Systems Analysis & Design
Data Persistence

CIS641

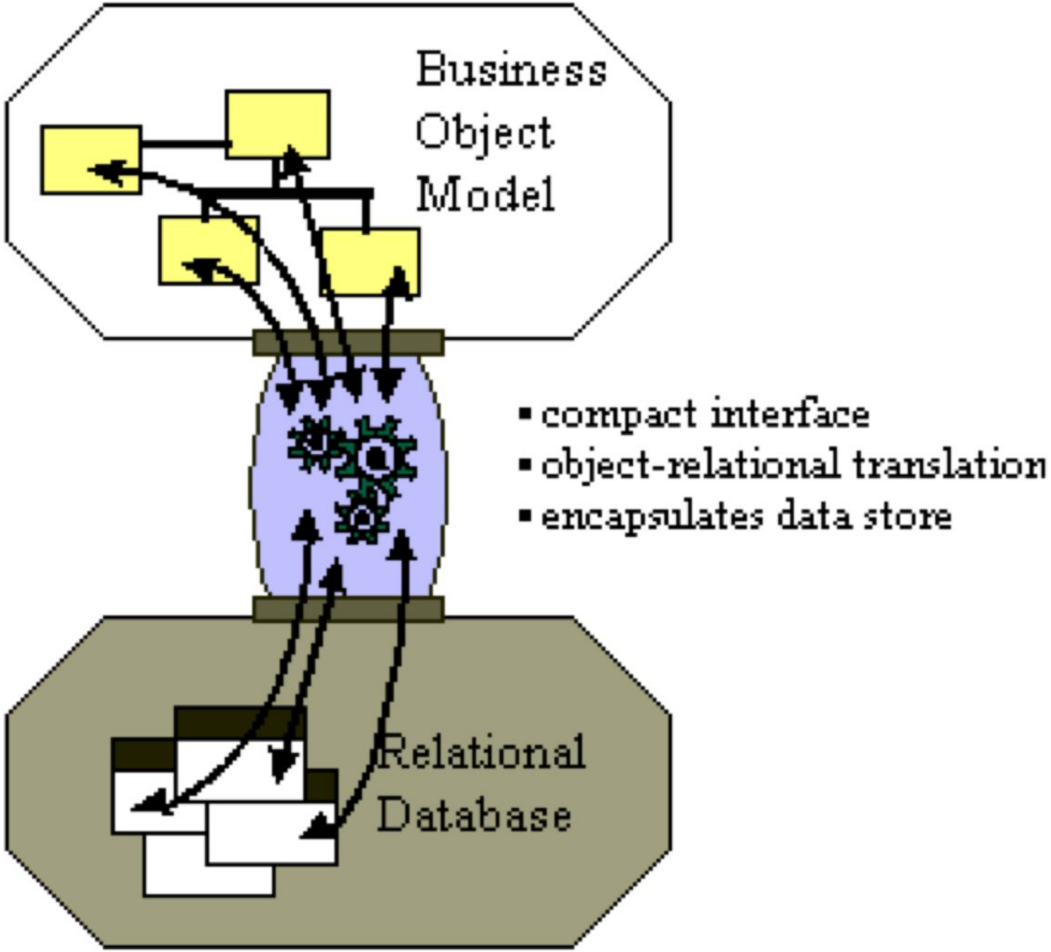
Erik Fredericks // frederer@gvsu.edu

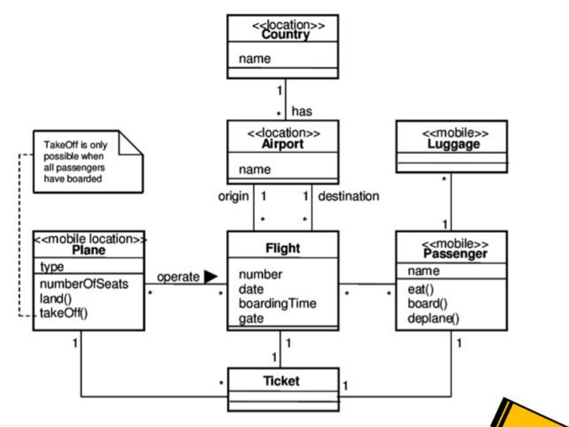
Adapted from materials provided by Gregory Schymik and the textbook (Systems Analysis and Design 5th/6th Ed.)

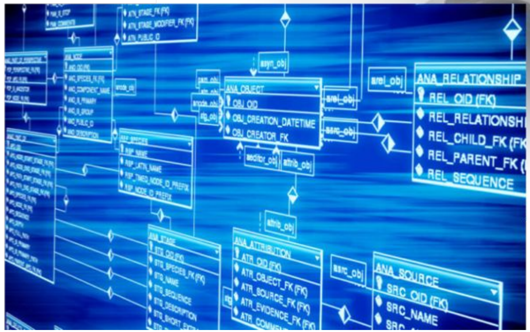
CHAPTER 9

Questions on class design

Example


Business
Object
Model
▪ compact interface
▪ object-relational translation
▪ encapsulates data store
Relational
Database


<<location>>
Country
name
1
*
has
<<location>>
Airport
name
origin
1
1
destination
*
*
<<mobile>>
Luggage
*
1
TakeOff is only possible when all passengers have boarded
<<mobile location>>
Plane
type
numberOfSeats
land()
takeOff()
operate
*
*
Flight
number
date
boardingTime
gate
*
*
<<mobile>>
Passenger
name
eat()
board()
deplane()
1
1
1
1
Ticket
*
1


ANA_RELATIONSHIP
REL_OID (FK)
REL_CHILD_FK (FK
REL_PARENT_FK (
REL_SEQUENCE
ANA_OBJECT
OBJ_OID
OBJ_CREATION_DATETIME
OBJ_CREATOR_FK
ANA_SOURCE
SRC_OID (FK)
SRC_NAME
arel_obj
attrib_obj

Introductory information for you all to peruse

Applications are of little use without data

- Data must be stored and accessed efficiently

Data management layer includes:

- Data access and manipulation logic
- Storage design

Four-step design approach:

- Selecting the format of the storage
- Mapping problem-domain objects to object persistence format
- Optimizing the object persistence format
- Designing the data access & manipulation classes

Or...

