BOSTON UNIVERSITY

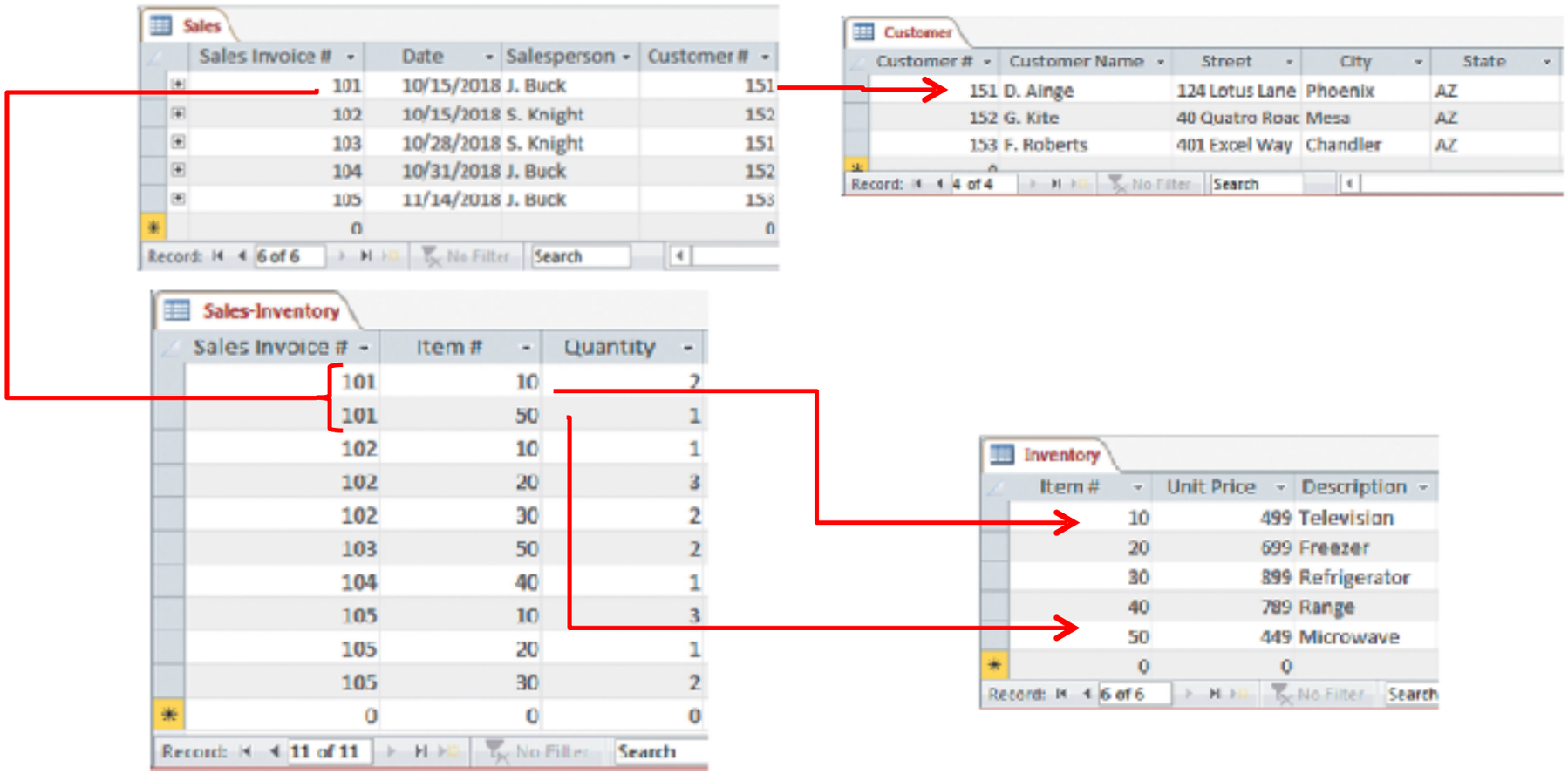
Department of Computer Science

CS591 Intro to mongoDB

@perrydBUCS

Traditional data sources: RDB

- An RDB is a relational database such as Oracle, DB2, or mySQL
- In RDBs data is stored in tables
- Each table contains a minimal amount of information and is related to other tables by a key
- For example, an Employee table and a Department table might be related by an EmployeeID
- Key values appear in each related table

