Lecture 07: Normalization and Data Modeling

DATA 351: Database Design

Lucas P. Cordova, Ph.D.

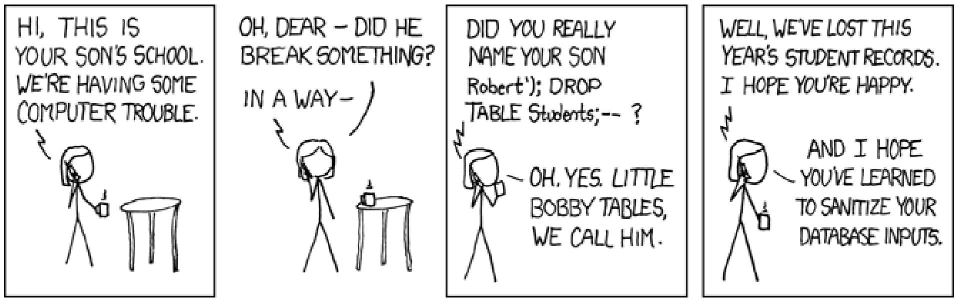
2026-02-02

This lecture covers database normalization and data modeling fundamentals. We explore the normalization process from messy flat files to well-structured relational schemas, practice with the Dunder Mifflin employee dataset, explore database design for the Oregon Turtle Conservation project, and learn to create Entity-Relationship diagrams using draw.io.

Table of contents

1 Overview

1.1 What We Will Cover


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Today we learn to structure data so it does not structure us.

2 Why Normalization Matters

2.1 The Spreadsheet Problem

You have probably seen spreadsheets like this in the wild:

OrderID	Customer	CustomerEmail	Product1	Qty1	Product2	Qty2	Product3	Qty3
1001	Acme Corp	orders@acme.com	Widget	5	Gadget	2		
1002	Acme Corp	orders@acme.com	Sprocket	10				
1003	Beta Inc	purchasing@beta.io	Widget	3	Gadget	1	Sprocket	7

This looks reasonable at first glance. Then someone orders four products.

2.2 Data Anomalies: The Horror Stories

When data is poorly structured, three types of anomalies haunt your database:

2.3 Update Anomaly

If Acme Corp changes their email address, you must update every single row where they appear. Miss one? Now you have conflicting data.

```
OrderID 1001: orders@acme.com
OrderID 1002: orders_new@acme.com  -- Oops!
```

2.4 Insert Anomaly

You want to add a new customer who has not placed an order yet. Where do you put them? The table requires an OrderID, but they have no orders.

You either:

- Invent a fake order (bad)
- Leave OrderID NULL (breaks the key)
- Create a separate customer table (correct)

2.5 Delete Anomaly

If you delete the only order from a customer, you lose all information about that customer. They vanish from your database like they never existed.

2.6 The Cost of Messy Data

Bad database design is expensive:

- Storage bloat from redundant data
- Slower queries from scanning duplicates
- Data inconsistency from update anomalies
- Developer headaches from complex application logic
- Lost data from delete anomalies

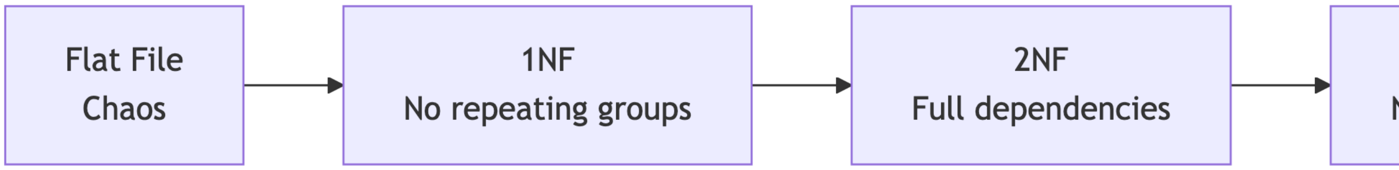
Normalization prevents all of these problems.

3 Normalization Overview

3.1 What Is Normalization?

Normalization is the process of organizing data to:

