IN3020&4020 – Database Systems (2024)
Part: Intro & Recap
Lecture 2: Relational Model

---

16 January

Egor V. Kostylev
IFI, University of Oslo
egork@ifi.uio.no



Lectures (reminder)

Physical:

- main part of the course, 2 a week
- an opportunity for a dialogue: interrupt me and ask questions
- I will ask questions sometimes

Lectures will be **recorded**:

- videos will be published (very) soon after the lecture
- do not forget about mandatories:
  you need to follow the course during the term to succeed

Materials to read

1. *Fundamentals of Database Systems* by R. Elmasri and S.B. Navathe (7th ed.): Chapter 5, Sections 5.1&5.2
2. *Database systems – the complete book* by H. Garcia-Molina, J.D. Ullman, and J. Widom (2nd ed.): Section 2.1, 2.2, 2.5 (+ 3.1, 7.1, 7.2)
3. *Foundations of Databases* by S. Abiteboul, R. Hull, and V. Vianu, Chapter 3

   `http://webdam.inria.fr/Alice/`
4. IN2090 – Databaser og datamodellering (Leif Harald Karlsen), Lecture 3

   `https://www.uio.no/studier/emner/matnat/ifi/IN2090/h21/timeplan/`
5. Wikipedia, etc.

There are slight variations (we will discuss)

Lecture plan

1. History of data models
2. Classic relational model (a version of)
3. Integrity constraints (keys and foreign keys)

1. History of data models

Data models

Database: collection of related data—that is, known facts with implicit meaning—organised in some way

Data model: the conceptual representation of data
usually, abstract representation

- concentrates on details important to usual database users
- hides the storage and implementation details that are not important to these users

Brief history (look them up)

Early days (1960's) of digital data management:

- Mainframe period, 1 'big' machine for (that-time) 1M$
- 512 Kbyte RAM, 50 Mbyte disk
- Other I/O-equipment magnetic & paper tape, punched-card reader and teletype (thus, unix has funny abbreviations such as `tty` for the «terminal»)

- hierarchical and network data models
- ad-hoc own query languages tailored for specific applications, largely procedural (i.e., not declarative)

Hierarchical database (actually, data) model

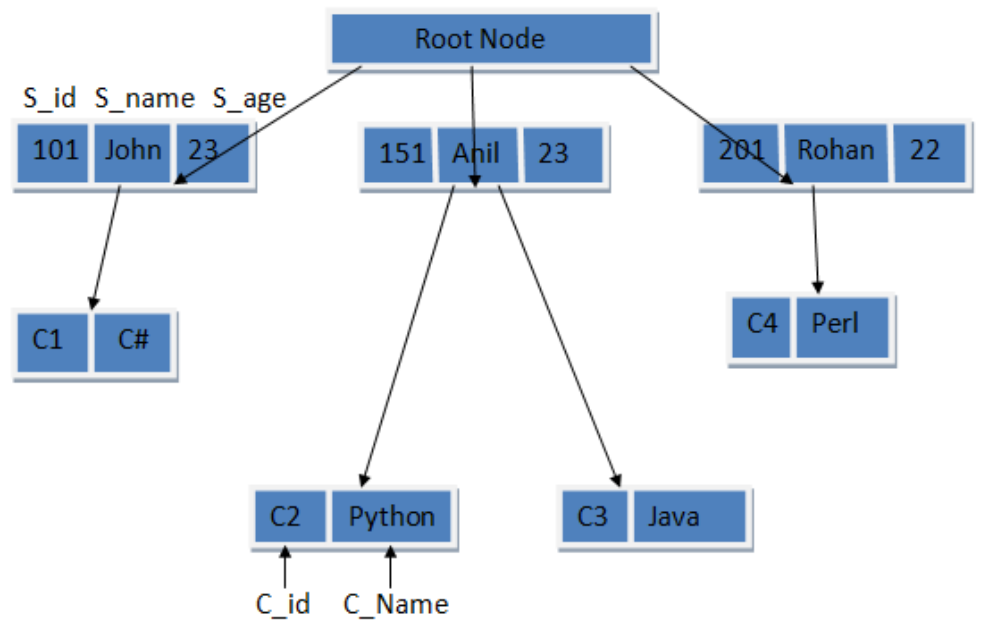
Underlies IBM's Information Management System (IMS) DBMS