How do we make things *stick* that we have been spending all this time designing?

Object persistence formats

Files (sequential and random access)

Object-oriented databases

Object-relational databases

Relational databases

“NoSQL” data stores

Let's start with **files**

Sequential access files

- Operations (read, write and search) are conducted one record after another (in sequence)
- Efficient for report writing
- Inefficient for searching (an average of 50% of records have to be accessed for each search)
- Unordered files add records to the end of the file
- Ordered files are sorted, but additions & deletions require additional maintenance

Random access files

- Efficient for operations (read, write and search)
- Inefficient for report writing

Application file types

Master Files

- Store core information (e.g., order and customer data)
- Usually held for long periods
- Changes require new programs

Look-up files (e.g., zip codes with city and state names)

Transaction files

- Information used to update a master file
- Can be deleted once master file is updated

Audit file—records data before & after changes

History file—archives of past transactions

Relational databases

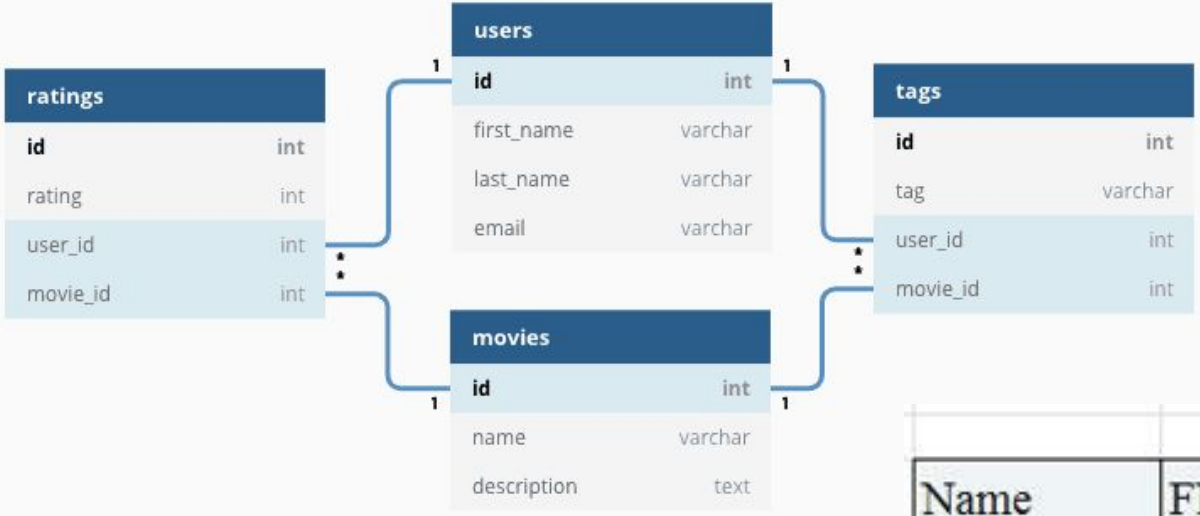
Most popular (?) way to store data for applications

Consists of a collection of tables

- Primary key uniquely identifies each row
- Foreign keys establish relationships between tables
  - Referential integrity ensures records in different tables are matched properly
  - Example: you cannot enter an order for a customer that does not exist

Structured Query Language (SQL) is used to access the data

- Operates on complete tables vs. individual records
- Allows joining tables together to obtain matched data



Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Object-relational databases

A relational database with ability to store objects

Accomplished using user-defined data types

- SQL extended to handle complex data types
- Support for inheritance varies

title	*author*
Compilers	Smith
Compilers	Jones
Networks	Jones
Networks	Frick

authors

title	*keyword*
Compilers	parsing
Compilers	analysis
Networks	Internet
Networks	Web

keywords

title	*pub-name*	*pub-branch*
Compilers	McGraw-Hill	New York
Networks	Oxford	London

books4

Figure 9.3 4NF version of the relation *flat-books* of Figure 9.2.

Object-oriented databases

Two approaches:

- Add persistence extensions to OO programming language
- Create a separate OO database

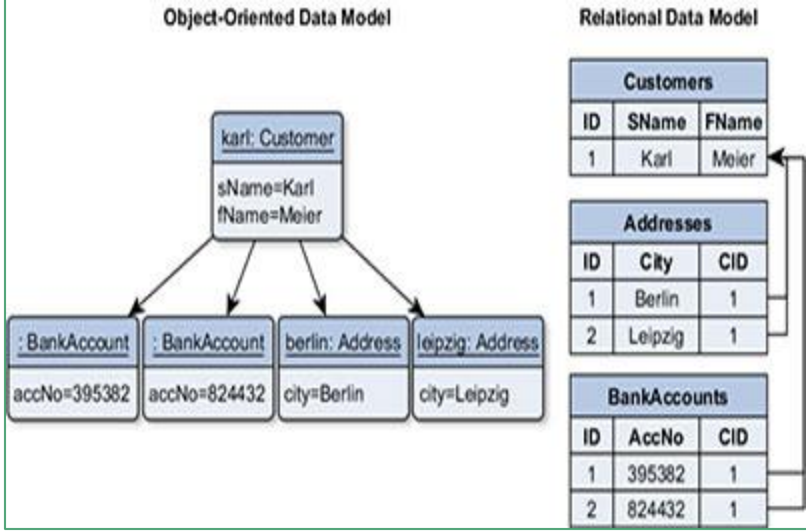
Utilize **extents**—a collection of instances of a class

- Each class is uniquely identified with an Object ID
- Object ID is also used to relate classes together (foreign key not necessary)