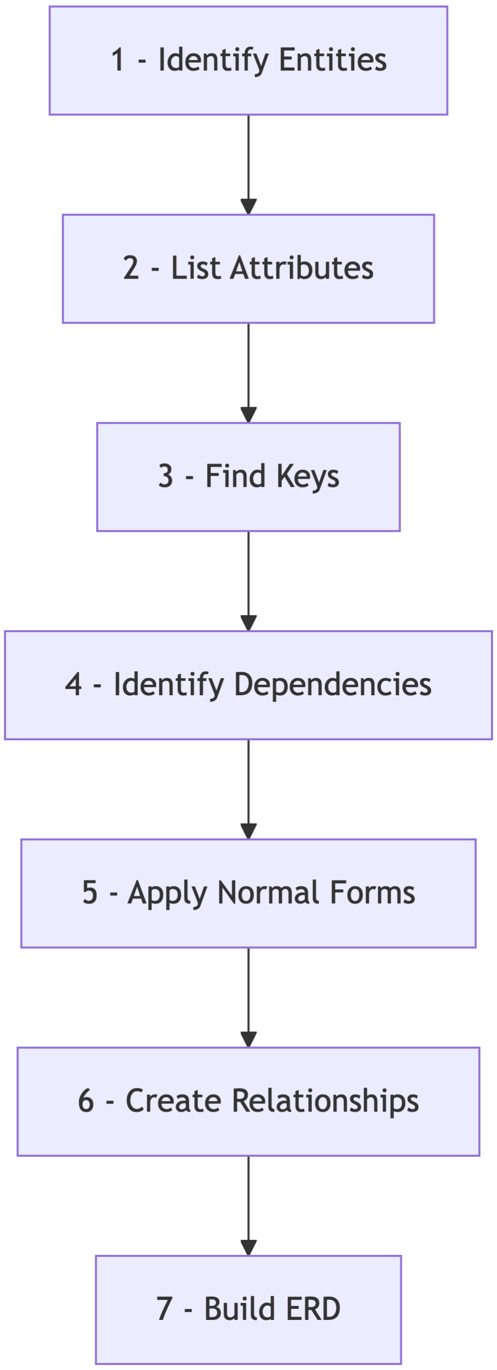
- Minimize redundancy (do not repeat yourself)
- Eliminate anomalies (prevent bad things)
- Ensure data integrity (keep it consistent)
- Enable efficient querying (make it fast)



Think of it as Marie Kondo for databases. Does this column spark joy in this table? No? It belongs somewhere else.

3.2 The Normalization Process Overview

The high-level workflow looks like this:



Step	Question to Ask
1. Identify Entities	What things exist in this domain?
2. List Attributes	What describes each entity?

Step	Question to Ask
3. Find Keys	What uniquely identifies each row?
4. Identify Dependencies	What determines what?
5. Apply Normal Forms	Should we split this table?
6. Create Relationships	How do we connect tables with foreign keys?
7. Build ERD	How do we document and communicate the design?

4 Normal Forms in Detail

4.1 What Is a Key?

A **key** is one or more attributes that uniquely identify a row in a table.

Keys are foundational to relational databases:

- They enforce uniqueness (no duplicate rows)
- They enable relationships between tables
- They are essential for normalization analysis

Without keys, we cannot distinguish one row from another.

4.2 Types of Keys

4.3 Primary Key

The chosen key used to uniquely identify rows in a table. Every table should have exactly one primary key.

```
employee(employee_id, full_name, email)
         -----------
              PK
```

4.4 Candidate Key

Any attribute or set of attributes that could serve as the primary key. A table may have multiple candidate keys.

In an `employee` table, both `employee_id` and `email` might be unique. Both are candidate keys, but we choose one as primary.

4.5 Super Key

Any set of attributes that uniquely identifies rows, including extra (non-minimal) attributes.

{employee_id, full_name} is a super key, but not a candidate key because `full_name` is unnecessary for uniqueness.

4.6 Foreign Key

An attribute in one table that references the primary key of another table. Creates relationships between tables.

```
employee.department_id -> department.department_id
```

4.7 Candidate Keys vs Primary Keys

Candidate keys are all possible keys. **Primary key** is the one you pick.

employee_id	email	full_name
1	michael@dm.com	Michael Scott
2	dwight@dm.com	Dwight Schrute
3	jim@dm.com	Jim Halpert

Candidate keys here: `employee_id`, `email`

Both uniquely identify rows. We typically choose `employee_id` as the primary key because:

- It is shorter and more efficient to index
- Email addresses can change
- Numeric IDs are easier to reference in foreign keys

4.8 Understanding Functional Dependencies

Before we dive into normal forms, we need to understand **functional dependencies**.

A functional dependency exists when one attribute determines another:

StudentID -> StudentName

This means: if you know the StudentID, you can determine exactly one StudentName.

Written as: StudentID *functionally determines* StudentName

Tip

Think of it like a function in programming: `getStudentName(studentID)` always returns the same name for the same ID.

4.9 Types of Dependencies

4.10 Full Dependency

An attribute depends on the entire primary key, not just part of it.

In a table with key (OrderID, ProductID):

- Quantity depends on BOTH OrderID AND ProductID (full)
- CustomerName depends only on OrderID (partial)

4.11 Partial Dependency

An attribute depends on only part of a composite key.