Used in bill of materials for Apollo space program

Data is represented in a tree-like structure

Edges are parent-child relationships

Own query language (called DL/I, tightly integrated into COBOL)



Network database model

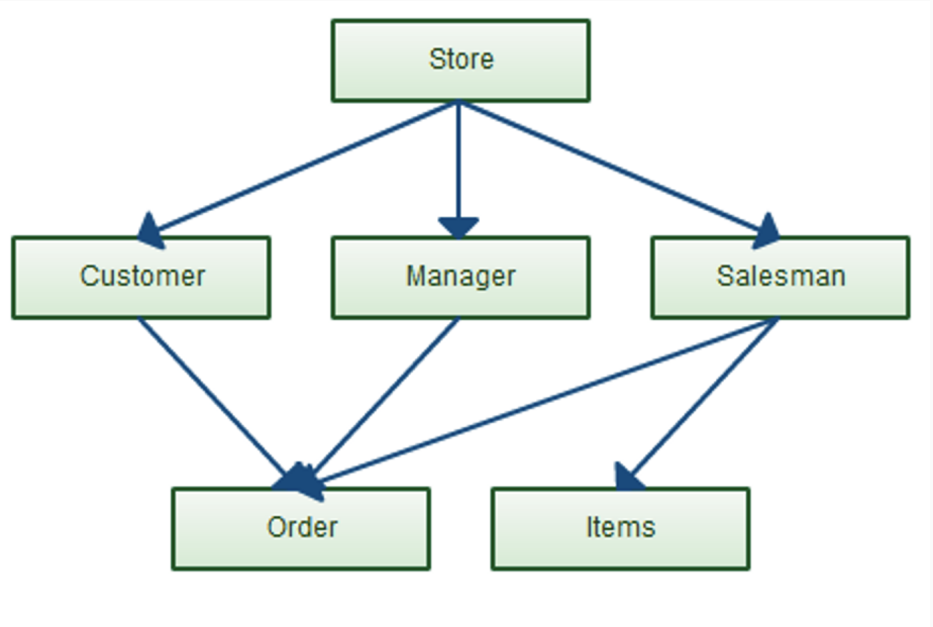
Underlies General Electric's Integrated Data Store (IDS) DBMS

Used by BT Customer Service System (and other projects)

Generalised hierarchical model: several parents of a child are allowed

Performance-oriented with low level of abstraction

Own (difficult to learn) query language



Codd's Relational Model

E. F. Codd, 1970: *A relational model of data for large shared data banks* (Commun. ACM 13, 6):

> *"Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed."*

What did Codd mean?

User should not see the internal representation (files, pointers):

- truly abstract data model

Relations as an abstract data structure:

- simple universal data model

Has a declarative query language of Relational Algebra expressions:

- has formal theoretical foundations
- simple and concise
- allows for efficient implementations: a lot of space for optimisation

Since then 'databases' usually mean 'relational databases':

- SQL is based on (variants of) relational model and algebra

Before we proceed to the details...

Since 2000 back to roots:

- XML is essentially a hierarchical model
- Knowledge Graphs (e.g., RDF) are essentially a network model

Question: Why they lost in the 70s, but regain popularity now?

Answer: Just *fancy*.

Before we proceed to the details...

