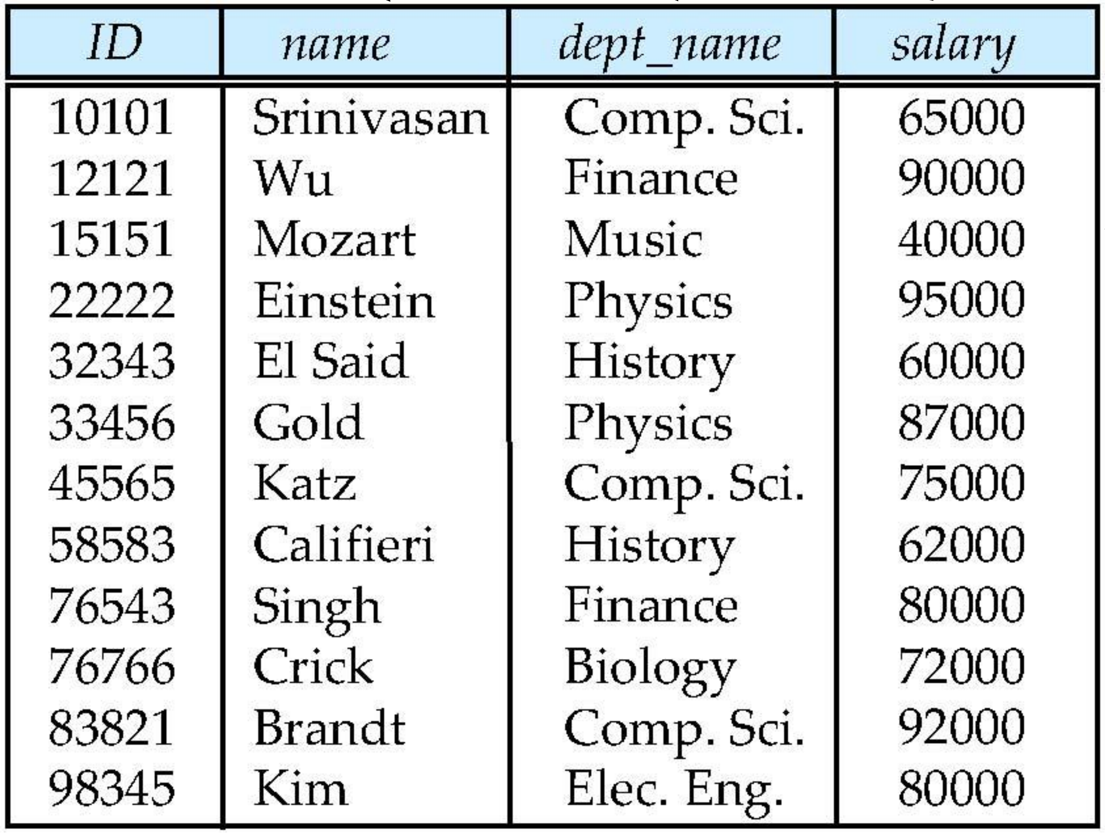
Relational Models

Database System Concepts, 6th Ed.

Example of a Relation

attributes
(or columns)



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

tuples
(or rows)

Attribute Types

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value ***null*** is a member of every domain. Indicated that the value is "unknown"
- The null value causes complications in the definition of many operations

Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are *attributes*

- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  Example:

  instructor = (*ID*, *name*, *dept_name*, *salary*)

- **Formally, given sets $D_1, D_2, \ldots. D_n$ a relation *r* is a subset of**

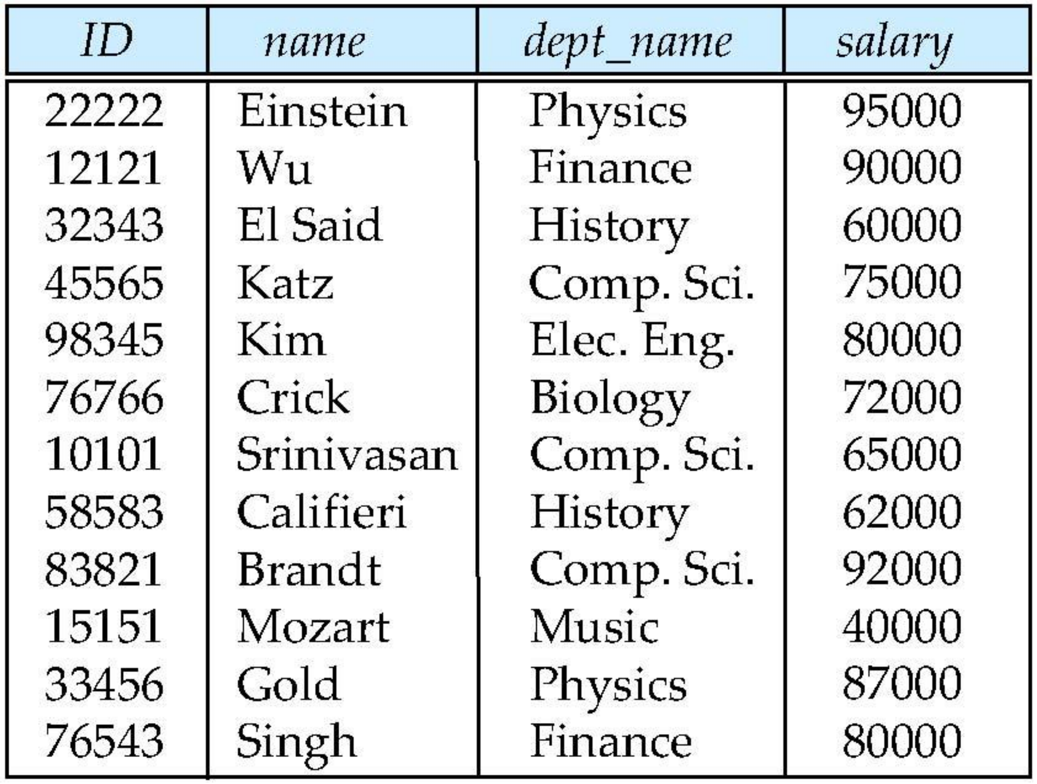
  $$D_1 \times D_2 \times \ldots \times D_n$$

  Thus, a relation is a set of *n*-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

- The current values (**relation instance**) of a relation are specified by a table
- An element ***t*** of ***r*** is a *tuple*, represented by a *row* in a table