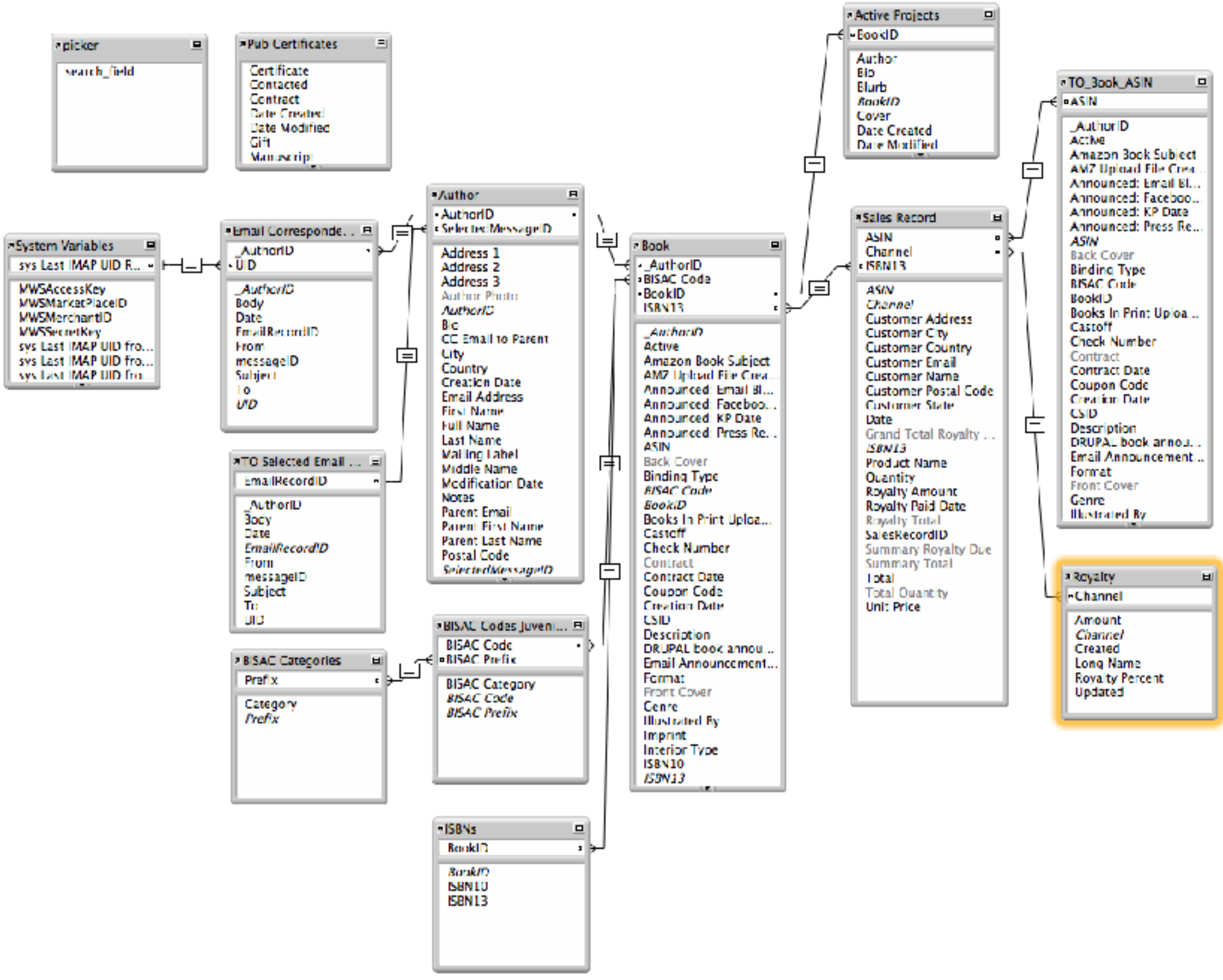


Source: Pearson Ed

- A query language (SQL: Structure Query Language) is used to manipulate data in the RDB
- For the db in the example we might write this to find all its ordered by a particular customer:

```
select customer.id, customer.name, invoice.id,
sales.invoice, sales.item, inventory.item,
inventory.description
FROM customer, invoice, sales, inventory
WHERE sales.invoice = invoice.id AND  sales.item =
inventory.item
```

