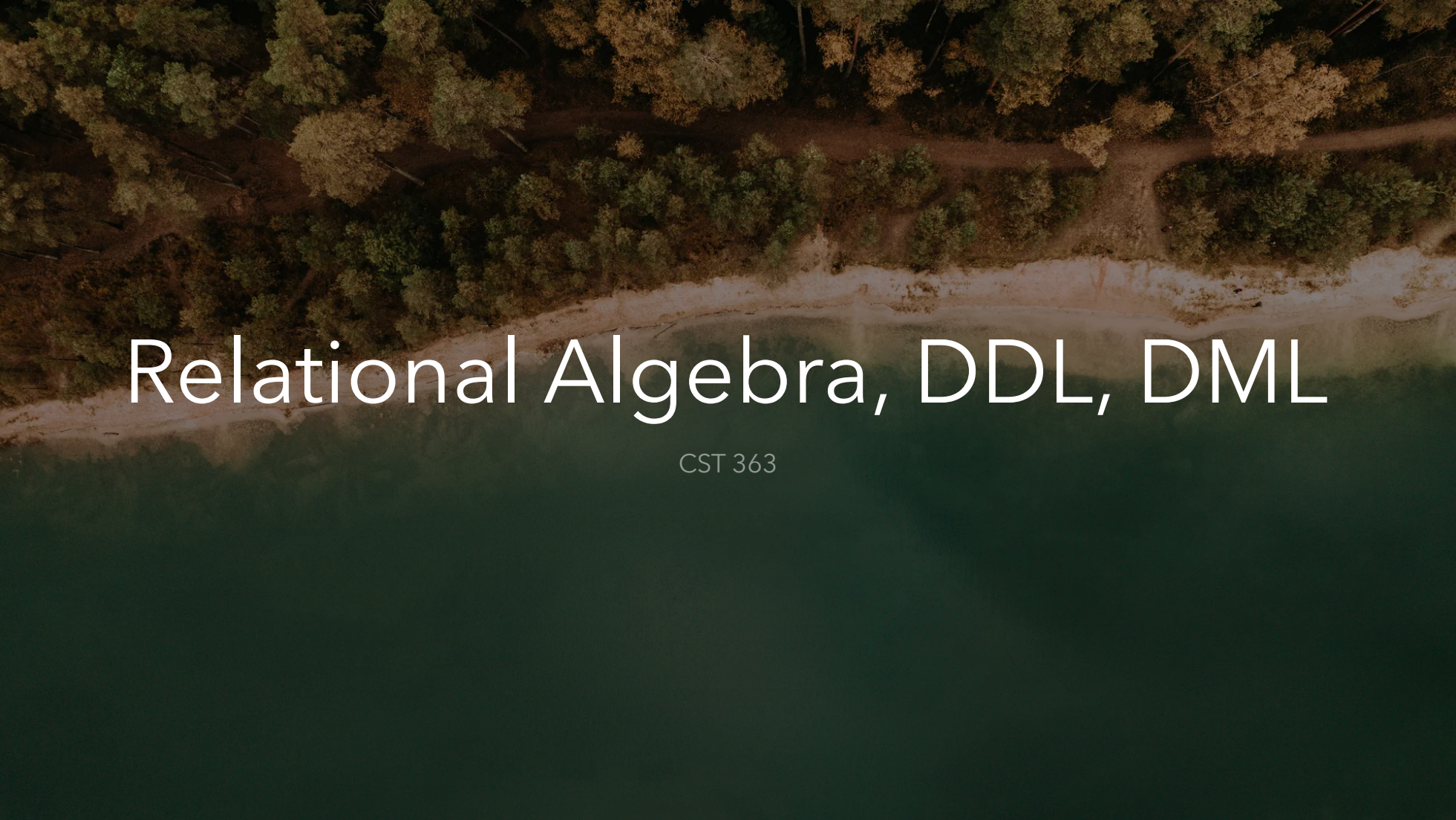


# Relational Algebra, DDL, DML

CST 363

# Database schema

- A **schema** is the design or structure of a specific database.
- An **instance** is a schema "instantiated" with data

Schema:

```
table roster {
   class: string,
   student: string
}
```

Instance:

| class | student |
|---|---|
| class_101 | student_1 |
| class_101 | student_2 |
| class_205 | student_3 |
| class_205 | student_4 |
| class_205 | student_5 |

# Database queries

- A query is a statement requesting the retrieval of information from a database.