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples



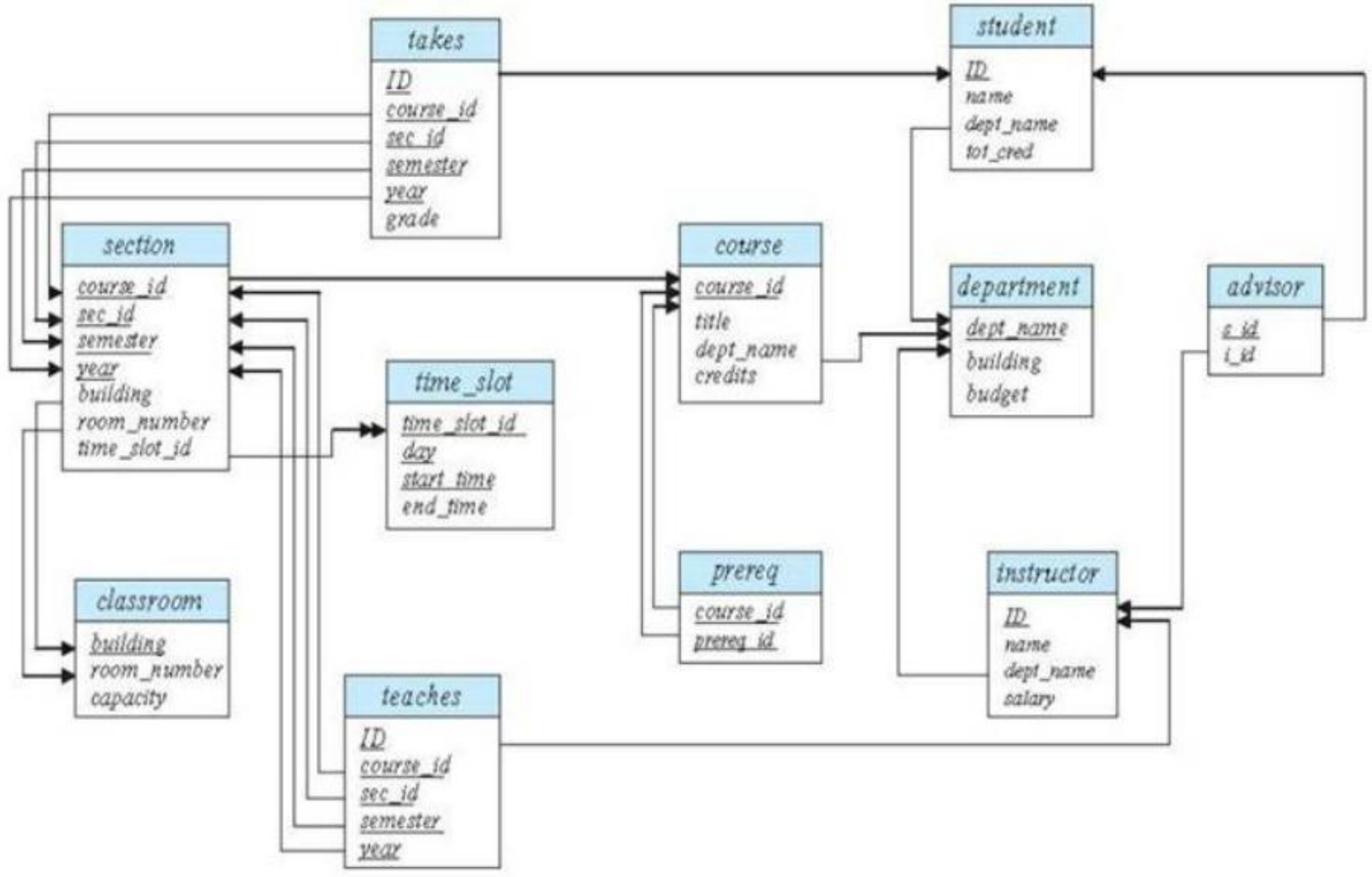
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music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

Keys

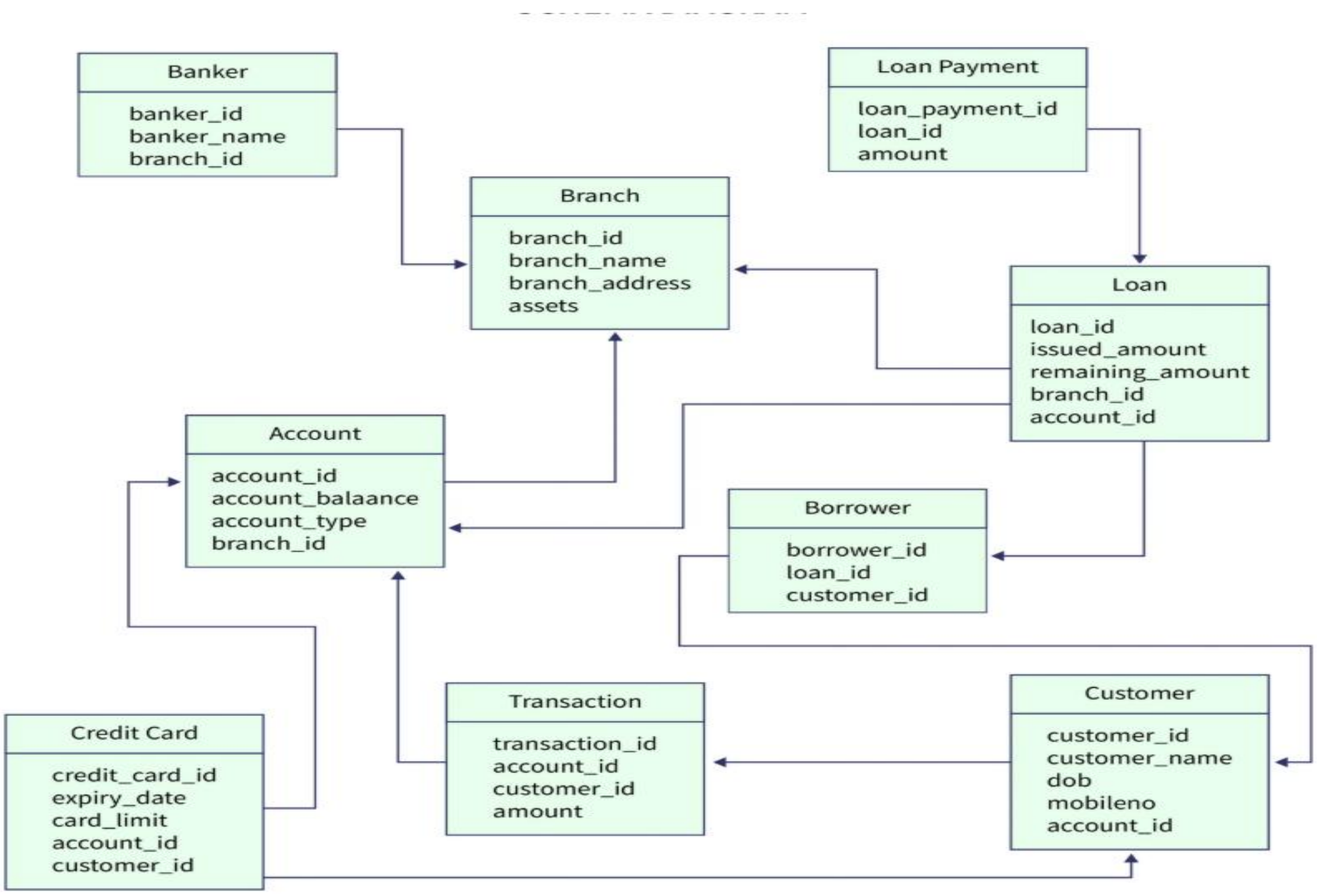
- Let K ⊆ R
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor*.
- Superkey *K* is a **candidate key** if *K* is minimal

