

# The Relational Model: Database Definition and Integrity Constraints

## Chapter 3

# Jennifer Widom Lectures

- Introduction and Relational Databases
- Relational Algebra
- SQL
- Constraints and Triggers

<https://lagunita.stanford.edu/courses/DB/2014/SelfPaced/about>

# 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
  - ✧ ...



# 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+



# 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+

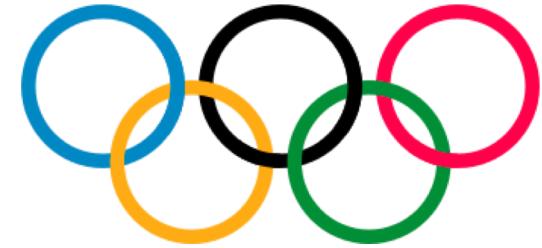


# Relational (Data) Model

- **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** (8<sup>th</sup> 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**)

# 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

Cardinality & Degree?

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

# 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 language



# 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

- Integrity Constraints
- Enforcing Constraints

create

## Add new records

Insert

## Retrieve records

Select

## Update records

Update

## Delete records

Delete

## Create a View

Create

## Update a View

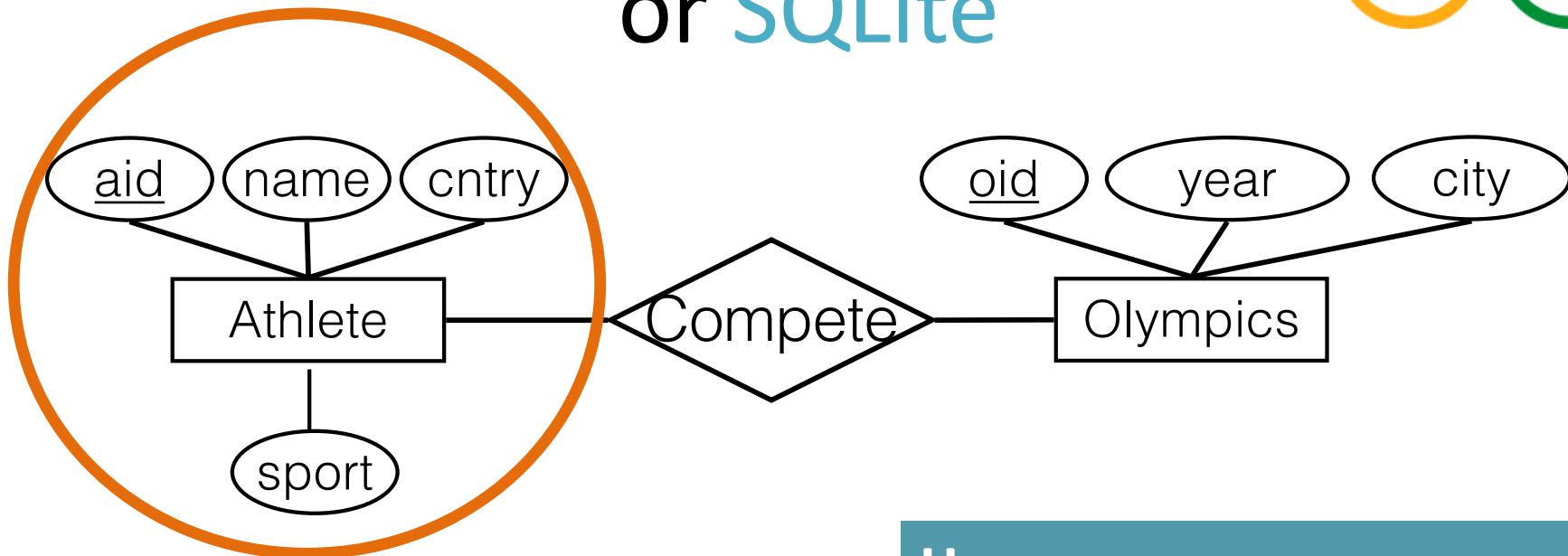
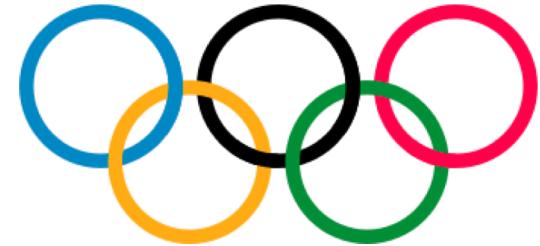
Update



