Fundamentals of Database Systems
COMPSCI 351

Instructor: Sebastian Link

The University of Auckland

The Relational Model of Data

What is the Relational Model of Data?

The simplest yet most important data model

Its inventor is E.F. Codd

- "A Relational Model for Large Shared Data Banks"
- published in *Communications of the ACM*, Vol 13, No 6, 1970

A Foundation for Database R&D

- distinguishes between syntax (schemata) and semantics (instances),
- enables physical data independence,
- basis for powerful languages, e.g. relational algebra and calculus,
- properties can be discovered and justified

What is the Relational Model of Data?

Simple yet powerful

- based on the single concept of a relation
- can represent a large variety of applications

Industry standard SQL founded on the relational model

- vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- syntheses emerging: object-oriented models, Web, NewSQL

A Simple Approach to Managing Data

- Information systems deal with storage and retrieval of data
    - use tuples to represent data about real-world objects

Example

The tuple

(13 Assassins, Miike Takashi, Japan, 2010)

represents the movie with title "13 Assassins", director "Miike Takashi", country of production "Japan", and production year "2010"

- Databases represent a whole collection of movies
    - e.g., from the New Zealand International Film Festival

The relational data model (RDM) is based on this approach

A Simple Approach to Managing Data

- Relations are sets of tuples
  - suitable to represent collections of data of any size
  - can be illustrated by tables

Example

Title	*Director*	*Country*	*Year*
13 Assassins	Miike Takashi	Japan	2010
A Cat in Paris	Alain Gagnol	Belgium	2010
Brother Number One	Annie Goldson	New Zealand	2011
La dolce vita	Federico Fellini	Italy	1960
Mana Waka	Merata Mita	New Zealand	1937
Nosferatu	Friedrich Murnau	Germany	1922
The Yellow Sea	Na Hong-jin	Korea	2010
Tyrannosaur	Paddy Considine	UK	2011

Tuples and Attributes

- Entries in a tuple capture some *property* of a real-world object

Example

In (13 Assassins, Miike Takashi, Japan, 2010) we have the *title* “13 Assassins”, *director* “Miike Takashi”, the *country* “Japan”, and *production year* “2010”

- These properties are called *attributes*

Tuples in a relation all have the same structure

- entries capture the same properties for all real-world objects
- e.g., for movies we capture the attributes *Title*, *Year*, *Director* and *Country*

Example: Tuples and Attributes

In table illustrations we use these properties (attributes) as column headers

Example

Title	*Director*	*Country*	*Year*
13 Assassins	Miike Takashi	Japan	2010
A Cat in Paris	Alain Gagnol	Belgium	2010
Brother Number One	Annie Goldson	New Zealand	2011
La dolce vita	Federico Fellini	Italy	1960
Mana Waka	Merata Mita	New Zealand	1937
Nosferatu	F.W. Murnau	Germany	1922
The Yellow Sea	Na Hong-jin	Korea	2010
Tyrannosaur	Paddy Considine	UK	2011

Relation Schemata

With every attribute we associate a *domain*, that is, a universal set containing all possible values of this attribute

Example

For movies, the attributes could have the following domains:

- $dom(\text{title}) = \texttt{string}$, $dom(\text{year}) = \texttt{nat}$,
- $dom(\text{director}) = \texttt{string}$, and $dom(\text{country}) = \texttt{string}$,

where `string` is the set of all strings over a fixed alphabet, while `nat` is just the set $\mathbb{N}$ of natural numbers.

Definition (Relation Schema)

A *relation schema* is a finite set, usually denoted by R. The elements of R are called *attributes*, and each attribute $A \in R$ is associated with a domain $dom(A)$.

