# Lecture 23: Database Management Systems

**Last Day:** Introduction

- Database processing
- Data models
- Entity-Relationship diagrams

**Today:** Relational Databases

- Relations
- Implementing ER diagrams
- Keys / Foreign keys
- Data Definition Language (DDL)
- Data Manipulation Language (DML)

Smith + Barnes, ch. 12.

# Relational Databases

- In relational databases, entities and relationships are represented with one construct: relations.

- A relation is a set of tuples.

- A tuple is an ordered sequence of values, $(v_1, v_2, \cdots, v_n)$.

- A relation is generally stored on disk as a file of records, where each record is a tuple.

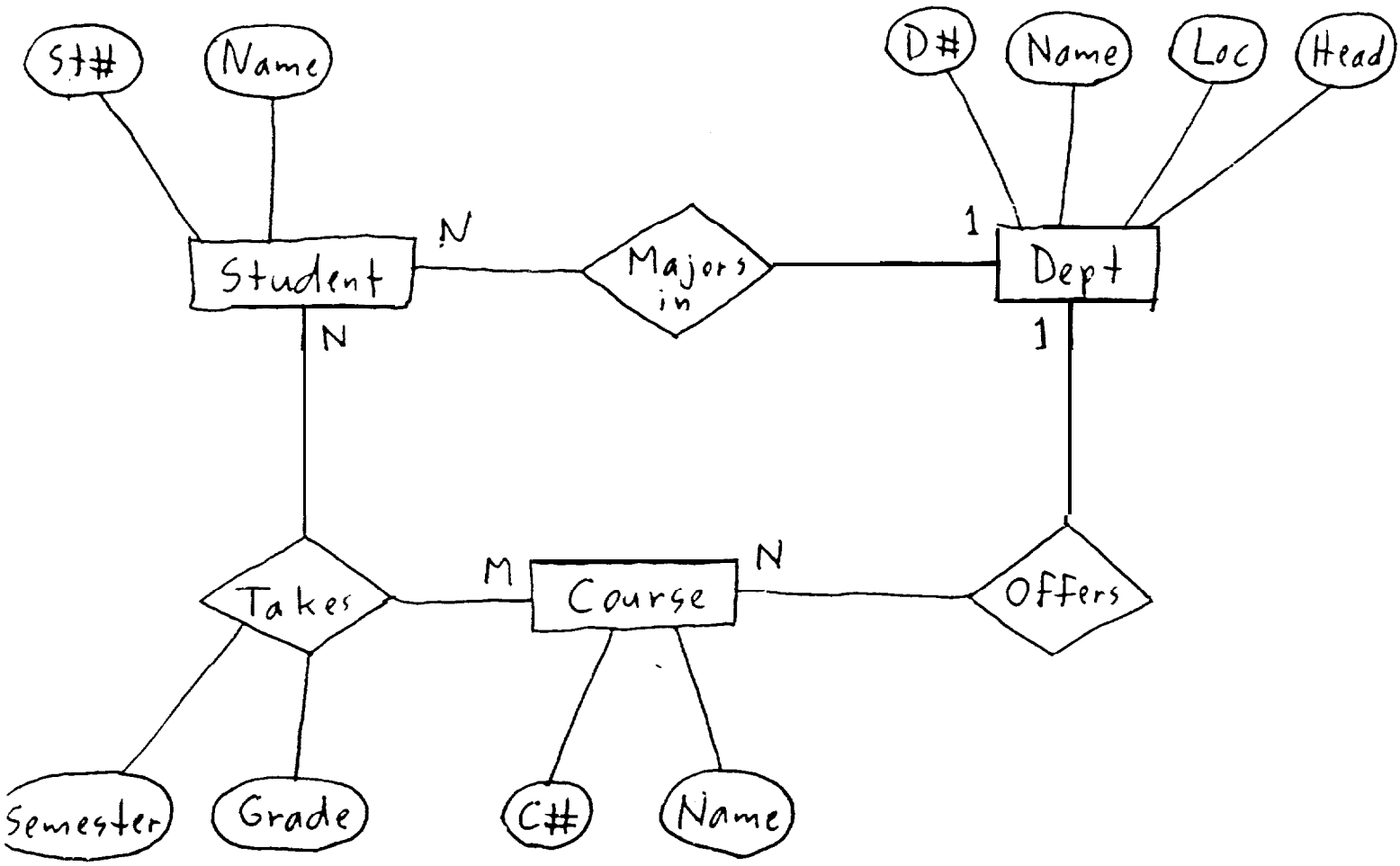
# Observation

- Because a relation is a set,
  - its tuples are <u>not</u> ordered;
  - it contains no duplicate tuples.