This is a problem. It means data is in the wrong table.

4.12 Transitive Dependency

A -> B -> C where A determines B, and B determines C, but B is not a key.

Example: StudentID -> DeptID -> DeptName

The student determines the department, and the department determines the department name. But DeptID is not a candidate key.

4.13 First Normal Form (1NF)

Rule: No repeating groups or arrays. Each cell contains exactly one atomic value.

4.14 Violation

EmpID	Name	PhoneNumbers
1	Michael	555-1234, 555-5678
2	Dwight	555-9999

Multiple values in one cell. SQL cannot query individual phone numbers easily.

4.15 Fixed (1NF)

EmpID	Name	PhoneNumber
1	Michael	555-1234
1	Michael	555-5678
2	Dwight	555-9999

One value per cell. Now we can search, filter, and index phone numbers.

4.16 1NF: Another Common Violation

Repeating columns are also a violation:

4.17 Violation

StudentID	Course1	Grade1	Course2	Grade2	Course3	Grade3
S001	Math	A	Physics	B		
S002	Math	B	Chemistry	A	Biology	A

What if a student takes 10 courses? 50 courses?

StudentID	Course	Grade

4.18 Fixed (1NF)

StudentID	Course	Grade
S001	Math	A
S001	Physics	B
S002	Math	B
S002	Chemistry	A
S002	Biology	A

Scalable. Clean. Queryable.

4.19 Second Normal Form (2NF)

Rule: Must be in 1NF, plus no partial dependencies on the primary key.

This only matters when you have a composite primary key.

4.20 Violation

Primary Key: (OrderID, ProductID)

OrderID	ProductID	ProductName	Quantity	UnitPrice
1001	P01	Widget	5	9.99
1001	P02	Gadget	2	14.99
1002	P01	Widget	10	9.99

ProductName and UnitPrice depend only on ProductID, not the full key!

OrderID	ProductID	Quantity

4.21 Fixed (2NF)

Orders Table:

OrderID	ProductID	Quantity
1001	P01	5
1001	P02	2
1002	P01	10

Products Table:

ProductID	ProductName	UnitPrice
P01	Widget	9.99
P02	Gadget	14.99

Now each table has a single theme.

4.22 Third Normal Form (3NF)

Rule: Must be in 2NF, plus no transitive dependencies.

Non-key attributes should depend only on the key, not on other non-key attributes.

4.23 Violation

EmpID	EmpName	DeptID	DeptName	DeptLocation
1	Michael	D01	Management	Floor 3
2	Dwight	D02	Sales	Floor 2
3	Jim	D02	Sales	Floor 2

EmpID -> DeptID -> DeptName (transitive dependency)

4.24 Fixed (3NF)

Employees Table:

EmpID	EmpName	DeptID
1	Michael	D01
2	Dwight	D02
3	Jim	D02

Departments Table:

DeptID	DeptName	DeptLocation
D01	Management	Floor 3
D02	Sales	Floor 2

Update the department once, and it is correct everywhere.

4.25 The 3NF Mantra

Bill Kent famously summarized 3NF as:

> "Every non-key attribute must provide a fact about the key, the whole key, and nothing but the key, so help me Codd."

- **The key:** Attribute depends on the primary key
- **The whole key:** Depends on ALL of the key (no partial dependencies)
- **Nothing but the key:** Does not depend on non-key attributes (no transitive dependencies)

4.26 Boyce-Codd Normal Form (BCNF)

Rule: A stricter version of 3NF. Every determinant must be a candidate key.

Most tables in 3NF are also in BCNF. The difference matters in edge cases with overlapping candidate keys.

4.27 Example Scenario

Consider course scheduling where:

- Each student takes one course per semester
- Each course is taught by one professor
- Each professor teaches only one course

Student	Course	Professor
Alice	Math101	Dr. Smith
Bob	Math101	Dr. Smith
Carol	Phys201	Dr. Jones

Candidate keys: (Student, Course) or (Student, Professor)

Professor -> Course is a dependency, but Professor is not a superkey.

4.28 BCNF Solution

Enrollment:

Student	Professor
Alice	Dr. Smith
Bob	Dr. Smith
Carol	Dr. Jones

Teaching:

Professor	Course
Dr. Smith	Math101
Dr. Jones	Phys201

Now every determinant is a candidate key.

4.29 When to Stop Normalizing

3NF is usually sufficient for most applications. BCNF is the practical maximum.

Higher normal forms exist (4NF, 5NF, 6NF) but are rarely needed in practice.

Warning: Over-normalization can hurt performance:

- Too many joins slow down queries
- Some denormalization is acceptable for read-heavy workloads
- Data warehouses often use denormalized star schemas

The goal is eliminating anomalies, not winning a normalization contest.

