COMP9311: Database Systems

Term 2 2019

Week 2 Tuesday (Relational Data Model)

By Helen Paik, CSE UNSW

Disclaimer: the course materials are sourced from

- previous offerings of COMP9311 and COMP3311
- Prof. Werner Nutt on Introduction to Database Systems (http://www.inf.unibz.it/~nutt/Teaching/IDBs1011/)

Relational Data Model

Chapter 5 of the textbook …

Different schemas are based on different levels of abstractions

Conceptual Schema → Logical Schema → Physical Schema

ER:
- Entities,
- Relationships,
- Attributes

Relations:
- tuples,
- attributes
- domain

≈ Tables/columns/rows

File organisation:
- File types
- Index structures

Relational Data Model Concepts

The relational data model is the most widely used data model for database systems.

The *relational data model* describes the world as

- a **collection** of inter-connected ***relations***

Goal of relational model:

- a simple, general data modelling formalism
- which maps easily to file structures (i.e. implementable)

Relational model has **two styles** of terminology:

- mathematical: relation, tuple, attribute, ...
- data-oriented: table, record, field/column, ...

Warning: textbooks alternate between the two; treat them as synonyms

Example relation: bank accounts – as a table …

Relation, Table

Attributes, Columns, Fields

Account

branchName	accountNo	balance
Downtown	A–101	500
Mianus	A–215	700
Perryridge	A–102	400
Round Hill	A–305	350
Brighton	A–201	900
Redwood	A–222	700
Brighton	A–217	750

Tuples, Rows, Records

- When a relation is thought of as a table of values, each row in the table represents a collection of related data values
- The table name and column names are used to help to interpret the meaning of the values in each row.

Relational Data Model Concepts

The relational model has one structuring mechanism ...

- **a *relation* corresponds to a mathematical "relation"**
- *a relation can also be viewed as a "table"*

Each *relation* (denoted *R,S,T,...*) has:

- a *name* (unique within a given database)
- a set of *attributes* (which can be viewed as column headings)

Each *attribute* (denoted *A,B,...* or $a_1,a_2,...$) has:

- a *name* (unique within a given relation)
- an associated *domain* (set of allowed values)

Relational Model (more formal definitions)

- A relational database consists of a set of “relations”
- A relation is made up of 2 parts:
  1) Schema : specifies name of relation, plus name and type of each attribute.
     - E.g. Students (name: string, sid: string, age: integer, gpa: real)
  2) Instance : a set of tuples (≈ table rows) containing a list of values for each attribute
     - #tuples = cardinality, #attributes = degree / arity

Relational Model (more formal definitions)

A relation schema R is formally denoted by:

$$R(A_1, A_2, A_3 \ldots, A_n)$$

where A_i denotes an attribute in R.

e.g., STUDENT(Name, Sid, Age, GPA)

Domain refers to the legal type and range of value of an attribute, denoted by $dom(A_i)$

- e.g., Attribute Age — Domain: [0-100]
- e.g., Attribute EmpName — Domain: 50 alphabetic chars
- e.g., Attribute Salary — Domain: non-negative integer
- Domain can also specify format of the data (e.g., (ddd)dd—dddd))

Sometimes R can be written as:

$$R(A_1{:}D_1, A_2{:}D_2, \ldots A_n{:}D_n)$$

where A_i denotes an attribute in R, D_i denotes the domain of A_i

e.g., STUDENT(Name: string, Sid: string, Age: integer, GPA: real)

relations, tuples and Cartesian product

Say … we have a relation $R(A_1, A_2, A_3 ..., A_n)$. A tuple t of $R(A_1, A_2, A_3 ..., A_n)$ is an "ordered" list of values $t = <v_1, v_2, v_3 ..., v_n>$ where each v_i is an element of $dom(A_i)$.

- e.g., t_1 = <Downtown, A-101, 500>, t_2 = <Mianus, A-205, 700>, ...

Relation instance r(R) is a set of n tuples in R

$$\mathbf{r(R) = \{t_1, t_2, t_3 ..., t_n\}}$$

Also, a relation instance r(R) is a mathematical relation of degree n on the domains $dom(A_1)$, $dom(A_2)$, ... , $dom(A_n)$, which is a subset of the Cartesian product (denoted by $\times$) of the domains that define R:

$$\mathbf{r(R) \subseteq (dom(A_1) \times dom(A_2) \times ... \times (dom(A_n))}$$

- The Cartesian product specifies all possible combinations of values from the underlying domains.
- A relation can take any subset of the possible combinations of values

relations, tuples and Cartesian product

Given the following two sets of values (representing domains):

- Name = { John, Tom }
- Grade = { A, B, C }

The Cartesian product of Name x Grade (i.e., 6 possible tuples)

Name x Grade = { <John, A> ... }

A relation StudentGrade (Name, Grade) can be defined as any subset of the above:

$\text{StudentGrade}_1 \subseteq$ Name x Grade = { ??}

$\text{StudentGrade}_2 \subseteq$ Name x Grade = { ?? }

- i.e., two 'tables' ...

