Widom Lectures

- Introduction and Relational Model
- Constraints and Triggers

http://online.stanford.edu/course/databases-self-paced

Reminder

Relational and Other Data Models

- **DBMS using the relational DM**
  - IBM DB2
  - MS SQL Server
  - Informix
  - Oracle
  - Sybase
  - Microsoft Access
  - Tandem
  - Teradata
  - SQLite
  - MySQL
  - PostgreSQL…

- Other data models
  - Hierarchical
    - IBM IMS
  - Network
    - IDMS, IDS
  - Object-oriented
    - ObjectStore
  - Object-relational
    - Oracle
  - …

Reminder

Relational (Data) Model

- The most widely-used model today
- **Data model** = a collection of concepts for describing data
  - A collection of relations
  - Relation = set of records – think of it as a table with rows and columns

Students

sid	name	login	age
13	Lisa	lsimp	40
41	Bart	bart	20

Courses

cid	cname	Cr.
E-484	EECS484	4
E-584	EECS584	3

Enrolled

sid	cid	Grade
41	E-484	A-
13	E-584	A+

Reminder

Relational (Data) Model

- **Schema** = a description of data in terms of a data model
  - Every relation has a schema
  - Specifies the name of the relation, the name and type of the columns (or fields or attributes)
  - Each row also called a tuple or a record

```
Students(sid:string, name:string, login:string, age:integer)
Courses(cid:string, cname:string, credits:integer)
Enrolled(sid:string, cid:string, grade:string)
```

Students

sid	name	login	age
13	Lisa	lsimp	40
41	Bart	bart	20

Courses

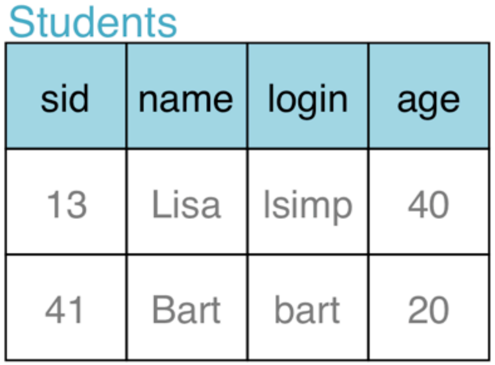
cid	cname	Cr.
E-484	EECS484	4
E-584	EECS584	3

Enrolled

sid	cid	Grade
41	E-484	A-
13	E-584	A+

Relational (Data) Model

- **Schema** = a description of data in terms of a data model
- **Instance** = a table, with rows (aka tuples, records), and columns (aka fields, attributes) that match the schema
  - – # of rows: cardinality
  - – # of columns: degree or arity



Students

sid	name	login	age
13	Lisa	lsimp	40
41	Bart	bart	20

New Scenario: Olympic Games

- Some history:
  - Inspired by the ancient Olympic Games, which were held in Olympia, **Greece** (8th century BC).

Example:
Instance of Athlete Relation

AID	Name	Country	Sport
1	Mary Lou Retton	USA	Gymnastics
2	Jackie Joyner-Kersee	USA	Track
3	Michael Phelps	USA	Swimming

What is the schema?

(aid: integer, name: `string`, country: `string`, sport: `string`)

Cardinality & Degree?

(A) Cardinality: 3, Degree: 3
(B) Cardinality: 3, Degree: 4
(C) Cardinality: 4, Degree: 3

Example:
Instance of Athlete Relation

AID	Name	Country	Sport
1	Mary Lou Retton	USA	Gymnastics
2	Jackie Joyner-Kersee	USA	Track
3	Michael Phelps	USA	Swimming

What is the schema? (aid: integer, name: `string`, country: `string`, sport: `string`)

Cardinality & Degree? Cardinality = 3, Degree = 4

Relational Query Languages

- Supports simple, powerful querying of data
- Queries written declaratively
  - In contrast to procedural methods
- DBMS is responsible for efficient evaluation
  - System can optimize for efficient query execution, and still ensure that the answer does not change
- SQL is the standard database query lan

Structured Query Language (SQL)

• Create a Table	Create
• Add new records	Insert
• Retrieve records	Select
• Update records	Update
• Delete records	Delete
• Create a View	Create
• Update a View	Update

Structured Query Language (SQL)

