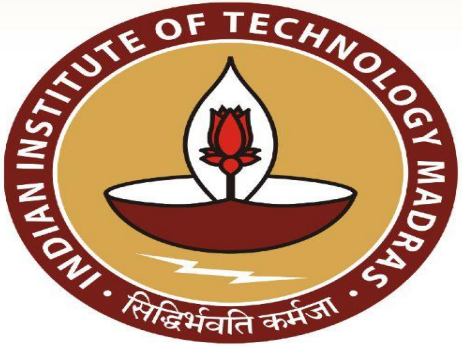


IIT Madras
BSc Degree

Models

Persistent Storage

- Example: Grades
- Need for persistent storage
- Requirements

Example: Gradebook

- Students: ID, name, address, …
- Courses: ID, name, department, year, …
- StudentCourse Relationship: which students are registered for which courses

Gradebook

	A	B
1	**Name**	**IDNumber**
2	Sunil Shashi	MAD001
3	Chetana Anantha	MAD002
4	Madhur Prakash	MAD003
5	Nihal Surya	MAD004
6	Shweta Lalita	MAD005
7	Raghu Balwinder	MAD006
8	Gulshan Kuldeep	MAD007
9	Kishan Shrivatsa	MAD008
10	Purnima Sunil	MAD009
11	Nikitha Madhavi	MAD010
12	Lilavati Prabhakar	MAD011
13	Rama Yamuna	MAD012

	A	B
1	**CourseID**	**Name**
2	EE1001	Introduction to Electrical Engineering
3	AM1100	Engineering Mechanics
4	MA1020	Functions of Several Variables
5	ME1100	Thermodynamics
6	BT1010	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

- Student - Course?

Relationships

- Student - Course?
- Separate entry with full details - student name, ID, address, course ID, name, department etc?

Relationships

- Student - Course?
- Separate entry with full details - student name, ID, address, course ID, name, department etc?
  - Redundant

Relationships

- Student - 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”

	A	B	C
4	MAD001	BT1010	78
5	MAD002	EE1001	30
6	MAD005	EE1001	68
7	MAD009	AM1100	62
8	MAD012	AM1100	77
9	MAD007	BT1010	41
10	MAD001	MA1020	56

Questions

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

Storage

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')]
```

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)]
```

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

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

Objects

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

- Add a new field to object easily

Persistence?

- In memory data structures lost when server shut down or restarted
- Save to disk? Structured data?
  - Python Pickle and similar modules
  - CSV - comma separated values
  - TSV - tab separated values
- Essentially same as spreadsheets: limited flexibility

Spreadsheet

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

Spreadsheet

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

Problems:

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

Relational Databases - SQL

- From IBM ~ 1970s
- Data 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

Relations

Spreadsheet

	A	B	C
4	MAD001	BT1010	78
5	MAD002	EE1001	30
6	MAD005	EE1001	68
7	MAD009	AM1100	62
8	MAD012	AM1100	77
9	MAD007	BT1010	41
10	MAD001	MA1020	56

	A	B
1	**CourseID**	**Name**
2	EE1001	Introduction to Electrical Engineering
3	AM1100	Engineering Mechanics
4	MA1020	Functions of Several Variables
5	ME1100	Thermodynamics
6	BT1010	Life Sciences

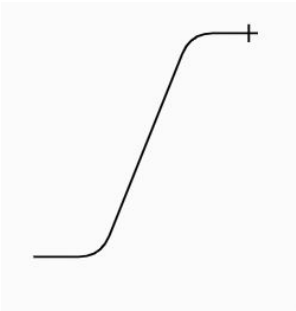
	A	B
1	**Name**	**IDNumber**
2	Sunil Shashi	MAD001
3	Chetana Anantha	MAD002
4	Madhur Prakash	MAD003
5	Nihal Surya	MAD004
6	Shweta Lalita	MAD005
7	Raghu Balwinder	MAD006
8	Gulshan Kuldeep	MAD007
9	Kishan Shrivatsa	MAD008
10	Purnima Sunil	MAD009
11	Nikitha Madhavi	MAD010
12	Lilavati Prabhakar	MAD011
13	Rama Yamuna	MAD012

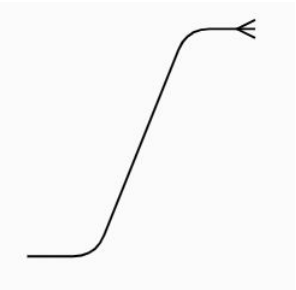
Relationship types

- 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 Inbox
- One-to-many (many-to-one):
  - one student 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 can have many students
  - Example: assign labels to emails - one email can have many labels and vice versa

