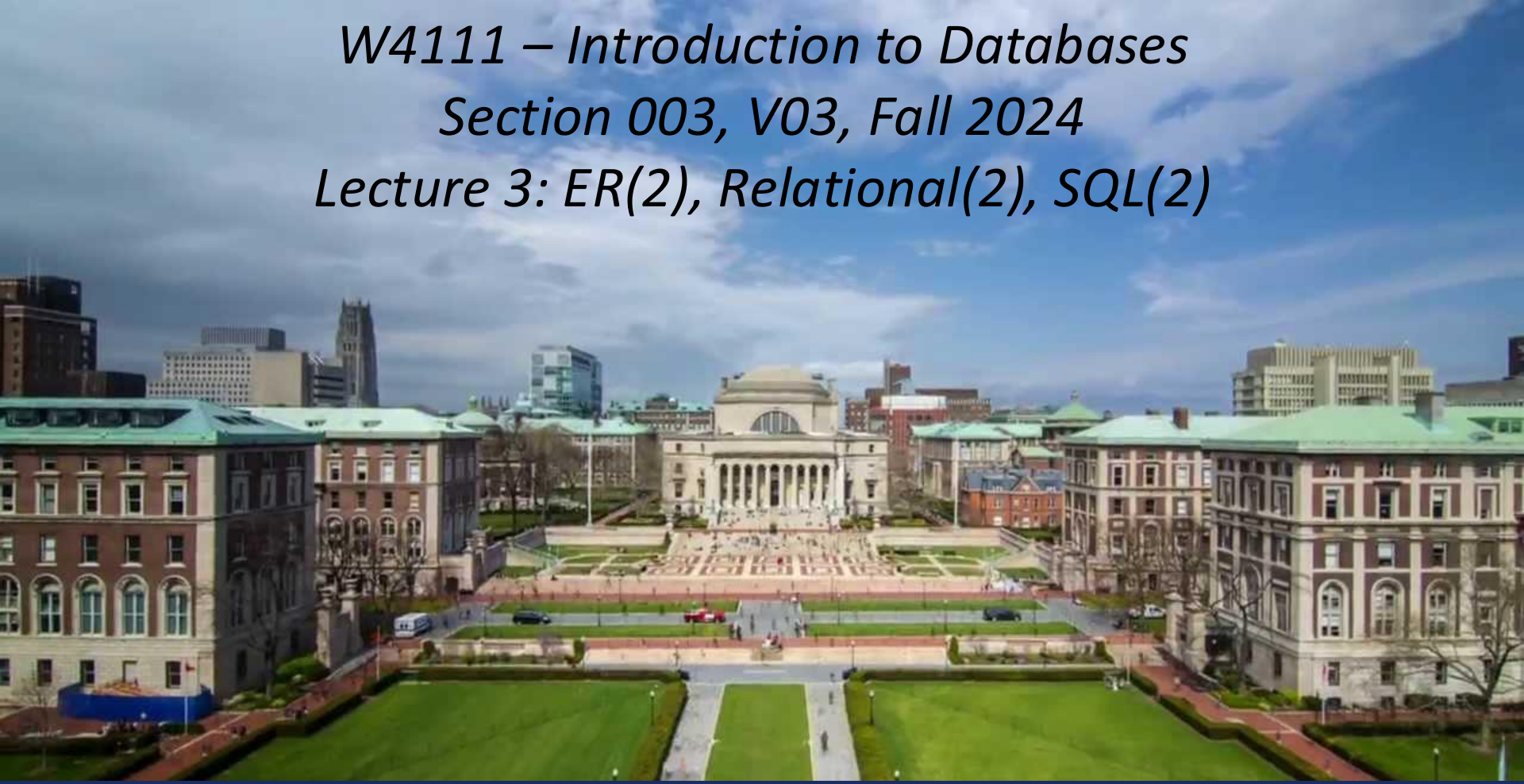
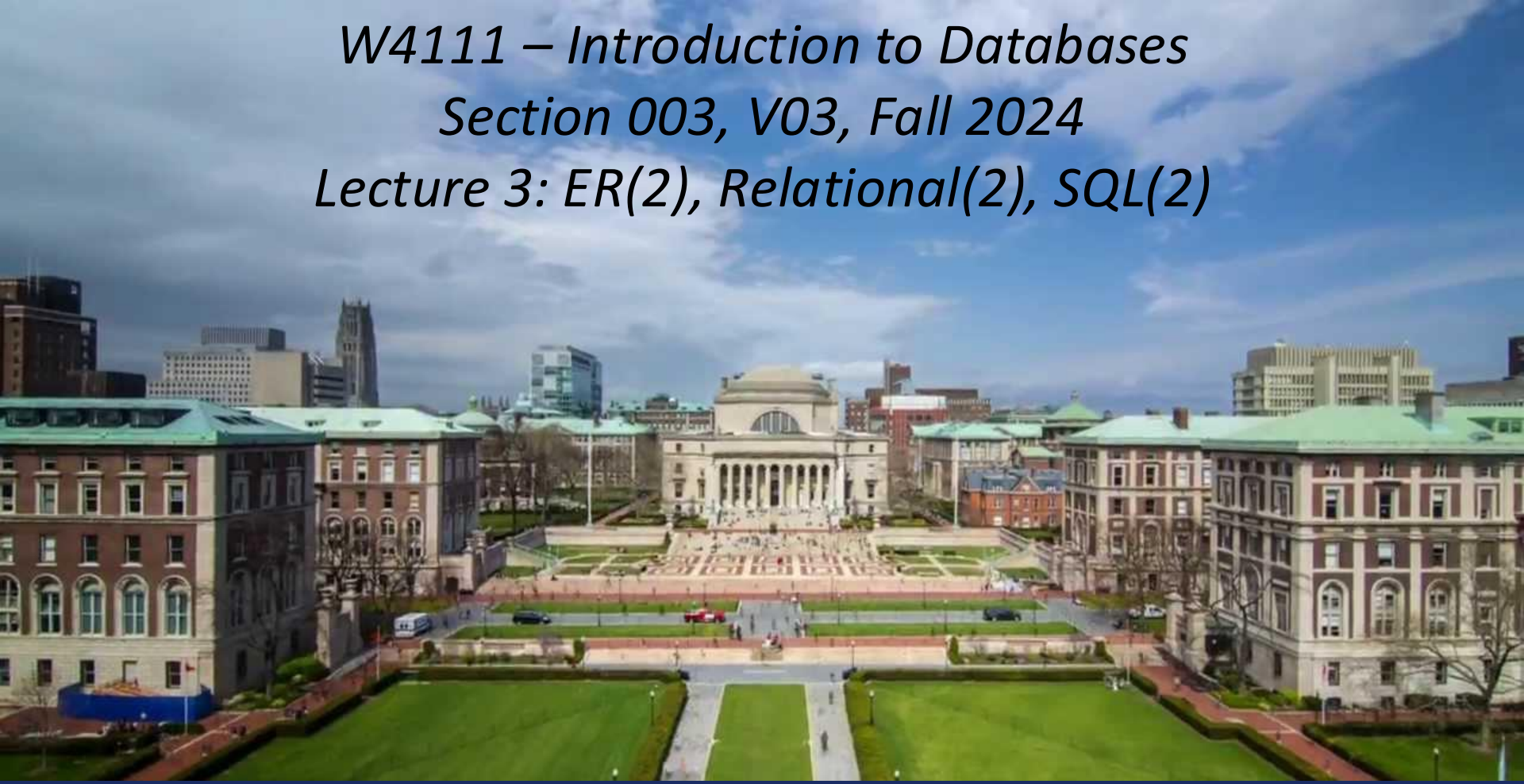
W4111 – Introduction to Databases
Section 003, V03, Fall 2024
Lecture 3: ER(2), Relational(2), SQL(2)



W4111 – Introduction to Databases
Section 003, V03, Fall 2024
Lecture 3: ER(2), Relational(2), SQL(2)



Today's Contents

Contents

- Introduction and course updates.
- ER Modeling (2).
- Relational model and algebra (2).
- SQL (2).
- Homework and projects.