- That earlier diagram is a form of ERD or *entity-relationship diagram*
- The ERD is a graphical way of showing how the tables are related
- *Related* in this case means that the tables share a common key
- Most DBs have tools to report ERDs, or you can start with an ERD and build tables from it
- Here's another, more formal ERD:


picker
search_field
Pub Certificates
Certificate
Contacted
Contract
Date Created
Date Modified
Gift
Manuscript
Active Projects
BookID
Author
Bio
Blurb
BookID
Cover
Date Created
Date Modified
TO_Book_ASIN
ASIN
_AuthorID
Active
Amazon Book Subject
AMZ Upload File Crea...
Announced: Email Bl...
Announced: Faceboo...
Announced: KP Date
Announced: Press Re...
ASIN
Back Cover
Binding Type
BISAC Code
BookID
Books In Print Uploa...
Castoff
Check Number
Contract
Contract Date
Coupon Code
Creation Date
CSID
Description
DRUPAL book annou...
Email Announcement...
Format
Front Cover
Genre
Illustrated By
System Variables
sys Last IMAP UID R...
MWSAccessKey
MWSMarketPlaceID
MWSMerchantID
MWSSecretKey
sys Last IMAP UID fro...
sys Last IMAP UID fro...
sys Last IMAP UID fro...
Email Corresponde...
_AuthorID
UID
_AuthorID
Body
Date
EmailRecordID
From
messageID
Subject
To
UID
Author
AuthorID
SelectedMessageID
Address 1
Address 2
Address 3
Author Photo
AuthorID
Bio
CC Email to Parent
City
Country
Creation Date
Email Address
First Name
Full Name
Last Name
Mailing Label
Middle Name
Modification Date
Notes
Parent Email
Parent First Name
Parent Last Name
Postal Code
SelectedMessageID
Book
_AuthorID
BISAC Code
BookID
ISBN13
_AuthorID
Active
Amazon Book Subject
AMZ Upload File Crea...
Announced: Email Bl...
Announced: Faceboo...
Announced: KP Date
Announced: Press Re...
ASIN
Back Cover
Binding Type
BISAC Code
BookID
Books In Print Uploa...
Castoff
Check Number
Contract
Contract Date
Coupon Code
Creation Date
CSID
Description
DRUPAL book annou...
Email Announcement...
Format
Front Cover
Genre
Illustrated By
Imprint
Interior Type
ISBN10
ISBN13
Sales Record
ASIN
Channel
ISBN13
ASIN
Channel
Customer Address
Customer City
Customer Country
Customer Email
Customer Name
Customer Postal Code
Customer State
Date
Grand Total Royalty ...
ISBN13
Product Name
Quantity
Royalty Amount
Royalty Paid Date
Royalty Total
SalesRecordID
Summary Royalty Due
Summary Total
Total
Total Quantity
Unit Price
Royalty
Channel
Amount
Channel
Created
Long Name
Royalty Percent
Updated
TO Selected Email ...
EmailRecordID
_AuthorID
Body
Date
EmailRecordID
From
messageID
Subject
To
UID
BISAC Codes Juveni...
BISAC Code
BISAC Prefix
BISAC Category
BISAC Code
BISAC Prefix
BISAC Categories
Prefix
Category
Prefix
ISBNs
BookID
BookID
ISBN10
ISBN13

Non-relational DBs

- For web-based apps the trend has been toward non-relational DBs such as mongoDB
- The DBs are document-based rather than table based
- The term **noSQL** is used for this style…it doesn't necessarily mean '*no* SQL' but rather '*non-relational*' or '*not only SQL*'
- noSQL DBs trade simplicity for performance
- There are quite a few of them… http://nosql-database.org

- The idea behind non-relational DBs is to store all of the data necessary for a record into a single object
- Relationships are folded into that single object
- In the sales example from earlier, the invoices and items would be folded into a single customer object with arrays for invoices
- We might also just have a single invoice object that includes everything about each sale
- There can potentially be a lot of duplication
  - For example, the description of an item, or the address of a customer, might be repeated in hundreds of documents
- Do we care?

