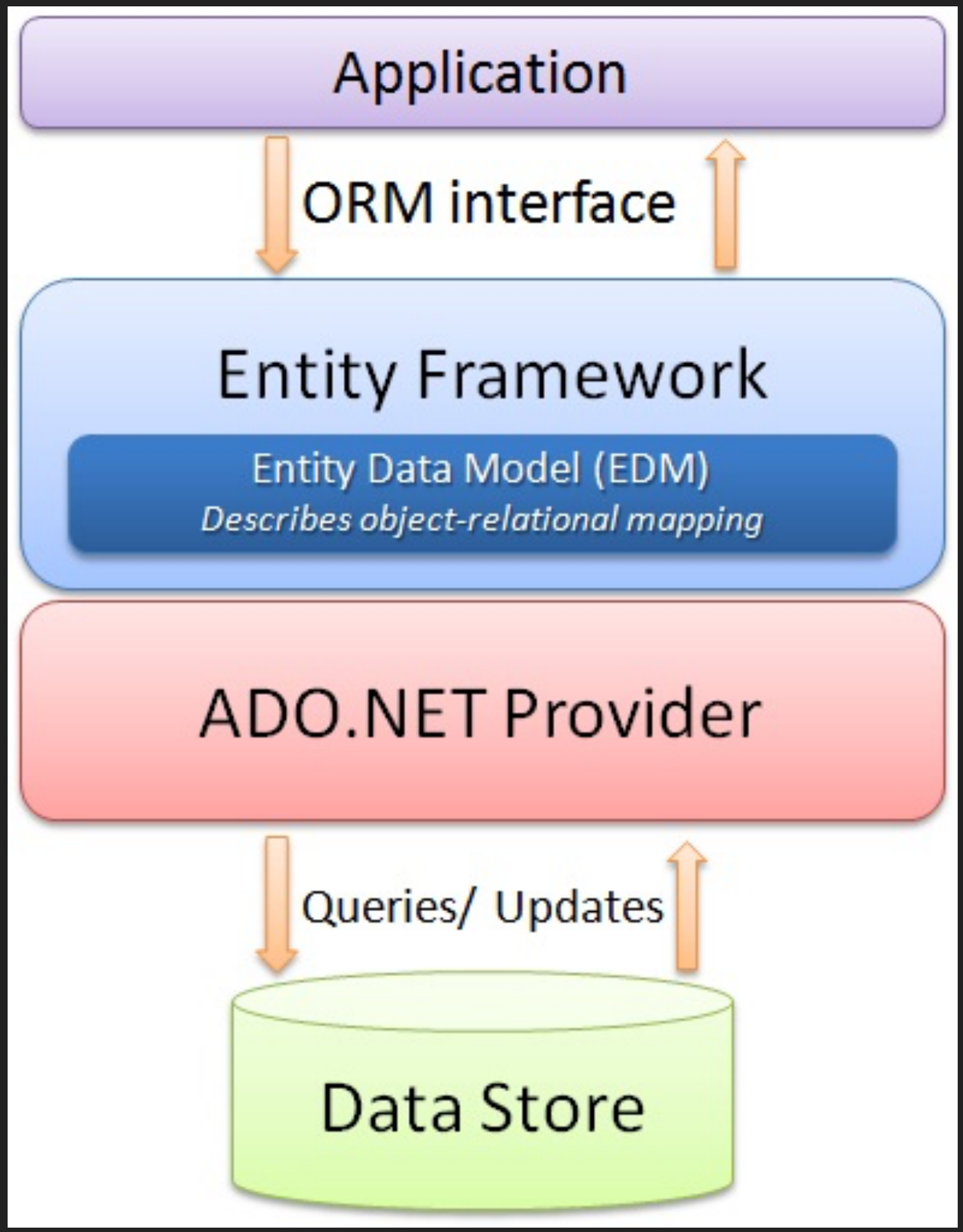
ENTITY FRAMEWORK

ENTITY FRAMEWORK

- Object Relational Mapper (ORM)
- Creates Models, Views and Controllers for the object model
- Part of ADO.NET

WHAT IS ENTITY FRAMEWORK?



OBJECT RELATIONAL MODEL

- System to convert data between incompatible types
- DB Entities and Relationships -> Classes
- Deals (primarily) with the data model

DATA MODEL VS BUSINESS/PROCESS MODEL

- Data Model
  - How data is stored, retrieved and updated
  - Maps back to a physical data store
- Business/Process Model
  - Rules governing how data is stored, retrieved and updated
  - Maps back to business rules