Since 2000 back to roots:

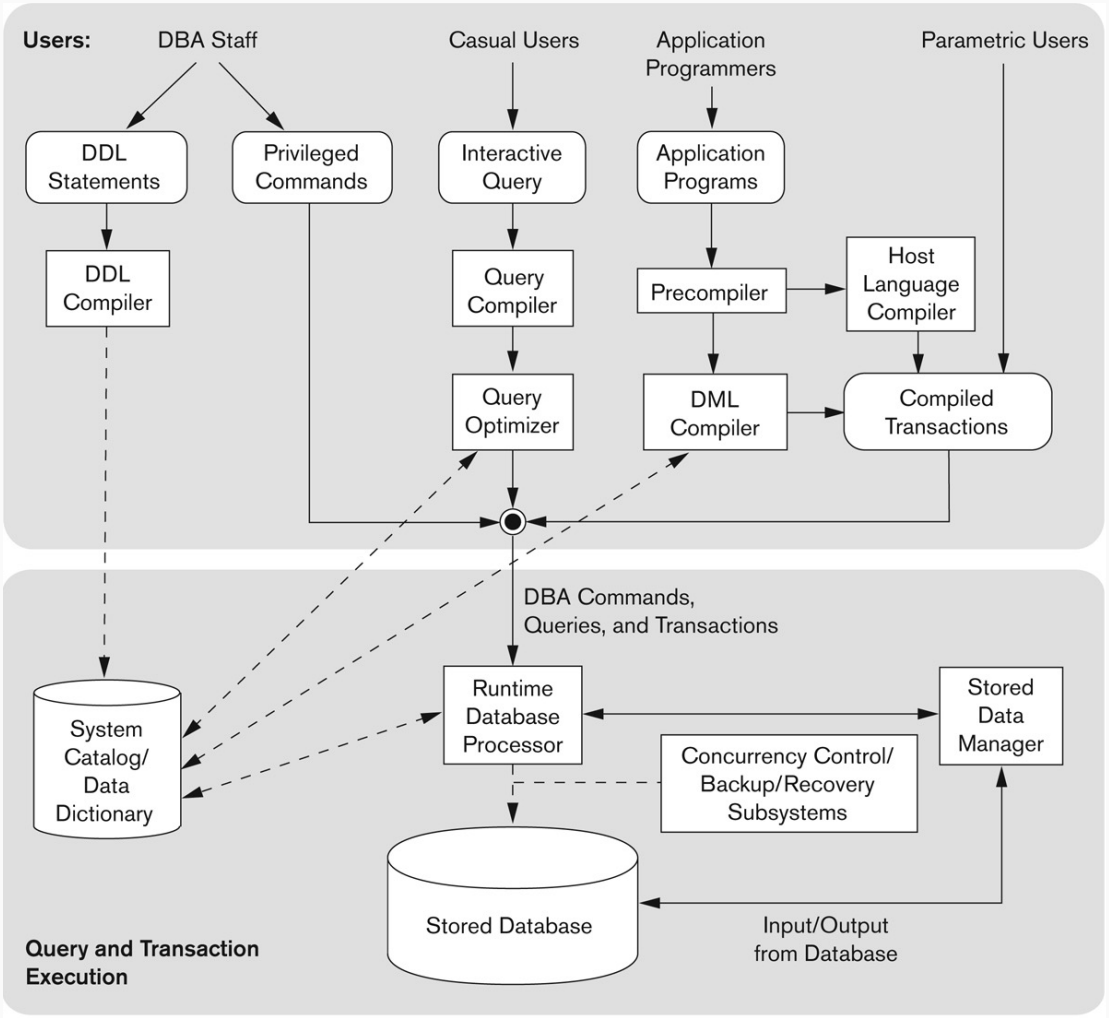
- XML is essentially a hierarchical model
- Knowledge Graphs (e.g., RDF) are essentially a network model

Question: Why they lost in the 70s, but regain popularity now?