  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**. Others are said to be **Alternate key.**
  - {ID} → Primary Key
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - Example – *dept_name* in *instructor* is a foreign key from *instructor* referencing *department*

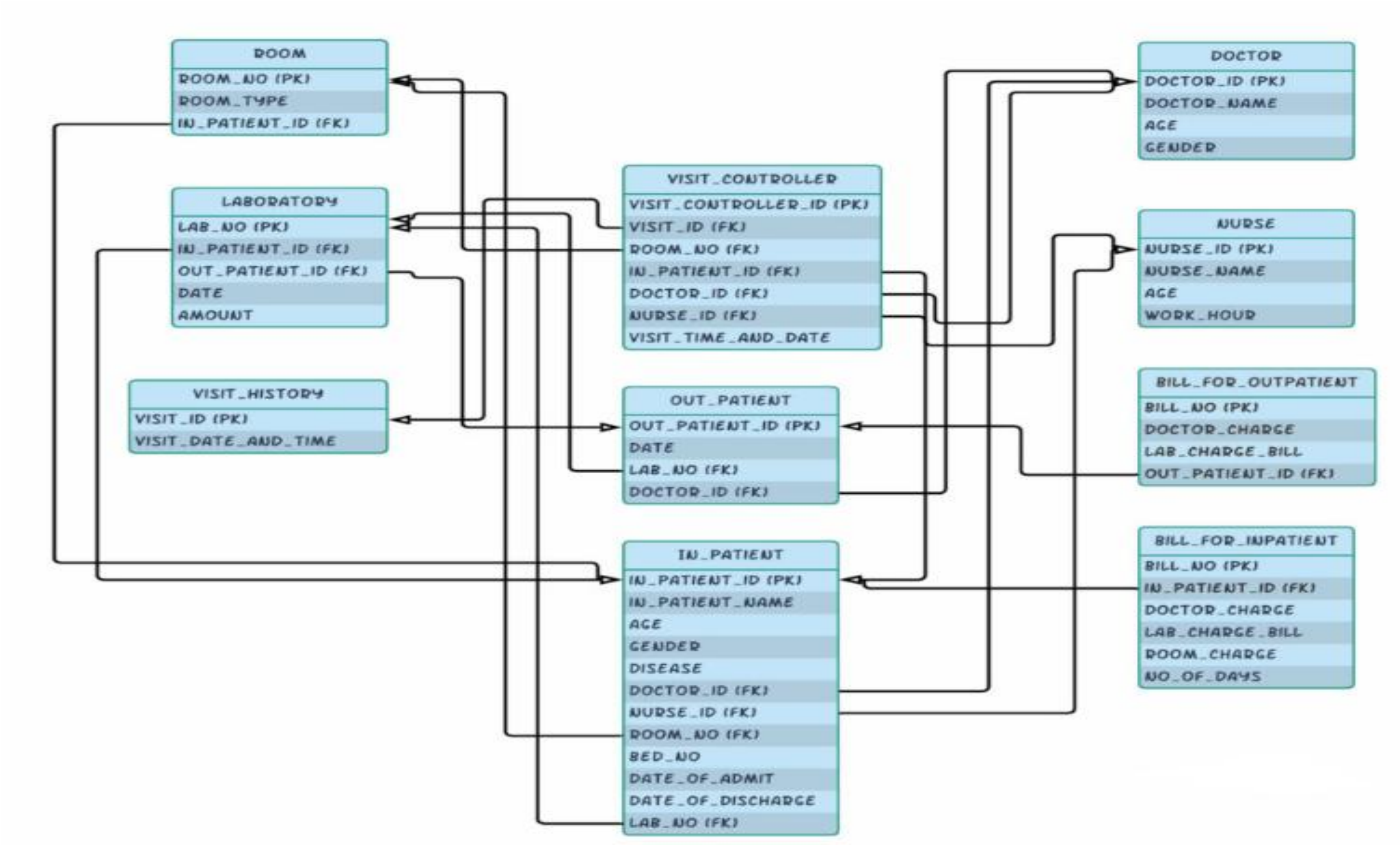
Relational Schema Diagram for University Database



Relational Schema Diagram for Bank Database



Relational Schema Diagram for Hospital Management



Mapping ER Model to Relational Model

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
- Each schema has a number of columns (generally corresponding to attributes), which have unique names.