Example of a Relation Schema

Example

We use the relation schema $\textsc{Movie} = \{$ title, director, country, year $\}$

Notation

- To emphasize the sequence of attributes, use $R(A_1, \ldots, A_n)$, e.g. $\textsc{Movie}$(title, director, country, year)
- To emphasize domains, write $R(A_1 : dom(A_1), \ldots, A_n : dom(A_n))$, e.g. $\textsc{Movie}$(title:`string`,director:`string`, country:`string`, year:`nat`)

Relations over a Relation Schema

A relation schema provides an abstract description of the tuples in a relation

Definition (Tuple and relation)

Let $R = \{A_1, \ldots, A_n\}$ be a relation schema. An R-**tuple** is an element t of the cartesian product

$$dom(A_1) \times \cdots \times dom(A_n).$$

An R-**relation** is a finite set r of R-tuples, that is, a finite relation

$$r \subseteq dom(A_1) \times \cdots \times dom(A_n).$$

Example: Tuple and Relation

Example

$$t = (\text{13 Assassins}, \text{Miike Takashi}, \text{Japan}, 2010)$$

is a $\textsc{Movie}$-tuple for the relation schema $\textsc{Movie}$(title, director, country, year)

Conventions

- Write $t = (A_1 : v_1, \ldots, A_n : v_n)$ to emphasize that v_i belongs to attribute A_i
- $t = (\mathit{title} : \text{13 Assassins}, \mathit{director} : \text{Miike Takashi}, \mathit{country} : \text{Japan}, \mathit{year} : 2010)$

An Alternative Definition for a Tuple

Definition

An R-tuple is a function

$$t : R \to \bigcup_{A \in R} dom(A)$$

mapping every attribute $A \in R$ to some element $t(A) \in dom(A)$.

Example

Title	*Director*	*Country*	*Year*
13 Assassins	Miike Takashi	Japan	2010
A Cat in Paris	Alain Gagnol	Belgium	2010
Brother Number One	Annie Goldson	New Zealand	2011
La dolce vita	Federico Fellini	Italy	1960
Mana Waka	Merata Mita	New Zealand	1937
Nosferatu	F.W. Murnau	Germany	1922

Database Schemata

A database contains more than just a single relation.
Consequently, we have several relation schemata in a database schema.

Definition (Database Schema)

A **database schema** is a finite set $\mathcal{S}$ of relation schemata.

Example

A database schema consisting of four relation schemata:

- MOVIE(title, year, country, run_time, genre)
- PERSON(id, first_name, last_name, year_born)
- DIRECTOR(id, title, year)
- ACTOR(id, title, year, role)

Relational Database

A relational database over a database schema consists of a relation for every relation schema that is a member of the database schema.

Definition (Relational Database)

Let $\mathcal{S}$ a database schema. An $\mathcal{S}$-**database**, usually denoted by $\mathcal{I}$, consists of just one R-relation $\mathcal{I}(R)$ for each relation schema R in $\mathcal{S}$, that is,

$$\mathcal{I} = \{\mathcal{I}(R) | R \in \mathcal{S}\}.$$

An Example for a Relational Database

MOVIE

title	year	country	run_time	genre
13 Assassins	2010	Japan	126	Drama
La dolce vita	1960	Italy	174	Classic
Mana Waka	1937	New Zealand	85	History
Nosferatu	1922	Germany	80	Horror
Tyrannosaur	2011	UK	91	Drama

DIRECTOR

id	title	year
1	13 Assassins	2010
3	La dolce vita	1960
6	Mana Waka	1937
7	Nosferatu	1922

PERSON

id	first_name	last_name	year_born
1	Miike	Takashi	1960
2	Koji	Yakusho	1956
3	Federico	Fellini	1920
4	Marcello	Mastroianni	1924
5	Anita	Ekberg	1931
6	Merata	Mita	1942
7	Friedrich	Murnau	1888
8	Max	Schreck	1879

ACTOR

id	title	year	role
2	13 Assassins	2010	Shinzaemon Shimada
4	La dolce vita	1960	Marcello Rubini
5	La dolce vita	1960	Sylvia
8	Nosferatu	1922	Graf Orlock
8	Nosferatu	1922	Nosferatu

$$R \mapsto \mathcal{I}(R)$$

Fast Data Access through Unique Column Combinations

Fast access to tuples is important for data processing (eg. queries and updates)

some column combinations have value combinations that uniquely identify tuples

- no two different movies have the same title, year, and country of production
- no two different people have the same id

minimal combinations of columns with this property are particularly interesting

- no two different movies have the same title and the same year of production
- but there are different movies with the same title and
- there are obviously different movies produced in the same year

Keys and Superkeys

Definition (Superkey)

A **superkey** over a relation schema R is a finite subset $K \subseteq R$ of R. An R-relation r is said to **satisfy** the superkey K over R if every pair of distinct tuples $t_1, t_2 \in r$ deviates on at least one attribute of K, that is, $t_1(A) \neq t_2(A)$ for some $A \in K$.