Inheritance is supported but is language dependent

Represent a small market share due to its steep learning curve

Object-Oriented Model



Object 1: Maintenance Report

Date	
Activity Code	
Route No.	
Daily Production	
Equipment Hours	
Labor Hours	

Object 1 Instance

01-12-01
24
I-95
2.5
6.0
6.0

Object 2: Maintenance Activity

Activity Code	
Activity Name	
Production Unit	
Average Daily Production Rate	

NoSQL

Newest type: used primarily for complex data types

- Does not support SQL
- No standards exist
- Support very fast queries

Data may not be consistent since there are no locking mechanisms

Types

- Key-value data stores
- Document data stores
- Columnar data stores

Immaturity of technology prevents traditional business application support

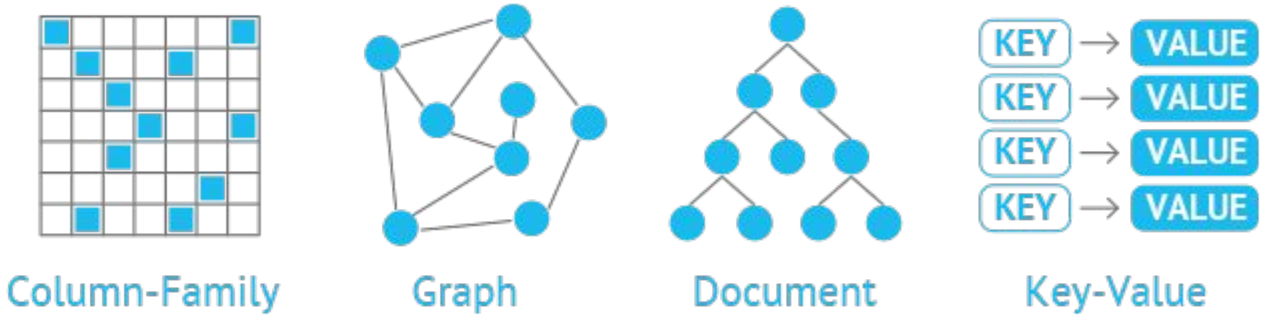
NoSQL

Designed for real-time management of voluminous, heterogeneous social media data on commodity servers

The most significant trade-off between SQL and NoSQL systems – i.e. relational databases vs. "everything else"

- Security and trustworthiness of vital, operational data for the agility, scalability and flexibility of big data.

NoSQL Database



NoSQL

NoSQL are typically "schema-less" (not really)

Classes will have to know the storage schema and translate it into the class diagram schema (nothing really different from RDBMS conceptually)

Some argue that the idea of NoSQL databases might fit better with iterative development approaches

- Object-relational Impedance Mismatch (how to handle inheritance)
    - "The object–relational impedance mismatch is a set of conceptual and technical difficulties that are often encountered when a relational database management system (RDBMS) is being served by an application program (or multiple application programs) written in an object-oriented programming language or style, particularly because objects or class definitions must be mapped to database tables defined by a relational schema."
        - % WikiPedia: https://en.wikipedia.org/wiki/Object%E2%80%93relational_impedance_mismatch
- OODBMSs seem to have failed to catch on

Demo!

MySQL (MariaDB):

- Standard SQL syntax
- https://www.linuxbabe.com/ubuntu/install-lamp-stack-ubuntu-20-04-server-desktop

Graph database: https://sandbox.neo4j.com/

- Uses Cypher language to lookup information

MongoDB:

- Document-based JSON
- https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-20-04
- https://ostechnix.com/rockmongo-graphical-mongodb-administration-tool/ (i couldn't get this to work)

BigQuery/BigTable (% the Goog)

- https://cloud.google.com/bigquery/docs/quickstarts/quickstart-web-ui

	Sequential and Random Access Files	Relational DBMS	Object Relational DBMS	Object-Oriented DBMS
Major Strengths	Usually part of an object-oriented programming language Files can be designed for fast performance Good for short-term data storage	Leader in the database market Can handle diverse data needs	Based on established, proven technology, e.g., SQL Able to handle complex data	Able to handle complex data Direct support for object orientation
Major Weaknesses	Redundant data Data must be updated using programs, i.e., no manipulation or query language No access control	Cannot handle complex data No support for object orientation Impedance mismatch between tables and objects	Limited support for object orientation Impedance mismatch between tables and objects	Technology is still maturing Skills are hard to find
Data Types Supported	Simple and Complex	Simple	Simple and Complex	Simple and Complex
Types of Application Systems Supported	Transaction processing	Transaction processing and decision making	Transaction processing and decision making	Transaction processing and decision making
Existing Storage Formats	Organization dependent	Organization dependent	Organization dependent	Organization dependent
Future Needs	Poor future prospects	Good future prospects	Good future prospects	Good future prospects

What are some considerations for **what type of data storage to select?**

So what do we do with this information?
Map from artifact to data layer!

Map objects to an OODBMS format

- Each concrete class has a corresponding object persistence class
- Add a data access and manipulation class to control the interaction

Map objects to an ORDBMS format

- Procedure depends on the level of support for object orientation by the ORDBMS

Map objects to an RDBMS format

ORDBMS

Rule 1: Map all concrete Problem Domain classes to the ORDBMS tables. Also, if an abstract problem domain class has multiple direct subclasses, map the abstract class to an ORDBMS table.

Rule 2: Map single-valued attributes to columns of the ORDBMS tables.

Rule 3: Map methods and derived attributes to stored procedures or to program modules.

Rule 4: Map single-valued aggregation and association relationships to a column that can store an Object ID. Do this for both sides of the relationship.

Rule 5: Map multivalued attributes to a column that can contain a set of values.

Rule 6: Map repeating groups of attributes to a new table and create a one-to-many association from the original table to the new one.

Rule 7: Map multivalued aggregation and association relationships to a column that can store a set of Object IDs. Do this for both sides of the relationship.