Representing Entity Sets

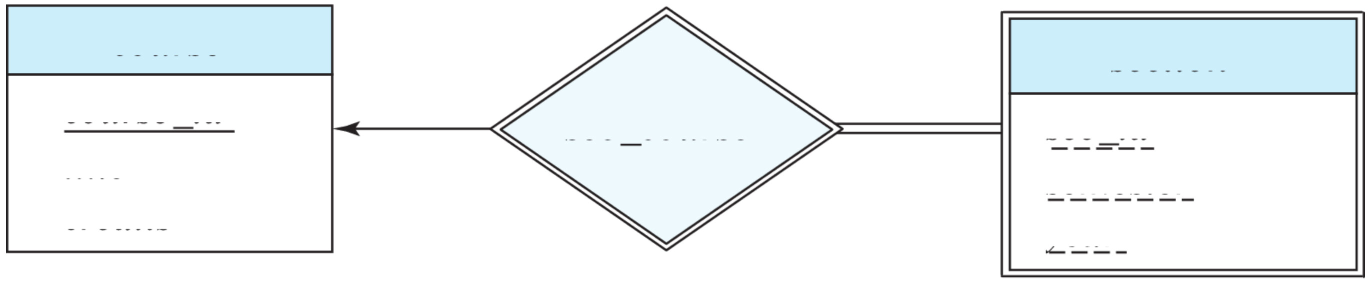
- A strong entity set reduces to a schema with the same attributes

  student(<u>ID</u>, name, tot_cred)

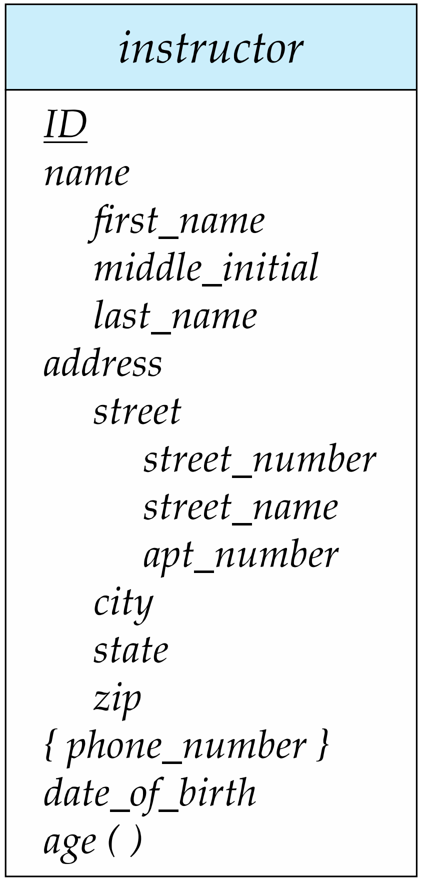
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set

  section (<u>course_id, sec_id, sem, year</u>)

- Example



Representation of Entity Sets with Composite Attributes



- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - Example: given entity set *instructor* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes *name_first_name* and *name_last_name*
    - Prefix omitted if there is no ambiguity (*name_first_name* could be *first_name*)
- Ignoring multivalued attributes, extended instructor schema is
  - *instructor(ID,*
    first_name, middle_initial, last_name,
    street_number, street_name,
    apt_number, city, state, zip_code,
    date_of_birth)

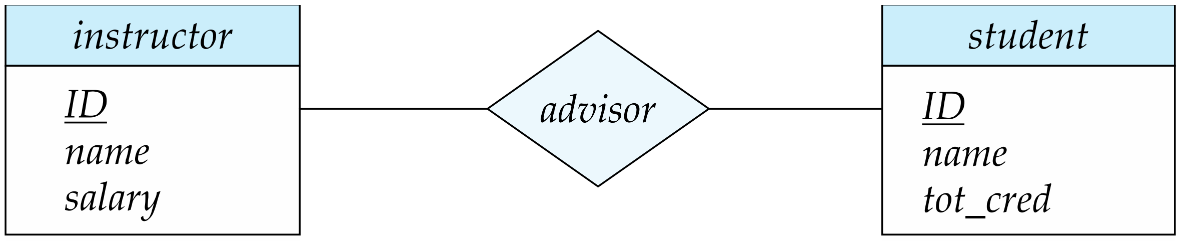
Representation of Entity Sets with Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
- Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
- Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
  inst_phone= (<u>*ID*</u>, <u>*phone_number*</u>)
- Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
  - For example, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
    (22222, 456-7890) and (22222, 123-4567)

Representing Relationship Sets

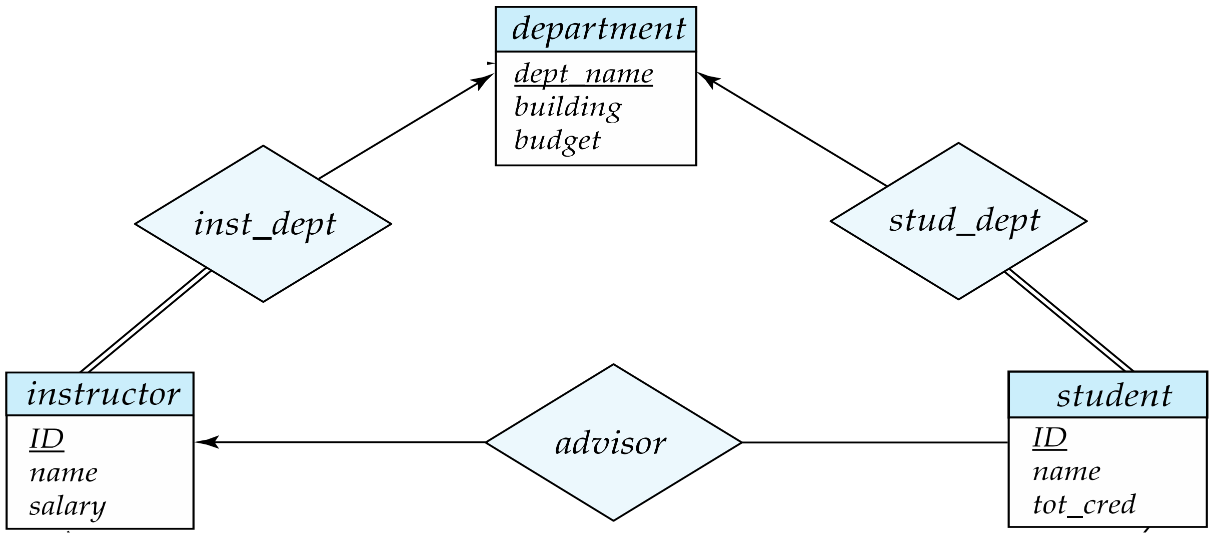
- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set *advisor*

advisor = (*s_id, i_id*)



Redundancy of Schemas

- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side
- Example: Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*
- Example