Real answer: we are on a different level:

- we understand much better what different data models (and query languages) are good for
- these models have universal and declarative query languages (XQuery, SPARQL, Cypher, etc.)
- the languages are much more efficient than the ones of 60's in all components

The place of the relation model in DBMS architecture



Abstract model: this is what all the users interact with, but *not* how data is stored in reality

2. Classical relational model

Relations: intuition and terminology

A relation can be seen as a table of values. Something like this:

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Relations: intuition and terminology

A relation can be seen as a table of values. Or this:

Tutorials

ID	site	tutorial	topic
1	w3schools	SQL_2003STD	Database
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Database
5	w3resource	MySQL	Database

Relations: intuition and terminology

A relation can be seen as a table of values. Something like this:

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

- **Tutorials**: relation name
- ID, site, tutorial, topic: attribute names (or just attributes),
  collectively: signature
- Relation name + signature: relation schema (or just relation)
- Rows: tuples, records, or instances
  collectively: relation state
- Elements of tuples: attribute values (or just attributes as well)

Relations: properties

Relation

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

- Each attribute has a domain (i.e., a set of potential values):
  integers, strings, etc. (often not written explicitly)
- Different attributes may have the same domain
- Attribute values are from the associated domain

Relations: properties

Relation

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

– Tuples are unordered and unique (i.e., not identical)

Relations: properties

Same relation

Tutorials:

ID	*site*	*tutorial*	*topic*
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
1	w3schools	SQL_2003STD	Databases
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

– Tuples are unordered and unique (i.e., not identical)

Relations: properties

Not a relation(!)

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
1	w3schools	SQL_2003STD	Databases
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases

– Tuples are unordered and unique (i.e., not identical)

Relations: properties

Also same relation

Tutorials:

site	*ID*	*tutorial*	*topic*
w3schools	1	SQL_2003STD	Databases
w3schools	2	HTML_5	WebDev
w3schools	3	CSS_3	WebDev
w3resource	4	SQL_2003STD	Databases
w3resource	5	MySQL	Databases

- Tuples are unordered and unique (i.e., not identical)
- Attributes are also unordered and unique

  (not so essential, but convenient for us;

  may be different in different formalisations)

Formally: relation schema

Domain: a set of values

- may be finite (e.g., countries $\{Norway, UK, India, \ldots\}$) or infinite (e.g., integers)
- atomic—that is, values have no essential internal structure (e.g., not sets or tuples themselves)
- usually, has a corresponding datatype in the query language

Attribute (i.e., 'column name'): a string with an associated domain

- domain of attribute A is written $dom(A)$
- for example, $dom(ID)$ is natural numbers

Relation schema: a relation name R (just string) and several attributes $A_1, A_2, \ldots, A_n$

- written as $R[A_1, A_2, \ldots, A_n]$ (or $R[A_2, A_1, \ldots, A_n]$, order is irrelevant)
- n is called the arity of R (question: can the arity be 0?)
- for example $Tutorials[ID, site, tutorial, topic]$ has arity 4

Formally: tuples

Tuple (or record, row, instance) over schema $R[A_1, \ldots, A_n]$: assignment of attribute values $v_1 \in dom(A_1), \ldots, v_n \in dom(A_n)$ to attributes $A_1, \ldots, A_n$:

- formally, can be written $v_1 \to A_1, \ldots, v_n \to A_n$
- we usually write $R\langle v_1, \ldots, v_n \rangle$ or even $\langle v_1, \ldots, v_n \rangle$
- example tuples over $Tutorials[ID, site, tutorial, topic]$:

$$\langle 1, w3schools, SQL_2003STD, Databases \rangle$$
$$\langle 2, w3schools, HTML_5, WebDev \rangle$$

Convenient notation (for later): if t is a tuple over $R[A_1, \ldots, A_n]$ and $\{A'_1, \ldots, A'_m\} \subseteq \{A_1, \ldots, A_n\}$ then $t[A'_1, \ldots, A'_m]$ is the *projection* of t to $\{A'_1, \ldots, A'_m\}$:

$$\langle 2, w3schools, HTML_5, WebDev \rangle[ID, topic] = \langle 2, WebDev \rangle$$