Introduction and course updates

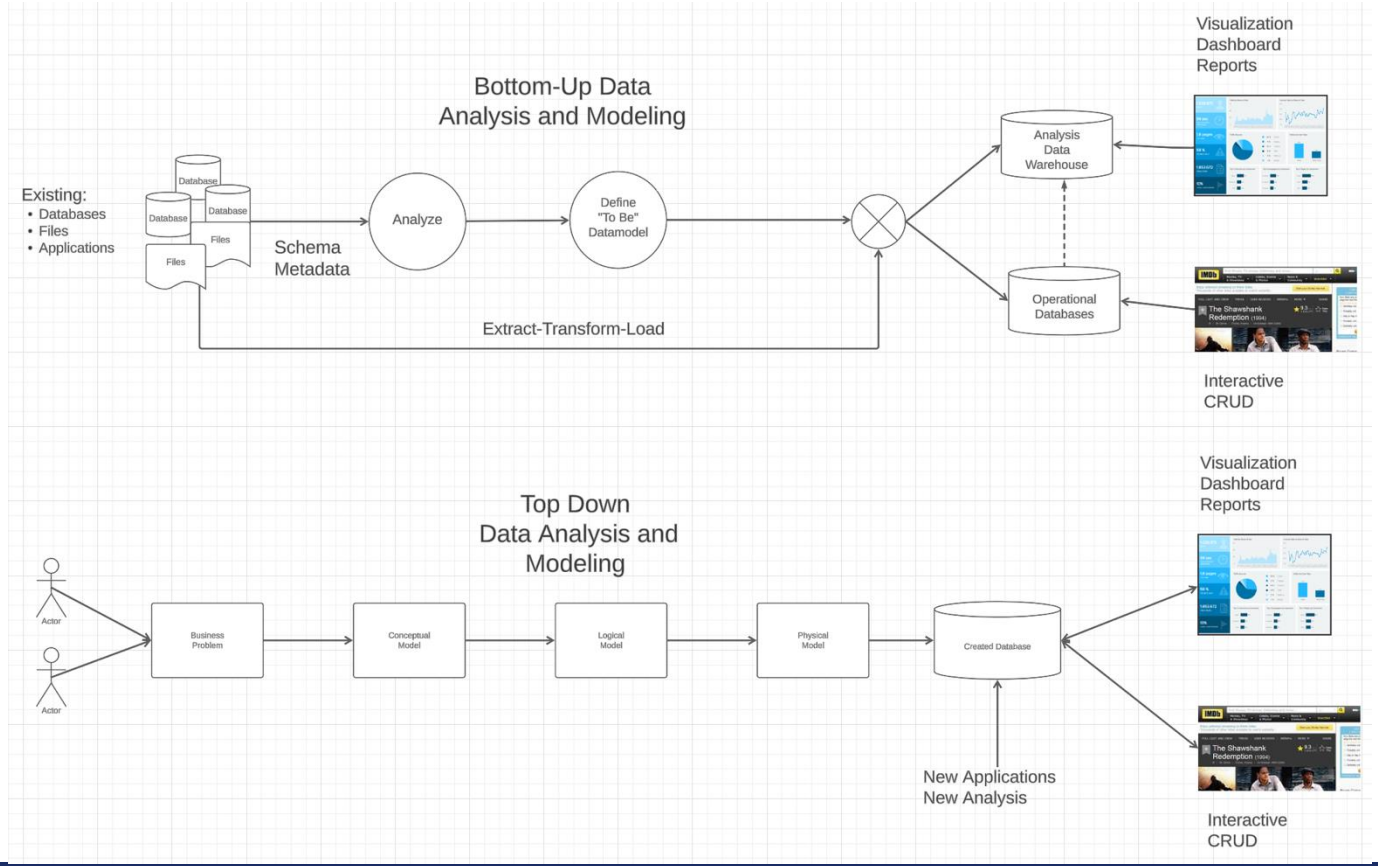
Course Updates

- Homework assignments:
  - In general, a homework has two parts, e.g. HW1a and HW1b.
    - "a" is written questions, and you have one week to complete.
    - "b" is practical "tasks," that also work towards a simple project at the end. You typically have two weeks to complete.
  - **Tentative** schedule:
    - HW1: 21-SEP
    - HW2: 5-OCT
    - HW3: 19-OCT
    - HW4: 2-NOV
    - HW5: 16-NOV
- The midterm is 18-OCT. We will come up with accommodations for students that have "legitimate," constraints, e.g. religious commitments.