- In contrast, in a file,
  - records are ordered (ie, there is a first record, a second record, etc.)
  - the same record may appear many times (in different locations in the file).

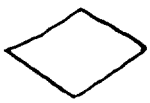
<u>Question</u>: How do we implement entity-relationship diagrams using relations?

# Entity Relationship Diagram : Example

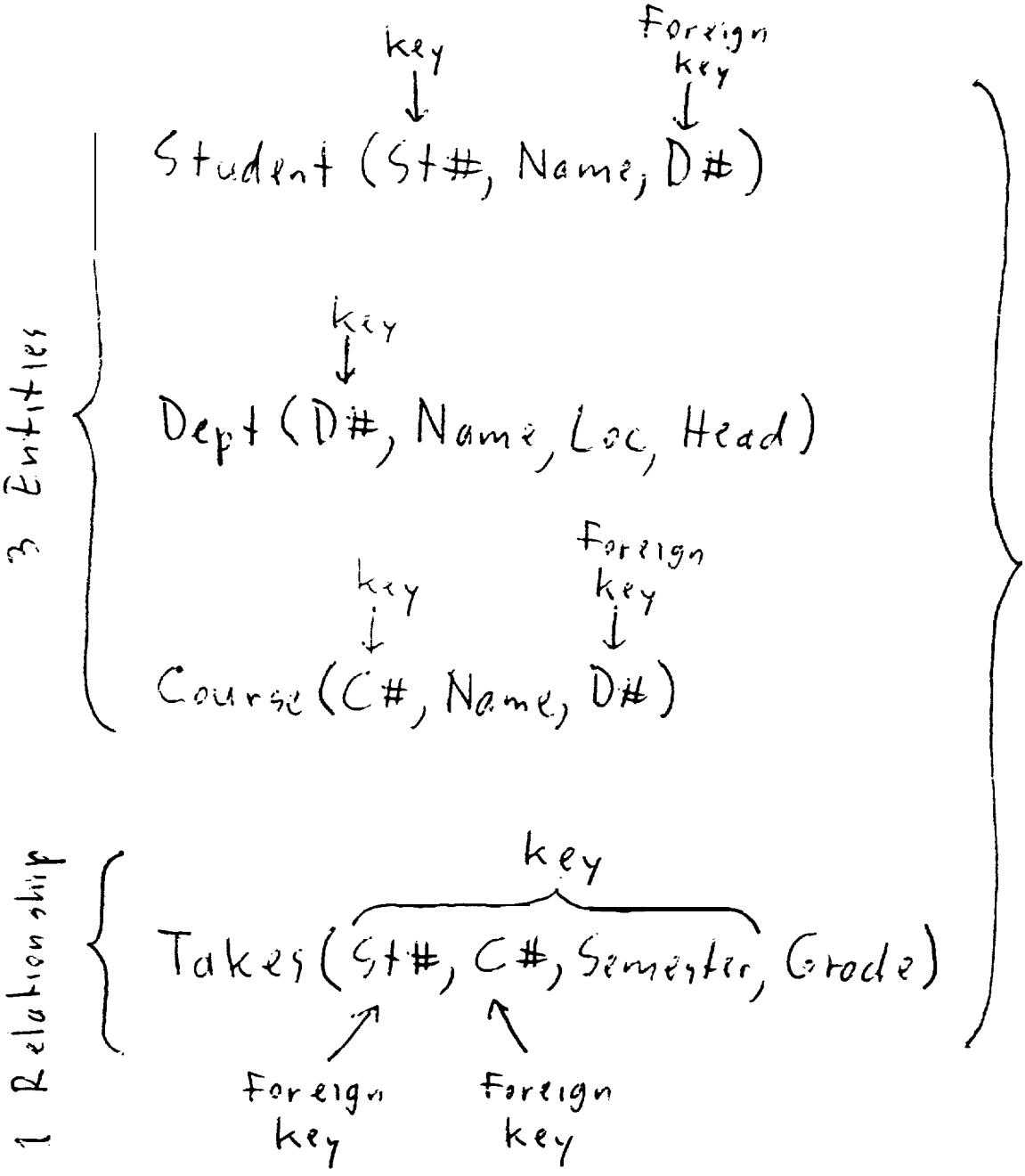


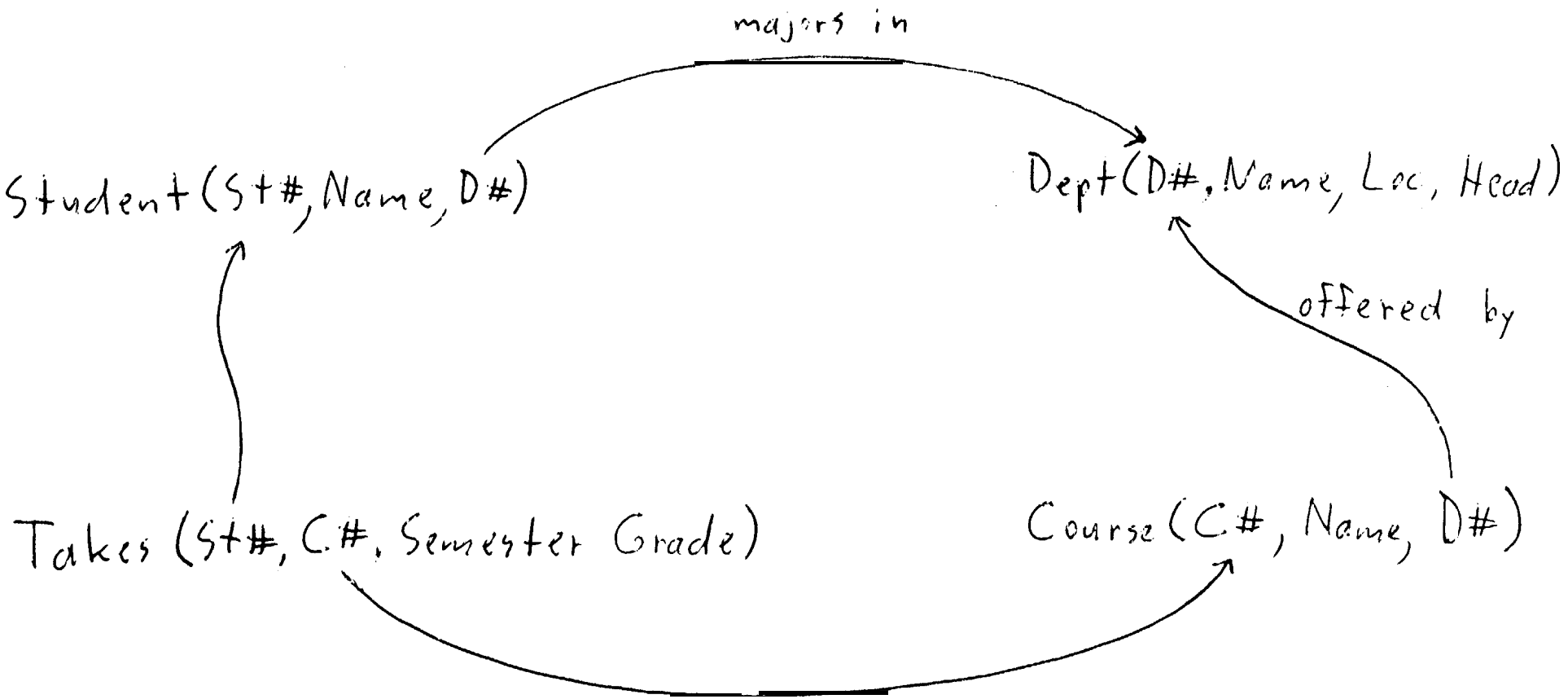

 – Attributes

 – Entities

 – Relationships

# Implementing ER Diagrams as Relations






Correspondence between the ER diagram and its relational implementation.

## Observations

N:1 relationships are implemented as Fields called **foreign keys**.

- A Foreign key identifies a record in **another** relation (ie, It is a **key** for another relation.

- eg. In the **Student** relation, the **D#** field implements the "Majors in" relationship.

- M:N relationships, like "Takes", are implemented as relations with **two** foreign keys.

# Terminology

| ER Model | Relational Model |
| --- | --- |
| Entity | Relation |
| Relationship | Relation / Foreign key |
| Attribute | Field |

# Observation

- Several Fields together can form a key and be used to index a file.

- eg. We can build a B-tree (or hash function) for Student based on St#.

- Likewise, we can build a B-tree (or hash function) for Takes based on the concatenation of St#, C# and Semester.

- In general, any non-empty set of fields can index a file.