Rule 8: For aggregation and association relationships of mixed type (one-to-many or many-to-one), on the single-valued side (1..1 or 0..1) of the relationship, add a column that can store a set of Object IDs. The values contained in this new column will be the Object IDs from the instances of the class on the multivalued side. On the multivalued side (1..* or 0..*), add a column that can store a single Object ID that will contain the value of the instance of the class on the single-valued side.

For generalization/inheritance relationships:

Rule 9a: Add a column(s) to the table(s) that represents the subclass(es) that will contain an Object ID of the instance stored in the table that represents the superclass. This is similar in concept to a foreign key in an RDBMS. The multiplicity of this new association from the subclass to the "superclass" should be 1..1. Add a column(s) to the table(s) that represents the superclass(es) that will contain an Object ID of the instance stored in the table that represents the subclass(es). If the superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..1, otherwise, it is 1..1. An exclusive-or (XOR) constraint must be added between the associations. Do this for each superclass.

or

Rule 9b: Flatten the inheritance hierarchy by copying the superclass attributes down to all of the subclasses and remove the superclass from the design.*

*It is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.

RDBMS

Rule 1: Map all concrete-problem domain classes to the RDBMS tables. Also, if an abstract Problem Domain class has multiple direct subclasses, map the abstract class to a RDBMS table.

Rule 2: Map single-valued attributes to columns of the tables.

Rule 3: Map methods to stored procedures or to program modules.

Rule 4: Map single-valued aggregation and association relationships to a column that can store the key of the related table, i.e., add a foreign key to the table. Do this for both sides of the relationship.

Rule 5: Map multivalued attributes and repeating groups to new tables and create a one-to-many association from the original table to the new ones.

Rule 6: Map multivalued aggregation and association relationships to a new associative table that relates the two original tables together. Copy the primary key from both original tables to the new associative table, i.e., add foreign keys to the table.

Rule 7: For aggregation and association relationships of mixed type, copy the primary key from the single-valued side (1..1 or 0..1) of the relationship to a new column in the table on the multivalued side (1..* or 0..*) of the relationship that can store the key of the related table, i.e., add a foreign key to the table on the multivalued side of the relationship.

For generalization/inheritance relationships:

Rule 8a: Ensure that the primary key of the subclass instance is the same as the primary key of the superclass. The multiplicity of this new association from the subclass to the "superclass" should be 1..1. If the superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..1, otherwise, it is 1..1. Furthermore, an exclusive-or (XOR) constraint must be added between the associations. Do this for each superclass.

OR

Rule 8b: Flatten the inheritance hierarchy by copying the superclass attributes down to all of the subclasses and remove the superclass from the design.*

* It is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.