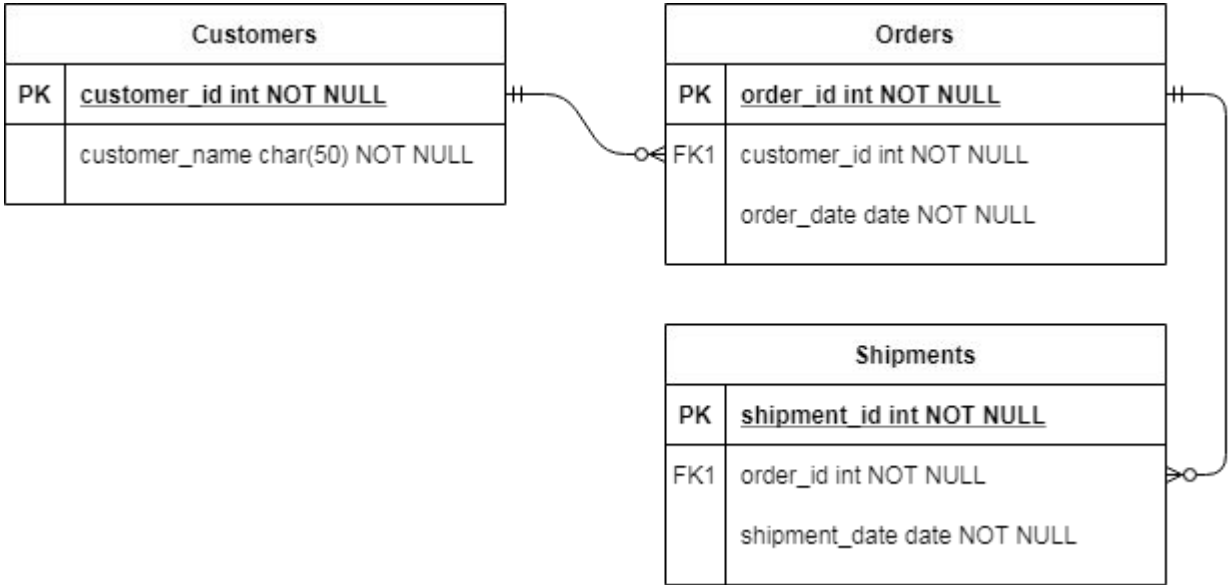
Diagrams

- Entity-Relationship
- UML
- Class relation
- ...