ER Modeling

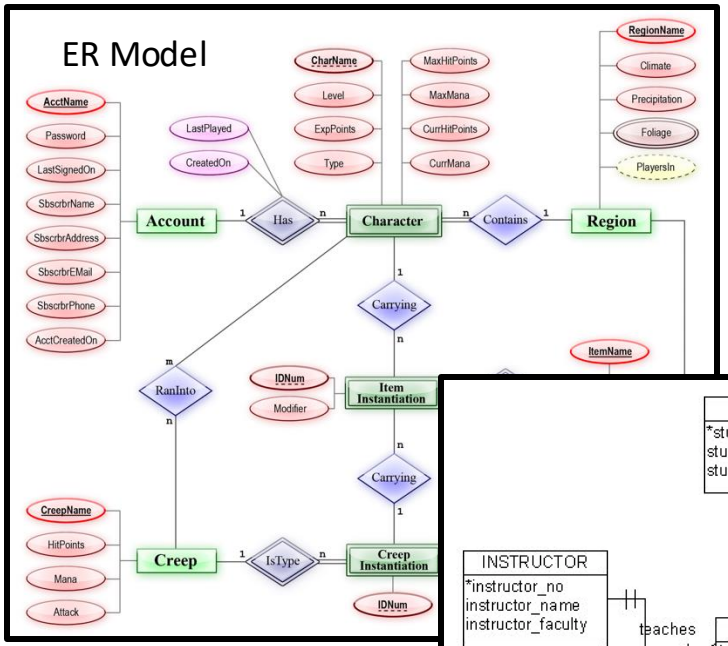
Concepts

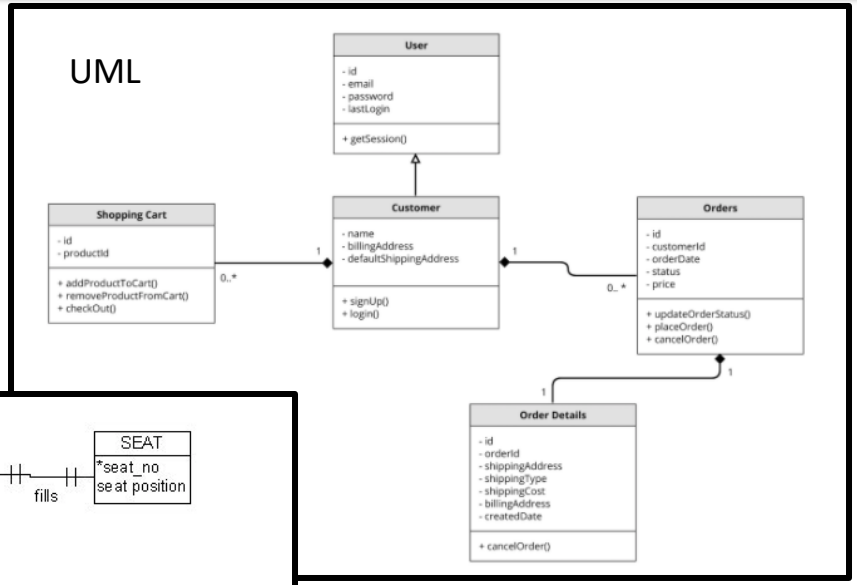
Modeling

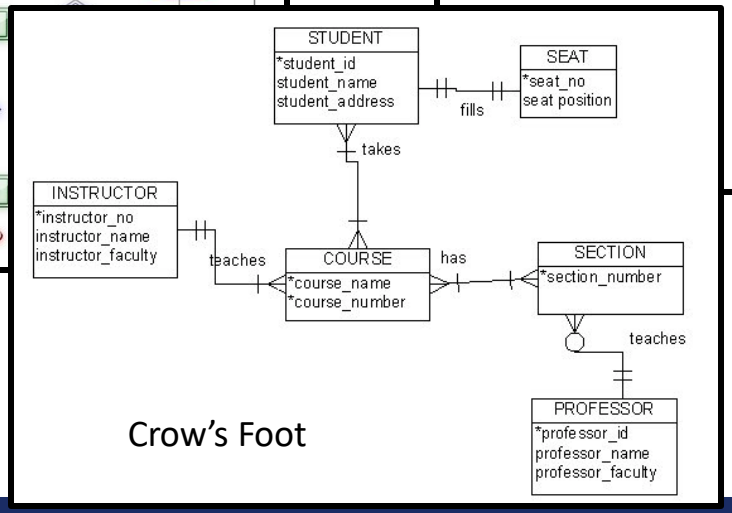


Most of the time there is a mix → Meet-in-the-Middle

Visual Notation – Many Notations





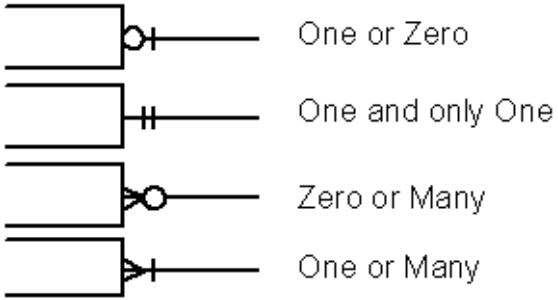


- "Other," i.e. PowerPoint is the most common modeling notation.
- It is easy to get "carried away."
- The trick is to do "just enough modeling."
- I mostly use Crow's Foot
  - It is "just enough"
  - But lacks some capabilities.
- The book uses ER notation.

Notation has Precise Meaning (Crow's Foot)

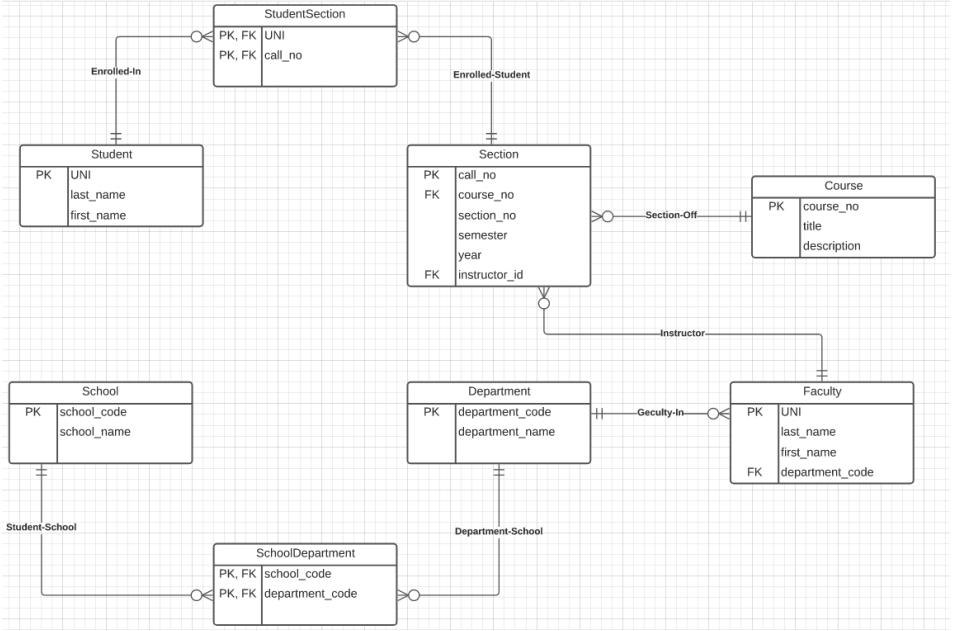
- Attribute annotations:
  - PK = Primary Key
  - FK = Foreign Key
- Line annotations:

Summary of Crow's Foot Notation



- We will learn over time and there are good tutorials (https://www.lucidchart.com/pages/er-diagrams) to help study and refresh.

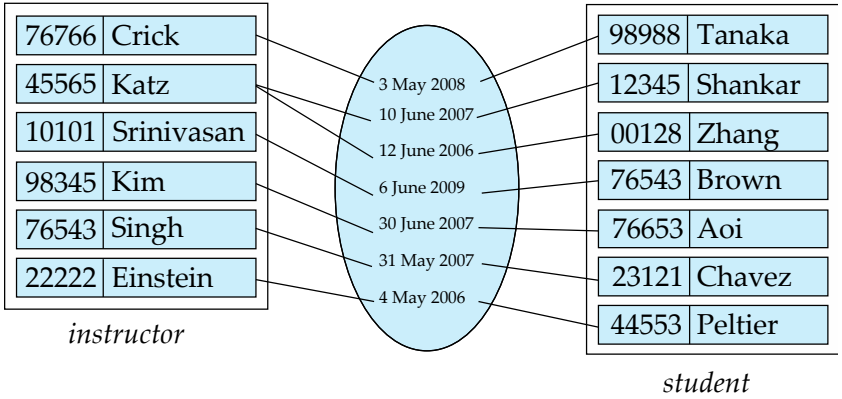
- We will spend a lot of time discussing keys.
- We will start in a couple of slides.



