Week 4 Lecture 1

Class	BSCCS2003
Created	@September 27, 2021 1:48 PM
Materials	
Module #	23
Type	Lecture
Week #	4

Persistent Storage

Example: Gradebook

- Students: ID, name, address, ...
- Courses: ID, name, department, year, ...
- Student-Course Relationship: which students are registered for which courses

Gradebook

Name	IDNumber
Sunil Shashi	MAD001
Chetana Anantha	MAD002
Madhur Prakash	MAD003
Nihal Surya	MAD004
Shweta Lalita	MAD005
Raghu Balwinder	MAD006
Gulshan Kuldeep	MAD007
Kishan Shrivatsa	MAD008
Purnima Sunil	MAD009
Nikitha Madhavi	MAD010
Lilavati Prabhakar	MAD011
Rama Yamuna	MAD012

Gradebook

CourseID	Name
EE1001	Introduction to Electrical Engineering
AM1100	Engineering Mechanics
MA1020	Functions of Several Variables
ME1100	Thermodynamics
BR1010	Life Sciences

Spreadsheets

- Arbitrary data organized into rows and columns
- Operations defined on cells or ranges
- Multiple inter-linked sheets within single spreadsheet

Any kind of tabular data - expressed in tables

Relationships

- Students - Course?
- Separate entry with full details - student name, ID, address, course ID, name, department, etc.
  - Redundant
- Separate table "joining" students with courses
  - Only ID specified
  - Relation specified with "Keys"

Aa A	B	# C
MAD001	BT1010	78
MAD002	EE1001	30
MAD005	EE1001	68
MAD009	AM1100	62
MAD012	AM1100	77
MAD007	BT1010	41

Questions

- How should the underlying data should be stored?
  - Can it be made persistent - survive server restart?
- How should the relations be represented?
- Structured ways to represent, manipulate data?

Week 4 Lecture 2

Class	BSCCS2003
Created	@September 27, 2021 2:41 PM
Materials	
Module #	24
Type	Lecture
Week #	4

Mechanisms for Persistent Storage

In-memory data structures

```
names = ['Alice', 'Bob', 'Charlie']
courses = ['Introduction to EE', 'Applied Mech', 'Calculus']
rels = [('Alice', 'Introduction to EE'),
        ('Bob', 'Calculus'),
        ('Alice', 'Calculus'),
        ('Charlie', 'Applied Mech')]
```

- Error prone - easy to make mistakes in entry or referencing
- Does not scale
- Duplicate names?

In-memory data structures: Keys

```
names = {0: 'Alice', 1: 'Bob', 2: 'Charlie'}
courses = {0: 'Introduction to EE', 1: 'Applied Mech', 2: 'Calculus'}
rels = [(0, 0), (1, 2), (0, 2), (2, 1)]
```

- Data entry errors less likely
- Duplicates not a problem - Unique key

Objects

```
class Student:
  id_next = 0 # Class variable
  def __init__(self, name):
    self.name = name
    self.id = Student.id_next
    Student.id_next += 1
```

- Auto-initialize ID to ensure unique
- Functions to set/get values

```
class Student:
  id_next = 0 # Class variable
  def __init__(self, name, hostel):
    self.name = name
    self.id = Student.id_next
    self.hostel = hostel
    Student.id_next += 1
```

- Add a new field to the object easily

But what about persistence?

- In-memory data structures are lost when the program is killed, server is shut down or rebooted (restarted)
- Save to disk? Structured data?
  - Python Pickle and similar modules
  - CSV - Comma Separated Values
  - TSV - Tab Separated Values
- Essentially the same as spreadsheets - limited flexibility

Spreadsheets

- Naturally represent tabular data
- Extension, adding fields are easy
- Separate sheets for relationships

Problems with spreadsheets:

- Lookups, cross-referencing harder than dedicated database
- Stored procedures - limited functionality
- Atomic operations - no clear definition