Consider relation R with attributes $a_1, a_2, \ldots a_n$

- *Relation schema* of R : $\boldsymbol{R}(a_1{:}D_1, a_2{:}D_2, \ldots a_n{:}D_n)$
- *Tuple* of R : an element of $D_1 \times D_2 \times \ldots \times D_n$ (i.e. list of values)
- *Instance* of R : subset of $D_1 \times D_2 \times \ldots \times D_n$ (i.e. set of tuples)

Domain declaration:
Name=String(30), DollarPrice=Decimal (10,2)

Relation schema:
Product(Prodname: Name, Price: DollarPrice, Category: Name, Manufacturer: Name)

Instance:

ProdName	Price	Category	Manufacturer
Gizmo	19.99	Gadgets	GizmoWorks
Power gizmo	29.99	Gadgets	GizmoWorks
SingleTouch	149.99	Photography	Canon
MultiTouch	203.99	Household	Hitachi

A relation = schema + instance

Relation Name

Attributes

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Tuples

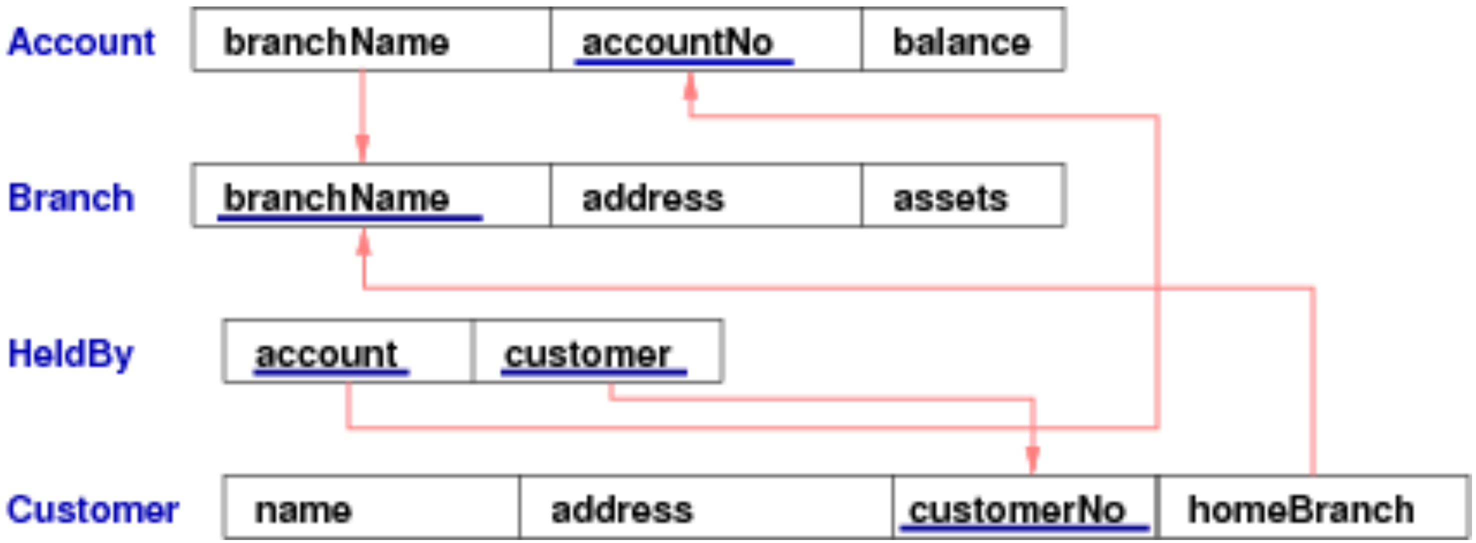
Analogy with programming languages:

schema = type
instance = value

Important distinction:

- Relation Schema = stable over long periods of time
- Relation Instance = changes constantly, as data is inserted/updated/deleted

Example Database Schema



Example Database (Instance)