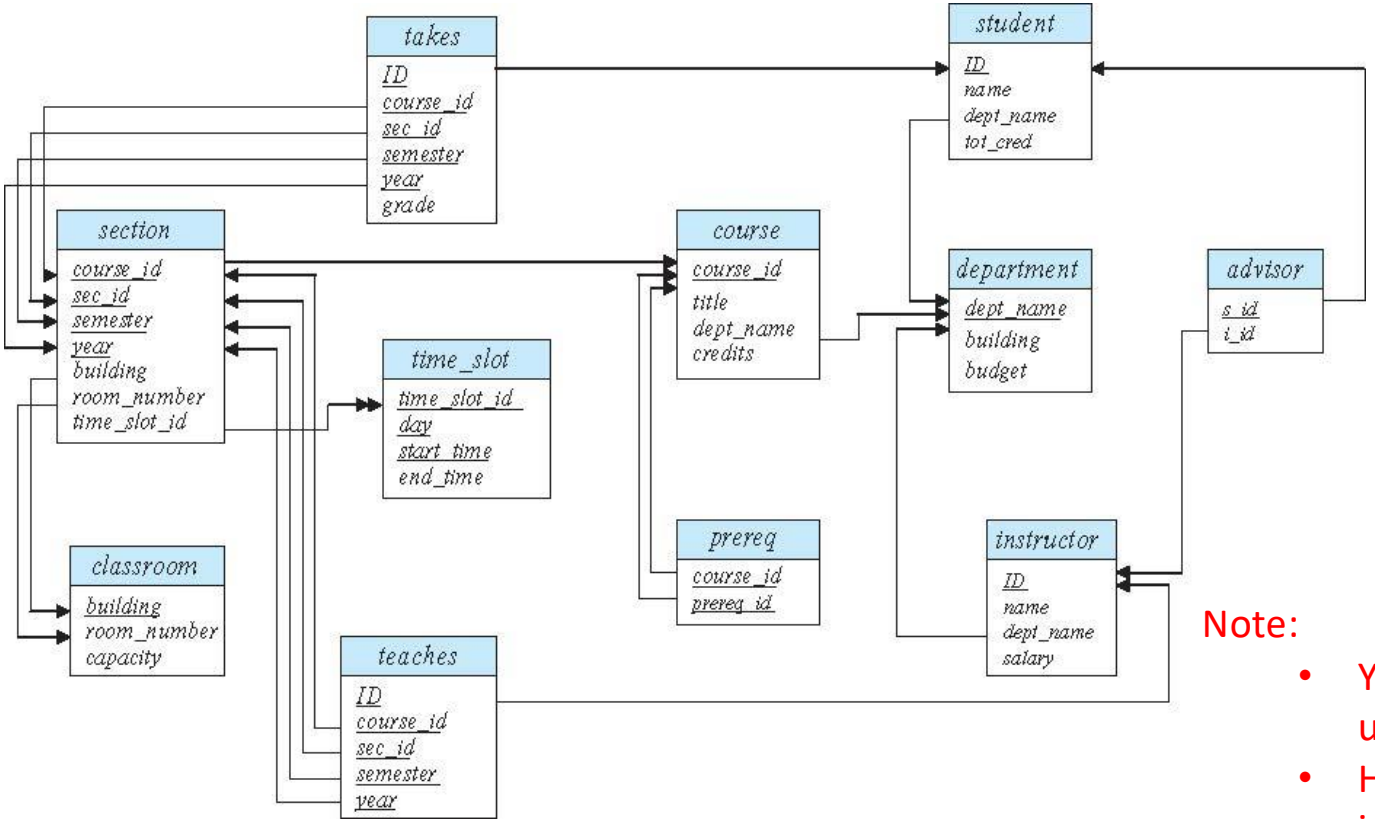
Entity-Relational Modeling Design Patterns

Relationship Sets (Cont.)

- An attribute can also be associated with a relationship set.
- For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor



Schema Diagram for University Database



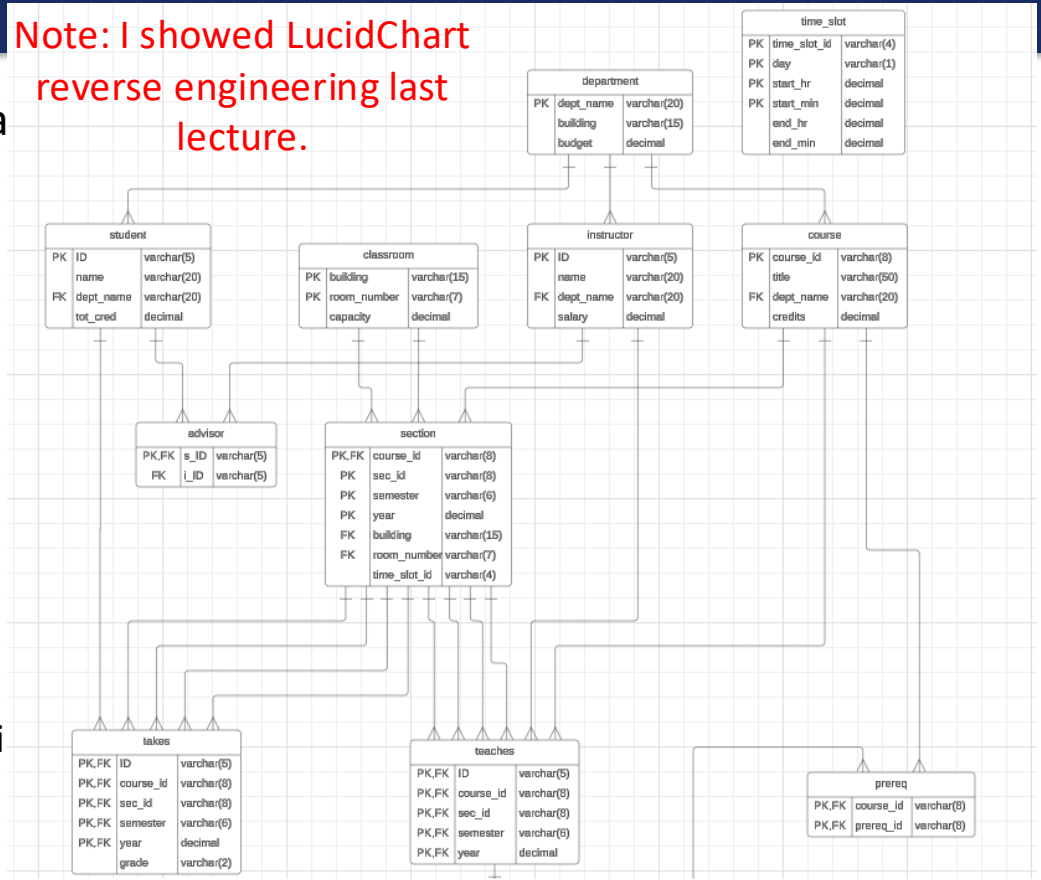
Note:

- You can see that DataGrip uses a similar notation.
- How creating a diagram in DataGrip.

Some Observations

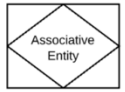
- Reverse engineering seems to have ma
- But, notice that sometimes:
  - Relationship are foreign keys, e.g. Section-Classroom
  - Are something else, e.g. Advisor
- This something else is called and Associative Entity.
- SQL requires and Associative Entity if
  - The relationship is many-to-many.
  - There are properties on the relationshi

Note: I showed LucidChart reverse engineering last lecture.



Associative Entity

- The ER model represents “associations/relationships” as
  - First class “things”
  - That are different from “entities.”
- The SQL model does not have “relationships” or associations as first-class types. You have
  - Tables
  - Columns
  - Keys
  - Constraints
  - … …
- You can implement some “relationships” using foreign keys. Others require something more complex – an *associative entity.*



Associative entity

Associative entities relate the instances of several entity types. They also contain attributes specific to the relationship between those entity instances.

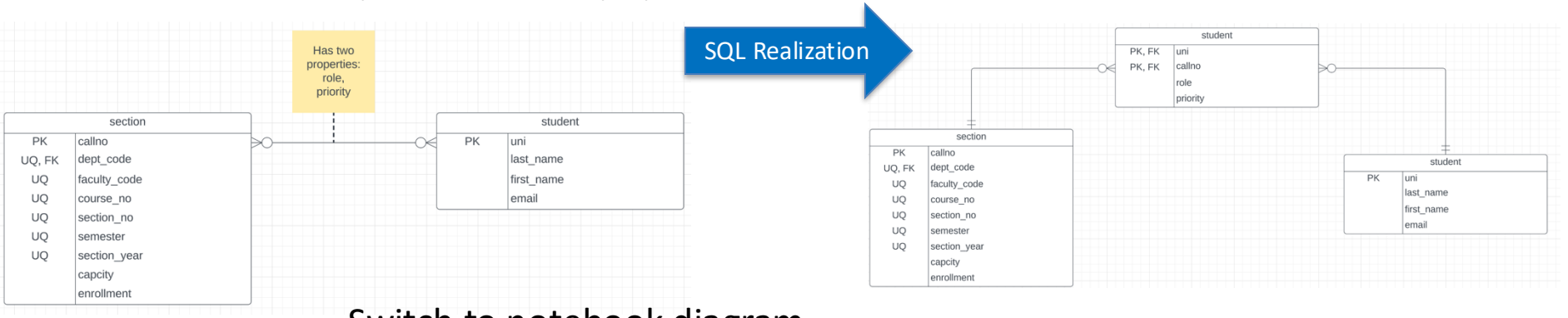
ERD relationship symbols

Within entity-relationship diagrams, relationships are used to document the interaction between two entities. Relationships are usually verbs such as assign, associate, or track and provide useful information that could not be discerned with just the entity types.