Example SQL queries:

```sql
SELECT *
FROM customers
WHERE country = 'Guam';
```

```sql
SELECT *
FROM customers
WHERE city = 'Berlin' OR city = 'Munich';
```

# Transactions

- A transaction is the *unit of change* in a database.
- Transactions contain a set of database operations.
- A transaction must suceed, or fail "completely".
- For example, a transaction might be used for moving a student from one class to another.

*more on transactions later in semester

# A Relational DB is a group of tables

- A column is also called an **attribute**
- Each attribute has a **domain**, which is a set of values that are allowed in the column.
- A row is also called a **tuple**.

instructor

| instructor_id | instructor_name | dept_name | salary |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | null |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | null | 62000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

# Tuples

Another table from the `courses` DB

- A tuple represents a relationship between the values of the tuple.
- A table represents a mathematical relation.
  - So a table is also called a "relation". This is why we say "relational database".

course

| course_id | title | dept_name | credits |
|---|---|---|---|
| BIO-101 | Intro. to Biology | Biology | 4 |
| BIO-301 | Genetics | Biology | 4 |
| BIO-399 | Computational Biology | Biology | 3 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |
| CS-319 | Image Processing | Comp. Sci. | 3 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 |
| FIN-201 | Investment Banking | Finance | 3 |
| HIS-351 | World History | History | 3 |
| MU-199 | Music Video Production | Music | 3 |
| PHY-101 | Physical Principals | Physics | 4 |

# Null values

- Sometimes we don't know a value, or the value doesn't exist.
- In relational DBs we deal with this by using special `null` values.

instructor

| instructor_id | instructor_name | dept_name | salary |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | null |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | null | 62000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

# Relation schema and instances

- **relation schema:** Gives the names and domains of the attributes (domains not shown here.)
  - `student(student_id, student_name, dept_name, tot_cred)`
- **relation instance:** an instance of a *relation schema* (also called a "table")

student

| student_id | student_name | dept_name | tot_cred |
|---|---|---|---|
| 128 | 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 |

# Database schema and instances

**database schema:** the relation schemas for all the relations in a database

```
student(student_id, student_name, dept_name, tot_cred)
department(dept_name, building, budget)
...
```

# Database schema and instances (cont.)

**database instance:** all the tables of the database

student

| student_id | student_name | dept_name | tot_cred |
|---|---|---|---|
| 128 | 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 |

department

| dept_name | building | budget |
|---|---|---|
| Biology | Watson | 90000 |
| Comp. Sci. | Taylor | 100000 |
| Elec. Engr. | Taylor | 85000 |
| Finance | Painter | 120000 |
| History | Painter | 50000 |
| Music | Packard | 80000 |
| Physics | Watson | 70000 |

# All the names

In databases there seem to be two or three ways to say everything.

- **schema** = relation schema
- **table** = relation = relation instance = instance
- tuple = **row**
- **attribute** = column

We'll normally use the terms in bold

# "Functions" on tables

- In programming we define objects, and methods on those objects.
- For example, for strings we have operations like
  - `substring(string, first, last)`
  - `concat(string1, string2)`
- In relational databases the objects are tables, and we define functions (operations) on tables.

# "Selection" operator: Pick out rows of a table

patient

| Patient No. | Last name | First name | Sex |
|---|---|---|---|
| 454 | Smith | John | M |
| 223 | Jones | Peter | M |
| 597 | Brown | Brenda | F |
| 234 | Jenkins | Alan | M |
| 244 | Wells | Christy | F |

Sex = F

| Patient No. | Last name | First name | Sex |
|---|---|---|---|
| 597 | Brown | Brenda | F |
| 244 | Wells | Christy | F |

# “Projection” operator: Pick out columns of a table

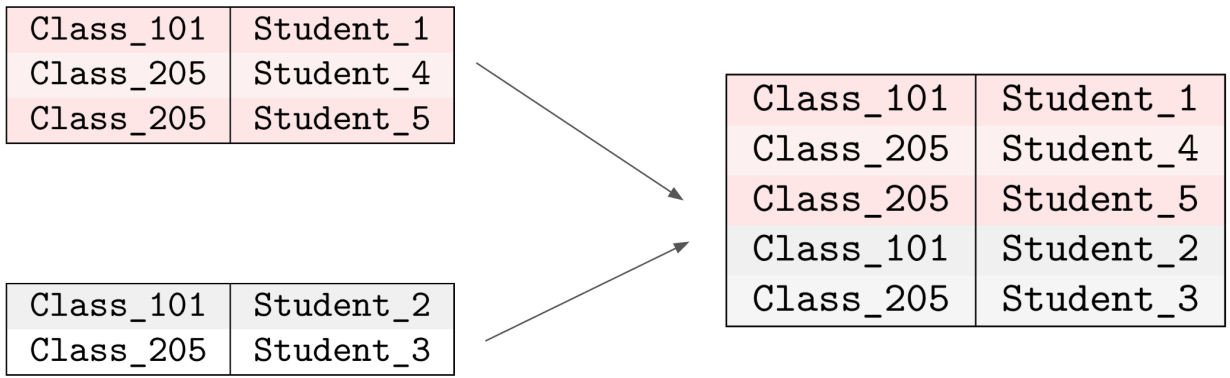
The operation takes a table as input and produces a table as output