When $m = 1$, we can write $t.A$ instead of $t[A]$

Formally: relations and databases

Relation state over relation schema $R[A_1, \ldots, A_n]$: a finite set I of tuples over $R[A_1, \ldots, A_n]$

- 'set' means no order of elements, no repetitions
- may be written $I \subseteq dom(A_1) \times \cdots \times dom(A_n)$
  (but formally not correct for us due to the order mismatch)

Relation: a relation schema + relation state

Database: finite set of relations (i.e., essentially, tables) with different names

Variants of the relational model

Many technical non-essential variations can be found:

- unordered vs. ordered signature (i.e., attributes, columns)
- named perspective (for attributes) vs. unnamed:
  - if the signature is ordered, names of attributes are not important and may be ignored (not the case for tables!)
  - moreover, attribute domains may be abstracted away (i.e., only one big domain is considered)
  - relation schema is just name and arity (written as R^2 or $R(\cdot, \cdot)$)
  - inspired by formal logic (which is foundations for databases): tuples are logical facts $R(a, b)$

Other details are (very) essential:

- no order, no repetitions of tuples in a relation, etc. (see below)

Semantics (the meaning) of databases

Intuitive: tuples represent certain facts that correspond to a real-world relationships

$$Tutorials\langle 1, w3schools, SQL_2003STD, Databases\rangle$$

means that there is a SQL_2003STD tutorial on Databases topic on w3schools site with ID 1 (of course, the fact may not hold, but the database creators claim it)

Closed world assumption is assumed in the relational databases: all facts not mentioned in the database, are claimed not to hold.
No tuple $Tutorials\langle 1, w3schools, Mars, Astronomy\rangle$—no tutorial.

Why to bother? Allows to intuitively answer many queries (e.g., queries with `NOT EXISTS`)

Different in other database models: RDF knowledge graphs, etc.

3. Integrity constraints

Key integrity constraints: intuition

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Attribute *ID* is special in this relation: has no repeated values

- called a (candidate) key (of the relation)

Question: any other keys in this relation?

Key integrity constraints: intuition

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Attribute *ID* is special in this relation: has no repeated values

- called a (candidate) key (of the relation)

Not a coincidence: we *want* it to be a real ID

- can require *ID* in schema *Tutorials*[*ID*, *site*, *tutorial*, *topic*] so that each relation over this schema has *ID* as key (i.e., enforce key constraint)
- use `PRIMARY KEY` or `UNIQUE` declaration in SQL

Both notions generalise to attribute sets

Candidate and super keys formally

Super key of a relation (schema + state): a subset X of the attributes of the relation such that if t_1 and t_2 are two different tuples then $t_1[X] \neq t_2[X]$

- observation: the relation signature is always a super key

Candidate key of a relation: a minimal super key

- that is, removing any attribute causes the remaining attributes to no longer be a super key

Candidate and super key examples

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Super keys: $\{ID\}$, $\{ID, site\}$, $\{site, tutorial\}$, $\{ID, site, tutorial, topic\}$, etc.

Not super keys: $\{site\}$, $\{tutorial, topic\}$, etc.

Candidate keys: $\{ID\}$, $\{site, tutorial\}$ Not unique!

Not candidate keys: $\{ID, site\}$, $\{tutorial, topic\}$, etc.

Key constraints formally

Extend our relational model notions:

Relation schema with a key constraint: a relation name R with attributes $A_1, A_2, \ldots, A_n$, and a key subset of $A_1, A_2, \ldots, A_n$

Relation over $R[A_1, A_2, \ldots, A_n]$ satisfies the key constraint if the key subset is a super key

A relation schema may have one or several key constraints declared; one is called primary key constraint (or just primary key)

- written as $R[\underline{A_1}, A_2, \ldots, \underline{A_n}]$, where the underlined attributes are the key subset (double underline is also used)
- for example $\mathit{Tutorials}[\underline{\mathit{ID}}, \mathit{site}, \mathit{tutorial}, \mathit{topic}]$

Question: Do you remember how to create a primary key in SQL? Other (candidate) key constraint?