Relationship Symbol	Name	Description
	Relationship	Relationships are associations between or among entities.
	Weak relationship	Weak Relationships are connections between a weak entity and its owner.

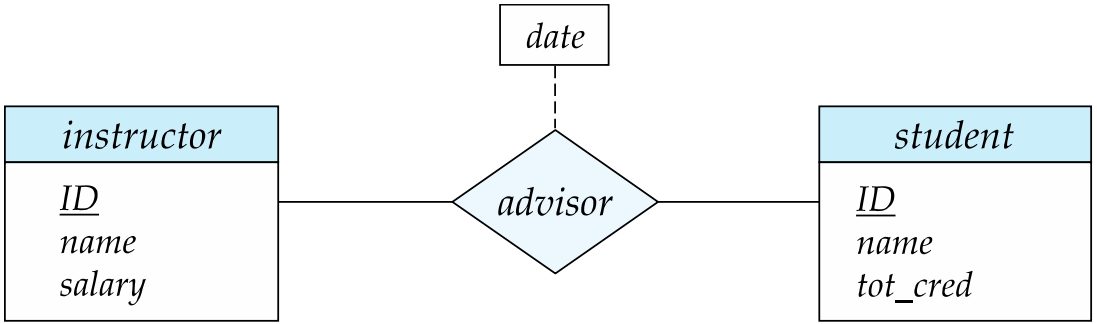
Associative Entity

- "An associative entity is a term used in relational and entity–relationship theory. A relational database requires the implementation of a base relation (or base table) to resolve many-to-many relationships. A base relation representing this kind of entity is called, informally, an associative table." (https://en.wikipedia.org/wiki/Associative_entity)
- Consider *Students – Sections:*
  - This is many-to-many. There is no way to implement in SQL. You see this in the *Advises* table in the sample database.
  - The "relationship/association" has properties that are not attributes of the connected entities.

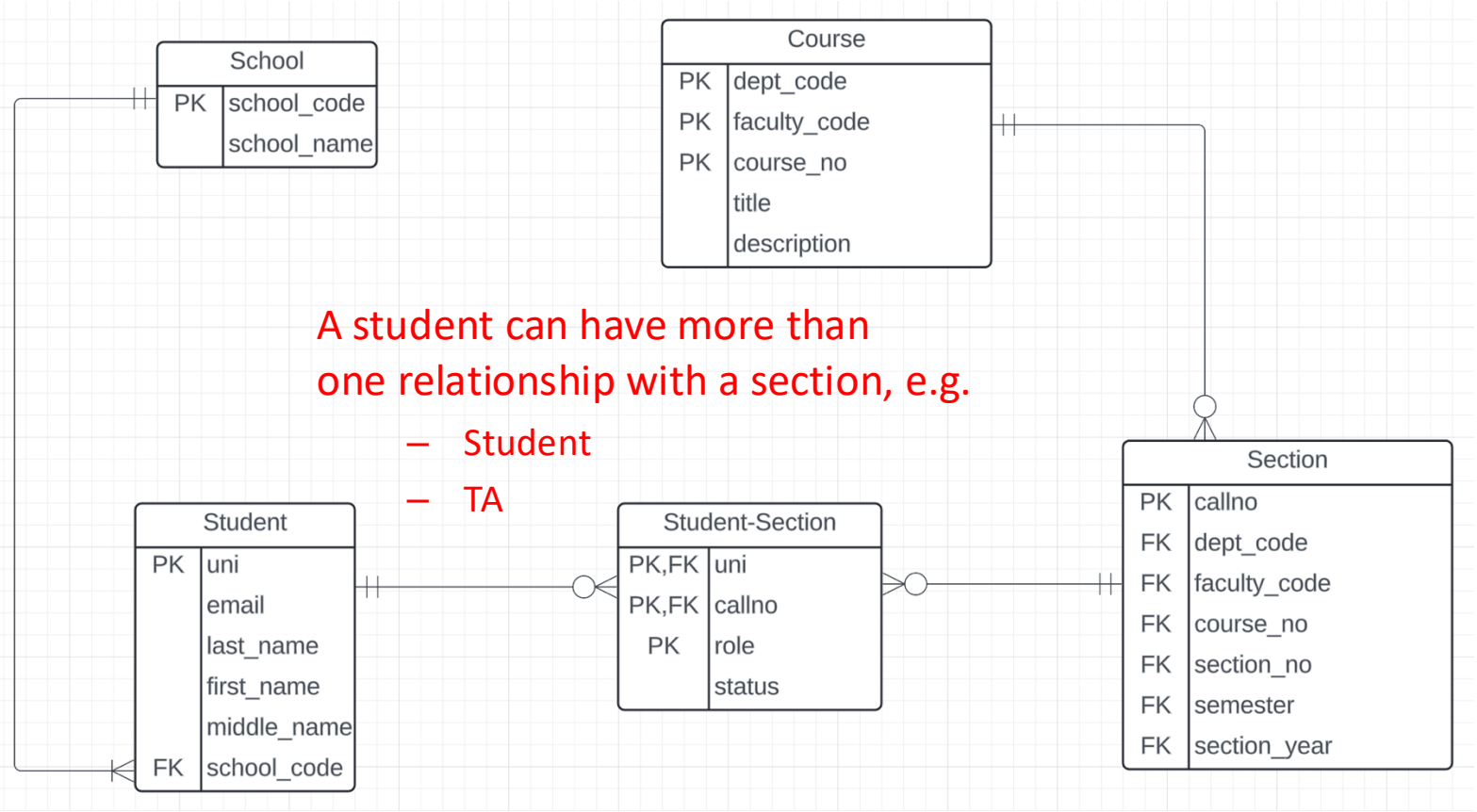


Switch to notebook diagram.

Relationship Sets with Attributes

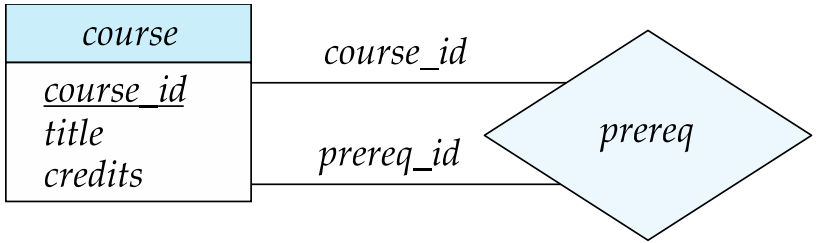


An Example from Columbia University Model



Roles

- Entity sets of a relationship need not be distinct
  - Each occurrence of an entity set plays a “role” in the relationship
- The labels “*course_id*” and “*prereq_id*” are called **roles**.