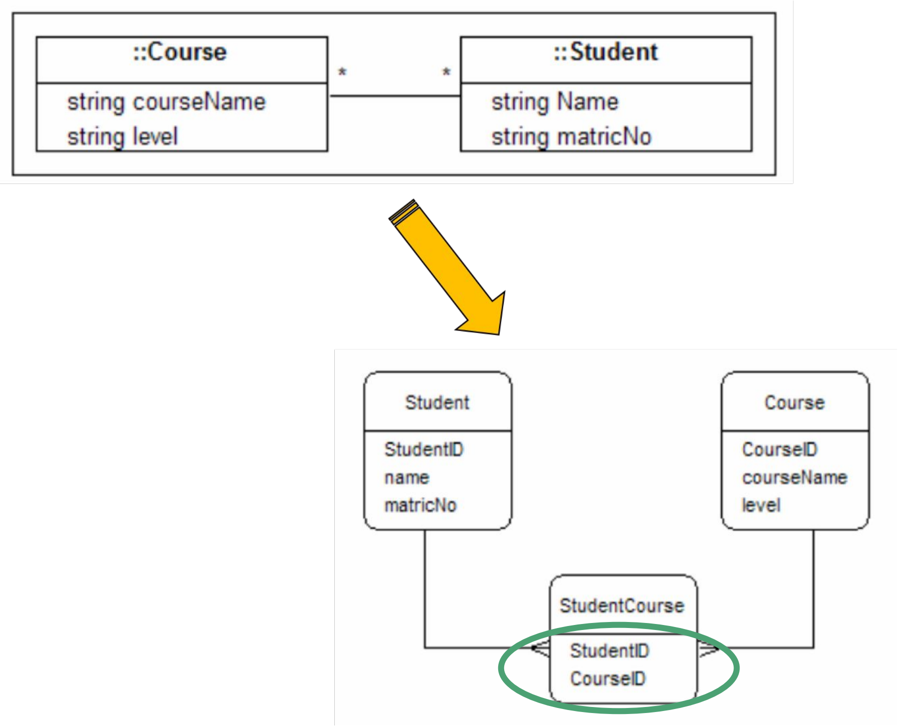
OR mapping

Define storage for our classes (attributes) **and their relationships**

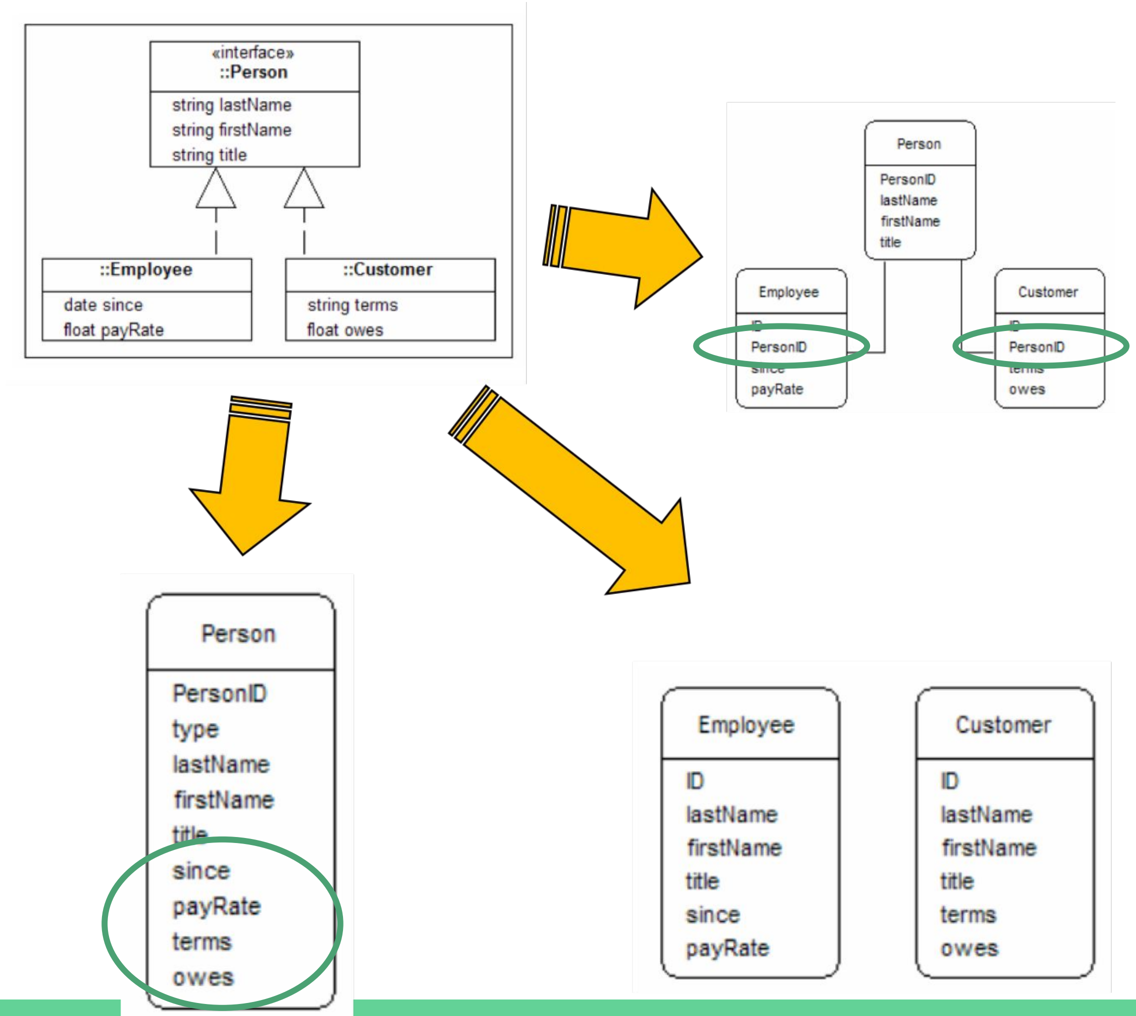
- Class ➜ table
- Single-valued attribute ➜ column
- Multi-valued attribute ➜ new table (one-to-many) add foreign key(s)

Some methods might need stored procedures!

OR mapping (many-to-many)



OR mapping (inheritance) (options)



Optimizing RDBMS storage

Primary (often conflicting) dimensions:

- Improve storage efficiency
  - Normalize the tables
  - Reduce redundant data and the occurrence of null values

- Improve speed of access
  - De-normalize some tables to reduce processing time
  - Place similar records together (clustering)
  - Add indexes to quickly locate records


Why can't you just be normal?

Order

-Order Number : unsigned long
-Date : Date
-Cust ID : unsigned long
-Last Name : String
-First Name : String
-State : String
-Tax Rate : float
-Product 1 Number : unsigned long
-Product 1 Desc. : String
-Product 1 Price : double
-Product 1 Qty. : unsigned long
-Product 2 Number : unsigned long
-Product 2 Desc. : String
-Product 2 Price : double
-Product 2 Qty. : unsigned long
-Product 3 Number : unsigned long
-Product 3 Desc. : String
-Product 3 Price : double
-Product 3 Qty. : unsigned long