DATA MODEL VS BUSINESS/PROCESS MODEL

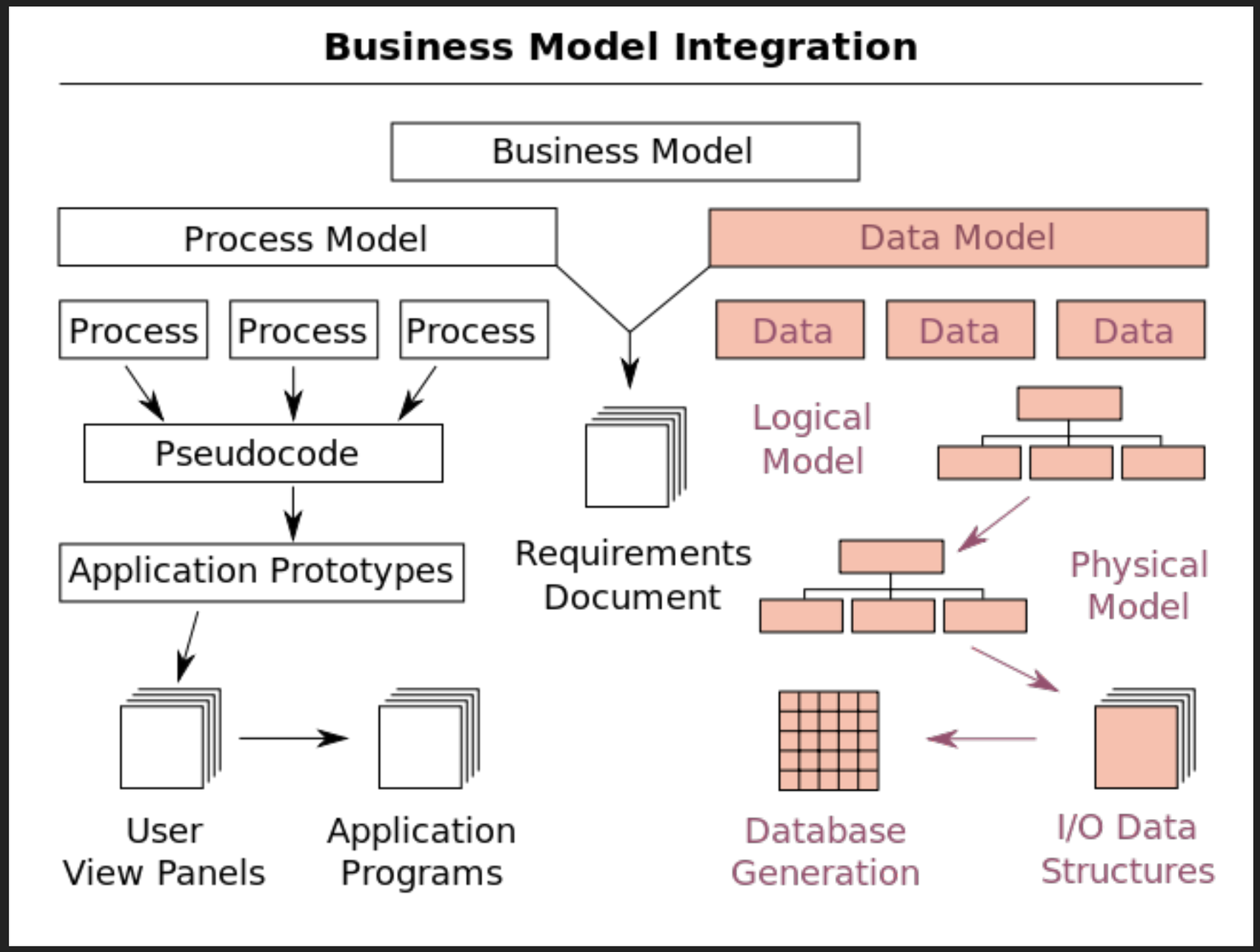
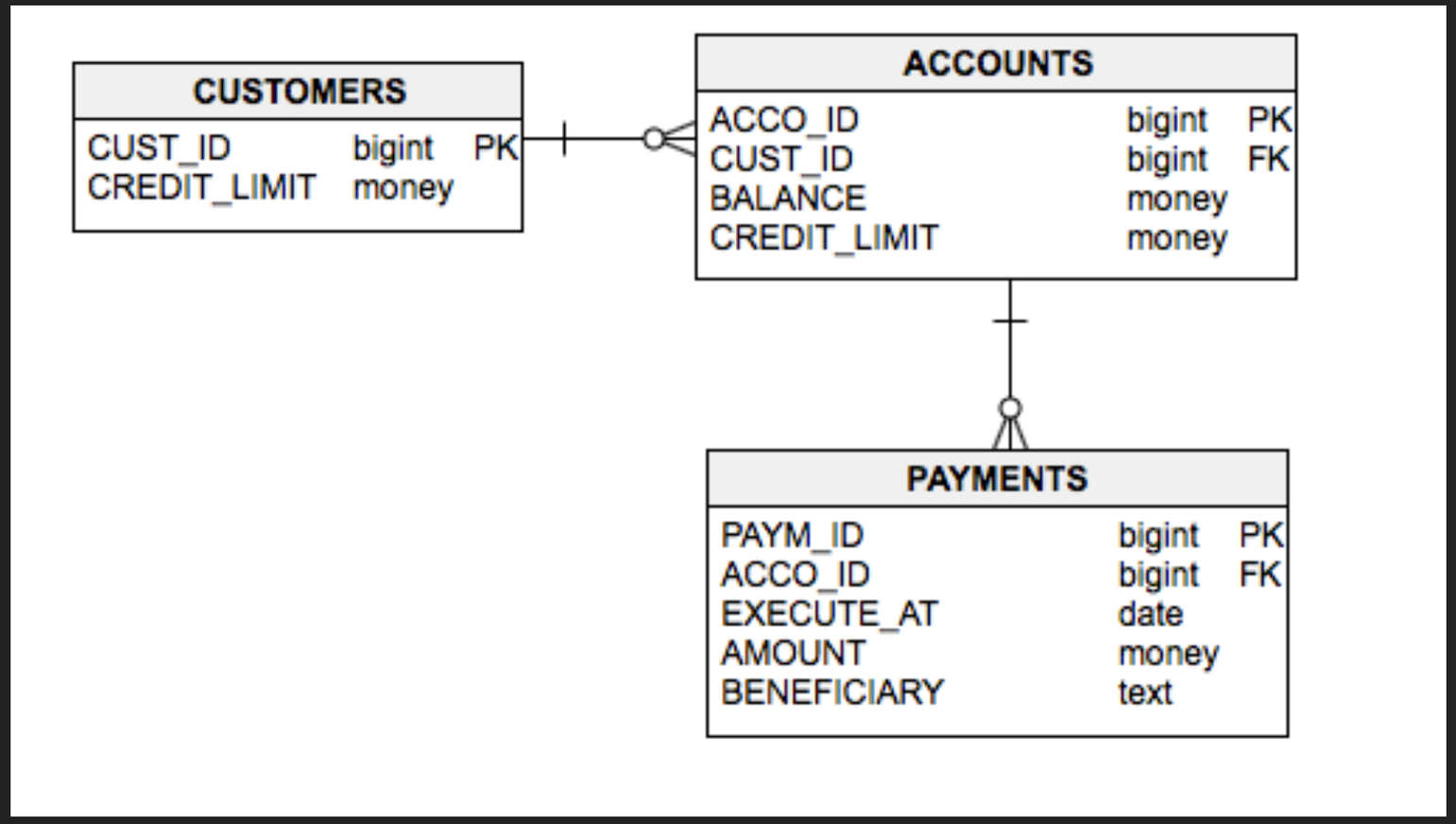