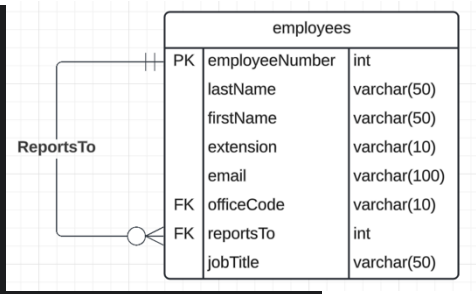
Another Example – Classic Models

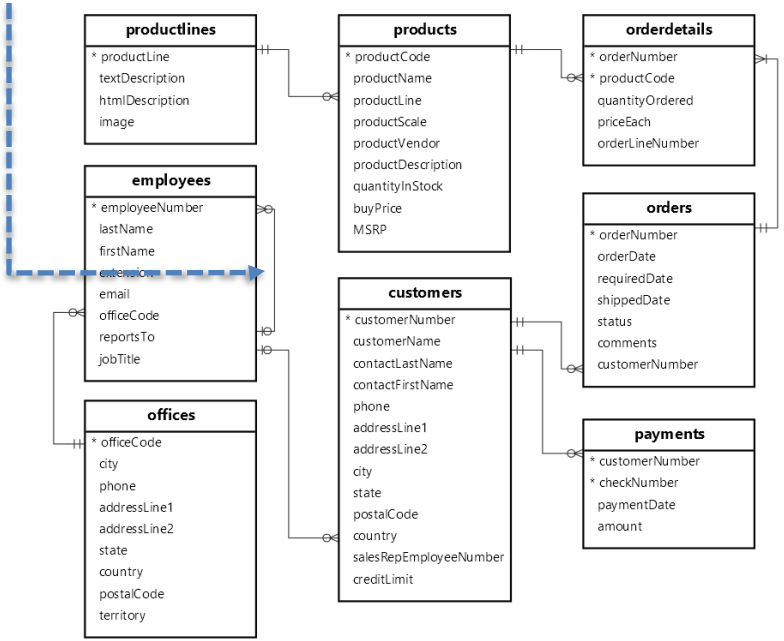
- Another simple but representative example we use is Classic Models (https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/).
- You can see an example of a non-distinct entity sets.

```
create table if not exists classicmodels.employees
(
    employeeNumber int          not null
        primary key,
    lastName       varchar(50)  not null,
    firstName      varchar(50)  not null,
    extension      varchar(10)  not null,
    email          varchar(100) not null,
    officeCode     varchar(10)  not null,
    reportsTo      int          null,
    jobTitle       varchar(50)  not null,
    constraint employees_ibfk_1
        foreign key (reportsTo) references classicmodels.employees (employeeNumber),
    constraint employees_ibfk_2
        foreign key (officeCode) references classicmodels.offices (officeCode)
);

create index officeCode
    on classicmodels.employees (officeCode);

create index reportsTo
    on classicmodels.employees (reportsTo);
```



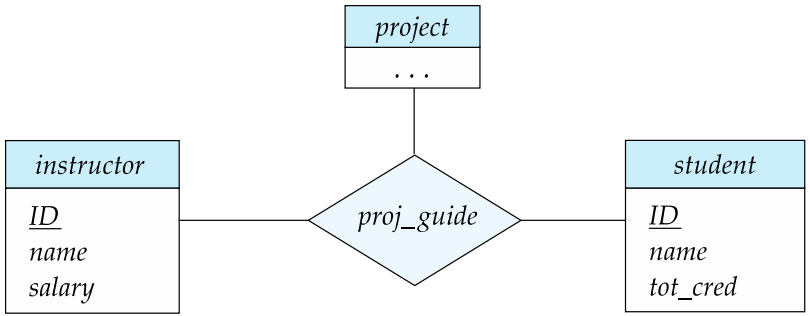


Degree of a Relationship Set

- Binary relationship
  - involve two entity sets (or degree two).
  - most relationship sets in a database system are binary.
- Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)
  - Example: *students* work on research *projects* under the guidance of an *instructor*.
  - relationship *proj_guide* is a ternary relationship between *instructor, student,* and *project*

Non-binary Relationship Sets

- Most relationship sets are binary
- There are occasions when it is more convenient to represent relationships as non-binary.
- E-R Diagram with a Ternary Relationship



Mapping Cardinality Constraints

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

Let’s Examine and Do Some Examples

Relational Model

Keys

Keys

- Let K $\subseteq$ R
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal
  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
  - which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation
  - Example: *dept_name* in i*nstructor* is a foreign key from *instructor* referencing *department*

Notation

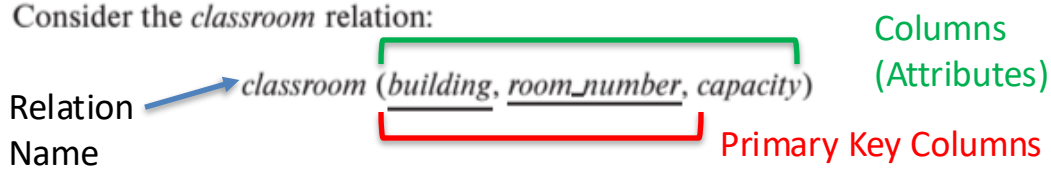
Classroom relation

building	room_number	capacity
Packard	101	500
Painter	100	125
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

classroom schema

It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept_name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined.

Consider the *classroom* relation:



- The primary key is a *composite key.* Neither column is a key (unique) by itself.
- Keys are statements about all possible, valid tuples and not just the ones in the relation.
  - Capacity is unique in this specific data, but clearly not unique for all possible data.
  - In this domain, there cannot be two classrooms with the same building and room number.
- Relation schema:
  - Underline indicates a primary key column. There is no standard way to indicate other types of key.
  - We will use **bold** to indicate foreign keys.
  - You will sometimes see things like *classroom(building:string, room_number:number, capacity:number)*

Observations

- Keys:
  - Will be baffling. It takes time and experience to understand/appreciate.
  - There are many, many types of keys with formal definitions.
  - I explain the formality but focus on the concepts and applications.

- No one uses the formal, relational model. So, why do we study it?
  - Is very helpful when understanding concepts that we cover later in the course, especially query optimization and processing.
  - There are many realizations of the model and algebra, and understanding the foundation helps with understanding language/engine capabilities.
  - The model has helped with innovating new approaches, and you may innovate in data and query models in your future.

A Note on Keys

- Relational model/algebra:
  - In the “pure” model, foreign keys reference primary keys.
  - This is not always the case in “real” databases and SQL.
  - There is no standard/common way to show foreign keys in relational schema. If I ask you to do it, you can pick a convention and add a note, e.g.

    Student(ID, dept_code, name, tot_cred)
- When doing modeling for implementations, there are many kinds/styles of key:
  - Alternate
  - Composite
  - Natural
  - Surrogate
  - Unique
  - … …

Relational Algebra (2)
More Advanced Operators

Relational Query Languages

- Procedural versus non-procedural, or declarative
- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate in this chapter on relational algebra
  - Not turning-machine equivalent
  - Consists of 6 basic operations

DFF Comments:

- You will sometimes see other operator, e.g. ← Assignment.
- Relational algebra focuses on *retrieve.* You can sort of do Create, Update, Delete.
- The SQL Language, which we will see, extends relational algebra.

Relational Algebra

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Join Operation

- The Cartesian-Product

  instructor X *teaches*

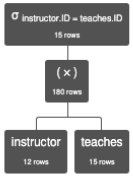
  associates every tuple of instructor with every tuple of teaches.
  - Most of the resulting rows have information about instructors who did NOT teach a particular course.
- To get only those tuples of "*instructor* X *teaches* " that pertain to instructors and the courses that they taught, we write:

  $\sigma_{instructor.id = teaches.id}$ (*instructor* x *teaches*))
  - We get only those tuples of "*instructor* X *teaches"* that pertain to instructors and the courses that they taught.
- The result of this expression, shown in the next slide

A fundamental definition:

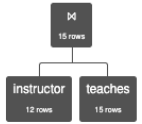
- $\sigma_{instructor.ID=teaches.ID}$ (instructor × teaches) = instructor ⋈ teaches
- ⋈ is the JOIN operations.

