the DB and in Python code that can be checked into version control
    - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
  - Stores migrations in the DB and in Python code that can be checked into version control
  - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
  - Stores migrations in the DB and in Python code that can be checked into version control
  - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
  - Migrations and inspectdb are primarily CLI-driven and code-based
  - *There is some support in PyCharm and a general VS Code Django plugin
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
  - Stores migrations in the DB and in Python code that can be checked into version control
  - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
  - Migrations and inspectdb are primarily CLI-driven and code-based
  - *There is some support in PyCharm and a general VS Code Django plugin
- Prioritizes object-oriented design over strict relational database design (“leaky abstraction”)

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
  - Stores migrations in the DB and in Python code that can be checked into version control
  - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
  - Migrations and inspectdb are primarily CLI-driven and code-based
  - *There is some support in PyCharm and a general VS Code Django plugin
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
    - Stores migrations in the DB and in Python code that can be checked into version control
    - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
    - Migrations and inspectdb are primarily CLI-driven and code-based
    - *There is some support in PyCharm and a general VS Code Django plugin
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
  - Stores migrations in the DB and in Python code that can be checked into version control
  - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
  - Migrations and inspectdb are primarily CLI-driven and code-based
  - *There is some support in PyCharm and a general VS Code Django plugin
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Pros and Cons of Django ORM Migrations/Inspectdb

Pros

- Cross-database support
- Django is designed to be used as a library
- Command-line scripting support
- Does more than just fwd/rev engineering schemas/DDL
    - Stores migrations in the DB and in Python code that can be checked into version control
    - Can migrate data, not only schema

Cons

- Not as beginner friendly as ER diagrams
- No GUI or visualizations
- Doesn't have IDE support comparable to ERDs*
    - Migrations and inspectdb are primarily CLI-driven and code-based
    - *There is some support in PyCharm and a general VS Code Django plugin
- Prioritizes object-oriented design over strict relational database design ("leaky abstraction")

Outline

Plan

1. Forward engineering: convert an ERD to SQL CREATE TABLE statements
   - We will show DBeaver ERD converted to PostgreSQL
2. Django sqlmigrate/migrate (ORM equivalent of forward engineering)
   - Classes in Python code converted to SQL CREATE TABLE STATEMENTS in SQLite
3. Reverse engineering
   - Django DB schema in SQLite converted to an ERD in DBeaver
4. Django inspectdb (ORM equivalent of reverse engineering)
   - DBeaver-generated database design in Postgres converted to Python classes

Plan

1. Forward engineering: convert an ERD to SQL CREATE TABLE statements
   - We will show DBeaver ERD converted to PostgreSQL
2. Django sqlmigrate/migrate (ORM equivalent of forward engineering)
   - Classes in Python code converted to SQL CREATE TABLE STATEMENTS in SQLite
3. Reverse engineering
   - Django DB schema in SQLite converted to an ERD in DBeaver
4. Django inspectdb (ORM equivalent of reverse engineering)
   - DBeaver-generated database design in Postgres converted to Python classes

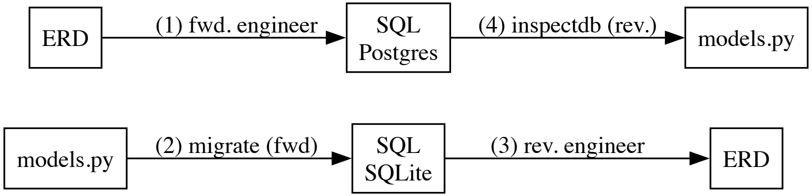
Plan

1. Forward engineering: convert an ERD to SQL CREATE TABLE statements
   - We will show DBeaver ERD converted to PostgreSQL
2. Django sqlmigrate/migrate (ORM equivalent of forward engineering)
   - Classes in Python code converted to SQL CREATE TABLE STATEMENTS in SQLite
3. Reverse engineering
   - Django DB schema in SQLite converted to an ERD in DBeaver
4. Django inspectdb (ORM equivalent of reverse engineering)
   - DBeaver-generated database design in Postgres converted to Python classes

Plan

1. Forward engineering: convert an ERD to SQL CREATE TABLE statements
   - We will show DBeaver ERD converted to PostgreSQL
2. Django sqlmigrate/migrate (ORM equivalent of forward engineering)
   - Classes in Python code converted to SQL CREATE TABLE STATEMENTS in SQLite
3. Reverse engineering
   - Django DB schema in SQLite converted to an ERD in DBeaver
4. Django inspectdb (ORM equivalent of reverse engineering)
   - DBeaver-generated database design in Postgres converted to Python classes

Demo Diagram



Plan

1. **Forward engineering: convert an ERD to SQL CREATE TABLE statements**
   - **We will show DBeaver ERD converted to PostgreSQL**
2. Django sqlmigrate/migrate (ORM equivalent of forward engineering)
   - Classes in Python code converted to SQL CREATE TABLE STATEMENTS in SQLite
3. Reverse engineering
   - Django DB schema in SQLite converted to an ERD in DBeaver
4. Django inspectdb (ORM equivalent of reverse engineering)
   - DBeaver-generated database design in Postgres converted to Python classes