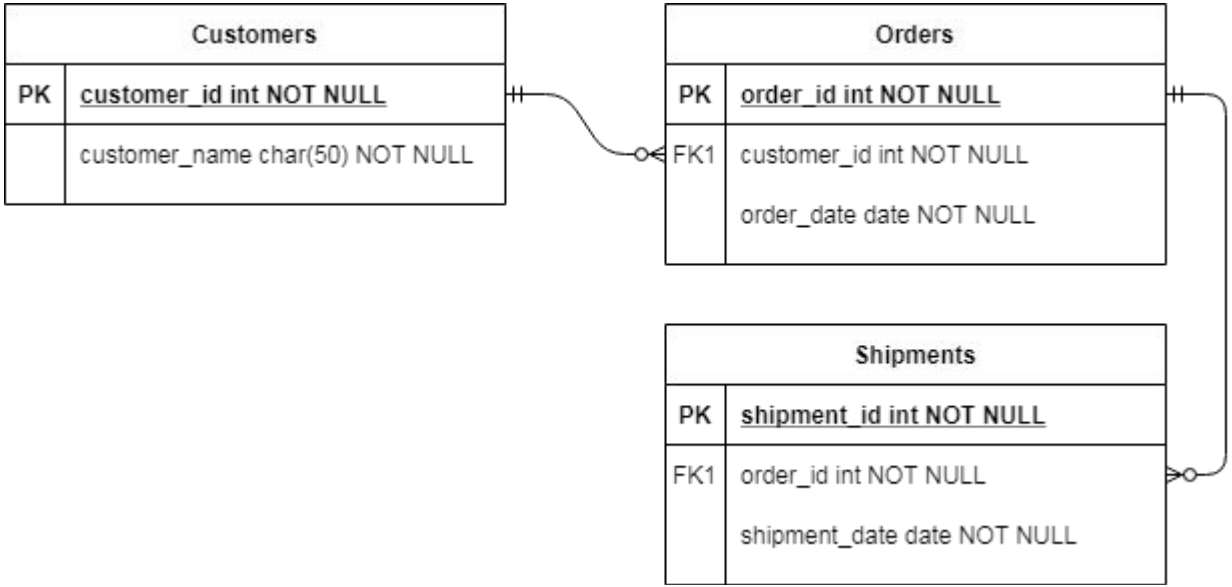




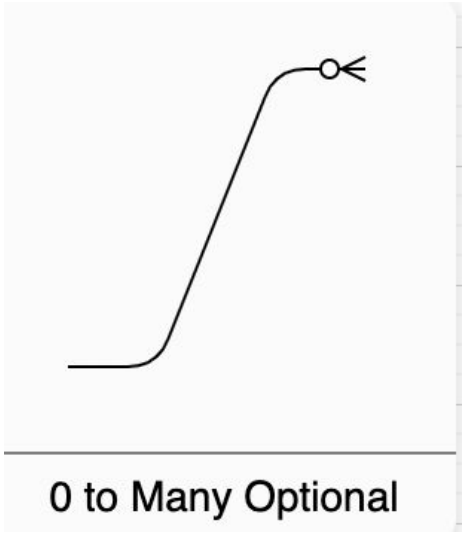
Entity-Relationship Diagram

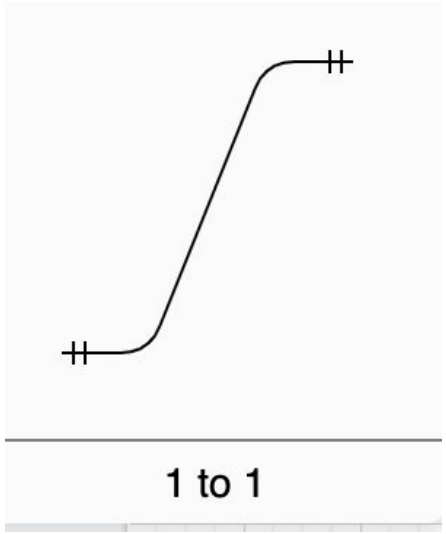


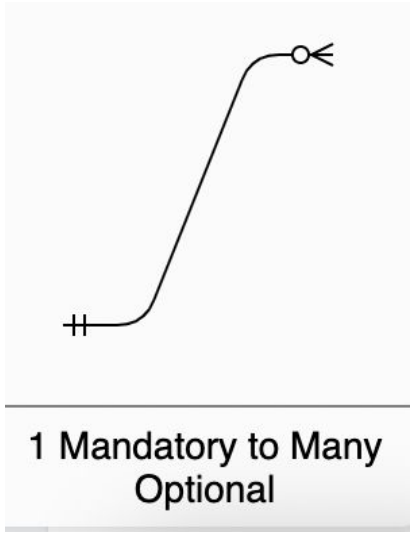
Entity-Relationship Diagram



ER Examples







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

SQL

Relational Databases

- From IBM ~ 1970s
- Data 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

Relational Databases

- From IBM ~ 1970s
- Data 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 on large databases

Relational Databases

- From IBM ~ 1970s
- Data 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 on large databases
  - **Foreign key**: connect to a different table - **Relationships**

Queries

- Retrieve data from the database:

eg. “Find students with name beginning with A”

“Find all courses offered in 2021”

Structured Query Language (SQL)

- 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
```

Student - Hostel mapping

```
select Students.name, Hostels.name
  from Students
  inner join Hostels
  on Students.hostelID = Hostels.ID
```

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

Cartesian Product

- N entries in table 1
- M entries in table 2
- M x N combinations - filter on them

Powerful SQL queries can be constructed

Example: find all students in Calculus

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

Example: find all students in Calculus

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

```
SELECT s.name
  FROM Students s
```

Example: find all students in Calculus

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

```
SELECT s.name
  FROM Students s
  JOIN StudentsCourses sc ON s.IDNumber = sc.studentID
```

Example: find all students in Calculus

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

```
SELECT s.name
  FROM Students s
  JOIN StudentsCourses sc ON s.IDNumber = sc.studentID
  JOIN Courses c ON c.ID = sc.courseID
```

Example: find all students in Calculus

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

```
SELECT s.name
  FROM Students 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