4.30 Normal Forms Summary

Form	Rule	Fixes
1NF	Atomic values only	Repeating groups
2NF	No partial dependencies	Composite key issues
3NF	No transitive dependencies	Non-key -> non-key
BCNF	Every determinant is a candidate key	Overlapping keys

5 Relational Notation

5.1 What Is Relational Notation?

Relational notation is a shorthand, textual way to represent a database table's structure, including its name, attributes, primary keys, and foreign keys.

It acts as a stepping stone between:

- Identifying entities and attributes (conceptual design)
- Creating ERD diagrams (visual/logical design)
- Writing SQL CREATE statements (physical implementation)

Think of it as pseudocode for database design. ERDs are the visual, logical representation of the same information that relational notation captures in text form.

5.2 Relational Notation Syntax

The basic format is:

```
table_name(attribute_1, attribute_2, attribute_3, ...)
```

Conventions:

- **Primary Key:** Underlined or marked with PK
- **Foreign Key:** Marked with FK, often with arrow to referenced table
- **Composite Key:** Multiple underlined attributes
- **Naming:** Use snake_case to align with PostgreSQL conventions

Example:

```
employee(employee_id, full_name, department_id, email)
         -----------              -------------
              PK                   FK -> department
```

5.3 Reading Relational Notation

Given this notation:

```
department(department_id, department_name, location)
           -------------

employee(employee_id, full_name, department_id, hire_date)
         -----------             -------------
                                  FK -> department
```

We understand:

- Two tables exist: `department` and `employee`
- Each department has an ID (PK), name, and location
- Each employee has an ID (PK), name, department reference (FK), and hire date
- Employees link to departments via `department_id`

5.4 Why Use Relational Notation?

Benefits:

- Quick to write and read
- Easy to iterate on design
- Technology agnostic
- Good for whiteboard discussions
- Natural precursor to ERD creation

You can sketch a database design in minutes without opening any tools.

6 Normalization Activity: Dunder Mifflin

6.1 Our Dataset: The Office

We have employee data from Dunder Mifflin Paper Company:

```
employee_id,full_name,department,email,hire_date,salary_usd,
is_manager,performance_rating,years_experience,commission_rate,last_login
1,Michael Scott,Management,michael.scott@dundermifflin.com,2005-03-24,
75000.00,true,3.2,15,,2026-01-15 09:12:00
2,Dwight Schrute,Sales,dwight.schrute@dundermifflin.com,2006-04-12,
62000.00,false,4.8,12,0.08,2026-01-16 08:01:00
...
```

Let us normalize this data step by step.

6.2 Step 1: Examine the Flat File

employee_id	full_name	department	email	hire_date	salary_usd	is_manager	performance_rating	years_experience	commission_rate	last_login
1	Michael Scott	Management	michael.scott@...	2005-03-24	75000.00	true	3.2	15		2026-01-15
2	Dwight Schrute	Sales	dwight.schrute@...	2006-04-12	62000.00	false	4.8	12	0.08	2026-01-16

employee_id	full_name	department	email	hire_date	salary_usd	is_manager	performance_rating	years_experience	commission_rate	last_login
3	Pam Beesly	Reception	pam.beesly@...	2007-07-02	42000.00	false	3.9	8		
4	Jim Halpert	Sales	jim.halpert@...	2005-10-05	61000.00	false	4.1	10	0.07	2026-01-16
5	Angela Martin	Accounting	angela.martin@...	2006-08-15	52000.00	false	4.5	11		2026-01-15

Questions to ask:

- What are the entities here?
- What determines what?
- Is there redundancy?

6.3 Step 2: Check 1NF

Is this data in First Normal Form?

Checklist:

- Each cell has one value? **Yes**
- No repeating groups? **Yes**
- All rows unique? **Yes** (employee_id is unique)

This data is already in 1NF. Good start.

6.4 Step 3: Identify Dependencies

What does employee_id determine?

```
employee_id -> full_name
employee_id -> department
employee_id -> email
employee_id -> hire_date
employee_id -> salary_usd
employee_id -> is_manager
employee_id -> performance_rating
employee_id -> years_experience
employee_id -> commission_rate
employee_id -> last_login
```

Are there other dependencies?

```
department -> ??? (Does department determine anything else?)
```

Think about it: does knowing the department tell us anything specific?

6.5 Step 4: Check 2NF

Is there a composite key? **No**, employee_id alone is the primary key.

With a single-column primary key, 2NF is automatically satisfied.

We are in 2NF.

6.6 Step 5: Check 3NF and Find Issues

Looking for transitive dependencies…

The `department` column contains the department name directly. What if departments have additional attributes?