Redundant Data

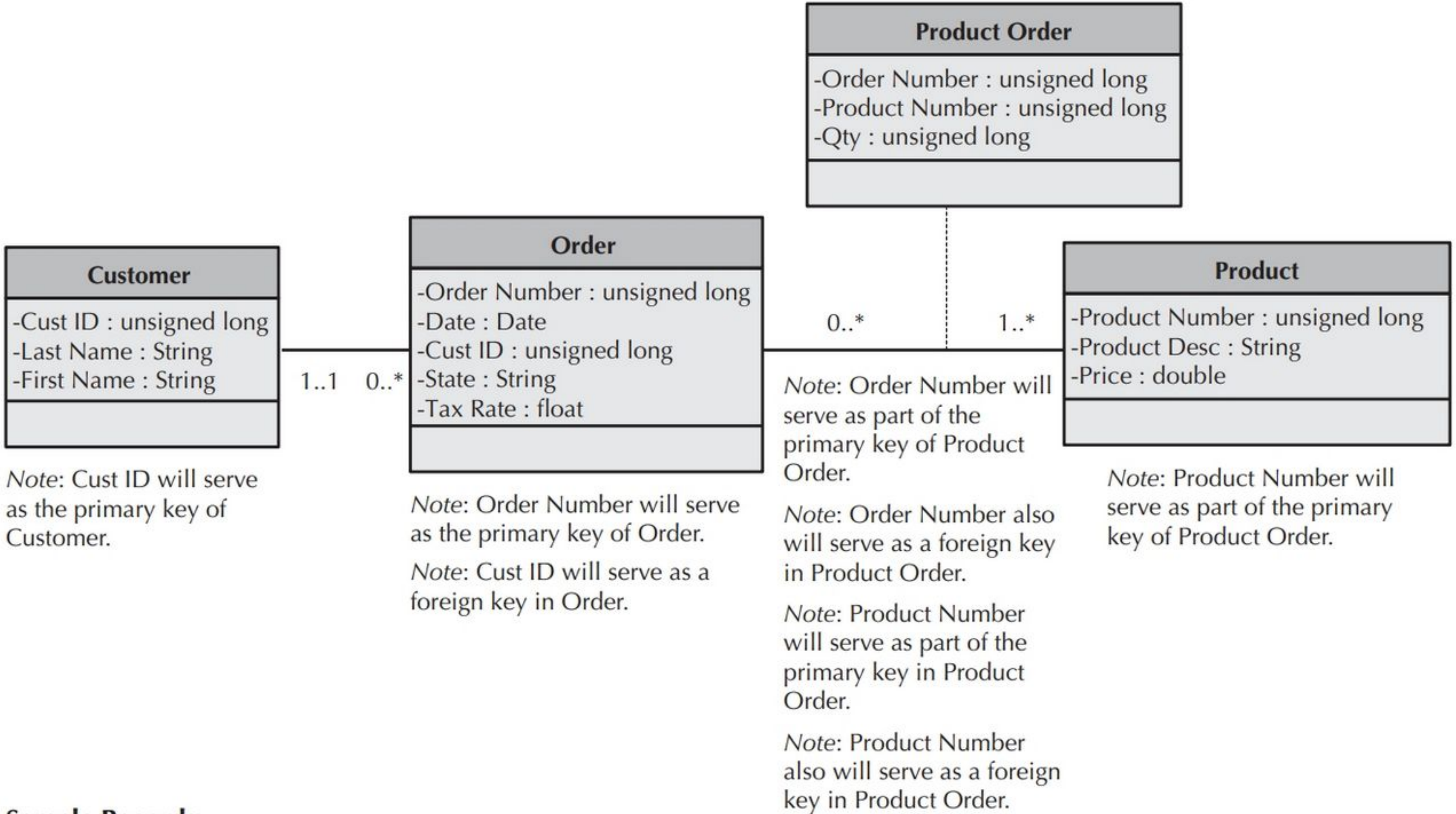
Null Cells

Sample Records:

Order Number	Date	Cust ID	Last Name	First Name	State	Tax Rate	Prod. 1 Number	Prod. 1 Desc.	Prod. 1 Price	Prod. 1 Qty.	Prod. 2 Number	Prod. 2 Desc.	Prod. 2 Price	Prod. 2 Qty.	Prod. 3 Number	Prod. 3 Desc.	Prod. 3 Price	Prod. 3 Qty.
239	11/23/00	1035	Black	John	MD	0.05	555	Cheese Tray	$45.00	2								
260	11/24/00	1035	Black	John	MD	0.05	444	Wine Gift Pack	$60.00	1								
273	11/27/00	1035	Black	John	MD	0.05	222	Bottle Opener	$12.00	1								
241	11/23/00	1123	Williams	Mary	CA	0.08	444	Wine Gift Pack	$60.00	2								
262	11/24/00	1123	Williams	Mary	CA	0.08	222	Bottle Opener	$12.00	2								
287	11/27/00	1123	Williams	Mary	CA	0.08	222	Bottle Opener	$12.00	2								
290	11/30/00	1123	Williams	Mary	CA	0.08	555	Cheese Tray	$45.00	3								
234	11/23/00	2242	DeBerry	Ann	DC	0.065	555	Cheese Tray	$45.00	2								
237	11/23/00	2242	DeBerry	Ann	DC	0.065	111	Wine Guide	$15.00	1	444	Wine Gift Pack	$60.00	1				
238	11/23/00	2242	DeBerry	Ann	DC	0.065	444	Wine Gift Pack	$60.00	1								
245	11/24/00	2242	DeBerry	Ann	DC	0.065	222	Bottle Opener	$12.00	1								
250	11/24/00	2242	DeBerry	Ann	DC	0.065	222	Bottle Opener	$12.00	1								
252	11/24/00	2242	DeBerry	Ann	DC	0.065	222	Bottle Opener	$12.00	1	444	Wine Gift Pack	$60.00	2				
253	11/24/00	2242	DeBerry	Ann	DC	0.065	222	Bottle Opener	$12.00	1	444	Wine Gift Pack	$60.00	1				
297	11/30/00	2242	DeBerry	Ann	DC	0.065	333	Jams & Jellies	$20.00	2								
243	11/24/00	4254	Bailey	Ryan	MD	0.05	555	Cheese Tray	$45.00	2								
246	11/24/00	4254	Bailey	Ryan	MD	0.05	333	Jams & Jellies	$20.00	3								
248	11/24/00	4254	Bailey	Ryan	MD	0.05	222	Bottle Opener	$12.00	1	333	Jams & Jellies	$20.00	2	111	Wine Guide	$15.00	1
235	11/23/00	9500	Chin	April	KS	0.05	222	Bottle Opener	$12.00	1								
242	11/23/00	9500	Chin	April	KS	0.05	333	Jams & Jellies	$20.00	3								
244	11/24/00	9500	Chin	April	KS	0.05	222	Bottle Opener	$12.00	2								
251	11/24/00	9500	Chin	April	KS	0.05	111	Wine Guide	$15.00	2								

FIGURE 9-11 Optimizing Storage

N
O
R
M
A
L



Sample Records:

Normalization

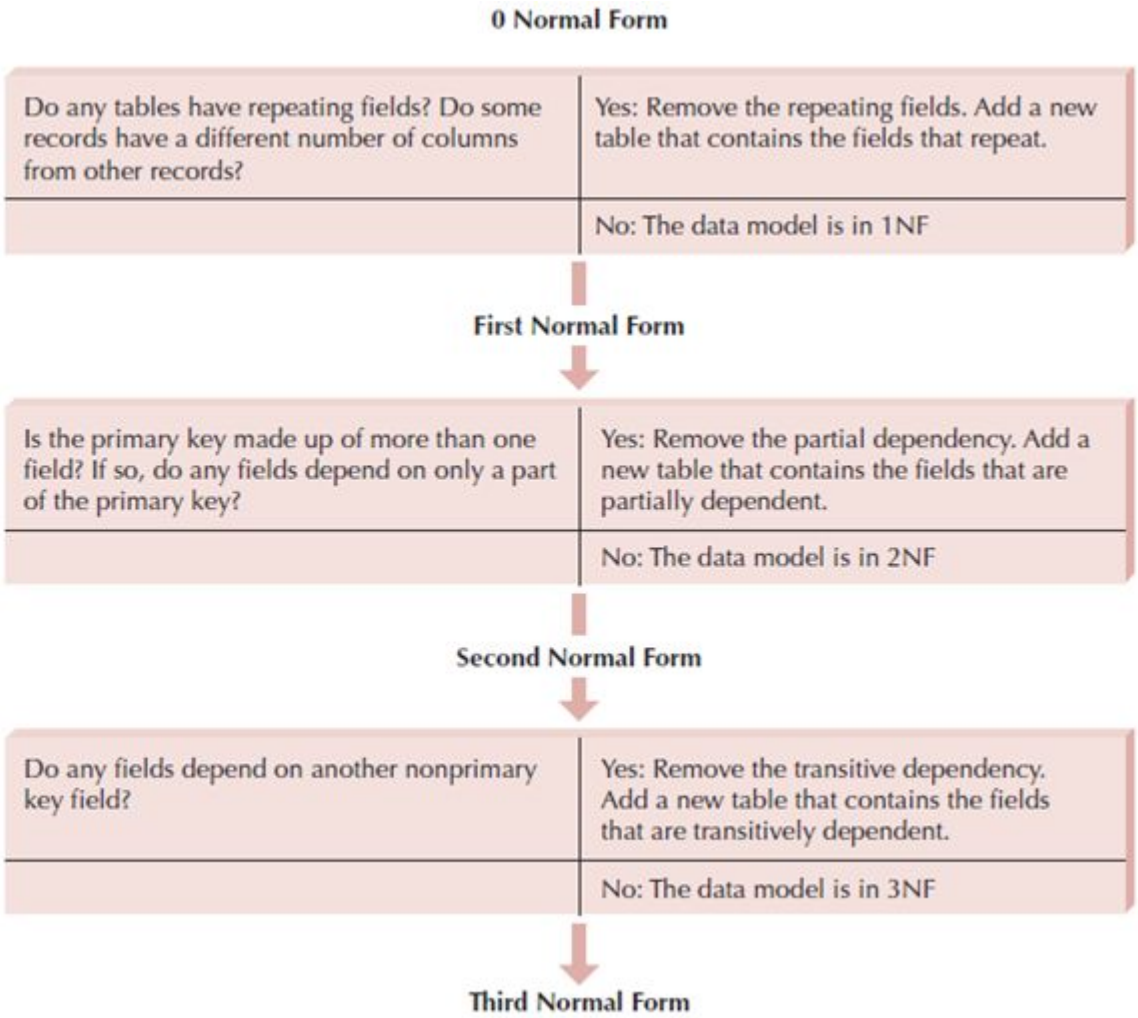
Store each data fact only once in the database

Reduces data redundancies and chances of errors

First four levels of normalization are

- **0 Normal Form**: normalization rules not applied
- **1 Normal Form**: no multi-valued attributes (each cell has only a single value)
- **2 Normal Form**: no partial dependencies (non-key fields depend on the entire primary key, not just part of it)
- **3 Normal Form**: no transitive dependencies (non-key fields do not depend on other non-key fields)

Steps


0 Normal Form
Do any tables have repeating fields? Do some records have a different number of columns from other records?
Yes: Remove the repeating fields. Add a new table that contains the fields that repeat.
No: The data model is in 1NF
First Normal Form
Is the primary key made up of more than one field? If so, do any fields depend on only a part of the primary key?
Yes: Remove the partial dependency. Add a new table that contains the fields that are partially dependent.
No: The data model is in 2NF
Second Normal Form
Do any fields depend on another nonprimary key field?
Yes: Remove the transitive dependency. Add a new table that contains the fields that are transitively dependent.
No: The data model is in 3NF
Third Normal Form

http://agiledata.org/essays/dataNormalization.html

Table 1. Data Normalization Rules.

Level	Rule
First normal form (1NF)	An entity type is in 1NF when it contains no repeating groups of data.
Second normal form (2NF)	An entity type is in 2NF when it is in 1NF and when all of its non-key attributes are fully dependent on its primary key.
Third normal form (3NF)	An entity type is in 3NF when it is in 2NF and when all of its attributes are directly dependent on the primary key.

Also: https://www.essentialsql.com/database-normalization

Optimizing data access speed

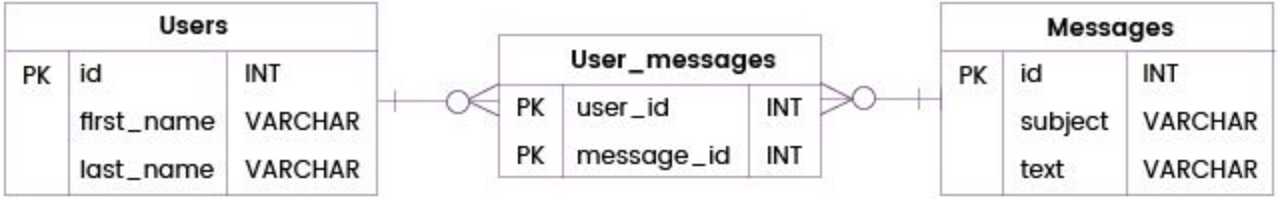
De-normalization

- Table joins require processing
- Add some data to a table to reduce the number of joins required (Increases data retrieval speed)
- Creates redundancy and should be used sparingly