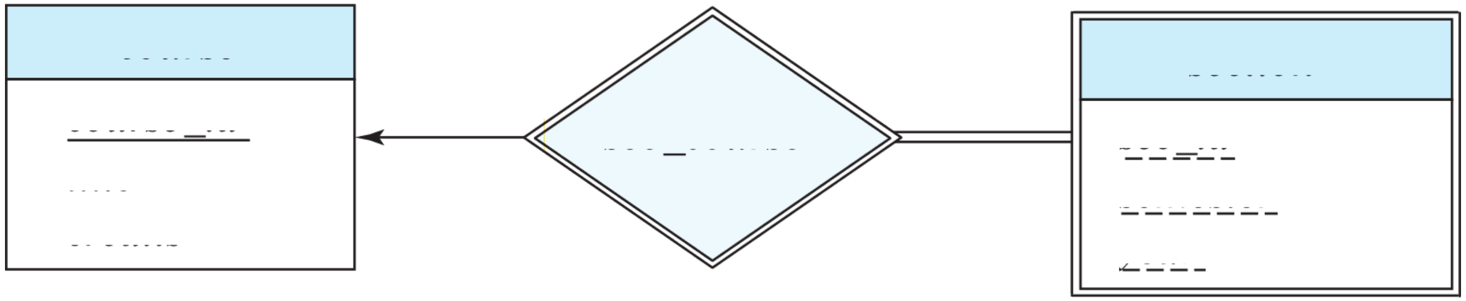


Redundancy of Schemas (Cont.)

- For one-to-one relationship sets, either side can be chosen to act as the "many" side
  - That is, an extra attribute can be added to either of the tables corresponding to the two entity sets
- If participation is *partial* on the "many" side, replacing a schema by an extra attribute in the schema corresponding to the "many" side could result in null values

Redundancy of Schemas (Cont.)

- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
- Example: The *section* schema already contains the attributes that would appear in the *sec_course* schema

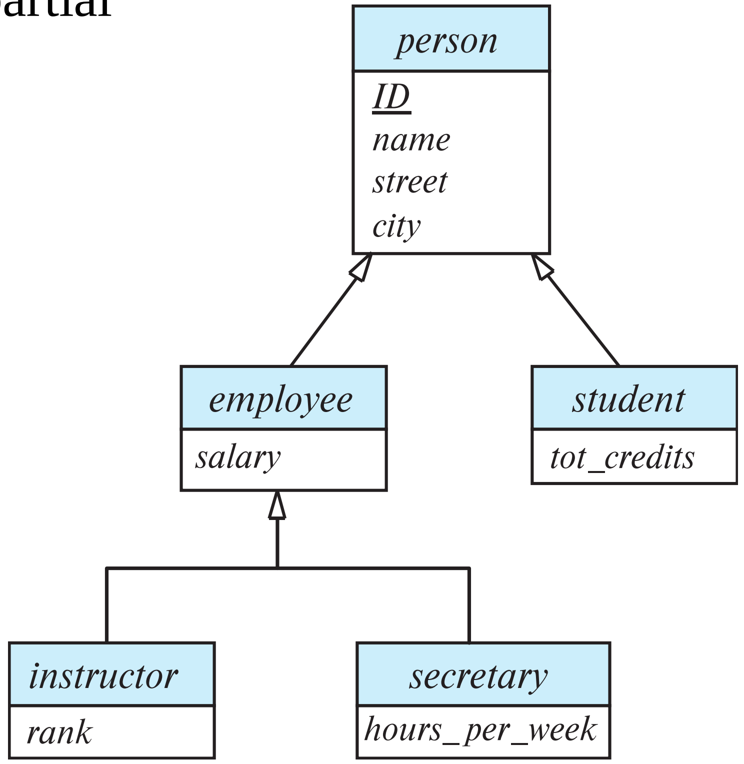


Specialization

- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (e.g., *instructor* "is a" *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

Specialization Example

- **Overlapping** – *employee* and *student*
- **Disjoint** – *instructor* and *secretary*
- Total and partial



Representing Specialization via Schemas

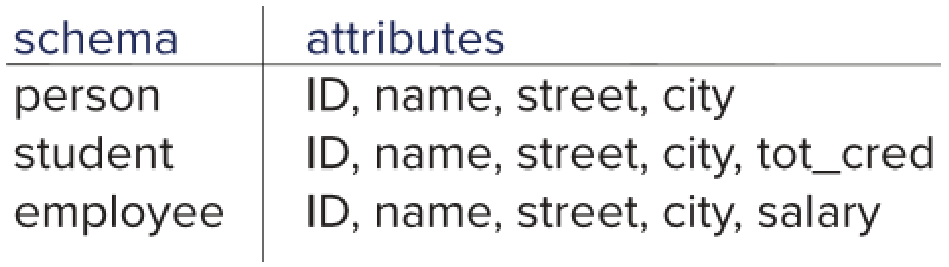
- Method 1:
  - Form a schema for the higher-level entity
  - Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

schema	attributes
person	ID, name, street, city
student	ID, tot_cred
employee	ID, salary

  - Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema

Representing Specialization as Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes



schema	attributes
person	ID, name, street, city
student	ID, name, street, city, tot_cred
employee	ID, name, street, city, salary

  - Drawback: *name*, *street* and *city* may be stored redundantly for people who are both students and employees

Generalization

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Completeness constraint

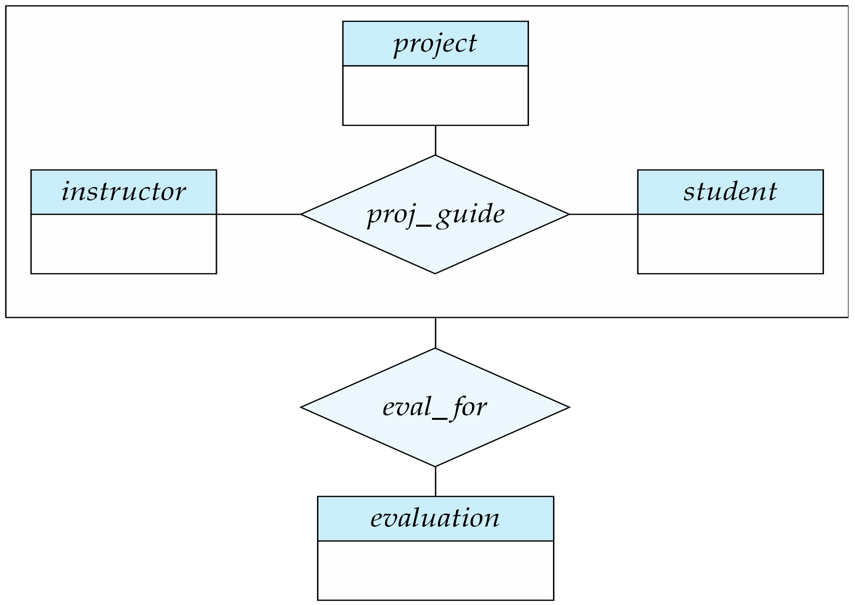
- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
  - **Total**: An entity must belong to one of the lower-level entity sets
  - **Partial**: An entity need not belong to one of the lower-level entity sets

Completeness constraint (Cont.)