Image c/o Wikipedia

DATA MODELS

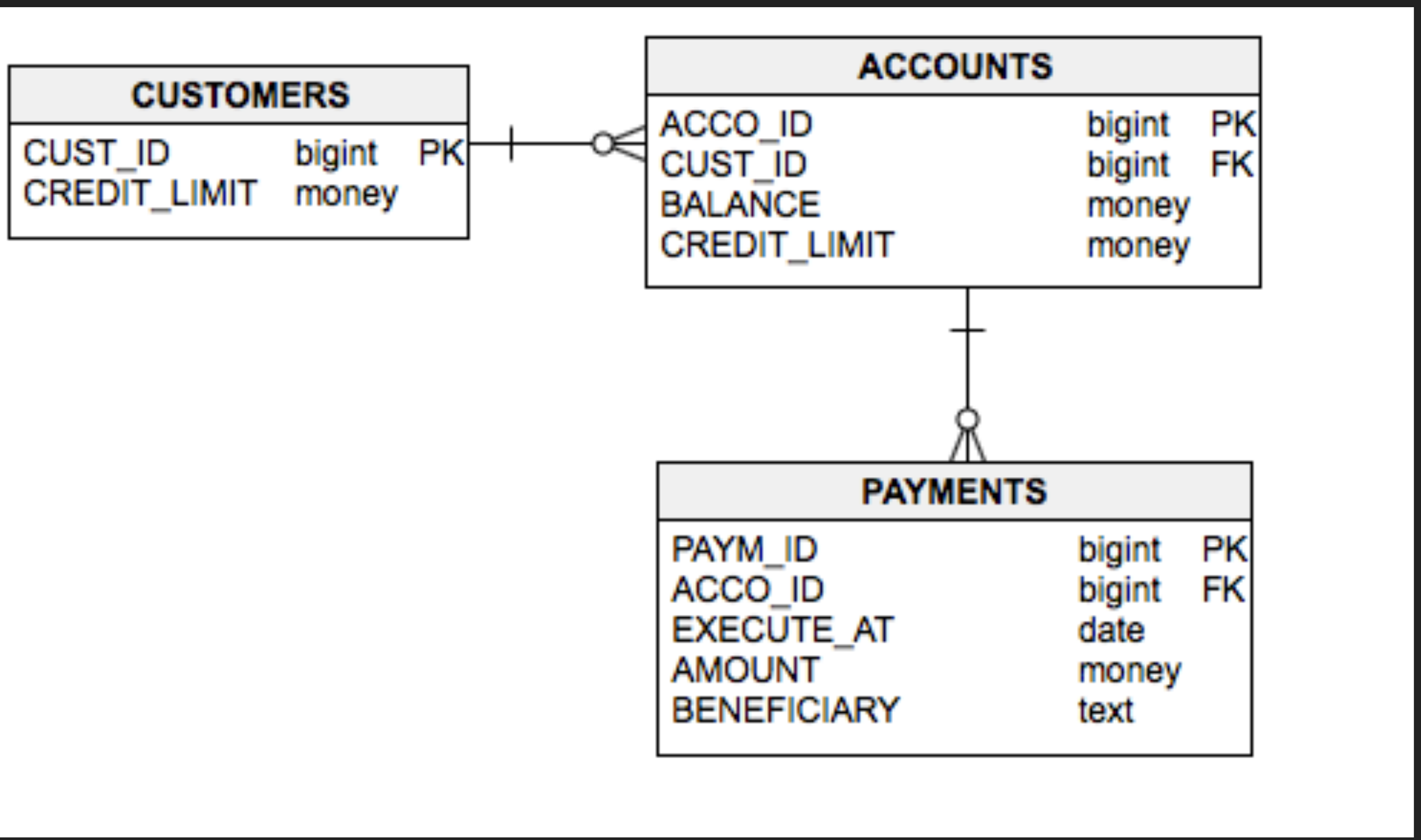
- The Relational Database representation is an ERD
- Includes the Data Access Layer



BUSINESS/PROCESS MODEL

- Models business rules/logic. UML.
- Enforces constraints and rules on how data is retrieved, entered and viewed
- Common scenarios include:
  - Business Logic
  - Authorisation
  - Security