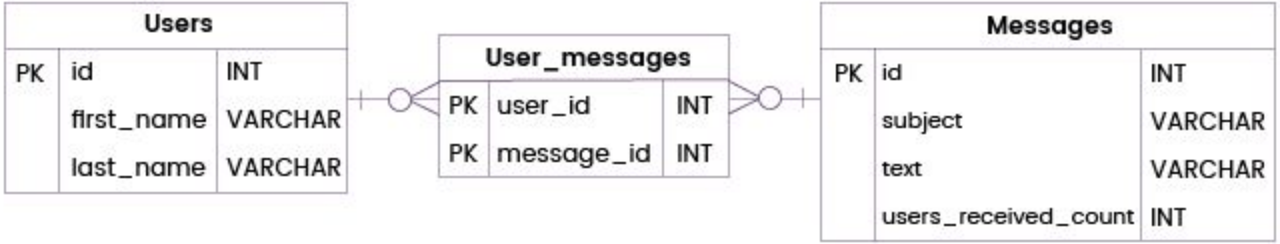
Clustering

- Place similar records close together on the disk
- Reduces the time needed to access the disk

Normalized database



Denormalized database

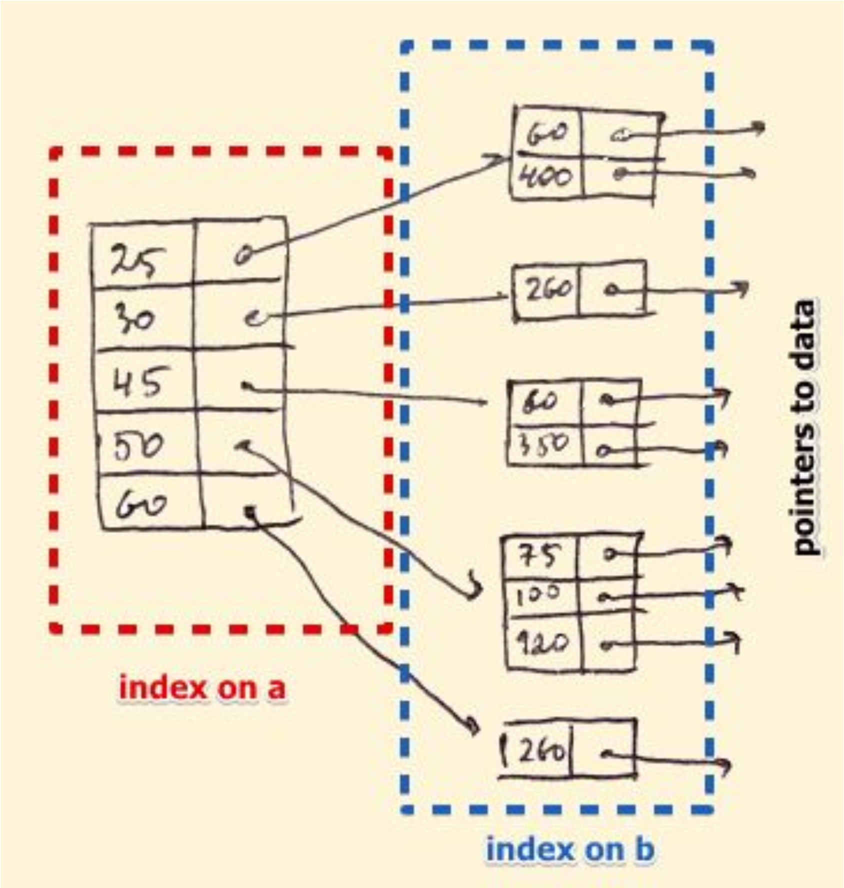


Optimizing data access speed

Indexing

- A small file with attribute values and a pointer to the record on the disk
- Search the index file for an entry, then go to the disk to retrieve the record
- Accessing a file in memory is much faster than searching a disk

Use indexes sparingly for transaction systems.
Use many indexes to increase response times in decision support systems.
For each table, create a unique index that is based on the primary key.
For each table, create an index that is based on the foreign key to improve the performance of joins.
Create an index for fields that are used frequently for grouping, sorting, or criteria.



25
30
45
50
60
60
400
260
60
350
75
100
120
1260
index on a
index on b
pointers to data

Optimizing data access storage

Estimating Data Storage Size

- Use volumetrics to estimate amount of raw data + overhead requirements

- This helps determine the necessary hardware capacity

Field	Average Size
Order Number	8
Date	7
Cust ID	4
Last Name	13
First Name	9
State	2
Amount	4
Tax Rate	2
Record Size	49
Overhead	30%
Total Record Size	63.7
Initial Table Size	50,000
Initial Table Volume	3,185,000
Growth Rate/Month	1,000
Table Volume @ 3 years	5,478,200

Designing data access / manipulation classes

Classes that translate between the problem domain classes and object persistent classes

ORDBMS: create one DAM for each concrete PD class

RDBMS: may require more classes since data is spread over more tables

- Class libraries (e.g., Hibernate) are available to help

NFRs and Data Management Layer design

Operational requirements:

- Affected by choice in hardware and operating system

Performance requirements:

- Speed & capacity issues

Security requirements:

- Access controls, encryption, and backup

Cultural & political requirements:

- May affect the data storage
    - e.g., expected number of characters for data field
    - required format of a data field
    - local laws pertaining to data storage
    - etc...

V&V THE DML!

Test the fidelity of the design before implementation

Verifying and validating the design of the data management layer falls into three basic groups:

- Verifying and validating any changes made to the problem domain
- Dependency of the object persistence instances on the problem domain must be enforced
- The design of the data access and manipulation classes need to be tested

Think of your term projects!

And how much they'd benefit from some *persistence*

Select **two** of your classes and translate them to **database tables**
(or a file structure if you prefer, but db tables are easier (to me at least))