Primary key examples

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Relation over $Tutorials[\underline{ID}, site, tutorial, topic]$ ($\{ID\}$ is a super key)

Not relation over $Tutorials[ID, \underline{site}, tutorial, topic]$ ($\{site\}$ is not a super key)

Relation over $Tutorials[\underline{ID, site, tutorial, topic}]$

Observation: since all attributes are always a super key, we can assume that if a primary key for a schema is not declared, then it is silently all the attributes

Foreign key (referential) integrity constraints intuition

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Students:

name	*tutorialID*
Alice	1
Alice	2
Bob	2

Values of attribute *tutorialID* are not arbitrary in relation *Students*: all values are assumed to be mentioned in *ID* of *Tutorials*

- can declare *tutorialID* to be a foreign key in *Students*[*name*, *tutorialID*] referring to the primary key *ID* of *Tutorials*[*ID*, *site*, *tutorial*, *topic*] so that the two relations in the database are required to be consistent in this sense

This notion generalises to attribute sets

Foreign key (referential) constraints formally

Sorry, a bit technical; but the idea is simple

Attributes $B'_1, \ldots, B'_k$ of relation over $S[B_1, \ldots, B_m]$ are a foreign key referring to a candidate key $A'_1, \ldots, A'_k$ of relation over $R[A_1, \ldots, A_m]$ if every tuple t_1 over $S[B_1, \ldots, B_m]$ refers to a tuple t_2 over $R[A_1, \ldots, A_n]$—that is, $t_1[B'_1, \ldots, B'_k] = t_2[A'_1, \ldots, A'_k]$

Foreign key constraint is a pair of sets of attributes $B'_1, \ldots, B'_k$ and $A'_1, \ldots, A'_k$ in relation schemas as above such that $A'_1, \ldots, A'_k$ is declared as key and $dom(B'_1) = dom(A'_1), \ldots, dom(B'_k) = dom(A'_k)$

Relations over $S[B_1, \ldots, B_m]$, $R[A_1, \ldots, A_n]$ satisfy this constraint if $B'_1, \ldots, B'_k$ is a foreign key referring to $A'_1, \ldots, A'_k$ for relations

A database (set of relations) can come with foreign keys, which should be satisfied

Foreign key (referential) constraints examples

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Students:

name	*tutorialID*
Alice	1
Alice	2
Bob	2

Satisfy the foreign key constraint where *tutorialID* of *Students* refers to *ID* of *Tutorials*

Foreign key (referential) constraints examples

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Students:

name	*tutorialID*
Alice	1
Alice	2
Bob	2
Bob	25

Do not satisfy the foreign key *tutorialID* of *Students* referring to *ID* of *Tutorials*