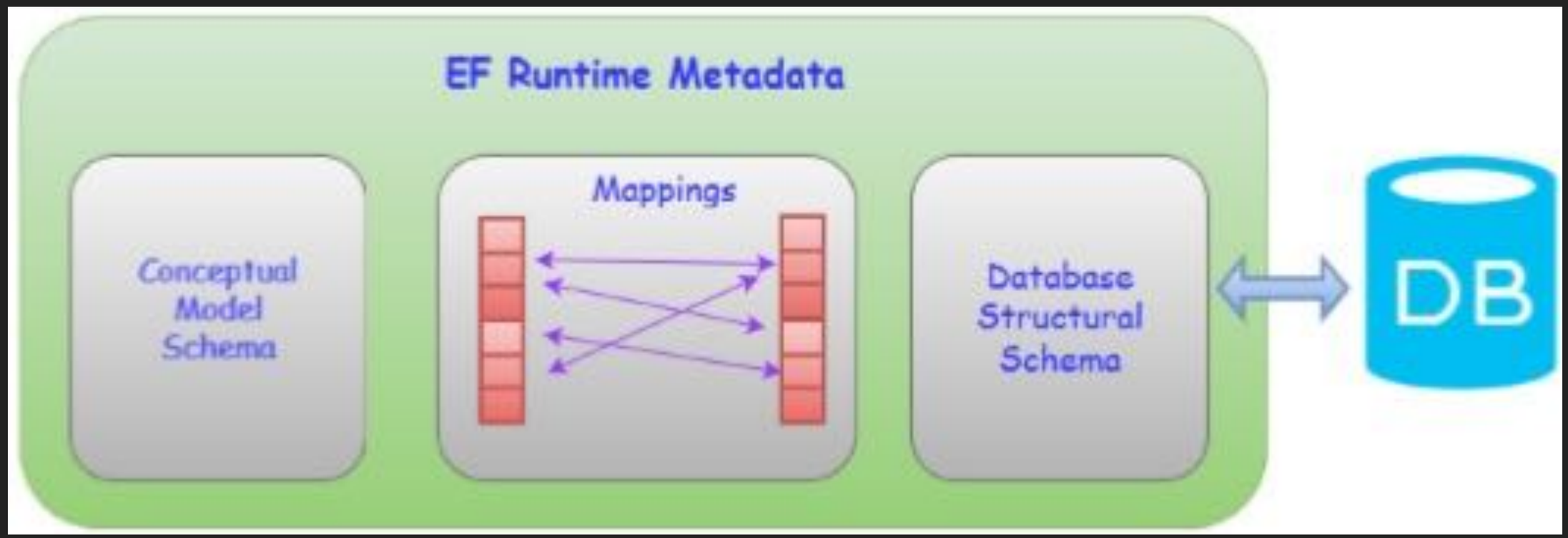
BUSINESS/PROCESS MODEL



WHAT DOES EF DO?

- Convert a database into classes
- Convert business objects into database tables and relationships
- Create business objects using a designer

- Maps DB schema into a conceptual model.
- Mappings can be edited so that a table can be mapped to multiple classes

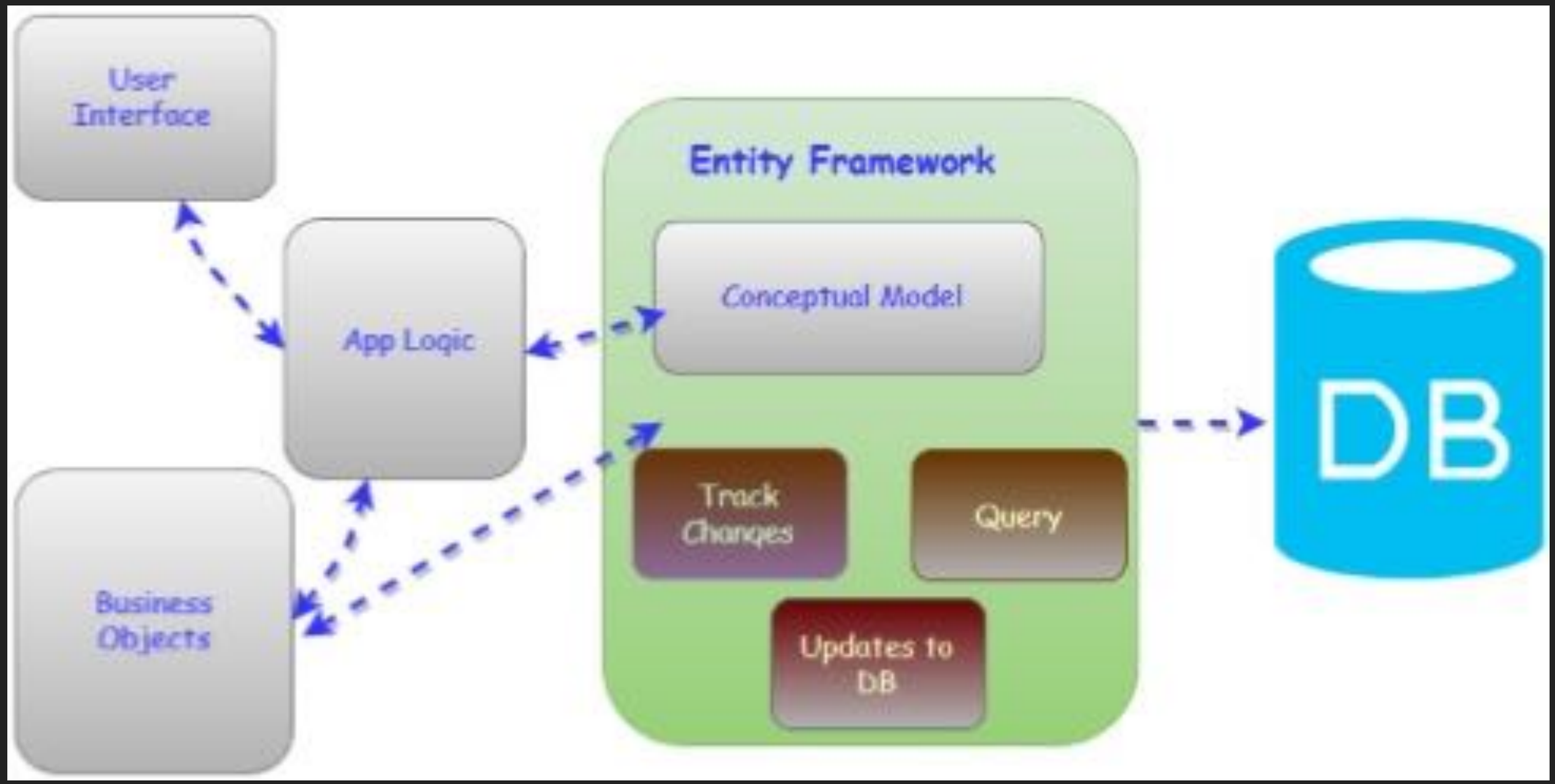


WHAT DOES EF DO?