Definition (Key)

A superkey K over R is said to be a **key** if there is no other superkey K' over R that is a proper subset of K, that is, $K' \subset K$.

Every key is a superkey, but only some superkeys are keys.

Example: Keys and Superkeys

Example

title	year	country
The Magnificent Seven	2016	USA
The Magnificent Seven	1960	USA
Psycho	1960	USA

Superkeys that are satisfied

- {title, year}
- {title, year, country}

Keys that are satisfied

- {title, year}

Superkeys that are violated

- {title, country}
- {year, country}

Discussion on Keys and Superkeys

Of practical interest are ...

- Keys that are satisfied *every* relation representing a real-world instance
- but not keys that are only satisfied by some particular relation (accidental keys)

Example

Is {title} a good key over MOVIE(title, year, country, run_time, genre)?

- The NZ film festival snapshot relation satisfies {title}
- Our last example shows there are different movies with the same title
- Are there different movies with the same title in the New Zealand Film Festival?

Comprehending keys is important and difficult in practice

What is a good key over ACTOR(id, title, year, role)?

Example

Business analysts need answers for the following questions:

- Can the same role in the same movie be played by different people?
  - If yes, then {title, year, role} is not a good key
  - If no, then {title, year, role} is a good key
- Can the same person in the same movie play different roles?
  - If yes, then {id, title, year} is not a good key
  - If no, then {id, title, year} is a good key
- Can the same person play the same role in different movies?
  - If yes, then {id, role} is not a good key
  - If no, then {id, role} is a good key

More comments on keys

- Database designers specify all keys that make sense
- The specification of keys restricts the number of possible database instances
- This helps reduce database instances to those which are more realistic, and helps identify objects in a database more efficiently
- In the literature:
  - our term *key* is sometimes referred to as *minimal key* or as *candidate key*
  - our term *superkey* is sometimes referred to as *key*

Keys enforce Codd's principle of entity integrity, that is, the unique identification of entities within a relation

Acessing Data Across Tables

Data entries in one table may identify tuples in other tables

Benefits

- This ensures Codd's principle of referential integrity, that is, the correct reference of entities across relations
  - for example, the id of an actor provides a reference to a unique person
- It eliminates data redundancy which speeds up updates
  - for example, we do not need to store the names of actors in the ACTOR table

Foreign Keys defined

Definition

A **foreign key** over a relation schema R in a database schema $\mathcal{S}$ is

- a sequence of attributes $A_1, \ldots, A_n \in R$ together with
- a key $K = \{B_1, \ldots, B_n\}$ on some relation schema $S \in \mathcal{S}$ where
- with $dom(A_i) = dom(B_i)$ for $i = 1, \ldots, n$.

This is usually denoted by $[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n]$.
The foreign key $[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n]$ over R is said to be **satisfied** by the database instance $\mathcal{I}$ of $\mathcal{S}$ if

- for each tuple $t \in \mathcal{I}(R)$ there is
- a tuple $s \in \mathcal{I}(S)$ such that
- $t(A_i) = s(B_i)$ for all $i = 1, \ldots, n$.

Examples for Foreign Keys

Example

Foreign keys on $\textsc{Director}$(id, title, year) are

- [id] $\subseteq$ $\textsc{Person}$[id]:
  the id of a director identifies a unique person
- [title,year] $\subseteq$ $\textsc{Movie}$[title,year]:
  the title and year of a director identify a unique movie

The specification of foreign keys restricts the possible database instances to those considered meaningful by the application domain.

Foreign key on DIRECTOR: $[title, year] \subseteq$ MOVIE$[title, year]$!