- Partial generalization is the default.
- We can specify total generalization in an ER diagram by adding the keyword **total** in the diagram and drawing a dashed line from the keyword to the corresponding hollow arrow-head to which it applies (for a total generalization), or to the set of hollow arrow-heads to which it applies (for an overlapping generalization).
- The *student* generalization is total: All student entities must be either graduate or undergraduate. Because the higher-level entity set arrived at through generalization is generally composed of only those entities in the lower-level entity sets, the completeness constraint for a generalized higher-level entity set is usually total

Aggregation

- Eliminate redundancy via *aggregation* without introducing redundancy, the following diagram represents:
  - A student is guided by a particular instructor on a particular project
  - A student, instructor, project combination may have an associated evaluation



Representing Aggregation as Schemas

- To represent aggregation, create a schema containing
  - Primary key of the aggregated relationship,
  - The primary key of the associated entity set
  - Any descriptive attributes
- In our example:
  - The schema *eval_for* is:

    eval_for (*s_ID, project_id, i_ID, evaluation_id*)
  - The schema *proj_guide* is redundant.

Relational Query Languages

- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power

Relational Algebra

- Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output.
- It uses operators to perform queries. An operator can be either **unary** or **binary**.
- They accept relations as their input and yield relations as their output.
- Relational algebra is performed recursively on a relation and intermediate results are also considered relations.
- The fundamental operations of relational algebra are as follows –
  - Select
  - Project
  - Union
  - Set difference
  - Cartesian product
  - Rename

Select Operation (σ) – selection of rows (Tuples)

It selects tuples that satisfy the given predicate from a relation.

Notation – $\sigma_p(r)$, Where **σ** (sigma) stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like – =, ≠, ≥, < , >, ≤.

For example –

$\boldsymbol{\sigma}_{\textit{subject = "database"}}$**(Books)**

output – Selects tuples from books where subject is 'database'.

$\boldsymbol{\sigma}_{\textbf{subject = "database" and price = "450"}}$**(Books)**

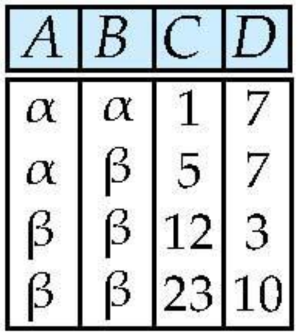
output – Selects tuples from books where subject is 'database' and 'price' is 450.

$\boldsymbol{\sigma}_{\textbf{subject = "database" and price = "450" or year > "2010"}}$**(Books)**

output – Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

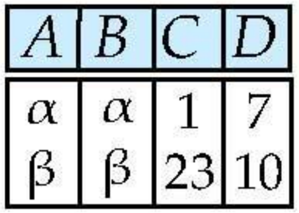
Select Operation (σ)

- Relation r



A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}(r)$



A	B	C	D
α	α	1	7
β	β	23	10

Project Operation (∏)– selection of columns (Attributes)

It projects column(s) that satisfy a given predicate.

Notation – $\prod_{A1,\ A2,\ An}$ (r),

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example –

$\prod_{subject,\ author}$ (Books)

Selects and projects columns named as subject and author from the relation Books.

Project Operation – selection of columns (Attributes)

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\prod_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Union (∪) Operation

It performs binary union between two given relations and is defined as r ∪ s = { t | t ∈ r or t ∈ s}

Notation – r U s, Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold –

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

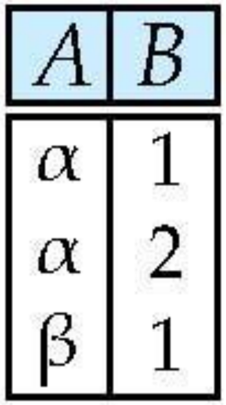
E.g.,

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Output – Projects the names of the authors who have either written a book or an article or both.

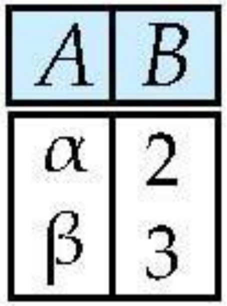
Union of two relations

- Relations *r, s:*



A	B
α	1
α	2
β	1

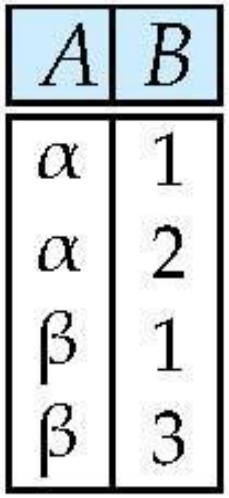
r



A	B
α	2
β	3

s

- r ∪ s:



A	B
α	1
α	2
β	1
β	3

Set difference (-) Operation

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

- **Notation – r – s**

  Finds all the tuples that are present in **r** but not in **s**.

E.g.,

$\prod_{\text{author}}$ (Books) – $\prod_{\text{author}}$ (Articles)

Output – Provides the name of authors who have written books but not articles.

Set difference of two relations

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r* – *s:*

A	B
α	1
β	1

Set intersection of two relations

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Cartesian product (X)

Combines information of two different relations into one.

Notation –

$$r \times s$$

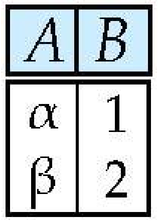
Where **r** and **s** are relations and their output will be defined as –
r X s = { q t | q ∈ r and t ∈ s}

$\boldsymbol{\sigma_{\text{author = 'tutorial'}}}$**(Books X Articles)**

Output – Yields a relation, which shows all the books and articles written by tutorial.