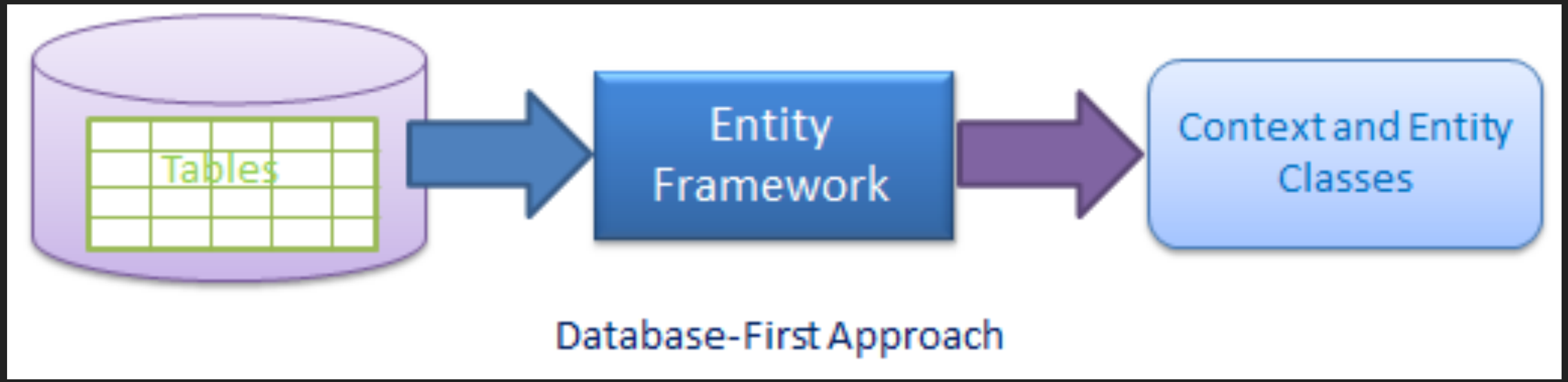
- Creates DAL
  - Handles queries, changes and updates
- Creates Data Model
- Can create simple views for CRUD
- Can incorporate all of the above into MVC
- Uses Linq-to-Entities for queries

CONCEPTUAL MODEL

- In EF we are interfacing with the conceptual model
- A model of the objects in the application
- Not interfacing directly with the database, or even a database handler

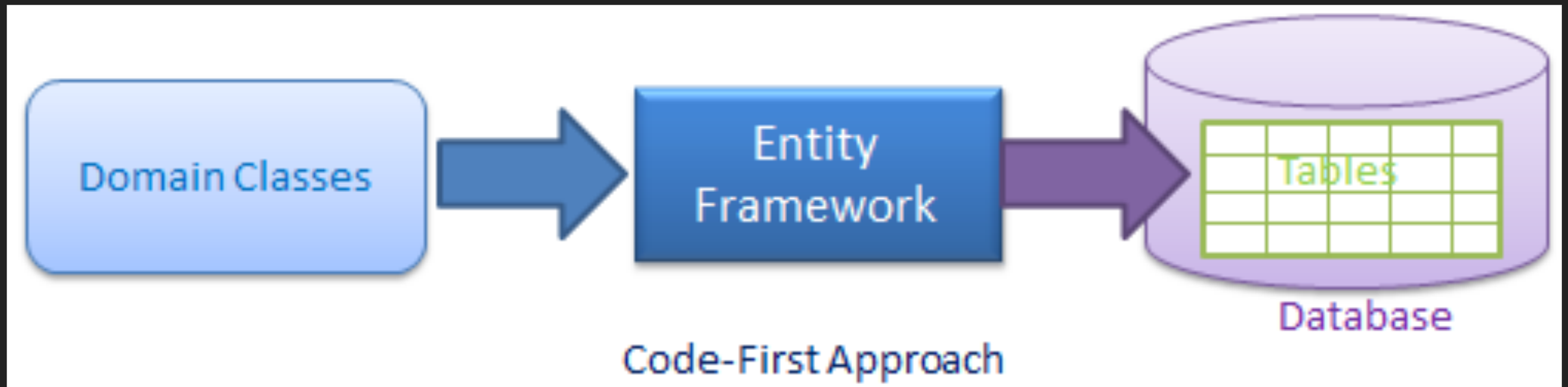

User Interface
App Logic
Business Objects
Entity Framework
Conceptual Model
Track Changes
Query
Updates to DB
DB