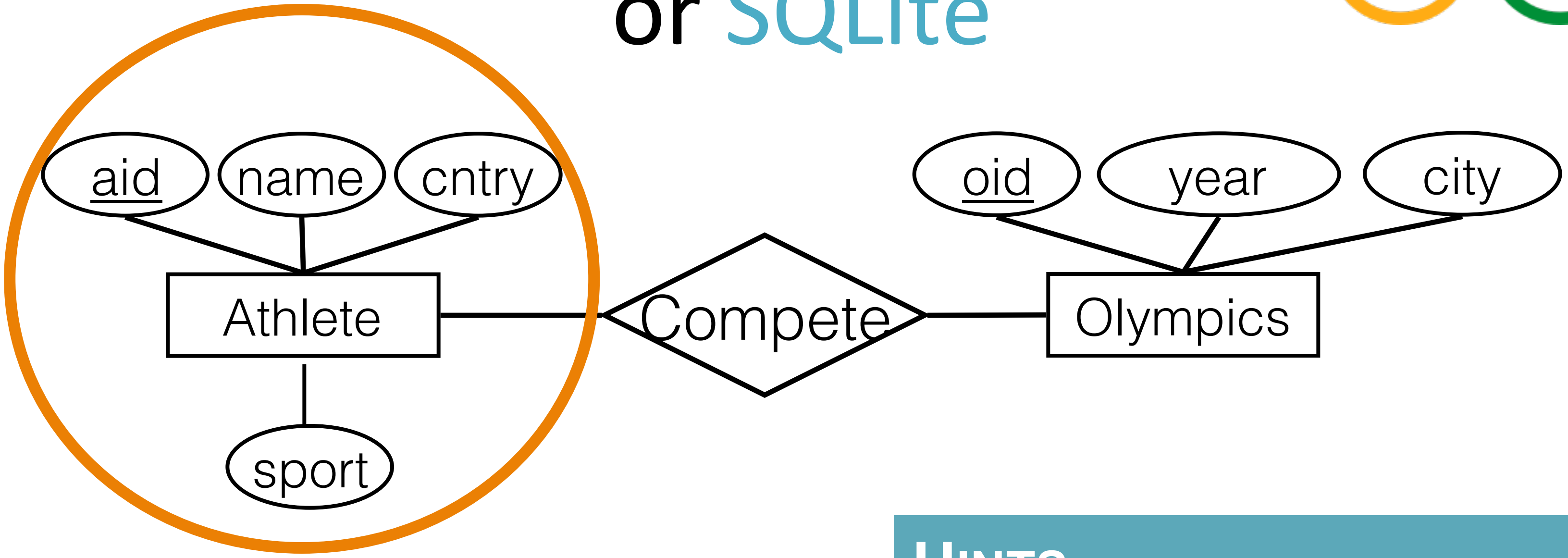
- Create a Table `Create`
  - Integrity Constraints
  - Enforcing Constraints
- Add new records `Insert`
- Retrieve records `Select`
- Update records `Update`
- Delete records `Delete`

- Create a View `Create`
- Update a View `Update`

Create a Table (Relation)

```
CREATE TABLE table_name (
   field1 TYPE,
   field2 TYPE,
   … … …
);
```

Try it out on paper or SQLite



```
CREATE TABLE table_name (
   field1 TYPE,
   field2 TYPE,
    … … …
);
```

HINTS

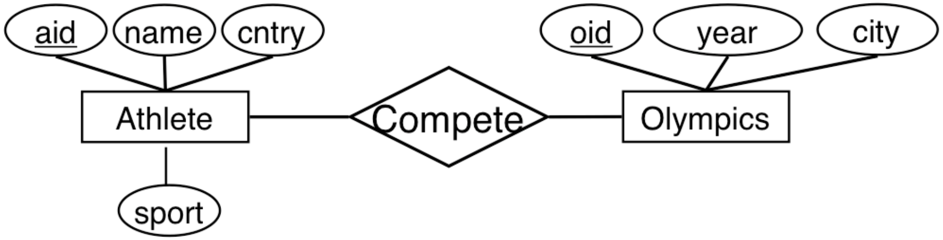
- Examples of types:
- char(20), integer, real, text, blob
- To create a DB named olympics:
  `>> sqlite3 olympics.db`

Download SQLite:
https://www.sqlite.org/download.html

Creating Relations in SQL

- Create the Athlete relation
  - Domain constraint (type) enforced when tuples added or modified

```
CREATE TABLE Athlete
(aid INTEGER,
 name CHAR(30),
 country CHAR(20),
 sport CHAR(20));
```