Consider:

- Department -> Department Manager
- Department -> Cost Center
- Department -> Floor Location

If we wanted to add these, we would repeat them for every employee in that department.

Problem Identified: Department should be its own entity.

6.7 Step 6: Decompose to 3NF

Let us create a separate department table:

Relational Notation:

```
department(department_id, department_name)
           -------------

employee(employee_id, full_name, department_id, email, hire_date,
         -----------             -------------
         salary_usd, is_manager, performance_rating,
         years_experience, commission_rate, last_login)

         FK: department_id -> department(department_id)
```

6.8 Step 7: Consider Additional Decomposition

Should we split further?

Sales-specific attributes:

- `commission_rate` only applies to Sales employees
- NULL for everyone else

Options:

1. Keep it simple (NULLs are okay)
2. Create a `sales_employee` subtype table

For this dataset, option 1 is reasonable. The NULL rate is low.

6.9 Step 8: Consider Login Tracking

The `last_login` column represents temporal data. What if we want login history?

Current: Single timestamp, overwrites each time

Better design for history:

```
login_history(login_id, employee_id, login_timestamp)
              --------  -----------
                            FK -> employee

employee(employee_id, ...)  -- Remove last_login
```

But for the current requirement (just last login), the original design works.

6.10 Final Normalized Design

```
department(department_id, department_name)
           -------------

employee(employee_id, full_name, department_id, email, hire_date,
         -----------             -------------
         salary_usd, is_manager, performance_rating,
         years_experience, commission_rate, last_login)

FK: employee.department_id -> department.department_id
```

6.11 From Design to Implementation

We have completed the logical design using relational notation:

```
department(department_id, department_name)
           -------------

employee(employee_id, full_name, department_id, email, hire_date,
         -----------             -------------
         salary_usd, is_manager, performance_rating,
         years_experience, commission_rate, last_login)

FK: employee.department_id -> department.department_id
```

Coming Soon: Physical Implementation

In an upcoming lecture, we will cover:

- **DDL (Data Definition Language):** CREATE TABLE, ALTER TABLE, DROP TABLE
- **Table Constraints:** PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL,

CHECK
- **Data Types:** Choosing appropriate PostgreSQL types
- **Importing Data:** Loading CSV files into normalized tables

For now, focus on understanding how to design the logical structure.

6.12 Activity: Your Turn

Working in pairs, answer these questions:

1. What other attributes might Department have in a real company?
2. Should `is_manager` be a separate table or role-based system?
3. How would you model an employee having multiple departments?
4. What index would you add and why?

Take 5 minutes, then we will discuss.

7 Entity-Relationship Diagrams

7.1 What Is an ERD?

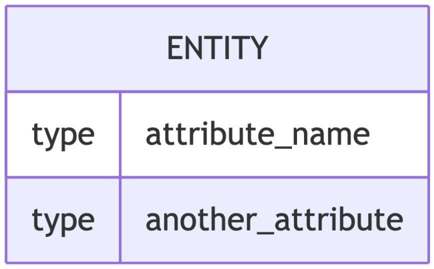
An **Entity-Relationship Diagram (ERD)** is a visual representation of your database structure.

ERDs show:

- Entities (tables)
- Attributes (columns)
- Relationships (foreign keys)
- Cardinality (one-to-one, one-to-many, many-to-many)

ERDs are the universal language of database design.

7.2 ERD Components



7.3 Entities

Rectangles represent tables. The entity name appears at the top.

7.4 Attributes

Listed inside the entity box with their data types.

7.5 Relationships

Lines connecting entities show how they relate. The line style indicates cardinality.

7.6 Cardinality Notation

- `||` One (mandatory)
- `|o` Zero or one (optional)
- `}|` Many (at least one)
- `}o` Zero or many

7.7 Crow's Foot Notation

The most common ERD notation style:

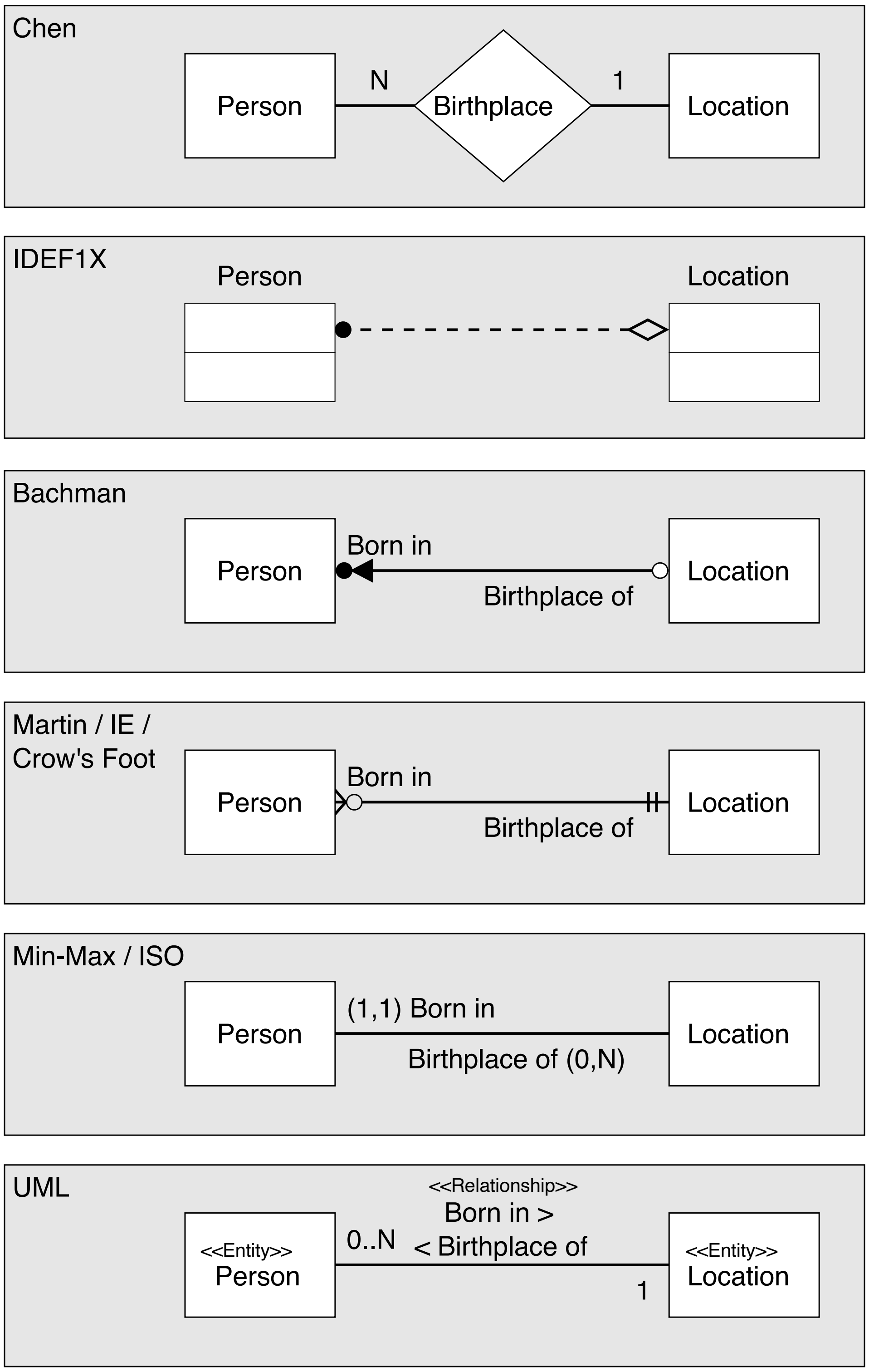
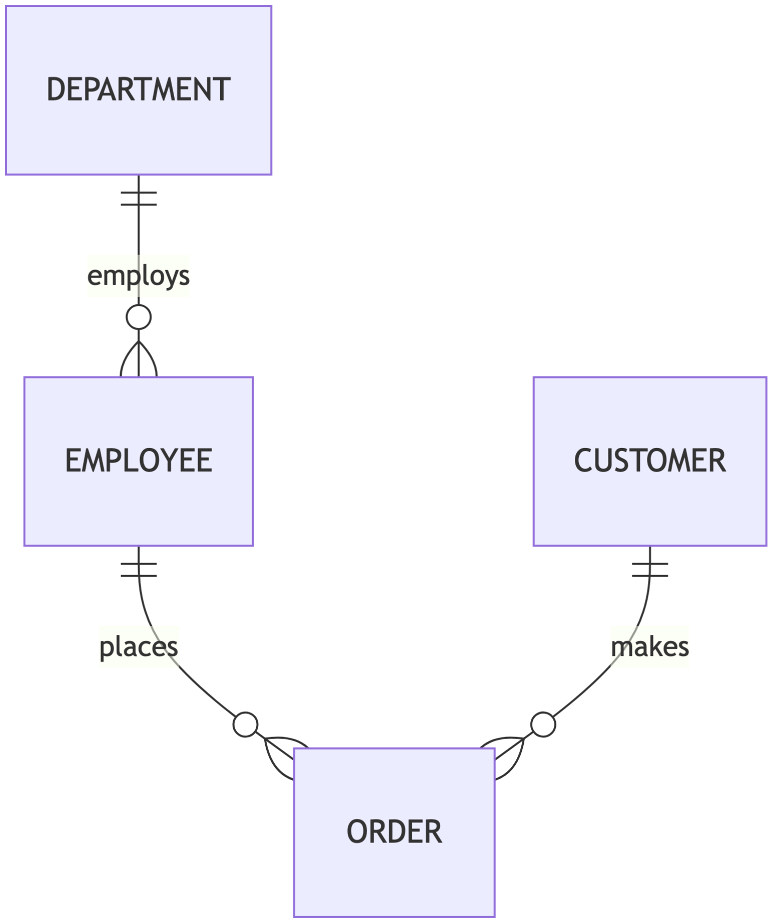