DATABASE FIRST

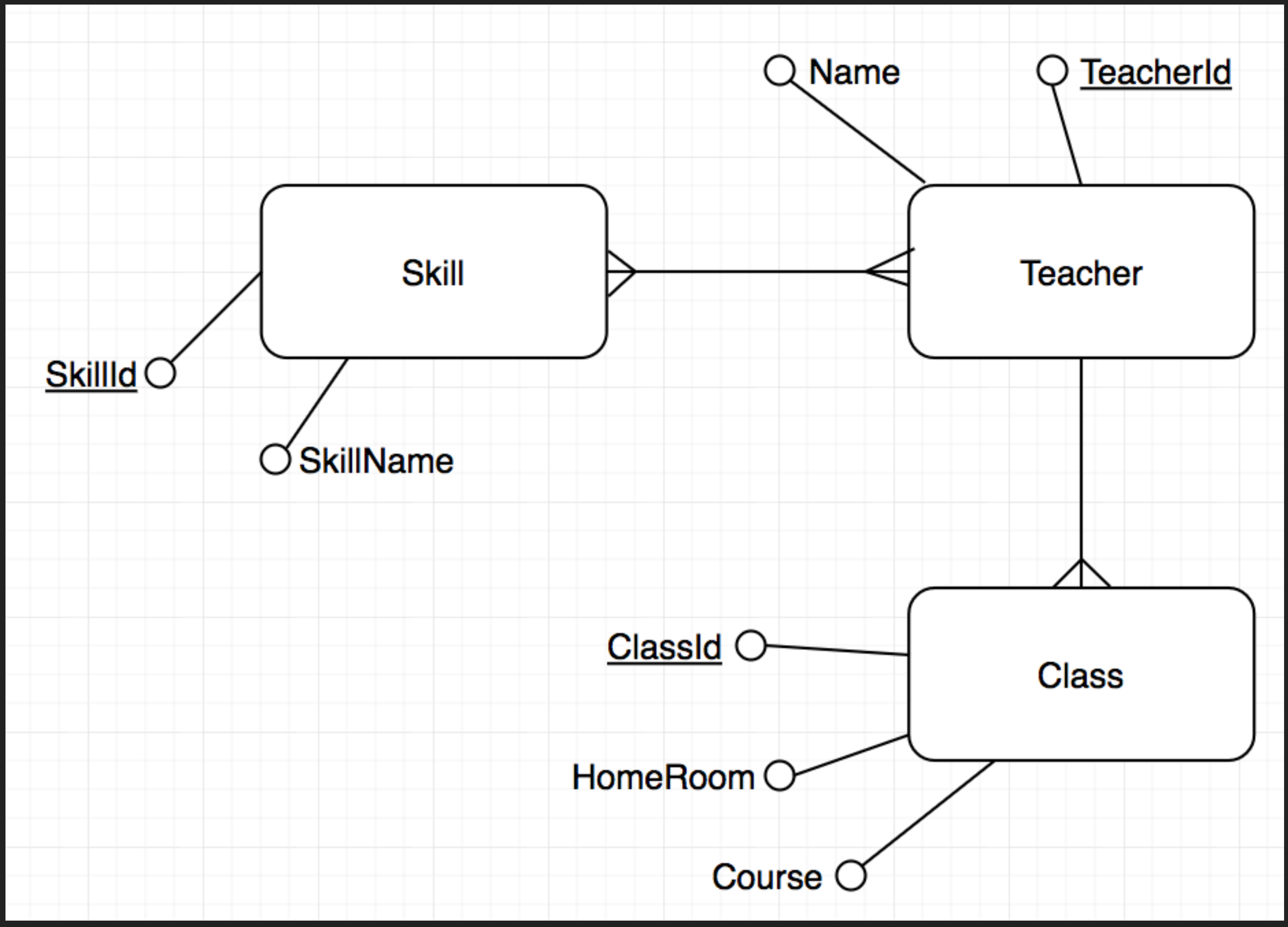


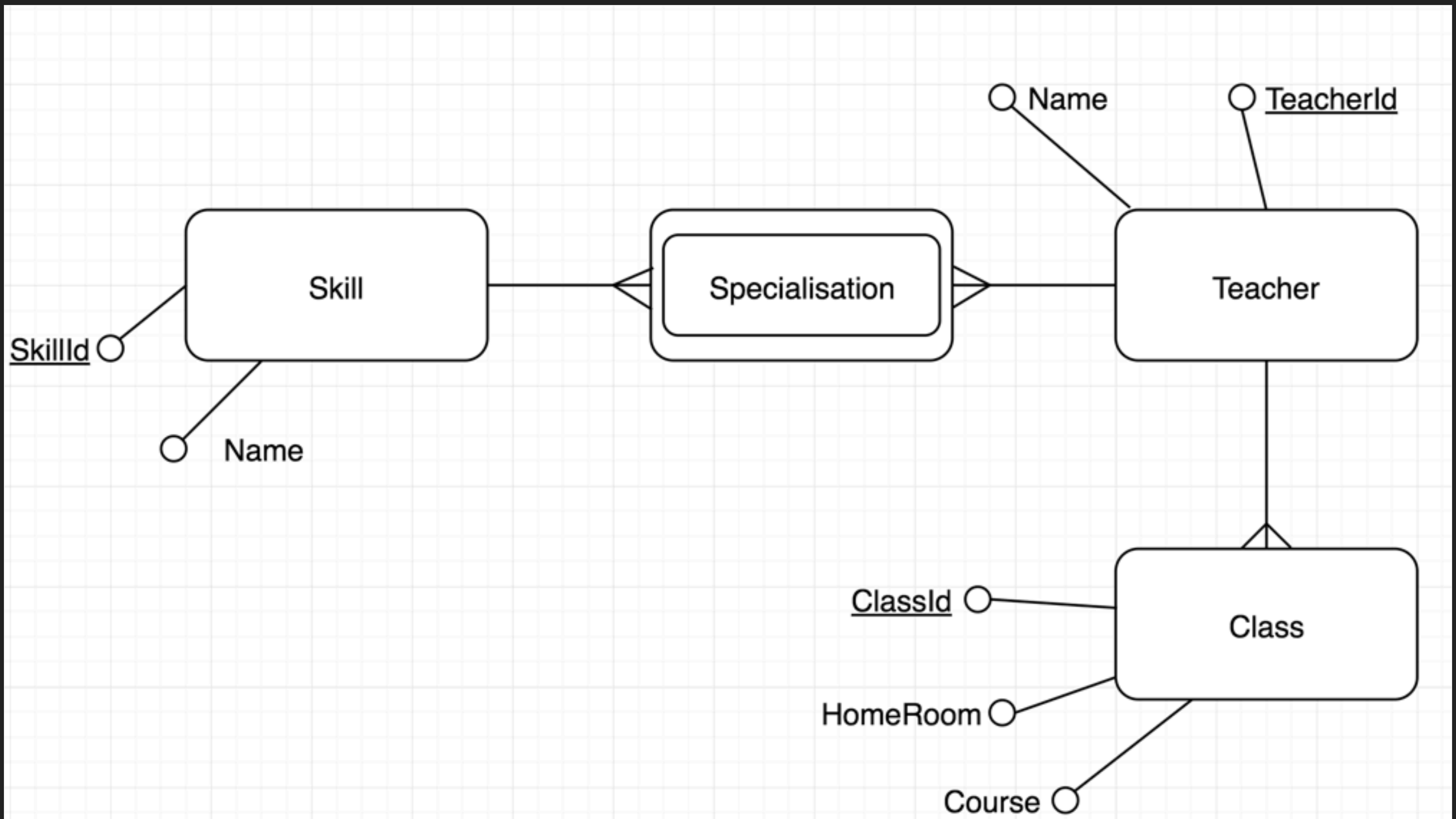
- Turns tables and relationships into conceptual model and a database context
- Able to handle views and stored procedures

