

Relational Model

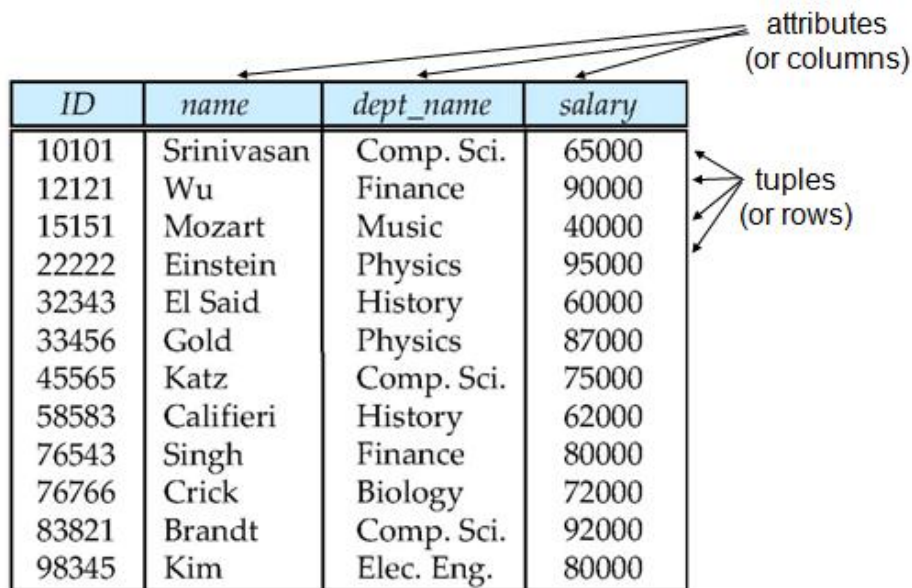
The relational model uses a collection of tables to represent both data and relationships among those data. Each table has multiple columns and each column has a unique name.

Characteristics

1. The primary data model for commercial data-processing applications.
2. It attained its primary position because of its simplicity, which eases the job of the programmer.
3. The relational model describes data at the logical and view levels, abstracting away low-level details of data storage.
4. The relational model is at a lower level than the E-R model. Database designs are often carried out in the E-R model and then translated to the relational model.

2.1 Structure of Relational Databases

A relational database consists of a collection of **tables**, each of which is assigned a unique name. In general, a row in a table represents a *relationship* among a set of values. Since a table is a collection of such relationships, there is close correspondence between the concept of *table* and the mathematical concept of relation, from which *relational* data model takes its name.



The diagram shows a table with four columns: ID, name, dept_name, and salary. The first four rows are highlighted in light blue. Arrows point from the text 'attributes (or columns)' to the column headers. Another set of arrows points from the text 'tuples (or rows)' to the first four rows of data.

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 2.1 The *instructor* relation

Relation and tuple: In relational model these are synonymous to **table** and **row**.

Attribute: In relational model we refer column headers as **attributes**.

Relation instance: It refers to a specific instance of a relation, i.e., containing a specific set of rows. Example: *instructor* instance above.

Domain: For each attribute, there is a set of permitted values, called **domain** (D) of that attribute. Thus, the domain of *salary* attribute of *instructor* relation is the set of all possible salary values. In general, a table of n attributes must be a subset of

$D_1 \times D_2 \times \dots \times D_{n-1} \times D_n$ (Cartesian product of a list of domains)

Atomic domain: A domain is atomic if elements of the domain are considered to be indivisible parts. Example: set of integers: 23, 45, 5, 78; full name of employees etc.

Non-atomic domain: If elements of a domain can be divided into several parts, the domain is called non-atomic domain. Example: set of all sets of integers: {23, 12, 4; 5, 65, 4; 34, 23, 98}, employee-id: HR001, IT005, a set of phone numbers.

- The important issue is not what the domain itself is, but rather how we use domain elements in our database. Example: *phone_number* +880-2-9670734.

Null value: One domain value that is a member of any possible domain is the null value, which signifies that the value is unknown or does not exist. Example: *phone_number*, *apartment_no*, *street_no* etc.

- Null values cause a number of difficulties while accessing or updating the database, thus should be eliminated if at all possible.
- The order in which tuples appear in a relation is irrelevant, since a relation is a set of tuples – sorted or unsorted does not matter.
- If two instances of the same set of tuples with one sorted on some attribute/attributes and with another unsorted, the two relations are same since both contains the same set of tuples.
- $t \in r$ to denote that tuple t is in relation r.

2.2 Database Schema

Database schema: Logical design of the database.

Database instance: A snapshot of the data in the database at a given instant in time.