- Turns out the answer is ‘not all the time’
- In typical web-based applications the focus is on a user and his or her interaction with a site
- For this use case, a fully denormalized data store (ala noSQL) can perform better than a fully relational (highly normalized) DB
- For the corner cases where we really *do* need relational data, we take the performance hit

ORMs, ORDs, and impedance

- In most case an RDB does not map one-to-one onto the objects in the application code
- Work is required to move data from app to DB; the pattern we use is an ORM (object-relational mapping), basically a way to take RDB table info and store it in an object
- The problem is that the data types in the DB don't necessarily match the way we want to use them in the app…this is called an *impedance mismatch*

- In document-based DBs (such as mongoDB), there are few or no restrictions on data types in a record, reducing or elimination impedance
- However, this means that there are basically no filters on what goes into the mongoDB document store…the risk is GIGO
- Libraries such as Mongoose provide ODM (object-document mapping) services on top of mongoDB
- ODM is the equivalent of ORM but for non-relational DBs
- Bottom line is that ODM on top of document-based DBs provide a cleaner, more performant way to handle data in most applications

A note about ODM

- It's certainly possible to throw random, possibly unrelated JSON at a document-based DB like mongoDB, and then go and operate on it
- However, mongoDB does essentially no type checking, and the risk is that you'll be working on undefined data
- Using ODM tools such as Mongoose allow you to place schema-based constraints on types and ensures that data in the store is uniform
- We'll talk about Mongoose tomorrow!

Using noSQL DBs

- We'll use mongoDB and Javascript for our examples
- In addition, Mongoose will provide ODM services through schemas
- The TL;DL is that we map Javascript objects 1:1 to database documents
- Mapping is done through JSON

CRUD with mongoDB

CREATE	obj.save( )
READ	db.find({key: value})
UPDATE	db.findByIdAndUpdate(id, updateObj)
DELETE	db.findByIdAndRemove(id)

Quick history

- 10gen started work in 2007 on a non-relation document store to use in their PaaS business
- First public release in 2009 as open source project
- mongoDB (from ‘humongous’) grew rapidly in popularity
- 10gen changed their name to MongoDB Inc in 2013

What it is

- Non-relational
- Document store
- No SQL
- Base system written in C++
- Drivers available for most current languages
- Independent of NodeJS

Installing mongoDB

- MacOS:
  - Download and install tarball with curl
  - OR use homebrew package manager (simplest but must install home-brew first)
- Windows:
  - Download and install package from mongoDB
- http://www.mongodb.org/downloads
- https://brew.sh
- https://docs.mongodb.com/getting-started/shell/tutorial/install-mongodb-on-windows

Data directory

- By default mongoDB looks for data in /data/db
- It'll exit at start if the directory doesn't exist
- You also can point to any directory at startup, i.e.
  - mongod& —dbpath ~/data
  - Might have to fiddle with permissions depending on path

Starting mongoDB

- The database runs as a daemon and listens for connection on port 27017 by default
  - This can be changed by editing resource file
  - You might need to hunt for mongod.conf…it'll end up in the install directory
    - Useful to create an alias in /etc to make it easy to find the second time (via ln -s)
- Startup
  - mongod&
  - mongod -dbpath ~/data& (if you've moved the data directory)

!!SECURITY!!

- The base installation has authentication TURNED OFF by default
- This means anyone with access to the port has access to all databases
- (This is why there was a huge mongoDB ransomware attack a few months ago)
- Authentication is set up in mongod.conf
  - Requires definition of users and roles
  - Docs are at https://docs.mongodb.com/manual/reference/configuration-options/#security-options
  - Lots of info on the web, too, especially after the attacks
- In class we'll use defaults, but if you deploy to something other than your local machine you'll want to set up authentication

Import sample collections from mongoDB

- Grab the sample data
  - `curl https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json > mongo-sample-collection.json`
- Import into running mongoDB instance
  - `mongoimport --db test --collection restaurants --drop --file ./mongo-sample-collection.json`

Open mongo shell

- Once mongod is running, connect to it from a command prompt
  - `mongo`
- Use `show dbs` to list available database
- Connect to a database with `use <name>`
  - `use test`
- View collections
  - `show collections`