JOIN Definition



σ instructor.ID = teaches.ID (instructor × teaches)

instructor.ID	instructor.name	instructor.dept_name	instructor.salary	teaches.ID	teaches.course_id	teaches.sec_id	teaches.semester	teaches.year
10101	'Srinivasan'	'Comp. Sci.'	65000	10101	'CS-101'	1	'Fall'	2009
10101	'Srinivasan'	'Comp. Sci.'	65000	10101	'CS-315'	1	'Spring'	2010
10101	'Srinivasan'	'Comp. Sci.'	65000	10101	'CS-347'	1	'Fall'	2009
12121	'Wu'	'Finance'	90000	12121	'FIN-201'	1	'Spring'	2010
15151	'Mozart'	'Music'	40000	15151	'MU-199'	1	'Spring'	2010
22222	'Einstein'	'Physics'	95000	22222	'PHY-101'	1	'Fall'	2009
32343	'El Said'	'History'	60000	32343	'HIS-351'	1	'Spring'	2010
45565	'Katz'	'Comp. Sci.'	75000	45565	'CS-101'	1	'Spring'	2010
45565	'Katz'	'Comp. Sci.'	75000	45565	'CS-319'	1	'Spring'	2010
76766	'Crick'	'Biology'	72000	76766	'BIO-101'	1	'Summer'	2009



instructor ⋈ teaches

instructor.ID	instructor.name	instructor.dept_name	instructor.salary	teaches.course_id	teaches.sec_id	teaches.semester	teaches.year
10101	'Srinivasan'	'Comp. Sci.'	65000	'CS-101'	1	'Fall'	2009
10101	'Srinivasan'	'Comp. Sci.'	65000	'CS-315'	1	'Spring'	2010
10101	'Srinivasan'	'Comp. Sci.'	65000	'CS-347'	1	'Fall'	2009
12121	'Wu'	'Finance'	90000	'FIN-201'	1	'Spring'	2010
15151	'Mozart'	'Music'	40000	'MU-199'	1	'Spring'	2010
22222	'Einstein'	'Physics'	95000	'PHY-101'	1	'Fall'	2009
32343	'El Said'	'History'	60000	'HIS-351'	1	'Spring'	2010
45565	'Katz'	'Comp. Sci.'	75000	'CS-101'	1	'Spring'	2010
45565	'Katz'	'Comp. Sci.'	75000	'CS-319'	1	'Spring'	2010
76766	'Crick'	'Biology'	72000	'BIO-101'	1	'Summer'	2009

$\sigma_{instructor.ID=teaches.ID}$ (instructor × teaches)

instructor $\bowtie$ teaches

instructor $\bowtie_{instructor.ID>teaches.ID}$ teaches

Join Operation (Cont.)

- The **join** operation allows us to combine a select operation and a Cartesian-Product operation into a single operation.
- Consider relations *r* (*R*) and *s* (*S*)
- Let "theta" be a predicate on attributes in the schema R "union" S. The join operation r $\bowtie_\theta$ s is defined as follows:

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

- Thus

$$\sigma_{instructor.id = teaches.id} (instructor \text{ x } teaches))$$

- Can equivalently be written as

$$instructor \bowtie_{Instructor.id = teaches.id} teaches.$$

The Dreaded Relax Calculator

- Let's look at an online tool that you will use.
- RelaX (https://dbis-uibk.github.io/relax/calc/local/uibk/local/0)
- The calculator:
  - Has an older version of the data from the recommended textbook. (https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0)
  - You can also upload new data.
- Some queries:
  - σ dept_name='Comp. Sci.' ∨ dept_name='History' (department)
  - π name, dept_name (instructor)
  - π ID, name (
    σ dept_name='Comp. Sci.' (instructor)
)

Some Terms

- "A NATURAL JOIN is **a JOIN operation that creates an implicit join clause for you based on the common columns in the two tables being joined**. Common columns are columns that have the same name in both tables." (https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljnaturaljoin.html)

- ⋈ ➔ Natural Join in relational algebra.

- *instructor* $\bowtie_{Instructor.id = teaches.id}$ *teaches* is called a *theta join.*

- So, think about it …
  - I showed you how to produce all possible pairs.
  - I showed you how to produce all naturally matching pairs.

What are all those other Symbols? (Will Cover)

- τ order by
- γ group by
- $\neg$ negation
- $\div$ set division
- $\bowtie$ natural join, theta-join
- $⟕$ left outer join
- $⟖$ right outer join
- $⟗$ full outer join
- $\ltimes$ left semi join
- $\rtimes$ right semi join
- $\triangleright$ anti-join

- Some of these are pretty obscure
  - Division
  - Anti-Join
  - Left semi-join
  - Right semi-join
- Most SQL engines do not support them.
  - You can implement them using combinations of JOIN, SELECT, WHERE, … …
  - But, I cannot every remember using them in applications I have developed.
- Outer JOIN is very useful, but less common. We will cover.
- There are also some “patterns” or “terms”
  - Equijoin
  - Non-equi join
  - Natural join
  - Theta join
  - … …
- I may ask you to define these terms on some exams because they may be common internships/job interview questions.

Let’s Do Some Examples

SQL

Data Definition Language

Data Definition Language

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **numeric**(3,1), allows 44.5 to be stores exactly, but not 444.5 or 0.32)
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- More are covered in Chapter 4.

MySQL Data Types (Subset)

MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)

Note: Scroll through https://www.mysqldatatypes.com/

Create Table Construct

- An SQL relation is defined using the **create table** command:

  create table *r*

  (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the data type of values in the domain of attribute A_i

- Example:

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20)**,**
  dept_name **varchar**(20),
  salary **numeric**(8,2))

Integrity Constraints in Create Table

- Types of integrity constraints
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references** *r*
  - **not null**
- SQL prevents any update to the database that violates an integrity constraint.
- Example:

create table *instructor* (
ID **char**(5),
name **varchar**(20) **not null,**
dept_name **varchar**(20),
salary **numeric**(8,2),
primary key (*ID*),
foreign key *(dept_name*) **references** *department);*

And a Few More Relation Definitions

- **create table** *student* (
  ID **varchar**(5),
  name **varchar**(20) not null,
  dept_name **varchar**(20),
  tot_cred **numeric**(3,0),
  primary key *(ID),*
  foreign key *(dept_name*) **references** *department*);

- **create table** *takes* (
  ID **varchar**(5),
  course_id **varchar**(8),
  sec_id **varchar**(8),
  semester **varchar**(6),
  year **numeric**(4,0),
  grade **varchar**(2),
  primary key *(ID, course_id, sec_id, semester, year)* ,
  foreign key (*ID*) **references** *student,*
  foreign key (*course_id, sec_id, semester, year*) **references** *section*);

And more still

- **create table** *course* (
  course_id **varchar**(8),
  title **varchar(**50),
  dept_name **varchar**(20),
  credits **numeric**(2,0),
  primary key *(course_id),*
  foreign key (*dept_name*) **references** *department*);

Integrity Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  - A checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $4.00 an hour
  - A customer must have a (non-null) phone number

Constraints on a Single Relation

- **not null**
- **primary key**
- **unique**
- **check** (P), where P is a predicate

Not Null Constraints

- **not null**
  - Declare *name* and *budget* to be **not null**

    name **varchar**(20) **not null**
    budget **numeric**(12,2) **not null**

Unique Constraints

- **unique** (A_1, A_2, …, A_m)
  - The unique specification states that the attributes A_1, A_2, …, A_m form a candidate key.
  - Candidate keys are permitted to be null (in contrast to primary keys).