Account

branchName	accountNo	balance
Downtown	A–101	500
Mianus	A–215	700
Perryridge	A–102	400
Round Hill	A–305	350
Brighton	A–201	900
Redwood	A–222	700

Branch

branchName	address	assets
Downtown	Brooklyn	9000000
Redwood	Palo Alto	2100000
Perryridge	Horseneck	1700000
Mianus	Horseneck	400000
Round Hill	Horseneck	8000000
North Town	Rye	3700000
Brighton	Brooklyn	7100000

Customer

name	address	customerNo	homeBranch
Smith	Rye	1234567	Mianus
Jones	Palo Alto	9876543	Redwood
Smith	Brooklyn	1313131	Downtown
Curry	Rye	1111111	Mianus

Depositor

account	customer
A–101	1313131
A–215	1111111
A–102	1313131
A–305	1234567
A–201	9876543
A–222	1111111
A–102	1234567

Attribute Values

Attribute values
- are atomic (no composite, multi-values)
- have a known domain
- can sometimes be “*null*”

Three meanings of null values
1. not applicable
2. not known
3. absent (not recorded)

Student

studno	name	hons	tutor	year	thesis title
s1	jones	ca	bush	2	*null*
s2	brown	cis	kahn	2	*null*
s3	smith	*null*	goble	2	*null*
s4	bloggs	ca	goble	1	*null*
s5	jones	cs	zobel	1	*null*
s6	peters	ca	kahn	3	*„A CS Survey“*

Database Updates

A database reflects the state of an aspect of the real world:

The world changes ➔ the database has to change

Updates to an instance:

- adding a tuple
- deleting a tuple
- modifying an attribute value of a tuple
- Updates to the data happen very frequently.

Updates to a schema:

- What could be updates to a schema?
- Updates to the schema: relatively rare, rather painful.

Order and Duplication

In tables (inside a database system):

- Order of attributes is fixed
- Order of rows is fixed (the order is determined by the system e.g., insertion order)
- Duplicate rows can exist

In mathematical relations:

- Order of tuples and duplicate tuples do not matter (it is a set!)
- Order of attributes is still fixed

Think back - relations as subsets of cartesian products

Tuple as elements of **String x Int x String x String**

E.g., t = (gizmo, 19.99, gadgets, GizmoWorks)

Relation = subset of **String x Int x String x String**

Order in the tuple is important !

- (gizmo, 19.99, gadgets, GizmoWorks)
- (gizmo, 19.99 , GizmoWorks, gadgets)

No explicit attributes, hidden behind positions …

Alternatively "Relations as Sets of Functions"

Fix the set A of attributes, e.g.

- A = {Name, Price, Category, Manufacturer}

Fix D as the union of the attribute domains, e.g.,

- D = dom(Name) $\cup$ dom(Price) $\cup$ dom(Category) $\cup$ dom(Manufacturer)

A tuple is a function t: A $\rightarrow$ D

E.g.

$$\{\text{Prodname} \longrightarrow \text{gizmo},\ \text{Price} \longrightarrow 19.99,\ \text{Category} \longrightarrow \text{gadgets},\ \text{Manufacturer} \longrightarrow \text{GizmoWorks}\}$$

Order in a tuple is not important,
attribute names are still important.

This is the model underlying SQL

Notation

Given Schema $R(A_1, A_2, A_3 ..., A_n)$ and tuple t that satisfies the schema

Then:

$t[A_i]$ = value of t for attribute A_i

$t[A_i, A_j, A_k]$ = subtuple of t, with values for A_i, A_j, A_k

Example: t = (gizmo, 19.99, gadgets, GizmoWorks)

- t[Price] = 19.99
- t[ProdName, Manufacturer] = (gizmo, GizmoWorks)

Two Definitions of Relations

- Positional tuples, without attribute names
- Tuples as mappings/functions of attributes

In theory and practice, both are used, e.g.,

- SQL: tuples as functions in
  - » SELECT-FROM-WHERE queries
- SQL: positional tuples in Boolean combinations of relations (union, intersection, difference)

You will see these features in the SQL language later.

Why Relational Model?

- Very simple model
- Often a good match for the way we think about our data
- Foundations in logic and set theory
- Abstract model that underlies SQL, the most important language in DBMSs today

Integrity Constraints

- **Ideal**: DB instance reflects the real world
- **In real life**: This is not always the case
- **Goal**: Find out, when DB is out of sync
- **Observation**: Not all mathematically possible instances make sense
- **Idea**:
  - Formulate conditions that hold for all plausible instances
  - Check whether the condition holds after an update
  - Such conditions are called integrity constraints!