MOVIE

title	year	country	run_time	genre
13 Assassins	2010	Japan	126	Drama
La dolce vita	1960	Italy	174	Classic
Mana Waka	1937	New Zealand	85	History
Nosferatu	1922	Germany	80	Horror
Tyrannosaur	2011	UK	91	Drama

DIRECTOR

id	title	year
1	13 Assassins	2010
3	La dolce vita	1960
6	Mana Waka	1937
7	Nosferatu	1922

PERSON

id	first_name	last_name	year_born
1	Miike	Takashi	1960
2	Koji	Yakusho	1956
3	Federico	Fellini	1920
4	Marcello	Mastroianni	1924
5	Anita	Ekberg	1931
6	Merata	Mita	1942
7	Friedrich	Murnau	1888
8	Max	Schreck	1879

ACTOR

id	title	year	role
2	13 Assassins	2010	Shinzaemon Shimada
4	La dolce vita	1960	Marcello Rubini
5	La dolce vita	1960	Sylvia
8	Nosferatu	1922	Graf Orlock
8	Nosferatu	1922	Nosferatu

Example for a Common Pitfall

MOVIE(title, year, country) with key {title, year}
ACTOR(id, title, year, role) with

- key {id, title, year, role} and
- foreign key [title, year] $\subseteq$ MOVIE[title, year]

The foreign key does not permit the same databases as the two inclusion dependencies

- [title] $\subseteq$ MOVIE[title] and
- [year] $\subseteq$ MOVIE[year]

MOVIE

title	year	country
Gran Torino	2008	USA
Moana	2016	USA

ACTOR

actor	title	year	role
11	Gran Torino	2016	Walt
24	Moana	2008	Maui

Integrity Constraints

Database schema must capture both structure and semantics of application

Integrity constraints enforce the business rules of applications in databases

- they are specified on the database schema
- classify databases into those that are
  - meaningful (i.e. those databases satisfying all constraints),
  - and not meaningful (i.e. those databases not satisfying some constraint)

Integrity Constraints continued

Databases are restricted to those considered meaningful for applications

Primary examples are: domain, key, and foreign key constraints

Integrity constraints interact with one another

- explicit enforcements of some constraints enforces others implicitly
- e.g., explicit enforcement of a key means implicit enforcement of its superkeys
- efficient database maintenance means minimization of costs to enforce constraints

Integrity constraints greatly determine the design of a database schema

- to process most common queries efficiently, and
- to process most common updates efficiently, but
- in many cases compromises are necessary

Example for Challenges in Schema Design

Application domain: *suppliers* deliver *articles* from a *location* at a *cost*

- for every article there is at most one supplier
- the article and location determine the cost
- the set of locations a supplier delivers from is independent of the set of articles delivered and costs charged for delivery

$\mathcal{S}_1 = \{R_1(\text{article, supplier, location, cost})\}$

article	*supplier*	*location*	*cost*
Kiwi	G6Fruitz	Tauranga	NZD1
Kiwi	G6Fruitz	Gisborne	NZD1

$\mathcal{S}_2 = \{R_2(\text{supplier, location}), R_3(\text{article, supplier, cost})\}$

supplier	*location*
G6Fruitz	Tauranga
G6Fruitz	Gisborne

article	*supplier*	*cost*
Kiwi	G6Fruitz	NZD1

Design choice depends on workload of database

- most common queries (e.g. choose $\mathcal{S}_1$ to query locations of articles)
- most common updates (e.g. choose $\mathcal{S}_2$ to update costs of articles)
- maintenance costs (efficiency of integrity enforcement)

Summary for the Relational Model of Data

- Relational DBMSs are based on the relational data model
- The relational data model is formally defined, its properties can be proven, explained and justified, and formal query languages such as relational calculus and algebra have been defined on it.
- The most important concepts in the relational data model are:
  - syntactic level: attributes, relation schemata, database schemata
  - semantic level: domains, tuples, relations, databases
- Integrity constraints play an important part in schema design
  - determine efficiency of updates
  - determine efficiency of queries
  - DBMSs offer support for enforcement of some constraints
    - domain constraints, key constraints, foreign key constraints