Relational Databases - SQL

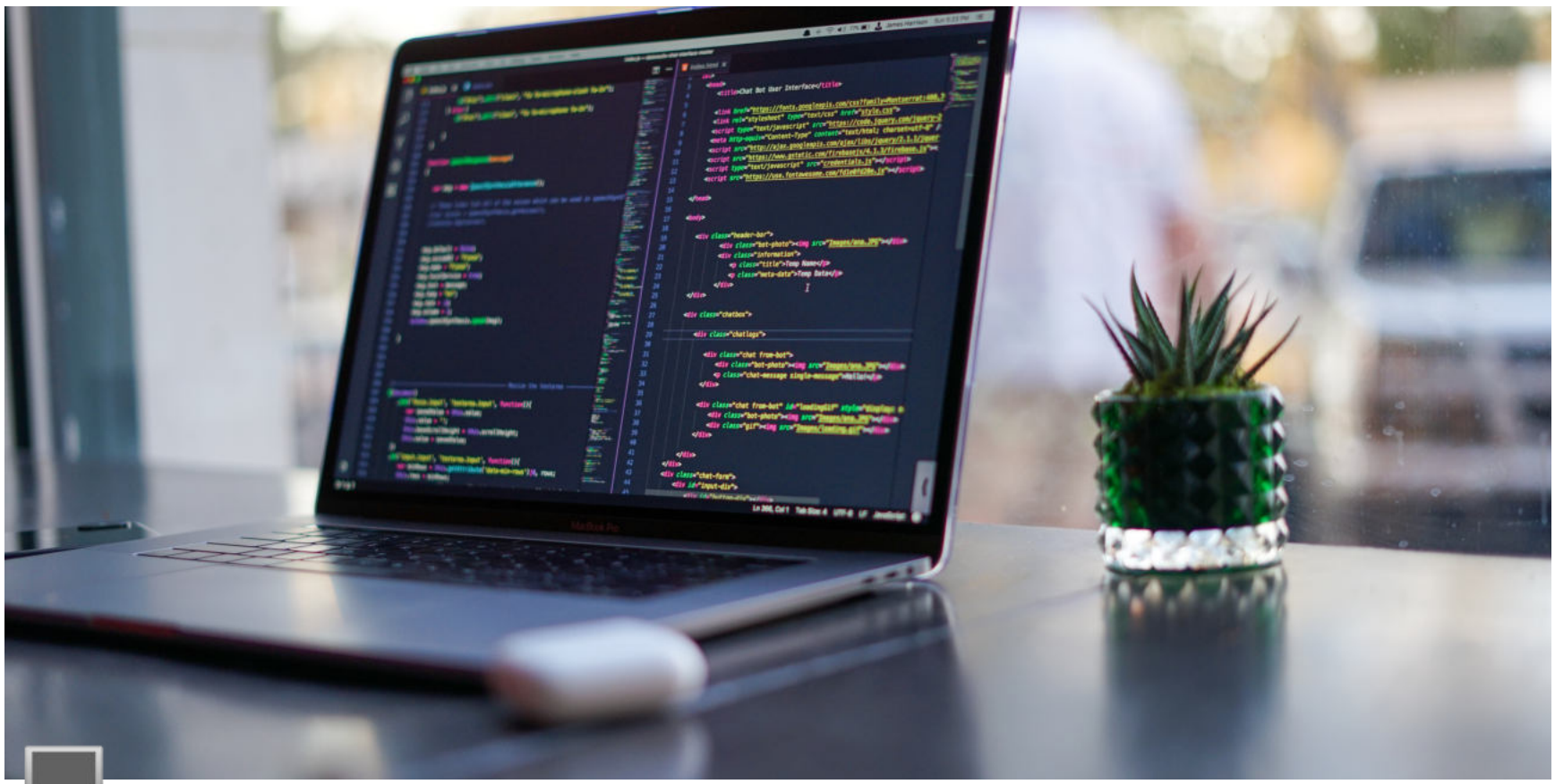
> Shameless plug, I do have notes for DBMS here

- From IBM ~ 1970s
- Data is stored in tabular format:
  - Columns of tables: fields (name, address, department, ...)
  - Rows of tables: individual entries (student1, student2, ...)

Unstructured Databases - NoSQL

- Easily add/change fields
- Arbitrary data
- NoSQL
  - MongoDB
  - CouchDB
  - ...