Relation schema:

- The concept of relation schema corresponds to the programming-language notion of type definition. Ex. char a, int x
- In general, a relation schema consists of a list of attributes and their corresponding domains. Schema for the *department* relation is *department (dept_name, building, budget)*
- The schema of a relation does not generally change.

Relation instance:

- The concept of relation instance corresponds to the programming-language notion of a value of a variable.
- Like variable, the contents of a relation instance may change with time as the relation is updated. We often simply say “relation” to mean “relation instance”.

University Schema:

- The schema for the *department* relation is *department (dept_name, building, budget)*
- Each course in a university may be offered multiple times, across different semesters, or even within a semester. We need a relation to describe each individual offering, or section, of the class. The schema is *section (course_id, sec_id, semester, year, building, room_number, time_slot_id)*
- We need a relation to describe the association between instructors and the class sections that they teach. The relation schema to describe this association is *teaches (ID, course_id, sec_id, semester, year)*

2.3 Keys

We must have a way to specify how tuples within a given relation is distinguished. It is expressed in terms of their attributes. That is, the value of the attributes of a tuple must be such that they can *uniquely identify* the tuple. In

other words, no two tuples in a relation are allowed to have exactly the same value for all the attributes.

1. Superkey:

Definition: A superkey is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.

- In *instructor* relation – *ID* is a superkey
- In *instructor* relation – *name* is **not** a superkey – several instructors may have same name.
- In *classroom* relation – $\{building, room_number\}$ is a superkey.
- A superkey may contain extraneous attributes. For example, the combination of *ID* and *name* is a superkey for the relation *instructor*. If *K* is a superkey, then so is any superset of *K* i.e. any superset of a superkey is also a superkey.

Formal definition: Let *R* denote the set of attributes in the schema of relation *r*. If we say that a subset *K* of *R* is a *superkey* for *r*, we are restricting consideration to instances of relation *r* in which no two distinct tuples have the same values on all attributes in *K*. That is, if t_1 and t_2 are in *r* and $t_1 \neq t_2$, then $t_1[K] \neq t_2[K]$.

2. Candidate Key:

Definition: The superkey, for which no proper subset is a superkey, is a candidate key. So the minimal superkeys are called candidate keys. It is possible that several distinct sets of attributes could serve as a candidate key.

- In *instructor* relation – both $\{ID\}$ and $\{name, dept_name\}$ are candidate keys. Although the attributes *ID* and *name* together can distinguish *instructor* tuples, their combination, $\{ID, name\}$, does not form a candidate key, since the attribute *ID* alone is a candidate key.
- In *classroom* relation – $\{building, room_number\}$ is also a candidate key.

3. Primary Key:

Definition: The primary key is to denote a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation.

- In *instructor* relation – $\{ID\}$ is a primary key.
- In *classroom* relation – $\{building, room_number\}$ is also a primary key.
- **So, Primary key \subseteq Candidate Key \subseteq Super key.**
- A key (primary, candidate or super) is the property of the entire relation, rather than of the individual tuples. Any two individual tuples in the relation

are prohibited from having the same value on the key attributes at the same time.

- Primary keys must be chosen with care (name Vs SSN in USA, name Vs NID(?) in Bangladesh). The primary key should be chosen such that its attribute values are never, or very rarely, changed.
- It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept_name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined..
- The designation of a key represents a constraint (primary key, unique key) in the real-world enterprise being modeled.

4. Foreign key:

Definition: A relation, say r_1 , may include among its attributes the primary key of another relation r_2 . This attribute is called a foreign key from r_1 , referencing r_2 . The relation r_1 is also called the **referencing relation** of the foreign key dependency and r_2 is called the **referenced relation** of the foreign key.

- The attribute *dept_name* in *instructor* is a foreign key from *instructor*, referencing *department*, since *dept_name* is the primary key of *department*.
- In any database instance, given any tuple, say t_a , from the *instructor* relation, there must be some tuple, say t_b , in the *department* relation such that the value of the *dept_name* attribute of t_a is the same as the value of the primary key, *dept_name*, of t_b .

<u>dept_name</u>	<u>building</u>	<u>budget</u>
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 2.5 The *department* relation

<u>ID</u>	<u>name</u>	<u>dept_name</u>	<u>salary</u>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 2.1 The *instructor* relation

Now consider the *section* and *teaches* relations.