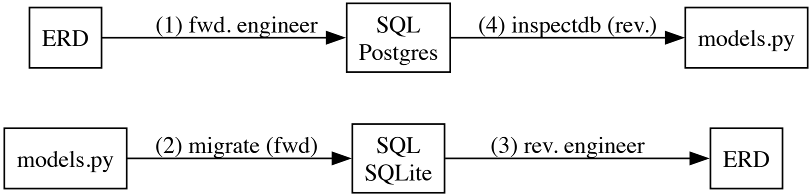
Forward Engineering: ERD to SQL

- Create a presenter table
  - id, name, and email columns
- Create a presentation table
  - id, title, room, and starts_at columns
- Create a presentation_presenter table (many-to-many mapping)
  - presentation_id and presenter_id columns
- See the generated SQL code

Plan

1. Forward engineering: convert an ERD to SQL CREATE TABLE statements
   - We will show DBeaver ERD converted to PostgreSQL
2. **Django sqlmigrate/migrate (ORM equivalent of forward engineering)**
   - **Classes in Python code converted to SQL CREATE TABLE STATEMENTS in SQLite**
3. Reverse engineering
   - Django DB schema in SQLite converted to an ERD in DBeaver
4. Django inspectdb (ORM equivalent of reverse engineering)
   - DBeaver-generated database design in Postgres converted to Python classes

Demo Diagram



Django Migrate

- Install django into a virtual environment
- Create a project called "scale" and an app called "talks"
- Create classes for presenter and presentation in `talks/models.py`
  - presentation will have a many-to-many field
- Add the "talks" app to the INSTALLED_APPS in `scale/settings.py`
- Make migrations, view SQL, then migrate
- Optional (if time): register the models in the admin panel

Django Setup

```
uv pip install django # install django

# start project
django-admin  startproject scale

# start app
cd edu
django-admin startapp talks

# create models by editing registration/models.py

# add 'talks' to INSTALLED_APPS in scale/settings.py

# make migrations (writes to app/migrations/0001_initial.py)
./manage.py makemigrations

# optionally, view sql
./manage.py sqlmigrate talks 0001

# migrate the DB
./manage.py migrate
```

Models.py

```
from django.db import models
# Create your models here.

class Presenter(models.Model):
    name = models.CharField(max_length=256)
    email = models.CharField(max_length=256)

class Presentations(models.Model):
    title = models.CharField(max_length=50)
    room = models.CharField(max_length=256)
    starts_at = models.DateTimeField()
    presenters = models.ManyToManyField(Presenter)
```

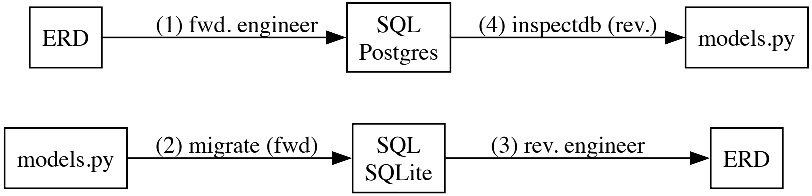
(Optional, If Time) Register the Models with the Django Admin Panel

```
# registration/admin.py
from django.contrib import admin
from registration.models import Presenter, Presentation
# or from .models import Presenter, Presentation
# Register your models here.
admin.site.register(Presenter)
admin.site.register(Presentation)
```

Plan

1. Forward engineering: convert an ERD to SQL CREATE TABLE statements
   - We will show DBeaver ERD converted to PostgreSQL
2. Django sqlmigrate/migrate (ORM equivalent of forward engineering)
   - Classes in Python code converted to SQL CREATE TABLE STATEMENTS in SQLite
3. **Reverse engineering**
   - **Django DB schema in SQLite converted to an ERD in DBeaver**
4. Django inspectdb (ORM equivalent of reverse engineering)
   - DBeaver-generated database design in Postgres converted to Python classes

Demo Diagram



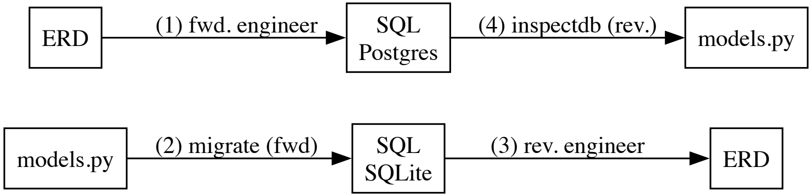
Reverse Engineering: View Django-Generated Tables in DBeaver

- In DBeaver, connect to the Django-generated SQLite database and view the ERD
- Notice: there are many tables created, not just our model classes, but lots of other ones
  - Django is "batteries included": it includes users, access permissions, content-types, etc
  - The tables have automatically generated ids, and the many-to-many table is automatically generated (and has an id)
  - Names are converted from Python class naming conventions to SQL naming conventions