Figure 1: Crow's Foot Notation Symbols

Symbol	Meaning
Line with perpendicular line	One (mandatory)
Line with circle	Zero (optional)
Line with crow's foot	Many

7.8 Reading ERD Relationships



Reading this diagram:

- One DEPARTMENT employs zero or many EMPLOYEEs
- One EMPLOYEE places zero or many ORDERs
- One CUSTOMER makes zero or many ORDERs

7.9 Creating ERDs with draw.io

draw.io (also known as diagrams.net) is a free, browser-based diagramming tool.

Why draw.io?

- Free and open source
- Works in browser (no installation)
- Has built-in ERD shapes
- Exports to PNG, PDF, SVG
- Integrates with Google Drive, GitHub

Access it at: https://app.diagrams.net

7.10 draw.io: Getting Started

7.11 Step 1: Create New

1. Go to app.diagrams.net
2. Choose where to save (local, Google Drive, etc.)
3. Click "Create New Diagram"
4. Select "Entity Relationship" template

7.12 Step 2: Add Entities

1. Find "Entity Relation" shapes in the left panel
2. Drag an "Entity" shape to the canvas
3. Double-click to rename
4. Click the + to add attributes

7.13 Step 3: Add Relationships

1. Hover over an entity until you see connection points
2. Drag a line from one entity to another
3. Right-click the line to change the line style
4. Add relationship labels

7.14 Step 4: Style and Export

1. Adjust colors and fonts as needed
2. File > Export as > PNG/PDF/SVG
3. Include in your documentation

7.15 draw.io ERD Example

Creating our Dunder Mifflin schema:

1. Create "Department" entity with:
    - department_id (PK)
    - department_name
2. Create "Employee" entity with:
    - employee_id (PK)
    - full_name
    - department_id (FK)
    - email
    - hire_date
    - salary_usd
    - … other attributes
3. Connect Employee to Department with one-to-many line

7.16 Alternative Tools

While we focus on draw.io, other options exist:

Tool	Pros	Cons
draw.io	Free, easy, browser-based	Manual relationship lines
Lucidchart	Polish, collaboration	Paid for full features
dbdiagram.io	Code-based syntax	Limited free tier
pgModeler	PostgreSQL specific	Steeper learning curve
MySQL Workbench	Reverse engineering	MySQL focused

For this course, draw.io is sufficient and recommended.

7.17 ERD Best Practices

1. **Consistent naming:** Use same conventions throughout
2. **Show cardinality:** Always indicate relationship types
3. **Include keys:** Mark PK and FK clearly
4. **Logical grouping:** Place related entities near each other
5. **Avoid crossing lines:** Rearrange to minimize line crossings
6. **Add descriptions:** Include relationship verb phrases
7. **Keep it readable:** Break into sub-diagrams if too complex

7.18 Activity: Create an ERD

Using draw.io, create an ERD for a simplified library system:

Entities:

- Book (isbn, title, publication_year)
- Author (author_id, name, birth_year)
- Member (member_id, name, email, join_date)
- Loan (loan_id, loan_date, return_date)

Relationships:

- A book can have multiple authors
- An author can write multiple books
- A member can have multiple loans
- Each loan is for one book

Take 10 minutes, then share your diagrams.

8 Applying Normalization: Oregon Turtle Conservation

8.1 A Real-World Normalization Challenge

Oregon's native freshwater turtles need help. The **Oregon Conservation and Recreation Fund (OCRF)** and **Oregon Department of Fish and Wildlife (ODFW)** track turtle sightings through a citizen science platform.

The raw sighting data arrives as a denormalized CSV file with issues we need to fix:

- Repeated submitter information across rows
- Multiple observations per submission
- Redundant location data
- Mixed temporal data (submission time vs. observation time)

This is exactly the kind of messy data you will encounter in practice.