# High-Level Database Languages

Relational database systems provide simple languages for

- Creating relations
- Indexing relations

} Data Definition Language (DDL)

- Querying relations
- Updating relations

} Data Manipulation Language (DML)

# Example: DDL Specification

Create Table Student:

St# (Integer, No Null)

Name (Char(20))

D# (Integer)

Create Table Dept:

D# (Integer, No Null)

Name (Char(20))

Loc (Char(5))

Head (Char(20))

# DDL Example (Cont.)

Create Table Course:

C# (Integer, No Null)

Name (Char(20))

D# (Integer)

Create Table Takes:

St# (Integer, No Null)

C# (Integer, No Null)

Semester (Char(1), No Null)

Grade (Real)

These DDL commands generate the Following relational database schema:

Student (St#, Name, D#)

Dept (D#, Name, Loc, Head)

Course (C#, Name, Subject, D#)

Takes (St#, C#, Semester, Grade)

# Data Base Instance (Relational Example)

**Student**

| St# | name | D# |
|---|---|---|
| 298311695 | Jim | 659 |
| 228142362 | Carol | 723 |
| 219367183 | Joan | 659 |
| 238431659 | Alex | 659 |
| 243857296 | Dave | 723 |

**Dept**

| D# | name | Loc | Head |
|---|---|---|---|
| 659 | CS | SF | Corneil |
| 723 | Math | SS | Jones |

**ourse**

| C# | name | Subject | D# |
|---|---|---|---|
| 158 | Programming | Computers | 659 |
| 238 | Discrete Math | Math | 723 |
| 164 | Algebra | Math | 723 |
| 385 | Computability | Computers | 659 |

**Takes**

| St# | C# | Semester | Grade |
|---|---|---|---|
| 298311695 | 164 | F | |
| 298311695 | 385 | S | |
| 228142362 | 164 | F | |
| 238431659 | 238 | F | |
| 243857296 | 158 | S | |
| 243857296 | 385 | S | |

# Data Manipulation Language (DML)

- Two basic operations:
  (1) Selecting data from a given relation.
  (2) Relating data in different relations.

- The most important commercial DML is SQL (Set Query Language).

- Syntax of SQL:

| Select Fields |
| --- |
| From relations |
| where conditions |

+ some frills.

# Single-Relation Queries

## Example 1: Selecting Fields.

"Retrieve the Name and Location of every department"

```
Select Name, Loc
From Dept
```

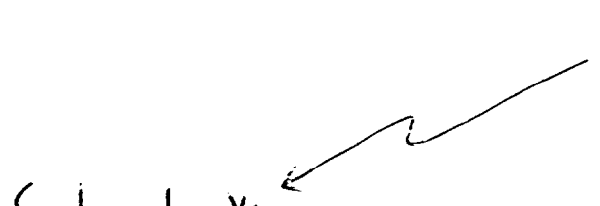
} SQL query

| Name | Loc |
|---|---|
| CS | SF |
| Math | SS |

} query answer

## Example 2: Selecting tuples

"Retrieve the record for the math dept."




```
Select *
From Dept
where Name = 'math'
```

} SQL query

| D# | Name | Loc | Head |
|---|---|---|---|
| 723 | math | 55 | Jones |

} query answer

**Note:** Both the input & output to an SQL query is a relation (ie, a table).

Example 3: Combining tuple & field selection

"Retrieve the head of the Math dept."

```
Select  Head
From    Dept
where   Name = 'math'
```

} SQL query

| Head |
| --- |
| Jones |

} query answer

## Example 4: Combining tuple & field selection

"Retrieve the Course# and Name of all courses about computers"

```
Select C#, Name
From Course
where Subject = 'Computers'
```

} SQL query

| C# | Name |
|---|---|
| 158 | Programming |
| 385 | Computability |

} Query answer

# Multi-Relation Queries

Example 1: "Retrieve the name of each student in the math dept."

Problem: Student data is in one relation, while Dept data is in another relation

Solution: First, "join" the two relations together, to produce one, large relation

Here is the join of the Student and Dept relations:

| St# | Name | D# | Name | Loc | Head |
|---|---|---|---|---|---|
| 298311695 | Jim | 659 | CS | SF | Cornell |
| 228142362 | Carol | 723 | Math | SS | Jones |
| 219367183 | Joan | 659 | CS | SF | Cornell |
| 238431659 | Alex | 659 | CS | SF | Cornell |
| 243957296 | Dave | 723 | Math | SS | Jones |

A subtle point: We now have two Fields called Name

To distinguish them, SQL uses

Student.Name and Dept.Name

The Following SQL query generates this "joined" table:

```
Select S#, Student.Name, D#, Dept.Name, Loc, Head
From Student, Dept
where Student.D# = Dept.D#
```

Specifies that tuples from Student & Dept with the same D# value are to be joined together into a single, long tuple

- We do not have to write all this out, since we do not want the entire joined table.

- Instead, we specify only what we want (the names of all math students) as follows:

SQL Query

```
Select  Student.Name        <- Display student name
From    Student, Dept       <- The tables involved.
where   Student.D# = Dept.D#   <- Join the two tables
and     Dept.Name = 'Math'     <- Select math tuples.
```

Query answer

| Student.Name |
| --- |
| Carol |
| Dave |