# 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:

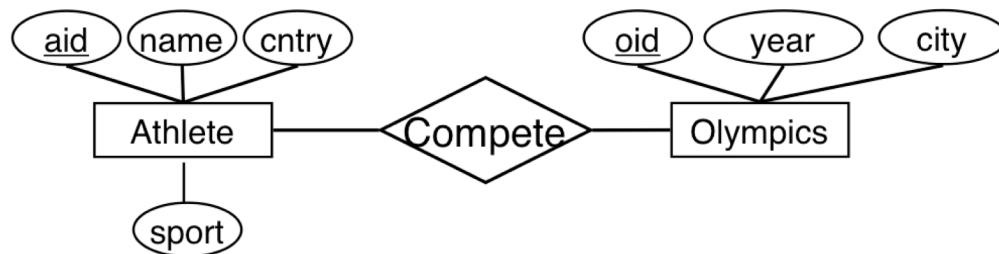
<https://www.sqlite.org/download.html>



# Creating Relations in SQL

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

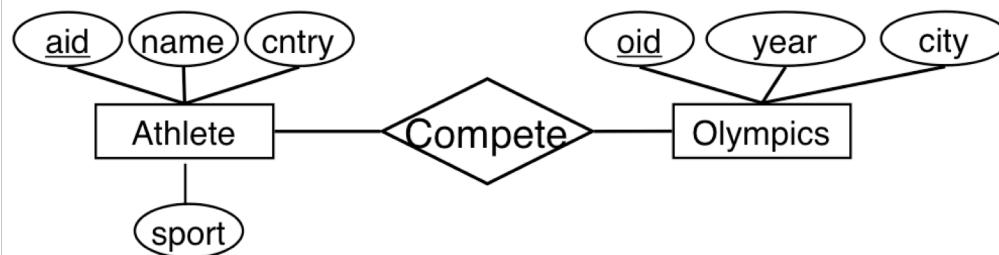
```
CREATE TABLE Athlete  
(aid INTEGER,  
name CHAR(30),  
country CHAR(20),  
sport CHAR(20));
```





# Creating Relations in SQL

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



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

```
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 the Olympics relation
- Create the Compete relation

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

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

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

# Structured Query Language (SQL)



- Create a Table
  - Integrity Constraints
  - Enforcing Constraints
- Add new records      create
- Retrieve records      Insert
- Update records      Select
- Delete records      Update
- Create a View      Delete
- Update a View      Create
- 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  $A_1, \dots, A_n$  such that:
  - no two tuples in **(any instance of)** R can have the same values for  $A_1, \dots, A_n$
- A set of attributes that satisfies the above condition, without the minimal requirement, is called a **superkey**.
- Every key is also a superkey.
- A relation can have more than one key:
  - One is designated as **primary key**.
  - Others are called **candidate keys**.

Examples: {aid}, {ssn},

- {aid} is a key in the Athlete relation
- {ssn} is a key for Citizen relation

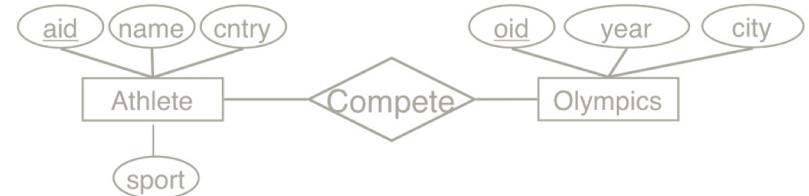


# PRIMARY KEY Constraint

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));
```

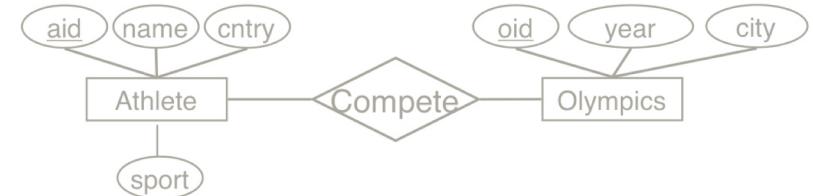




# Not NULL Constraint

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));
```

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

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



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

# 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)
```



# Referential Integrity in SQL

- 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 to the Compete relation if we add a new athlete (with a new ID)?



# Referential Integrity in SQL

- 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?

# Primary Keys and ICs

- In (most) DBMSs the primary key (or any of its parts, if it is composite) cannot get NULL value.
  - In this class, we will stick with this rule (despite the fact that there are exceptions)
- Although you can choose how to handle UPDATE/DELETE actions when there are references (via foreign keys), some options may be in conflict with the primary key (PK) constraints
  - The system will not allow changes that violate the PK constraints.

# 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

- 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

Students				Courses			Enrolled		
sid	name	login	age	cid	cname	Cr.	sid	cid	Grade
13	Lisa	lsimp	40	E-484	EECS484	4	41	E-484	A-
41	Bart	bart	20	E-584	EECS584	3	13	E-584	A+

