The Relational Model

Chapter 3

Relational Databases

- Most common data model in modern DBMS
- Many commercial systems
  - Oracle, MS SQL Server, IBM DB2, more…
- Also open source
  - MySQL, PostgreSQL, SQLite, …

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

- Set semantics: (classical relational model, like ER model) *Every row is unique*
- Multiset semantics: (modern systems, SQL) *Duplicate rows allowed*

Terminology

Athlete Relation **schema**:
Athlete(aid: integer, name: string, country: string, sport:string)

Athlete Relation **instance:**

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

Cardinality: Number of rows
Degree: Number of columns

Cardinality = 3, Degree = 4

Structured Query Language (SQL)

- A standard declarative language for relational databases to update/query tables
  - Create a Table
  - Add new records (INSERT)
  - Retrieve records (SELECT)
  - Update records (UPDATE)
  - Delete records (DELETE)

Create a Table (Relation)

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

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);
```

Trying these out

- Sqlite3 is a personal database in which the entire database is stored in a single file on your computer
- Install sqlite3 if you have a personal computer
- Alternative: login to login.engin.umich.edu and use sqlite3 in a terminal window.

Example: To work with a database athlete.db:

```
% sqlite3 athlete.db
```

Trying this out: Sample Database

- We have posted a file athlete_create.sql with these commands and some data inserted into the tables.
- Recommendation: Install sqlite3 (see Piazza)
- You can load it into SQLite as follows:

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

- Alternatively: enter into Oracle's sqlplus:

```
START athlete_create.sql
```

Integrity Constraints: Examples

- How do we specify that certain attributes are 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 do say that the Athlete ID and Olympic ID values in compete relation must be valid references?
  - Referential Integrity

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 is one that satisfies all specified ICs.
  - DBMS must not allow illegal instances.

Integrity Constraint: Primary and Candidate Keys

- A key for a relation R is the *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 $A_1,\ldots,A_n$
- E.g., {ssn} is a key for Citizen relation
- {ssn, name} is **not** a key, but a superkey – not minimal.
- A relation can have more than one key; one is designated as primary key. Others are called candidate keys

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 in tables indicates that the value is unknown or inapplicable.

Primary Keys Properties

- PRIMARY KEY columns can never be null
  - Databases automatically enforce this
- PRIMARY KEYS need not be an integer ID, though they often are for entities
- IDs when 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 specified as the *primary key*.

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

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’.
- E.g., *aid* in Competes relation is a foreign key referring to Athlete
  - If all foreign key constraints are enforced, *referential integrity* (no dangling references) is achieved.

Enforcing ICs

- Whenever we modify 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!

- 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 or RESTRICT (action is rolled back);
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple to NULL or a default value)

```
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?

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:
  - DELETE FROM Athlete WHERE name='Michael Phelps';
  - UPDATE Athlete SET aid=5 WHERE aid=4;

(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
  - Next week!

Views

- View is used just like a relation, but we store a *definition*, rather than a set of tuples

```
CREATE  VIEW  Athens_Olympians
          AS SELECT A.aid, A.name, A.country
          FROM Athlete A, Competes C, Olympics O
          WHERE A.aid = C.aid AND C.oid = O.oid
                              AND O.year = 2004
```

- Provide external data independence
- Security
- Views can be dropped using DROP VIEW
- How to drop a table if there is a view on it?
  - DROP TABLE command has options to let user specify this

Views

What does this query compute?

```
SELECT name
FROM Athens_Olympians
WHERE country = 'USA';
```

Find the names of all athletes from country ‘USA’ who participated in the 2004 Olympics.

Destroying & Altering Relations

```
DROP TABLE Olympics
```

Destroys the relation Olympics.

(Schema information and tuples are deleted)

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

Alters Athlete schema by adding a new column

What do we put in the new field?

a null value: ‘unknown’ or ‘inapplicable’

Adding & Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO  Athlete (aid, name, country, sport)
VALUES  (4, 'Johann Koss', 'Norway', 'Speedskating')
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Athlete A
WHERE A.name =  ‘Smith’
```

Updates on Views?

- User perspective: view is like a table
  - Fine for queries, but what about updates?
- Can be tricky to figure out how updates map back to data stored in base tables

Updates on Views - Example

Athlete

aid	name	age
1	Alice	18
2	Alice	25
3	Bob	22

Sport

sid	name	sport
1	Alice	tennis
2	Alice	frisbee
3	Bob	skating

ActiveStudents

name	age	sport
Alice	18	tennis
Alice	18	frisbee
Alice	25	tennis
Alice	25	frisbee
Bob	22	skating

```
CREATE VIEW ActiveStudents
AS SELECT A.name, A.age, S.sport
FROM Athlete A, Sport S
WHERE A.name = S.name
```

What if we want to delete the row (Alice, 18, tennis) from ActiveStudents ?

Updates on Views

- *Key is to guarantee that update can be mapped to precisely one tuple in one base table*
- SQL-99
  - Can update field of a view if it is obtained from exactly one base table, and primary key of that table included in view.

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
  - In addition, we *always* have domain constraints, e.g., INTEGER fields must always contain integer values
- Views can be used for External schemas, Logical data independence

Looking Forward…

- Suggested exercises: 3.1, 3.3, 3.5, 3.7, 3.9, 3.19
- Next time: Translating ER diagrams to relational Tables
  - See Chapter 3.5

The SQL Query Language

- Find all athletes from USA:

```
SELECT *
FROM Athlete A
WHERE A.country = ‘USA’
```

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

- Print only the names and sports:

```
SELECT A.name, A.sport
FROM Athlete A
WHERE A.country = ‘USA’
```

Name	Sport
Mary Lou Retton	Gymnastics
Jackie Joyner-Kersee	Track
Michael Phelps	Swimming

Querying Multiple Relations

- What does the following query compute?

```
SELECT O.year
FROM Athletes A, Olympics O, Compete C
WHERE A.aid = C.aid AND O.oid = C.oid
            AND A.name = ‘Michael Phelps’
```

Find the years when Michael Phelps competed in the Olympics