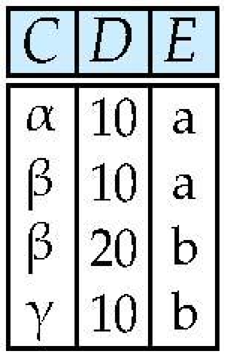
joining two relations -- Cartesian-product

- Relations *r, s*:



A	B
α	1
β	2

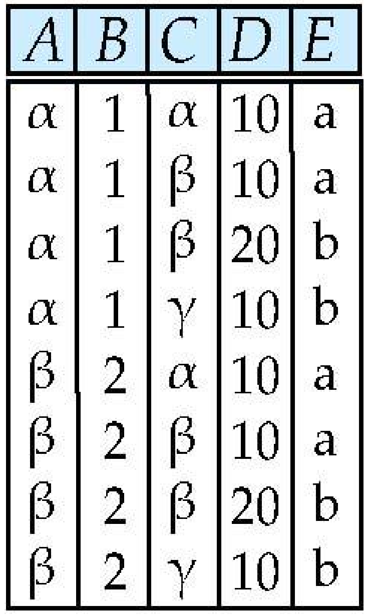
r



C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:



A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian-product – naming issue

- Relations *r*, *s*:

A	*B*
α	1
β	2

r

B	*D*	*E*
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	*r.B*	*s.B*	*D*	*E*
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Renaming Operation (ρ)

The results of relational algebra are also relations but without any name.

The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ.

Notation – ρ_{x} (E),

Where the result of expression **E** is saved with name of **x.**

Renaming Operation (ρ)

- Allows us to refer to a relation, (say E) by more than one name.

$$\rho_x(E)$$

returns the expression *E* under the name *X*

- Relations *r*

A	B
α	1
β	2

r

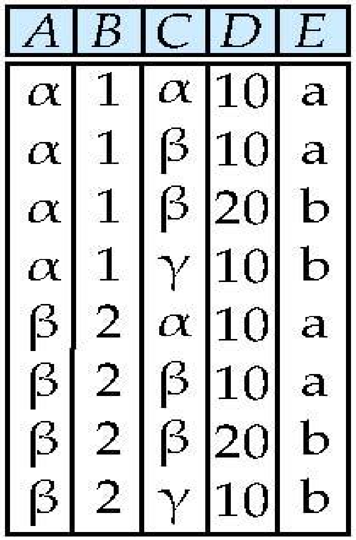
- $r \times \rho_s(r)$

r.A	r.B	s.A	s.B
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

Composition of Operations

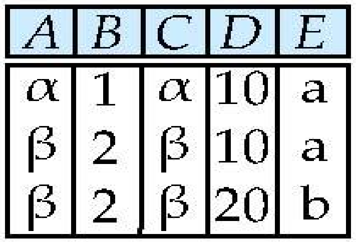
- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$

- $r \times s$



A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$



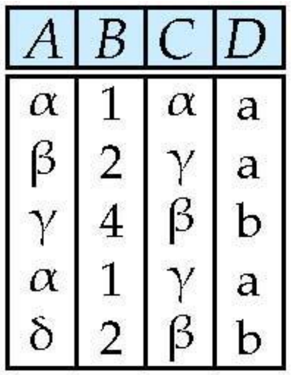
A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Joining two relations – Natural Join

- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, the "natural join" of relations *R* and *S* is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*

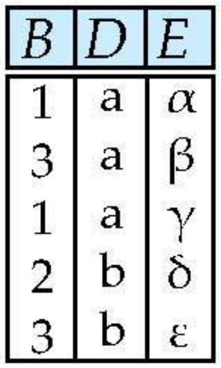
Natural Join Example

- Relations r, s:



A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

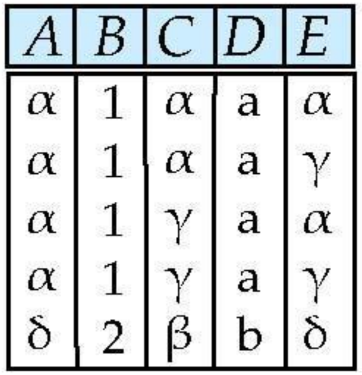
r



B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- Natural Join
  - r ⋈ s



A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Notes about Relational Languages

- Each Query input is a table (or set of tables)
- Each query output is a table.
- All data in the output table appears in one of the input tables
- Relational Algebra is not Turning complete

Summary of Relational Algebra Operators

Symbol (Name)	Example of Use
σ (Selection)	$^{\sigma}$ salary > = 85000 $^{(instructor)}$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$^{\Pi}$ *ID, salary* $^{(instructor)}$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
x (Cartesian Product)	*instructor* x *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
∪ (Union)	$^{\Pi}$ *name* $^{(instructor)}$ ∪ $^{\Pi}$ *name* $^{(student)}$
	Output the union of tuples from the *two* input relations.
- (Set Difference)	$^{\Pi}$ *name* $^{(instructor)}$ -- $^{\Pi}$ *name* $^{(student)}$
	Output the set difference of tuples from the two input relations.
⋈ (Natural Join)	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

Practice Question

Relation schemas:

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

Answer the following questions using relational algebra queries.

a) Get the complete details of all flights to New Delhi.

b) Get the details about all flights from Chennai to New Delhi.

c) Find only the flight numbers for passenger with pid 123 for flights to Chennai before 06/11/2020.

d) Find the passenger names for passengers who have bookings on at least one flight.

e) Find the passenger names for those who do not have any bookings in any flights.

Practice Question

a) Get the complete details of all flights to New Delhi.

$\sigma_{dest='NewDelhi'}(\text{flight})$

b) Get the details about all flights from Chennai to New Delhi.

$\sigma_{src='Chennai' \wedge dest='NewDelhi'}(\text{flight})$

c) Find only the flight numbers for passenger with pid 123 for flights to Chennai before 06/11/2020.

$\pi_{fid}(\sigma_{pid='123' \wedge fdate<'2020-11-06' \wedge dest='Chennai'}(\text{booking} \bowtie \text{flight}))$

d) Find the passenger names for passengers who have bookings on at least one flight.

$\pi_{pname}(\sigma_{fid>1}(\text{passenger} \bowtie \text{booking}))$

e) Find the passenger names for those who do not have any bookings in any flights.

$\pi_{pname}(\text{passenger}) - \pi_{pname}(\sigma_{fid>1}(\text{passenger} \bowtie \text{booking}))$

Practice Question

f) Find the agency names for agencies that located in the same city as passenger with passenger id 123.

g) Get the details of flights that are scheduled on both dates 01/12/2020 and 02/12/2020 at 16:00 hours.

h) Get the details of flights that are scheduled on either of the dates 01/12/2020 or 02/12/2020 or both at 16:00 hours.

i) Find the details of all male passengers who are associated with Jet agency.

j) Find the agency names for agencies who do not have any bookings for passenger with id 123.