| Patient No. | Last name | First name | Sex |
|---|---|---|---|
| 454 | Smith | John | M |
| 223 | Jones | Peter | M |
| 597 | Brown | Brenda | F |
| 234 | Jenkins | Alan | M |
| 244 | Wells | Christy | F |

"Patient No.", "First name"

| Patient No. | First name |
|---|---|
| 454 | John |
| 223 | Peter |
| 597 | Brenda |
| 234 | Alan |
| 244 | Christy |

# "Union" operator: Add the rows of two tables



# Question about "projection" operator

| Patient No. | Last name | First name | Ward No. |
|---|---|---|---|
| 454 | Smith | John | 6 |
| 223 | Jones | Peter | 8 |
| 597 | Brown | Brenda | 3 |
| 234 | Jenkins | Alan | 7 |
| 244 | Brown | Chris | 6 |

What happens if we perform a `projection` on this table using attribute 'Last name'?

| Last name |
|---|
| Smith |
| Jones |
| Brown |
| Jenkins |
| Brown |

**Is this a problem?**

# Removing duplicate rows

- As part of applying an operation of relation algebra, duplicate rows should be removed.
- Duplicate rows aren't technically allowed in a valid table.
- In practice a DB system may allow duplicate rows.

# Defining the operations

`selection(table, condition)`

- return the rows of table that meet the condition
- example: `t = selection(student, tot_cred > 100)`

`projection(table, attributes)`

- returns the specified columns of table
- example: `t = projection(student, ["student_name", "student_id"])`

`union(table1, table2)`

- returns the table containing the rows of table1 and table2
- example: `t = union(student1, student2)`

# Answering questions with the functions

What are the names of students with more than 100 credits?

- To get the answer:
  - select the rows of the `student` table with `tot_cred > 100`
  - get the `name` attribute

```
projection(selection(student, tot_cred > 100), ["student_name"])
```

# Another example

What courses are offered in Spring '09, and what buildings are they in?

| course_id | sec_id | semester | year | building | room_number | time_slot_id |
|---|---|---|---|---|---|---|
| BIO-101 | 1 | Summer | 2009 | Painter | 514 | B |
| BIO-301 | 1 | Summer | 2010 | Painter | 514 | A |
| CS-101 | 1 | Fall | 2009 | Packard | 101 | H |
| CS-101 | 1 | Spring | 2010 | Packard | 101 | F |
| CS-190 | 1 | Spring | 2009 | Taylor | 3128 | E |
| CS-190 | 2 | Spring | 2009 | Taylor | 3128 | A |
| CS-315 | 1 | Spring | 2010 | Watson | 120 | D |
| CS-319 | 1 | Spring | 2010 | Watson | 100 | B |

```
t = selection(section, semester="Spring" and year=2009)
projection(t, ["course_id", "building"])
```

# Exercise

Write the operations to get the IDs of students who took course CS-101 in Fall 2009.

| ID | course_id | sec_id | semester | year | grade |
|---|---|---|---|---|---|
| 128 | CS-101 | 1 | Fall | 2009 | A |
| 128 | CS-347 | 1 | Fall | 2009 | A- |
| 12345 | CS-101 | 1 | Fall | 2009 | C |
| 12345 | CS-190 | 2 | Spring | 2009 | A |
| 12345 | CS-315 | 1 | Spring | 2010 | A |
| 12345 | CS-347 | 1 | Fall | 2009 | A |

```
projection(selection(takes, course_id="CS-101"
  and semester="Fall" and year=2009), ["ID"])
```

# A 'union' example

Get the ID of courses taught in Fall 2009 or Spring 2010

section

| course_id | sec_id | semester | year | building | room_number | time_slot_id |
|---|---|---|---|---|---|---|
| BIO-101 | 1 | Summer | 2009 | Painter | 514 | B |
| BIO-301 | 1 | Summer | 2010 | Painter | 514 | A |
| CS-101 | 1 | Fall | 2009 | Packard | 101 | H |
| CS-101 | 1 | Spring | 2010 | Packard | 101 | F |
| CS-190 | 1 | Spring | 2009 | Taylor | 3128 | E |

```
t1 = selection(section, semester="Fall" and year=2009)
t2 = selection(section, semester="Spring" and year=2010)
projection(union(t1,t2), ["course_id"])
```