The check clause

- The **check** (P) clause specifies a predicate P that must be satisfied by every tuple in a relation.
- Example: ensure that semester is one of fall, winter, spring or summer

```
create table section
    (course_id varchar (8),
     sec_id varchar (8),
     semester varchar (6),
     year numeric (4,0),
     building varchar (15),
     room_number varchar (7),
     time slot id varchar (4),
     primary key (course_id, sec_id, semester, year),
     check (semester in ('Fall', 'Winter', 'Spring', 'Summer')))
```

Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If “Biology” is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for “Biology”.
- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a **foreign key** of R if for any values of A appearing in R these values also appear in S.

Referential Integrity (Cont.)

- Foreign *keys can be* specified as part of the SQL **create table** statement

  foreign key (*dept_name*) **references** *department*

- By default, a foreign key references the primary-key attributes of the referenced table.
- SQL allows a list of attributes of the referenced relation to be specified explicitly.

  foreign key (*dept_name*) **references** *department* (*dept_name*)

Cascading Actions in Referential Integrity

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.
- An alternative, in case of delete or update is to cascade

```
create table course (
        (…
         dept_name varchar(20),
         foreign key (dept_name) references department
              on delete cascade
              on update cascade,
          . . .)
```

- Instead of cascade we can use :
  - **set null**,
  - **set default**

Integrity Constraint Violation During Transactions

- Consider:

  create table *person* (
  ID **char**(10),
  name **char**(40),
  mother **char**(10),
  father **char**(10),
  primary key *ID,*
  foreign key *father* **references** *person,*
  foreign key *mother* **references** *person*)

- How to insert a tuple without causing constraint violation?
  - Insert father and mother of a person before inserting person
  - OR, set father and mother to null initially, update after inserting all persons (not possible if father and mother attributes declared to be **not null**)
  - OR defer constraint checking

Let’s Do Some Examples

- Generated values for section ID.
- Enumerations for faculty code and section.
- Check constrains.
- … …

JOIN Operator

Joined Relations

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause
- Three types of joins:
  - Natural join
  - Inner join
  - Outer join

Notes:

- You will also hear terms like equi-join, non-equi-join, theta join, semi-join, … …
- I ask for definitions on exams, but you can just look them up.

Natural Join in SQL

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column.
- List the names of instructors along with the course ID of the courses that they taught
  - **select** *name*, *course_id*
    from *students, takes,*
    where *student.ID = takes.ID*;
- Same query in SQL with “natural join” construct
  - **select** *name*, *course_id*
    from *student* **natural join** *takes*;

Natural Join in SQL (Cont.)

- The **from** clause can have multiple relations combined using natural join:

  select $A_1, A_2, \ldots A_n$
  from r_1 **natural join** r_2 **natural join .. natural join** r_n
  where P ;

Student Relation

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Takes Relation

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*null*

student **natural join** *takes*

ID	*name*	*dept_name*	*tot_cred*	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	*null*

Dangerous in Natural Join

- Beware of unrelated attributes with same name which get equated incorrectly
- Example -- List the names of students instructors along with the titles of courses that they have taken
  - Correct version

    select *name*, *title*
    from *student* **natural join** *takes*, *course*
    where *takes.course_id* = *course.course_id*;

  - Incorrect version

    select *name*, *title*
    from *student* **natural join** *takes* **natural join** *course*;

    - This query omits all (student name, course title) pairs where the student takes a course in a department other than the student's own department.
    - The correct version (above), correctly outputs such pairs.

Natural Join with Using Clause

- To avoid the danger of equating attributes erroneously, we can use the "**using**" construct that allows us to specify exactly which columns should be equated.
- Query example

  select *name*, *title*
  from (*student* **natural join** *takes*) **join** *course* **using** (*course_id*)

Join Condition

- The **on** condition allows a general predicate over the relations being joined
- This predicate is written like a **where** clause predicate except for the use of the keyword **on**
- Query example

  select *
  from *student* **join** *takes* **on** *student_ID = takes_ID*

  - The **on** condition above specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal.
- Equivalent to:

  select *
  from *student , takes*
  where *student_ID = takes_ID*

Join Condition (Cont.)

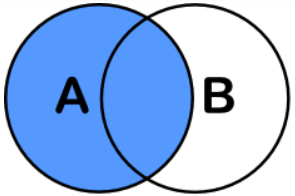
- The **on** condition allows a general predicate over the relations being joined.
- This predicate is written like a **where** clause predicate except for the use of the keyword **on**.
- Query example

  select *
  from *student* **join** *takes* **on** *student_ID = takes_ID*

  - The **on** condition above specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal.
- Equivalent to:

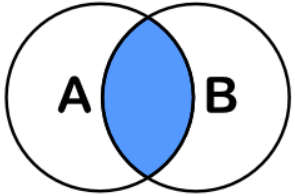
  select *
  from *student , takes*
  where *student_ID = takes_ID*

One Way to Think About Joins

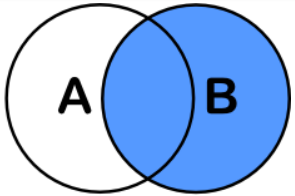



```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
```

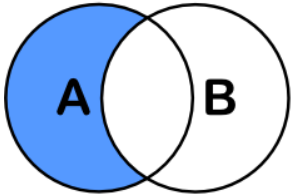
CHEATSHEET
SQL
JOINS



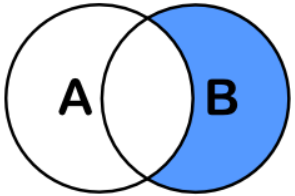

```
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
```



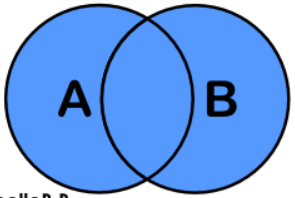

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
```



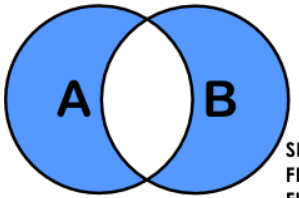

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

Database Modification

Modification of the Database

- Deletion of tuples from a given relation.
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation

Deletion

- Delete all instructors

  delete from *instructor*

- Delete all instructors from the Finance department

  delete from *instructor*
  where *dept_name*= 'Finance';

- *Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building.*

  delete from *instructor*
  where *dept name* **in** (**select** *dept name*
  from *department*
  where *building* = 'Watson');

Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

  delete from *instructor*
  where *salary* < (**select avg** (*salary*)
  from *instructor*);

  - Problem: as we delete tuples from *instructor*, the average salary changes
  - Solution used in SQL:
    1. First, compute **avg** (salary) and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

Insertion

- Add a new tuple to *course*

  insert into *course*
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- or equivalently

  insert into *course* (*course_id*, *title*, *dept_name*, *credits*)
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- Add a new tuple to *student* with *tot_creds* set to null

  insert into *student*
  values ('3003', 'Green', 'Finance', *null*);

Insertion (Cont.)

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.

```
insert into instructor
    select ID, name, dept_name, 18000
    from   student
    where  dept_name = 'Music' and total_cred > 144;
```

- The **select from where** statement is evaluated fully before any of its results are inserted into the relation.

Otherwise queries like

```
insert into table1 select * from table1
```

would cause problem

Updates

- Give a 5% salary raise to all instructors

```
update instructor
  set salary = salary * 1.05
```

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor
  set salary = salary * 1.05
 where salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
update instructor
set salary = salary * 1.05
where salary <  (select avg (salary)
                 from instructor);
```

Updates (Cont.)

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

```
update instructor
   set salary = salary * 1.03
   where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - The order is important
  - Can be done better using the **case** statement (next slide)

Wrap Up