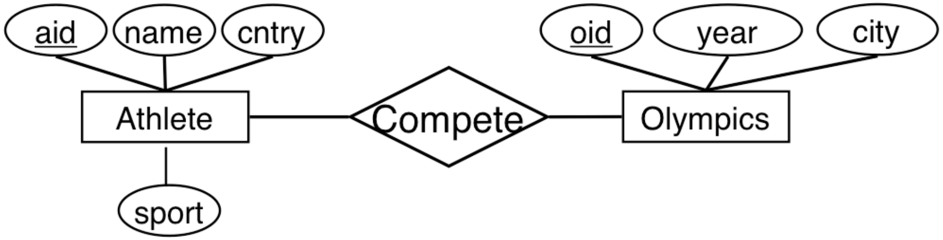
Creating Relations in SQL

- Create the Athlete relation
  - Domain constraint (type) enforced when tuples added or modified

```
CREATE TABLE Athlete
(aid INTEGER,
 name CHAR(30),
 country CHAR(20),
 sport CHAR(20));
```

- Create the Olympics relation

```
CREATE TABLE Olympics
(oid INTEGER,
 year INTEGER,
 city CHAR(20));
```



Creating Relations in SQL

- Create the Athlete relation
  - Domain constraint (type) enforced when tuples added or modified

```
CREATE TABLE Athlete
(aid INTEGER,
 name CHAR(30),
 country CHAR(20),
 sport CHAR(20));
```

- Create the Olympics relation

```
CREATE TABLE Olympics
(oid INTEGER,
 year INTEGER,
 city CHAR(20));
```

- Create the Compete relation

```
CREATE TABLE Compete
(aid INTEGER,
 oid INTEGER);
```

Structured Query Language (SQL)

✓ • Create a Table	`Create`
– Integrity Constraints	
– Enforcing Constraints	
• Add new records	`Insert`
• Retrieve records	`Select`
• Update records	`Update`
• Delete records	`Delete`
• Create a View	`Create`
• Update a View	`Update`

Creating Relations: Constraints

- How to specify certain attributes as keys?
  - e.g., athlete ID (aid) or olympics ID (oid)
  - We must prevent duplicate keys, e.g. two athletes with the same ID in the database

- How to say that the Athlete ID and Olympic ID values in Compete relation must have valid references?

Integrity Constraints (ICs)

- IC: condition that must be true for *any* instance of the database; e.g., domain constraints
  - ICs are specified when schema is defined
  - ICs are checked when relations are modified
- A legal instance of a relation satisfies *all* specified ICs
  - DBMS must not allow illegal instances

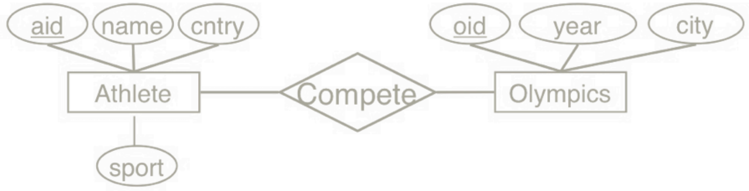
Integrity Constraint: Primary and Candidate Keys

- A **key** for a relation R
  - = minimal set of attributes $A_1, \ldots, A_n$ such that:
  - no two tuples in (any instance of) R can have the same values for $A_1, \ldots, A_n$
- A relation can have more than one key:
  - One is designated as primary key.
  - Others are called candidate keys.

Examples: {aid}, {ssn}, {ssn, name}

- {aid} is a key in the Athlete relation
- {ssn} is a key for Citizen relation
- {ssn, name} is **not** a key, but a superkey – not minimal

PRIMARY KEY Constraint



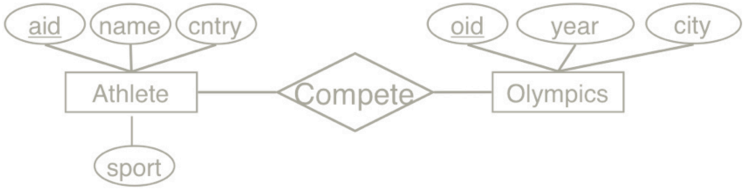
Several ways of specifying the constraint:

```
CREATE TABLE Athlete
(aid INTEGER PRIMARY KEY,
 name CHAR(30),
 country CHAR(20),
 sport CHAR(20));

CREATE TABLE Athlete
(aid INTEGER,
 name CHAR(30),
 country CHAR(20),
 sport CHAR(20),
 PRIMARY KEY(aid));
```

NOT NULL Constraint

Disallow null values for a field



```
CREATE TABLE Athlete
 (aid INTEGER PRIMARY KEY,
  name CHAR(30) NOT NULL,
  country CHAR(20),
  sport CHAR(20));
```

- NULL value = the value is unknown or inapplicable
- Example:
  - country and sport can be NULL (= not known or unspecified)
  - But, name must be specified.