Plan

1. Forward engineering: convert an ERD to SQL CREATE TABLE statements
   - We will show DBeaver ERD converted to PostgreSQL
2. Django sqlmigrate/migrate (ORM equivalent of forward engineering)
   - Classes in Python code converted to SQL CREATE TABLE STATEMENTS in SQLite
3. Reverse engineering
   - Django DB schema in SQLite converted to an ERD in DBeaver
4. **Django inspectdb (ORM equivalent of reverse engineering)**
   - **DBeaver-generated database design in Postgres converted to Python classes**

Demo Diagram



Django's InspectDB: Generating Python Classes from SQL Tables

```
uv pip install psycopg2 # install postgres triver

# in registration/admin.py,  connect to Postgres instead of SQLite
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
       "NAME": "postgres",
       "USER": "kaze7539",
#     "PASSWORD": "mypassword",
       "HOST": "127.0.0.1",
       "PORT": "5432",     } }

# in commmand line,  run
./manage.py inspectdb
```

Bonus: Pyreverse

- Pyreverse is a part of PyLint
- Instead of reverse engineering from SQL to ERD, pyreverse generates UML diagrams from Python code
    - Another type of "reverse engineering"
    - This is very useful for visualizing large projects with complex object-oriented inheritance

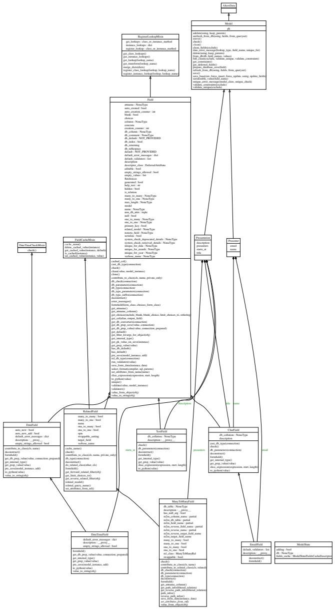
Pyreverse

```
uv pip install pylint  # install pylint module

# in command line,  run
pyreverse --all-ancestors --show-associated 1 talks/models.py

# convert dot to svg
dot -T svg -O classes.dot
```

Pyreverse Generated UML Diagram



Outline

Summary

- We discussed the similarities and differences between Django's migration features and ER diagrams forward and reverse engineering features
  - Think of forward and reverse engineering more broadly
- We discussed different ERD software
- We saw how we can use Django to convert between SQL and Python code
- We also saw pyreverse
- AI statement:

Summary

- We discussed the similarities and differences between Django's migration features and ER diagrams forward and reverse engineering features
  - Think of forward and reverse engineering more broadly
- We discussed different ERD software
- We saw how we can use Django to convert between SQL and Python code
- We also saw `pyreverse`
- AI statement:

Summary

- We discussed the similarities and differences between Django's migration features and ER diagrams forward and reverse engineering features
  - Think of forward and reverse engineering more broadly
- We discussed different ERD software
- We saw how we can use Django to convert between SQL and Python code
- We also saw `pyreverse`
- AI statement:

Summary

- We discussed the similarities and differences between Django's migration features and ER diagrams forward and reverse engineering features
  - Think of forward and reverse engineering more broadly
- We discussed different ERD software
- We saw how we can use Django to convert between SQL and Python code
- We also saw `pyreverse`
- AI statement:
  - Visualization (reverse engineering) may be a way to better understand AI generated code
    Code generation (forward engineering) can be seen as an early form of AI (80's AI: expert systems)

Summary

- We discussed the similarities and differences between Django's migration features and ER diagrams forward and reverse engineering features
  - Think of forward and reverse engineering more broadly
- We discussed different ERD software
- We saw how we can use Django to convert between SQL and Python code
- We also saw `pyreverse`
- AI statement:
  - Visualization (reverse engineering) may be a way to better understand AI generated code
  - Code generation (forward engineering) can be seen as an early form of AI (80's AI: expert systems)

Summary

- We discussed the similarities and differences between Django's migration features and ER diagrams forward and reverse engineering features
  - Think of forward and reverse engineering more broadly
- We discussed different ERD software
- We saw how we can use Django to convert between SQL and Python code
- We also saw `pyreverse`
- AI statement:
  - Visualization (reverse engineering) may be a way to better understand AI generated code
  - Code generation (forward engineering) can be seen as an early form of AI (80's AI: expert systems)

Summary

- We discussed the similarities and differences between Django's migration features and ER diagrams forward and reverse engineering features
  - Think of forward and reverse engineering more broadly
- We discussed different ERD software
- We saw how we can use Django to convert between SQL and Python code
- We also saw `pyreverse`
- AI statement:
  - Visualization (reverse engineering) may be a way to better understand AI generated code
  - Code generation (forward engineering) can be seen as an early form of AI (80's AI: expert systems)

Summary

- We discussed the similarities and differences between Django's migration features and ER diagrams forward and reverse engineering features
    - Think of forward and reverse engineering more broadly
- We discussed different ERD software
- We saw how we can use Django to convert between SQL and Python code
- We also saw `pyreverse`
- AI statement:
    - Visualization (reverse engineering) may be a way to better understand AI generated code
    - Code generation (forward engineering) can be seen as an early form of AI (80's AI: expert systems)

Questions

- Any questions or comments?