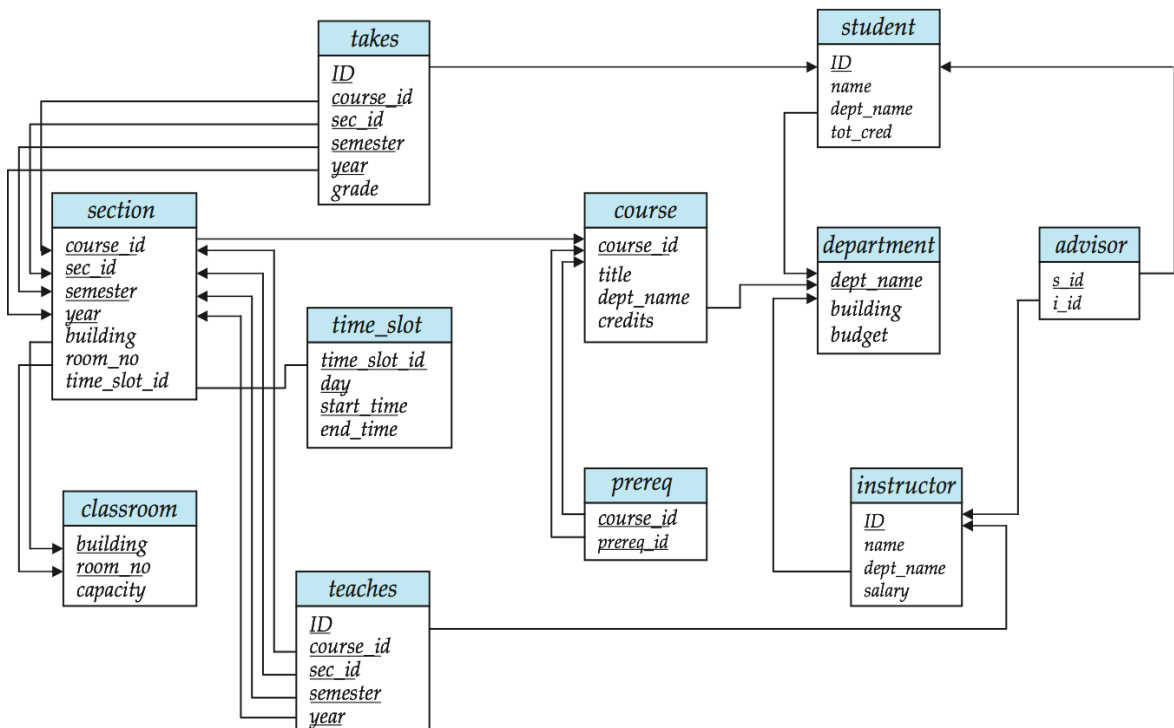
<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>building</i>	<i>room_number</i>	<i>time_slot_id</i>
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 2.6 The *section* relation

- It would be reasonable to require that if a section exists for a course, it must be taught by at least one instructor; however, it could possibly be taught by more than one instructor. To enforce this constraint, we would require that if a particular (*course_id*, *sec_id*, *semester*, *year*) combination appears in *section*, then the same combination must appear in *teaches*.

<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 2.7 The *teaches* relation



- Referential integrity constraints other than foreign key constraints are not shown explicitly in schema diagrams. A different diagrammatic representation called the entity-relationship diagram will let us represent several kinds of constraints (types of relationships – one to one, one to many, many to one and many to many) , including general referential integrity constraints.
- Schema of the university database is listed below:

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Figure 2.9 Schema of the university database.

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure 2.2 The *course* relation

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figure 2.3 The *prereq* relation

2.5 Relational Query Languages

A query language is a language in which user requests information from the database. This is a part of DML (Data Manipulation Language). These languages are usually on a level higher than that of a standard programming language.

Procedural Language: The user instructs the system to perform a sequence of operations on the database to compute the desired result ('what' and 'how'). Example: Relational algebra.

Nonprocedural Language: The user describes the desired information without giving a specific procedure for obtaining that information ('what' without specifying 'how'). Example: Tuple relational calculus, Domain relational calculus.

- Most commercial relational database systems offer a query language that includes elements of both procedural and non-procedural approaches. Example: SQL and PL/SQL

Pure Language: The relational algebra, the tuple relational calculus and the domain relational calculus are called the pure languages. These query languages are terse and formal, lacking the "syntactic sugar" of commercial languages, but they illustrate the fundamental techniques for extracting data from the database.

Fundamental Relational-Algebra Operations

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as a result.

Fundamental operations are:

1. select (unary)
2. project (unary)
3. rename (unary)
4. cartesian product (binary)
5. union (binary)
6. set-difference (binary)

Continue with the slides: 2.10 to 2.18

Here is the end of Introduction to Relational Model