Primary Keys Properties

- Can **never** be null (databases automatically enforce this)
  - NO parts of a composite primary key can be NULL.
- Need not be an integer ID
  - though they often are for efficient search
- IDs used as primary keys do not necessarily auto-increment in databases.
  - Additional features of SQL must be used to make them auto-increment. You will see that in the projects.

Candidate Keys

- Candidate keys specified using UNIQUE
- One of the candidate keys is chosen as the *primary key*.

```
CREATE TABLE Athlete
      (aid INTEGER,
       name CHAR(30) NOT NULL,
       country CHAR(20),
       sport CHAR(20),
      UNIQUE (name, country),
      PRIMARY KEY (aid));
```

In English, what restriction does the candidate key impose here?

WARNING: If used carelessly, ICs can prevent storing instances that arise in practice!

Foreign Keys in SQL

- Only people listed in Athletes relation should be allowed to compete

```
CREATE TABLE Compete
    (aid INTEGER,  oid INTEGER,
      PRIMARY KEY  (aid, oid),
      FOREIGN KEY (aid) REFERENCES Athlete);
```

- … and only in games stored in the Olympics relation

```
CREATE TABLE Compete
    (aid INTEGER,  oid INTEGER,
      PRIMARY KEY  (aid, oid),
      FOREIGN KEY (aid) REFERENCES Athlete,
      FOREIGN KEY (oid) REFERENCES Olympics);
```

Foreign Keys: Definition and Rules

- **Foreign key** = set of fields in one relation that is used to refer to a tuple in another relation.
- Must refer to primary key of the second relation
  - Like a 'logical pointer'
- Example:

```
CREATE TABLE Athlete
(aid INTEGER PRIMARY KEY,
name CHAR(30),
country CHAR(20),
sport CHAR(20));
```

```
CREATE TABLE Compete
    (aid INTEGER,   oid INTEGER,
     PRIMARY KEY  (aid, oid),
     FOREIGN KEY (aid) REFERENCES Athlete,
     FOREIGN KEY (oid) REFERENCES Olympics);
```

If all foreign key constraints are enforced, referential integrity (no dangling references) is achieved.

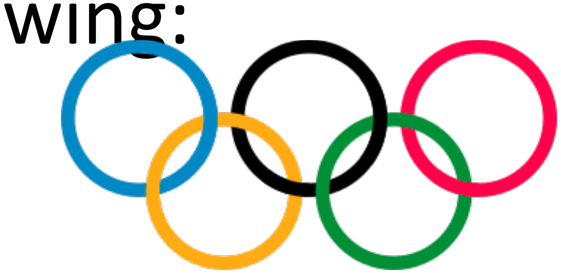
Sample Database

- On Canvas: athlete_create.sql
  - Contains commands to create a database with these three relations
  - and some data
- You can load it into SQLite as follows:

```
% sqlite3 athlete.db
.read athlete_create.sql
```

- If you use sqlplus, use the following:

```
START athlete_create.sql
```



Structured Query Language (SQL)

✓ • Create a Table	`Create`
– Integrity Constraints	
– Enforcing Constraints	
• Add new records	`Insert`
• Retrieve records	`Select`
• Update records	`Update`
• Delete records	`Delete`
• Create a View	`Create`
• Update a View	`Update`

Enforcing ICs

- Whenever we modify the database
  - must check for violations of ICs

- Enforcing Domain, Primary Key, Unique ICs is straightforward
  - Reject offending UPDATE / INSERT command

Enforcing Referential Integrity

- If a Compete tuple is inserted with no corresponding Athlete aid:
    - Insert operation is REJECTED!

```
CREATE TABLE Compete
    (aid INTEGER,  oid INTEGER,
      PRIMARY KEY  (aid, oid),
      FOREIGN KEY (aid) REFERENCES Athlete,
      FOREIGN KEY (oid) REFERENCES Olympics);
```

Enforcing Referential Integrity

- If a Compete tuple is inserted with no corresponding Athlete aid:
  - Insert operation is REJECTED!