Last line could alternatively be written:

```
union(projection(t1,["course_id"]), projection(t2,["course_id"]))
```

# Cartesian Product

| id | name |
|---|---|
| 1 | Cesar |
| 2 | Rivka |
| 3 | Arturo |

| course | id | time |
|---|---|---|
| DB | 1 | 10 |
| OS | 2 | 4 |

| id | name | course | id | time |
|---|---|---|---|---|
| 1 | Cesar | DB | 1 | 10 |
| 1 | Cesar | OS | 2 | 4 |
| 2 | Rivka | DB | 1 | 10 |
| 2 | Rivka | OS | 2 | 4 |
| 3 | Arturo | DB | 1 | 10 |
| 3 | Arturo | OS | 2 | 4 |

Question:
If you take the product of a table with 5 rows and a table with 8 rows, how many rows do you get?

# Using cross product

What are the names of CS instructors and what classes do they teach?

| instructor_id | instructor_name | dept_name | salary |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | null |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | null | 62000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

| ID | course_id | sec_id | semester | year |
|---|---|---|---|---|
| 10101 | CS-101 | 1 | Fall | 2009 |
| 10101 | CS-315 | 1 | Spring | 2010 |
| 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | FIN-101 | 1 | Spring | 2010 |
| 15151 | MUS-199 | 1 | Spring | 2010 |
| 22222 | PHY-101 | 1 | Fall | 2009 |

```
t1 = cross(instructor, teaches)
t2 = selection(t1, instructor.ID = teaches.ID and dept_name = "Comp. Sci.")
projection(t2, ["name", "course_id"])
```

# Saying it in plain Greek

To make it look more like math, you can write the operations using Greek letters and symbols

- `selection(section, year=2015)`
  - $\sigma_{\text{year}=2015}(\text{section})$
  - sigma ($\sigma$) – "selection" starts with "s"

- `projection(section, ["course_id"])`
  - $\pi_{\text{course\_id}}(\text{section})$
  - pi ($\pi$) – "projection" starts with "p"

- $\pi_{\text{course\_id}}(\sigma_{\text{year}=2015(\text{section})})$
  - First, filters the `section` table to only the rows where year = 2015.
  - Then, from those remaining rows, keep only the `course_id` column.

# Part 2: SQL (DDL/DML)

# What is SQL?

- SQL = "Structured Query Language"
- Originally developed at IBM in the early 70's
- Pronounced "sequel" or S-Q-L
- A special-purpose language that lets you express questions about your relational data in a precise way

SQL serves as the interface between the database and users/applications

# Parts of SQL

SQL is not just for writing queries:

- Data-definition language (DDL)
  - create/delete/modify schemas
  - define integrity constraints
  - define views
  - drop tables
- Data-manipulation language (DML)
  - define queries
  - modify tables (insert/delete/modify tuples)
- TCL (Transaction Control Language): `BEGIN`, `COMMIT`, `ROLLBACK`
- DCL (Data Control Language): `GRANT`, `REVOKE`

# Defining a relation schema

*Question*: What information do you provide when you define a relation schema?

- name of the table
- names and domains of each attribute
- primary key
- optionally:
  - whether an attribute is allowed to have a null value
  - foreign keys

# SQL for defining a relation schema

Relation schema: `department(dept_name, building, budget)`

DDL table definition:

```
CREATE TABLE department (
   dept_name VARCHAR(20) PRIMARY KEY,
   building VARCHAR(15),
   budget NUMERIC(12, 2)
);
```

- For each attribute we give the attribute name and type.
- We also give the primary key.

# Another 'create table' example

```
CREATE TABLE student (
   student_id VARCHAR(20) PRIMARY KEY,
   name VARCHAR(20) NOT NULL,
   dept_name VARCHAR(20),
   tot_cred NUMERIC(3, 0),
   FOREIGN KEY (dept_name) REFERENCES department(dept_name)
);
```

- The foreign key constraint says:
  - every `dept_name` value in the student table must equal the key of some row in the department table.
- The `NOT NULL` constraint says:
  - no name value in the student table can be `NULL`
- PostgreSQL requires referenced tables exist before creating FKs — `department` needs to be created first

# SQL Constraints Review

- `PRIMARY KEY` = `UNIQUE` + `NOT NULL`
- `FOREIGN KEY` enforces referential integrity
- `NOT NULL` disallows missing values

# Attribute types

Recall that attributes can only be simple values, like integers, and strings.*

Here are some of the main types allowed by SQL:

- `VARCHAR(n)` : a variable-length string of length at most `n`
- `INT` : an integer — `integer` is also okay
- `NUMERIC(p, d)` : a fixed-point number of `p` total digits, with `d` digits to right of the decimal point.
  - Example: 123.45 is `numeric(5,2)`
- `FLOAT(p)` : a floating-point number with either 4 byte or 8 byte precision
  - alternatively can use `REAL` (4 bytes) and `DOUBLE PRECISION` (8 bytes)

*In the relational model, attributes are atomic (simple) values.