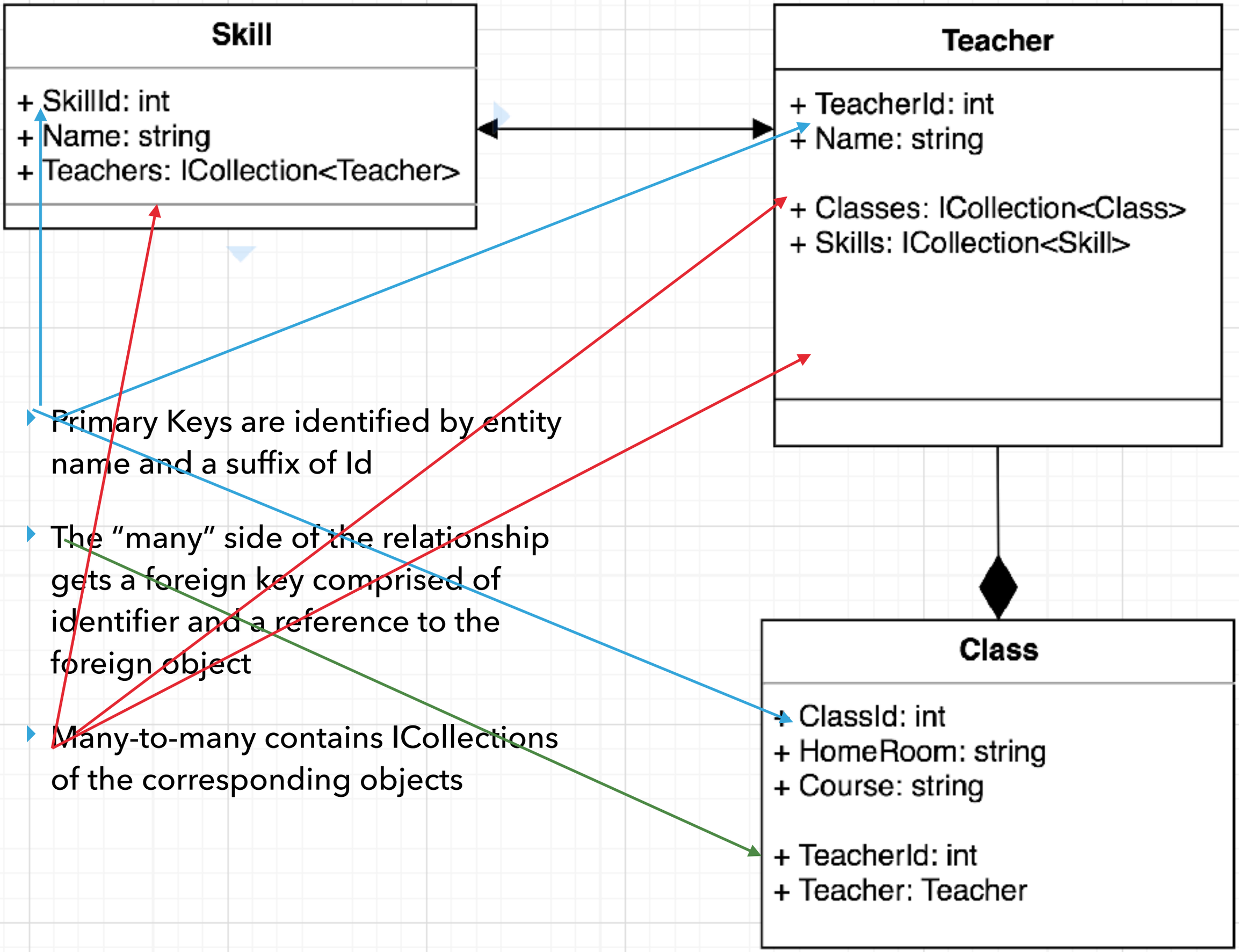
CODE FIRST



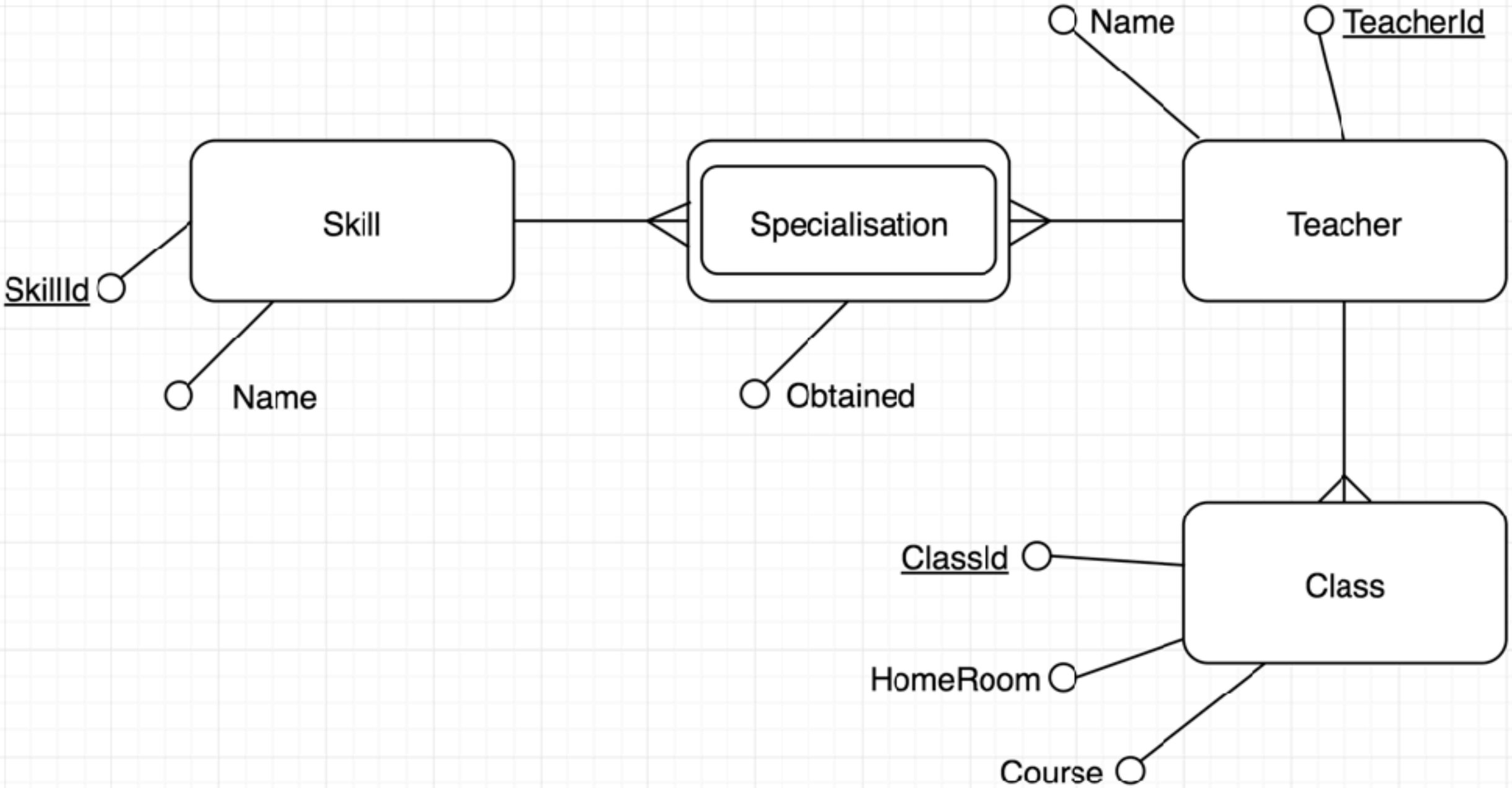
- Takes class definitions and creates a relational database from them
- Able to create all types of relationship