- What if an Athlete tuple is deleted? Possible actions:
  - **Disallow deletion** if a Compete tuple refers to athlete
  - **Delete all** Compete tuples that refer to deleted athlete
  - **Set to default or null** value for all references to the deleted athlete

- Similar choices on update of primary key of Athlete

Referential Integrity in SQL

- SQL supports all four options on deletes and updates
  - Default is NO ACTION: action is rolled back;
    Similar to RESTRICT: action is disallowed.
  - CASCADE: also delete all tuples that refer to deleted tuple
  - SET NULL / SET DEFAULT: sets foreign key value of referencing tuple

```
CREATE TABLE Compete
    (aid INTEGER,  oid INTEGER,
      PRIMARY KEY  (aid, oid),
      FOREIGN KEY (aid)
      REFERENCES Athlete
      ON DELETE CASCADE
      ON UPDATE SET NULL)
```

What happens with an associated Compete tuple if we modify an athlete's ID?

Referential Integrity in SQL

- SQL Supports all four options on deletes and updates
  - Default is NO ACTION: action is rolled back;

    Similar to RESTRICT: action is disallowed.
  - CASCADE: also delete all tuples that refer to deleted tuple
  - SET NULL / SET DEFAULT: sets foreign key value of referencing tuple

```
CREATE TABLE Compete
    (aid INTEGER,  oid INTEGER,
      PRIMARY KEY  (aid, oid),
      FOREIGN KEY (aid)
      REFERENCES Athlete
      ON DELETE CASCADE
      ON UPDATE NO ACTION)
```

What happens if we modify an athlete's ID with an associated Compete tuple?

Table Constraints

- More general than key constraints
- Can use a query to express constraint
  - Constraints checked each time table updated
  - CHECK constraint always true for empty relation

```
CREATE TABLE  Sailors
        ( sid  INTEGER,
        sname  CHAR(10),
        rating  INTEGER,
        age  REAL,
        PRIMARY KEY  (sid),
        CHECK  ( rating >= 1
                AND rating <= 10 )
```

```
CREATE TABLE  Reserves
        ( sname  CHAR(10),
        bid  INTEGER,
        day  DATE,
        PRIMARY KEY  (bid,day),
        CONSTRAINT  noInterlakeRes
        CHECK  (`Interlake’  <>
                        ( SELECT  B.bname
                        FROM  Boats B
                        WHERE  B.bid=bid)))
```

Constraints Over Multiple Relations

- For general constraint over multiple tables, use an assertion

Number of boats plus number of sailors is < 100

```
CREATE ASSERTION  smallClub
CHECK
((SELECT COUNT (S.sid) FROM Sailors S) +
  (SELECT COUNT (B.bid) FROM Boats B) < 100)
```

Triggers vs. Constraints

- Often used to maintain consistency
  - Can you use a foreign key?
  - Foreign keys are defined declaratively
- Constraints are easier to understand than triggers
- Triggers are more powerful.
  - Often used to fill out fields in a form
  - Check complex actions (such as credit limit in a shopping application)
  - Check preferred customer status
  - Generate logs for auditing and security checks.

Try it out

- Modify athlete_create.sql so that it has the UPDATE and DELETE constraints in COMPETE relation as in the previous slide.
  - Modified file available in athlete_modified.sql.
- Try the following and check COMPETE:

```
sqlite> DELETE FROM Athlete WHERE name='Michael
Phelps';
sqlite> UPDATE Athlete SET aid=5 WHERE aid=4;
```

Implementation Note: In SQLite, make sure you issued "`PRAGMA foreign keys = ON;`" command to enforce foreign key constraints. By default, SQLite ignores them for backward compatibility.

Implementation Notes

- Oracle's sqlplus:
  - You cannot use NO ACTION constraints. They are the default and thus not needed.
  - String literals like 'USA' must use **single quotes**, not double quotes
- SQLite:
  - You need `PRAGMA foreign_keys = ON;` to enforce foreign key constraints. This is for backward compatibility.

Where do ICs Come From?