Practice Question

f) Find the agency names for agencies that located in the same city as passenger with passenger id 123.

$\pi_{aname}(\sigma_{acity=pcity\ \wedge pid='123'}(\text{ passenger} \bowtie \text{agency}))$

g) Get the details of flights that are scheduled on both dates 01/12/2020 and 02/12/2020 at 16:00 hours.

$\sigma_{fdate='2020-12-01' \wedge time='16:00'}\ (\text{flight}) \cap \sigma_{fdate='2020-12-02' \wedge time='16:00'}\ (\text{flight})$ **OR**

$\sigma_{fdate='2020-12-01' \wedge fdate='2020-12-02' \wedge time='16:00'}(\text{flight})$

h) Get the details of flights that are scheduled on either of the dates 01/12/2020 or 02/12/2020 or both at 16:00 hours.

$\sigma_{time='16:00' \wedge (fdate='2020-12-01' \vee fdate='2020-12-02')}(\text{flight})$

Practice Question

i) Find the details of all male passengers who are associated with Jet agency.

$\boldsymbol{\sigma_{pgender='Male'}(passenger \bowtie (\sigma_{aname='Jet'}(agency) \bowtie booking))}$ **OR**

$\boldsymbol{\sigma_{pgender='Male' \wedge aname='Jet'}(passenger \bowtie agency)}$

j) Find the agency names for agencies who do not have any bookings for passenger with id 123.

$\boldsymbol{\pi_{aname}(agency) - \pi_{aname}(\sigma_{pid=123}(booking \bowtie agency))}$

Tuple Relational Calculus

- Tuple Relational Calculus (TRC) is a **non-procedural query language** used to retrieve data from relational databases by **describing the properties of the required data** (not how to fetch it).
- It is **based on first-order predicate logic** and uses **tuple variables** to represent rows of tables.
- **Syntax:** The basic syntax of TRC is as follows:

  { t | P(t) }

  - **t:** Tuple variable (row placeholder)
  - **P(t):** Predicate condition to satisfy
  - **{}:** Denotes a set of result tuples

Tuple Relational Calculus

- **Logical Operators in TRC:**
  - **∧**: AND
  - **∨**: OR
  - **¬**: NOT
- **Quantifiers:**
  - **∃ t ∈ r (Q(t))** → There exists a tuple t in relation r satisfying predicate Q(t)
  - **∀ t ∈ r (Q(t))** → For all tuples t in relation r, predicate Q(t) holds
- **For example**, let's say we have a table called "Employees" with the attributes EID, Name, Salary, DepartmentID
- To retrieve the names of all employees who earn more than $50,000 per year, we can use the following TRC query:

  { t | Employees(t) ∧ t.Salary > 50000 }

Tuple Relational Calculus

Tuple Relational Query

- In Tuple Calculus, a query is expressed as

  $$\{t \mid P(t)\}$$

  - t represents the resulting tuples.
  - P(t) is a predicate (a condition that must be true for t to be included in the result
- P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT(¬).
- It also uses quantifiers:
  - $\exists\ t \in r\ (Q(t))$ = "there exists" a tuple in t in relation r such that predicate Q(t) is true.
  - $\forall\ t \in r\ (Q(t))$ = Q(t) is true "for all" tuples in relation r.

Tuple Relational Calculus

Table Customer

Customer name	Street	City
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana
Ria	A5	Patiala

Table Account

Account number	Branch name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000
1114	ABC	7000

Table Branch

Branch name	Branch City
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar

Table Loan

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Tuple Relational Calculus

Table Borrower

Customer name	Loan number
Saurabh	L33
Mehak	L49
Ria	L98

Table Depositor

Customer name	Account number
Saurabh	1111
Mehak	1113
Suniti	1114

Tuple Relational Calculus

- Find the loan number, branch, and amount of loans greater than or equal to 10000 amount.

  {t| t ∈ loan ∧ t[amount]>=10000}

- Find the loan number for each loan of an amount greater or equal to 10000.

  {t| ∃ s ∈ loan(t[loan number] = s[loan number] ∧ s[amount]>=10000)}

Tuple Relational Calculus

- Find the names of all customers who have a loan and an account at the bank.

  {t | ∃ s ∈ borrower(t[customer-name] = s[customer-name]) ∧ ∃ u ∈ depositor(t[customer-name] = u[customer-name])}

- Find the names of all customers having a loan at the "ABC" branch.

  {t | ∃ s ∈ borrower(t[customer-name] = s[customer-name] ∧ ∃ u ∈ loan(u[branch-name] = “ABC” ∧ u[loan-number] = s[loan-number]))}

Tuple Relational Calculus

Students(student_id, name, age, department)

Courses(course_id, course_name, instructor)

Enrollments(student_id, course_id, semester, year)

1. Find all students in the "CSE" department.

   {t | t ∈ Students ∧ t.department="CSE“}

2. Find names of students older than 20.

   {t.name | t ∈ Students ∧ t.age>20}

3. Find student(s) who are older than at least one "CSE" student.

{s | s∈Students ∧ ∃c ∈ Students(c.department="CSE"∧ s.age > c.age)}

Tuple Relational Calculus

4. Find all students younger than 18.

$\{t \mid t \in Students \wedge t.age < 18\}$

5. Get names of students enrolled in the year 2025.

$\{s.name \mid s \in Students \wedge \exists e \in Enrollments(s.student\_id = e.student\_id \wedge e.year = 2025)\}$

6. Find students who have enrolled in the course "DBMS".

$\{s \mid s \in Students \wedge \exists e \in Enrollments \exists c \in Courses(s.student\_id = e.student\_id \wedge e.course\_id = c.course\_id \wedge c.course\_name = \text{"DBMS"})\}$

Tuple Relational Calculus

7. Find instructors who teach at least one course.

$\{c.instructor \mid c \in Courses\}$

8. Find students who are not enrolled in any course.

$\{s \mid s \in Students \land \neg\exists e \in Enrollments\ (s.student\_id = e.student\_id)\}$

9. Find students enrolled in every course offered by instructor "Smith".

$\{s \mid s \in Students \land \forall c \in Courses(c.instructor = \text{"Smith"} \rightarrow \exists e \in Enrollments(e.student\_id = s.student\_id \land e.course\_id = c.course\_id))\}$