Name
TeacherId
Skill
Teacher
SkillId
SkillName
ClassId
Class
HomeRoom
Course

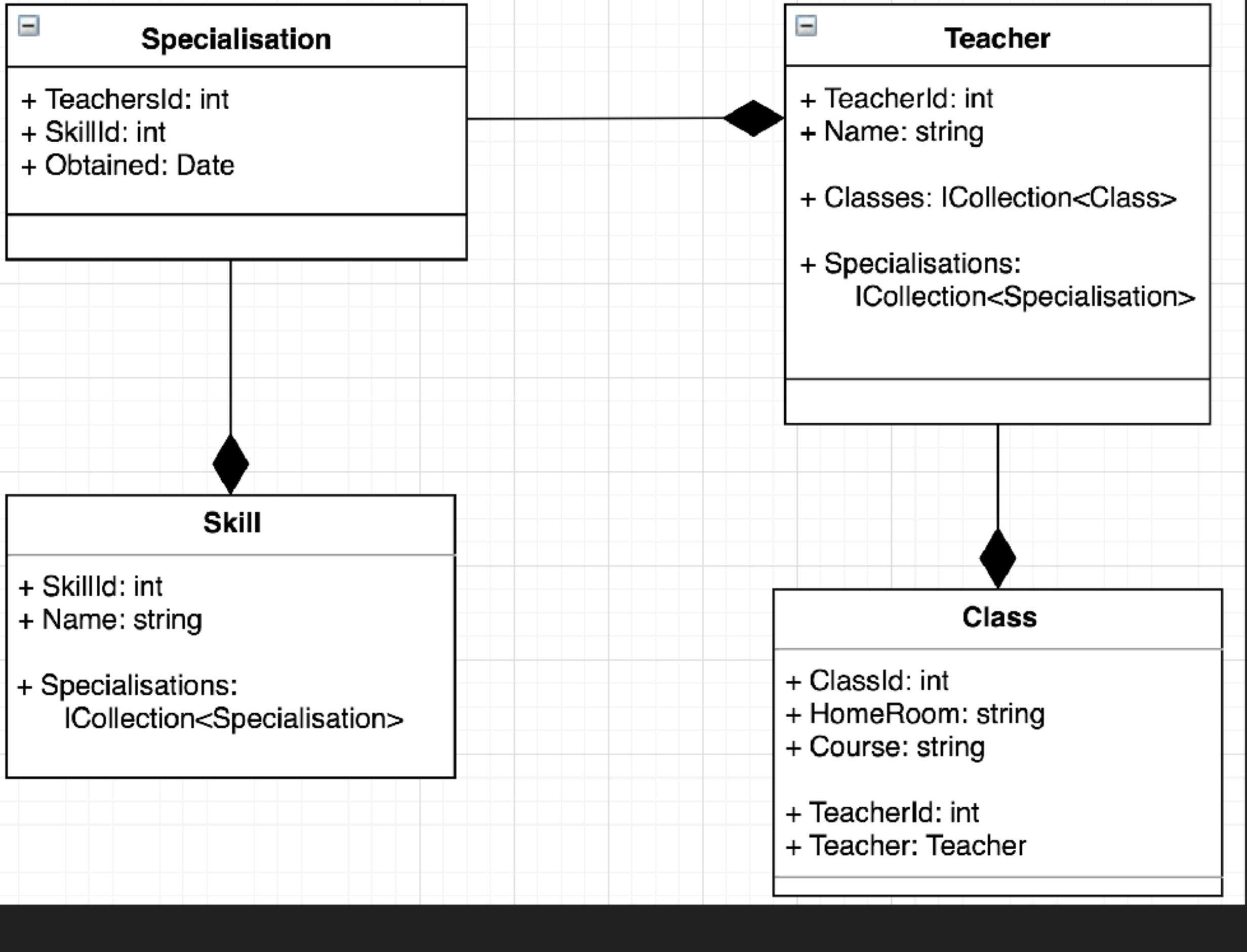

Name
TeacherId
Skill
Specialisation
Teacher
SkillId
Name
ClassId
Class
HomeRoom
Course


Skill
+ SkillId: int
+ Name: string
+ Teachers: ICollection<Teacher>
Teacher
+ TeacherId: int
+ Name: string
+ Classes: ICollection<Class>
+ Skills: ICollection<Skill>
Primary Keys are identified by entity name and a suffix of Id
The “many” side of the relationship gets a foreign key comprised of identifier and a reference to the foreign object
Many-to-many contains ICollections of the corresponding objects
Class
+ ClassId: int
+ HomeRoom: string
+ Course: string
+ TeacherId: int
+ Teacher: Teacher

MANY-TO-MANY WITH EXTRA ATTRIBUTE



- Creates the lookup table as a seperate class


Specialisation
+ TeachersId: int
+ SkillId: int
+ Obtained: Date
Teacher
+ TeacherId: int
+ Name: string
+ Classes: ICollection<Class>
+ Specialisations:
ICollection<Specialisation>
Skill
+ SkillId: int
+ Name: string
+ Specialisations:
ICollection<Specialisation>
Class
+ ClassId: int
+ HomeRoom: string
+ Course: string
+ TeacherId: int
+ Teacher: Teacher