- Based on real-world enterprise being modeled
- An IC is a statement about all possible instances!
- We can **check** a database instance to see if an IC is **violated**, but we can NEVER infer that an IC is true by looking at an instance.
- Key and foreign key ICs are the most common
- Also table constraints and assertions

Destroying & Altering Relations

- To destroy the relation Olympics.
  - Schema information and tuples are deleted

```
DROP TABLE Olympics
```

- To alter the Athlete schema by adding a new column

```
ALTER TABLE Athlete
    ADD COLUMN age: INTEGER
```

- What do we put in the new field?
- A null value: ‘unknown’ or ‘inapplicable’

Relational Model: Summary

Students

sid	name	login	age
13	Lisa	lsimp	40
41	Bart	bart	20

Courses

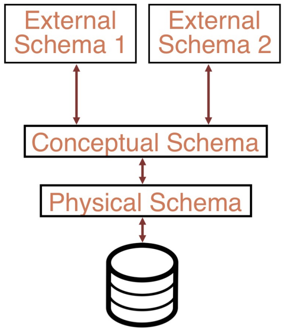
cid	cname	Cr.
E-484	EECS484	4
E-584	EECS584	3

Enrolled

sid	cid	Grade
41	E-484	A-
13	E-584	A+

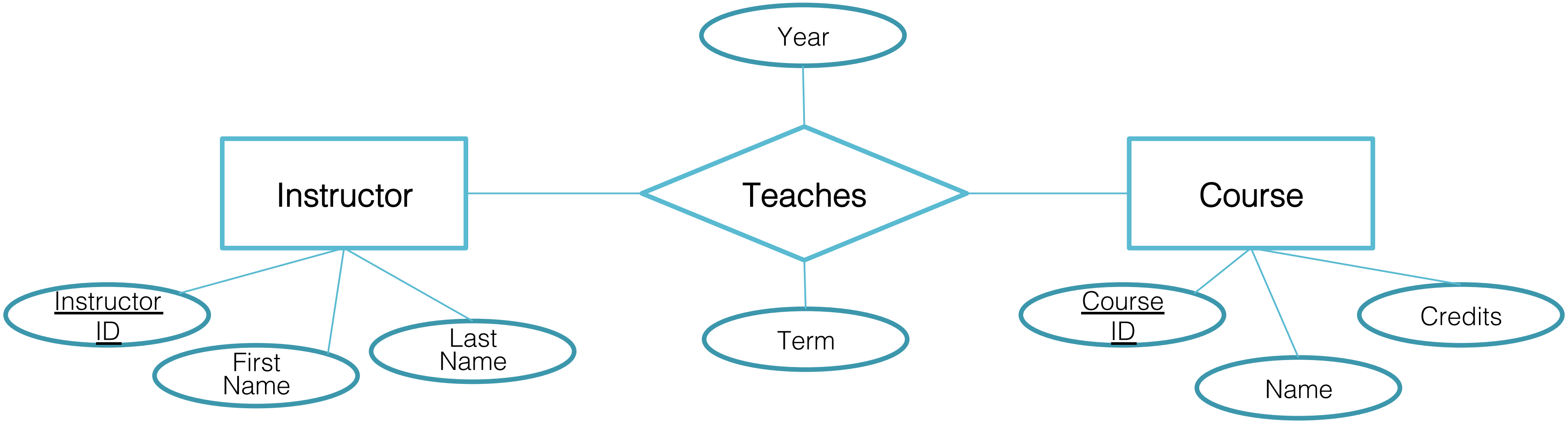
- A tabular representation of data
- Simple and intuitive
- Currently the most widely used database model
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - We always have domain constraints

    e.g. `INTEGER` fields must always contain integer values
- Views can be used for external schemas, and provide logical data independence



Next

- Discussion: Project 1 intro
- Monday: Translation from ER diagrams to Relations



Instructor ID	First Name	Last Name
394953	John	Smith
454544	Sara	King
439849	Alex	Dee
….	….	….
….	….	….

Instructor ID	Course ID	Year	Term
454544	E302	2009	F
394953	C210	2010	W
439849	M184	2010	F
….	….	….	….
….	….	….	….

Course ID	Name	Credits
M184	Calculus	3
C210	Physics	4
E302	Algorithms	4
….	….	….
….	….	….

Terminology Parade

- Database: A set of relations or tables in the database:
- Relation: Defined by:
  - Schema: Describes the columns and constraints
    - Relation name
    - Name and domain (i.e., type) for each column
    - E.g., Student (sid: integer, name: string, gpa: real)
  - Instance: A table, with rows (aka tuples, records), and columns (aka fields, attributes) that match the schema
    - # Rows = cardinality
    - # Columns = degree / arity
- Set semantics: (classical relational model) *Every row is unique*
- Multiset semantics: (modern systems, SQL) *Duplicate rows allowed*

Integrity Constraints

- Describes conditions that must be satisfied by every legal instance
- Types of integrity constraints
  - Domain constraints
  - Primary key constraints
  - Foreign key constraints
  - **General constraints**