- Flexible, but potential loss of validation



Week 4 Lecture 3

Class	BSCCS2003
Created	@September 27, 2021 3:39 PM
Materials	
Module #	25
Type	Lecture
Week #	4

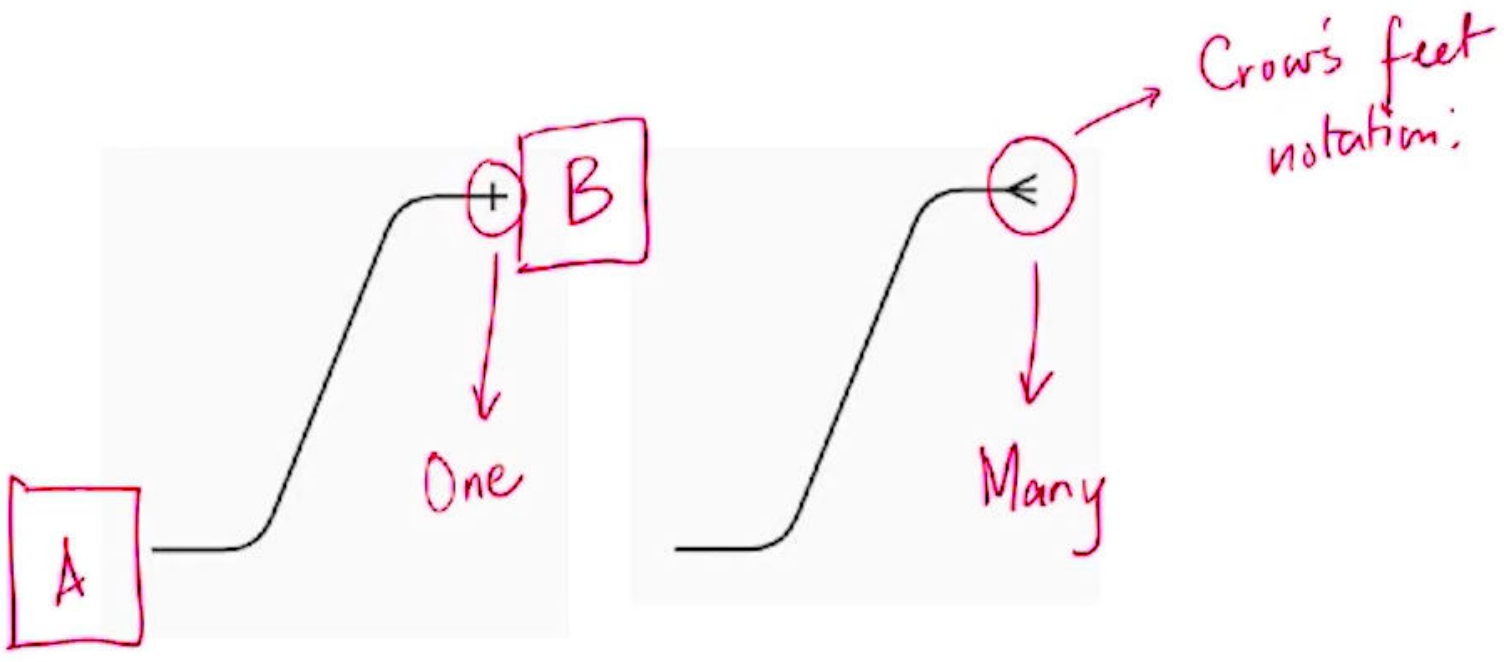
Relations and ER diagram

Relationship types

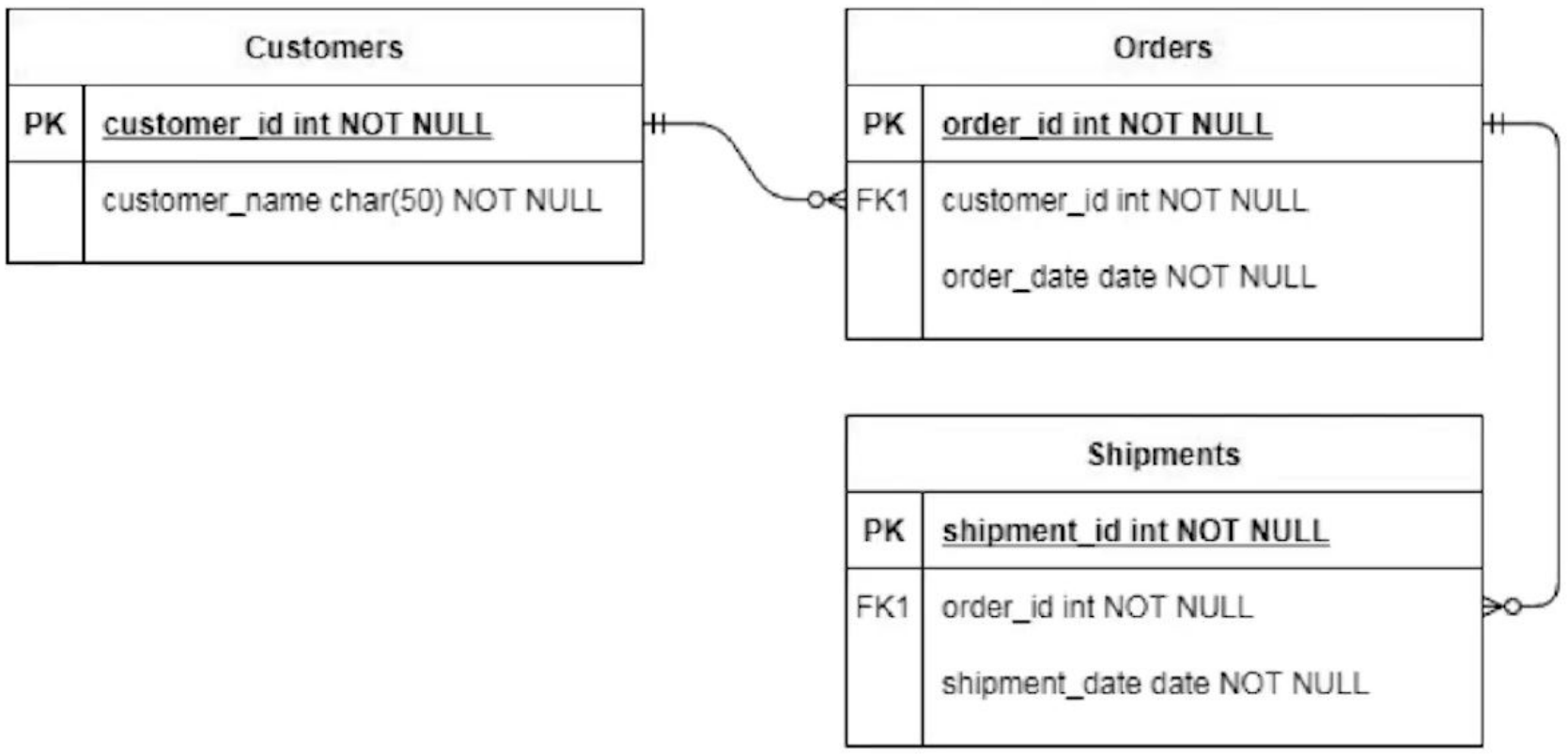
- One to One:
  - One student has one roll number
  - One roll number uniquely identifies one student
  - **Example:** assign unique message-ID to each email in the inbox
- One to Many (Many to One):
  - One stays in only one hostel
  - One hostel has many students
  - **Example:** save emails in folders - one email is in only one folder
- Many to Many:
  - One student can register for many courses
  - One course be taken by many students
  - **Example:** assign labels to emails - one email can have many labels and vice-versa

Diagrams

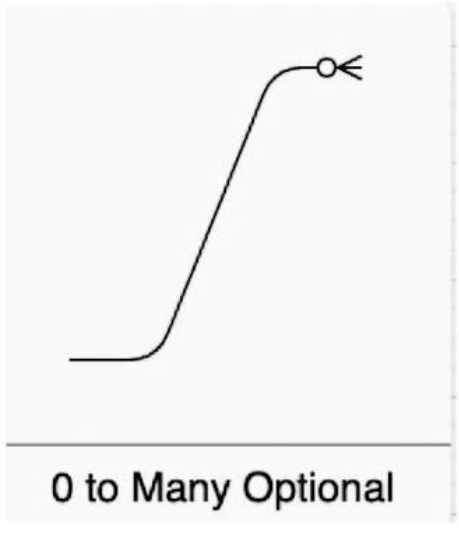
- Entity-Relationship (ER)
- Unified Modeling Language (UML)
- Class relation