Foreign keys observations

Sometimes, foreign keys are allowed to refer only to primary keys

- see our example

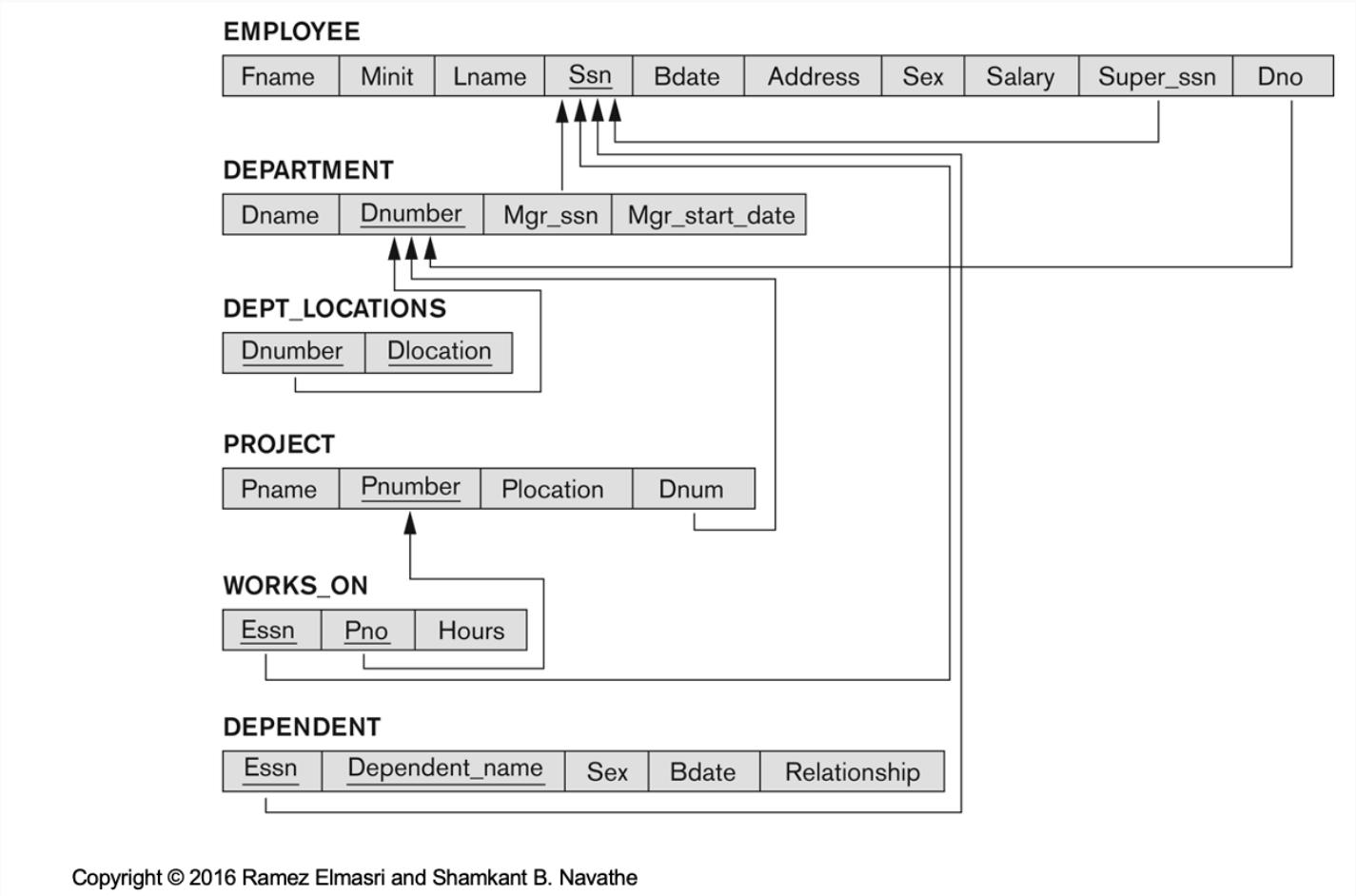
Corresponding attributes do not need to have the same name

- see our example

It is allowed to refer to the same table

- imagine a table of employees with an *ID* and *managerID* attributes (the big boss has to manage himself/herself)

Complex schema example



Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Foreign keys are convenient to draw with arrows

In this figure, all foreign keys consist of one attribute (simple case)

Who needs to care about keys?

Question: Does a casual database user who writes a query over a database needs to know about keys?

Answer: Not really

Question: Does a database manager who inputs data into a database needs to know about keys?

Answer: Yes, of course

Question: Does a DBMS programmer who writes a (`SELECT`) query engine needs to bother about keys?

Answer: Yes, because keys are the knowledge that may help a lot to evaluate queries efficiently, and we will see how

What we have learned

1. History of data models
2. Classic relational model (a version of)
3. Integrity constraints (keys and foreign keys)

Next time: SQL introduction (SQL data model, SQL sublanguages)