PostgreSQL also supports richer types (e.g., arrays/JSON), but we'll focus on relational design first.

# Yet another example

```
CREATE TABLE takes (
   student_id VARCHAR(5),
   course_id VARCHAR(8),
   section_id VARCHAR(8),
   semester VARCHAR(6),
   section_year NUMERIC(4, 0),
   grade VARCHAR(2),
   PRIMARY KEY (student_id, course_id, section_id, semester, section_year),
   FOREIGN KEY (course_id, section_id, semester, section_year)
       REFERENCES section(course_id, section_id, semester, section_year) ON DELETE CASCADE,
   FOREIGN KEY (student_id)
       REFERENCES student(student_id) ON DELETE CASCADE
);
```

- primary key has multiple attributes
- two foreign keys

# Cascaded deletes

| ID | name | dept_name | salary |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |

| ID | course_id | sec_id | semester | year |
|---|---|---|---|---|
| 10101 | CS-101 | 1 | Fall | 2009 |
| 10101 | CS-315 | 1 | Spring | 2010 |
| 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | FIN-101 | 1 | Spring | 2010 |
| 15151 | MUS-199 | 1 | Spring | 2010 |
| 22222 | PHY-101 | 1 | Fall | 2009 |

- `ID` in the **teaches** table is a foreign key that "points to" the `ID` field of the **instructor** table.
- What to do when the first row is deleted from the instructor table?
  - Default: delete is rejected if referenced.
    - `CASCADE` : referencing rows are deleted automatically.
      - `CASCADE` is powerful; use intentionally.

# Inserting rows into a table

| patient_no | last_name | first_name | sex | date_of_birth | ward |
|---|---|---|---|---|---|
| 454 | Smith | John | M | 14.08.78 | 6 |
| 223 | Jones | Peter | M | 07.12.85 | 8 |
| 597 | Brown | Brenda | F | 17.06.61 | 3 |
| 234 | Jenkins | Alan | M | 29.01.72 | 7 |
| 244 | Wells | Chris | F | 25.02.95 | 6 |

```
CREATE TABLE patient (
   patient_no INTEGER PRIMARY KEY,
   last_name VARCHAR(64) NOT NULL,
   first_name VARCHAR(64) NOT NULL,
   sex VARCHAR(1),
   date_of_birth DATE,
   ward INTEGER
);
```

# Inserting rows into a table (cont.)

SQL also has statements to delete and modify rows, as well as batch operations.

```
CREATE TABLE patient (
   patient_no INTEGER PRIMARY KEY,
   last_name VARCHAR(64) NOT NULL,
   first_name VARCHAR(64) NOT NULL,
   sex VARCHAR(1),
   date_of_birth DATE,
   ward INTEGER
);
```

```
INSERT INTO patient VALUES (454, 'Smith', 'John', 'M', '1978-08-14', 6);
INSERT INTO patient VALUES (223, 'Jones', 'Peter', 'M', '1985-12-07', 8);
INSERT INTO patient VALUES (597, 'Brown', 'Brenda', 'F', '1961-06-17', 3);
INSERT INTO patient VALUES (234, 'Jenkins', 'Alan', 'M', '1972-01-29', 7);
INSERT INTO patient VALUES (244, 'Wells', 'Chris', 'F', '1995-02-25', 6);
```

# More efficient insert

- PostgreSQL supports multi-row insert

```
INSERT INTO patient (patient_no, last_name, first_name, sex, date_of_birth, ward)
VALUES
  (454, 'Smith', 'John', 'M', '1978-08-14', 6),
  (223, 'Jones', 'Peter', 'M', '1985-12-07', 8);
```

# Deleting or Modifying a table

```
DROP TABLE IF EXISTS instructor;
```

- delete the `instructor` schema and its tuples

```
ALTER TABLE instructor ADD office VARCHAR(5);
```

- adds attribute 'office' to the instructor schema
- value of office is initialized to 'null'

# Summary

- `CREATE TABLE` – define a relation schema
- `INSERT` – insert rows into a relation instance
- `DROP TABLE` – delete relation schemas
- `ALTER TABLE` – modify relation schemas