8.2 The Conservation Context

Two native species are in decline:

- **Northwestern Pond Turtle** (*Actinemys marmorata*)
- **Western Painted Turtle** (*Chrysemys picta bellii*)

Threatened by two invasive species:

- **Red-eared Slider** (*Trachemys scripta elegans*)
- **Common Snapping Turtle** (*Chelydra serpentina*)

Citizens report sightings via mobile app. ODFW biologists review and intervene when necessary.

8.3 The Current Data Structure

The CSV contains denormalized data where a single submission spans multiple rows:

Submission ID	First Name	Email	Latitude	Longitude	Observation ID	Species	Behavior	Count
101	Jane	jane@...	45.123	-122.456	1001	Western-Pond	Basking	3
101	Jane	jane@...	45.123	-122.456	1002	RedEared-Slider	Swimming	1
102	Bob	bob@...	44.987	-123.111	1003	Western-Painted	Basking	2

Notice how Jane's submission information repeats because she observed two species.

8.4 Identifying Normalization Issues

What problems do you see?

8.5 Redundancy

Submitter information (name, email, phone) repeats for every observation within a submission. Location data also repeats.

If Jane submits a report with 5 species, her contact info appears 5 times.

8.6 Update Anomaly Risk

If Jane changes her email, we must update every row where she appears. Miss one? Data inconsistency.

8.7 Entity Mixing

The flat file conflates three distinct concepts:

- **Submitter** (the person reporting)
- **Submission** (a single report event)
- **Observation** (each species sighted)

8.8 Lookup Candidates

Repeated text values that should be normalized:

- Species names
- Behavior types
- Status values (New, Pending, Approved, Rejected)
- Action taken values

8.9 Your Assignment Preview

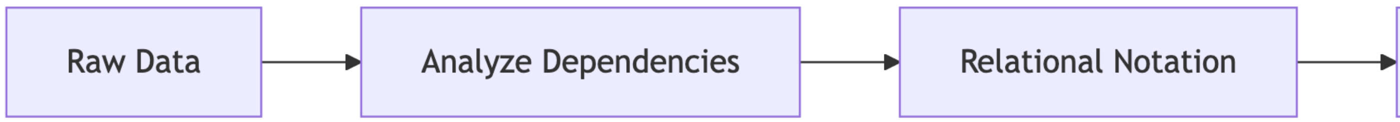
Your assignment will be to design a normalized schema for this turtle sighting data:

1. Analyze the denormalized CSV file
2. Identify all functional dependencies
3. Create a complete Entity-Relationship Diagram normalized to 3NF
4. Document your design decisions and assumptions

This is an **ERD-only assignment**. Focus on the logical design; we will implement the physical schema in a later assignment.

9 Putting It All Together

9.1 The Complete Workflow



This is the professional database design workflow. We cover the first four steps in this lecture and the final two in the upcoming DDL and data import lecture.

9.2 From Spreadsheet to Schema

Starting point: Messy spreadsheet from business users

1. **Understand the data:** What entities exist? What are the relationships?
2. **Identify dependencies:** What determines what?
3. **Apply normal forms:** Decompose until 3NF/BCNF
4. **Document in relational notation:** Quick text representation
5. **Create ERD:** Visual/logical documentation
6. **Write SQL DDL:** CREATE TABLE statements (upcoming lecture)
7. **Migrate data:** Load and transform from source (upcoming lecture)
8. **Validate:** Test queries, check constraints

9.3 Common Pitfalls to Avoid

- Normalizing too early (understand data first)
- Over-normalizing (not everything needs 5 tables)
- Ignoring performance (sometimes denormalization is okay)
- Forgetting NULL handling (design for missing data)
- Skipping documentation (future you will thank present you)
- Not considering growth (will this scale?)

9.4 Key Takeaways

1. **Normalization prevents anomalies** - Update, insert, delete problems go away
2. **3NF is usually sufficient** - Do not over-engineer
3. **Relational notation is fast** - Sketch designs quickly
4. **ERDs communicate designs** - Universal visual language
5. **Practice makes perfect** - The more you normalize, the better you get

9.5 Looking Ahead

Next steps:

- Complete the Oregon Turtle Conservation ERD assignment (Assignment 4)

Questions?

10 References

10.1 References

1. Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6), 377-387.
2. Kent, W. (1983). A Simple Guide to Five Normal Forms in Relational Database Theory. *Communications of the ACM*, 26(2), 120-125.
3. Silberschatz, A., Korth, H., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill.
4. Date, C. J. (2003). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.
5. Oregon Department of Fish and Wildlife. (n.d.). *Oregon Turtle Conservation*. https://www.dfw.state.or.us/wildlife/turtle/
6. diagrams.net. (n.d.). *Free Online Diagram Software*. https://www.diagrams.net