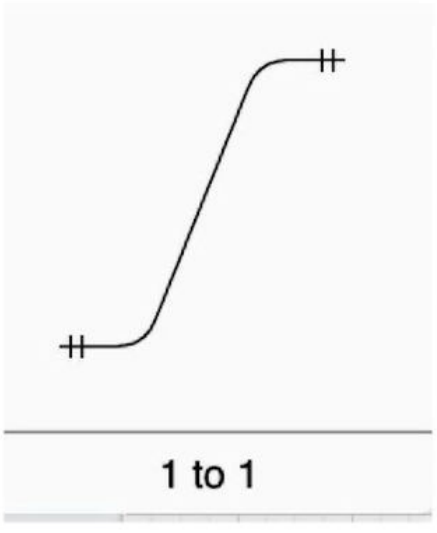


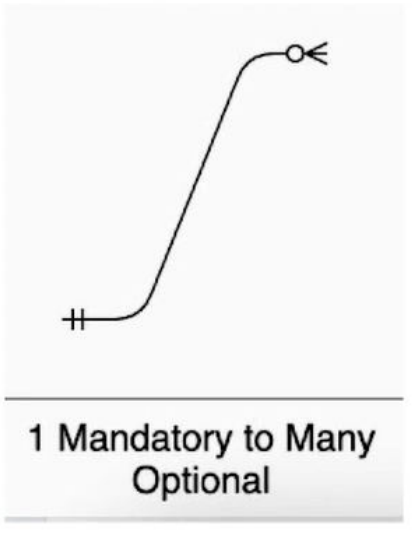
Entity-Relationship Diagram



ER examples







Tool: Draw.io → https://app.diagrams.net

Week 4 Lecture 4

Class	BSCCS2003
Created	@September 27, 2021 4:29 PM
Materials	
Module #	26
Type	Lecture
Week #	4

SQL

Relational Databases - SQL

- From IBM ~ 1970s
- Data is stored in tabular format:
  - Columns of tables: fields (name, address, department, ...)
  - Rows of tables: individual entries (student1, student2, ...)
- Key: unique way of accessing a given row
  - **Primary Key:** important for fast access to large databases
  - **Foreign Key:** connect to a different table - Relationships

Queries

- Retrieve data from a database

eg: `"Find students with name beginning with A"`

`"Find all courses offered in 2021"`

Structured Query Language

- English-like, but structured
- Quite verbose
- Specific mathematical operations:
  - Inner join

- Outer join

Example: Inner join

Name	IDNumber	hostelID
Sunil Shashi	MAD001	1
Chetana Anantha	MAD002	2
Madhur Prakash	MAD003	2
Nihal Surya	MAD004	3
Shweta Lalita	MAD005	2
Raghu Balwinder	MAD006	3
Gulshan Kuldeep	MAD007	1
Kishan Shrivatsa	MAD008	1
Purnima Sunil	MAD009	2
Nikitha Madhavi	MAD010	1
Lilavati Prabhakar	MAD011	3
Rama Yamuna	MAD012	3

ID	Name	Capacity
1	Jamuna	300
2	Ganga	300
3	Brahmaputra	500

Student - Hostel mapping

```
SELECT Students.name, Hostels.name
FROM Students
INNER JOIN Hostels
ON Students.hostelID = Hostels.ID;
```

```
---- OUTPUT ----
Sunil Shashi, Jamuna
Chetana Anantha, Ganga
```

Cartesian product

- N entries in table 1
- M entries in table 2
- M $\times$ N combinations - filter on them

Powerful SQL queries can be constructed

Example: find all the students in Calculus

- Find ID number for the course
- Look up StudentsCourses table to find all entries with this course ID
- Look up Students to find the names of students with these IDs

```
SELECT s.name
FROM Student s
JOIN StudentsCourses sc ON s.IDNumber = sc.studentID
JOIN Courses c ON c.ID = sc.courseID
WHERE c.name = 'Calculus';
```

Summary

- Models - persistent data storage
- Mechanisms:
  - csv
  - spreadsheets
  - SQL
  - NoSQL

- Entities and Relationships
    - Different ways of representing

No details on display, views or what kind of updates permitted