Common Types of Integrity Constraints

Domain Constraints

- “No employee is younger than 15 or older than 80”

Referential Integrity (also “foreign key constraints”)

- “Employees can only belong to a department that is mentioned in the Department relation”

Functional Dependencies (FDs)

- *“Employees in the same department have the same boss”*

Superkeys and keys (special case of FDs)

- *“Employees with the same tax code are identical”*

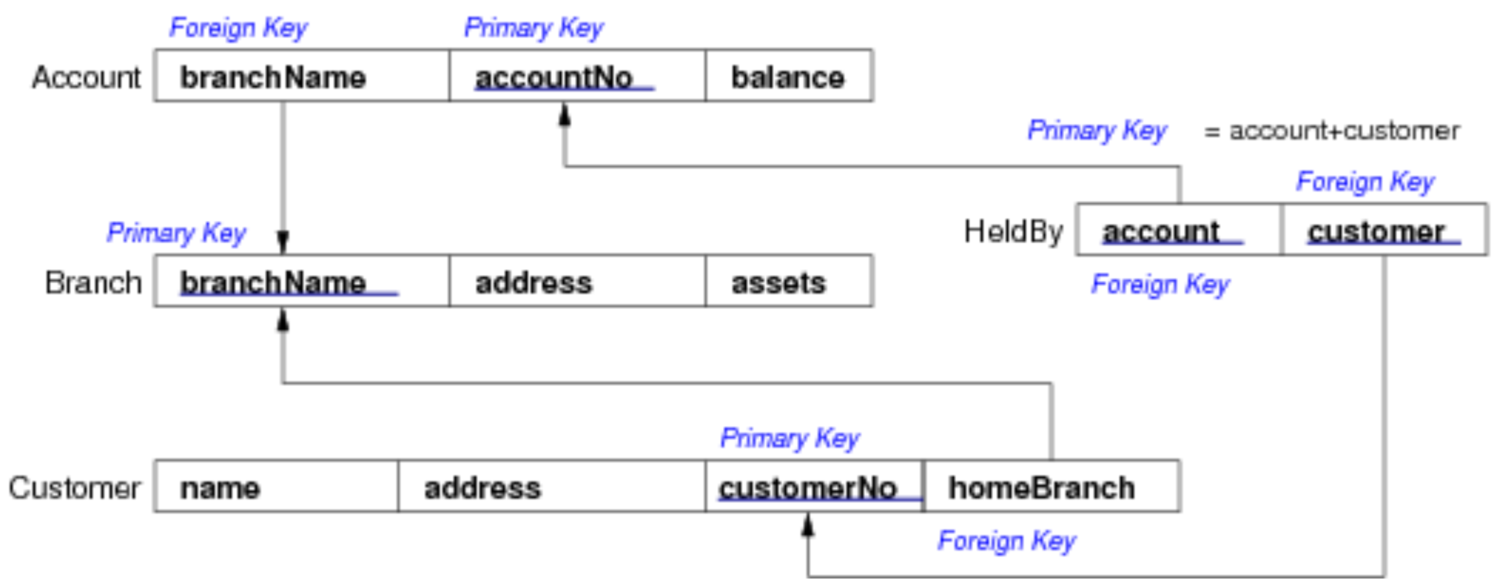
Integrity constraints (ICs) are part of the schema

We allow only instances that satisfy the ICs

Referential Integrity

Referential integrity constraints

- describe references between relations (tables)
- are related to notion of *a foreign key* (FK)



Referential Integrity

A set of attributes F in relation R_1 is a *foreign key* for R_2 if:

- the attributes in F correspond to the primary key of R_2
- the value for F in each tuple of R_1
  - either occurs as a primary key in R_2
  - or is entirely NULL

Foreign keys are critical in relational DBs; they provide ...

- the "glue" that links individual relations (tables)
- the way to assemble query answers from multiple tables
- the relational representation of ER relationships

Relational Database System

A *relational database schema* is

- a set of relation schemas $\{ R_1, \quad R_2, \ \dots \ R_n \}$, and
- a set of integrity constraints

A *relational database instance* is:

- a set of relation instances $\{ r_1(R_1), \quad r_2(R_2), \ \dots \ r_n(R_n) \}$
- where all of the integrity constraints are satisfied

One of the important functions of a relational DBMS:

- ensure that all data in the database satisfies constraints