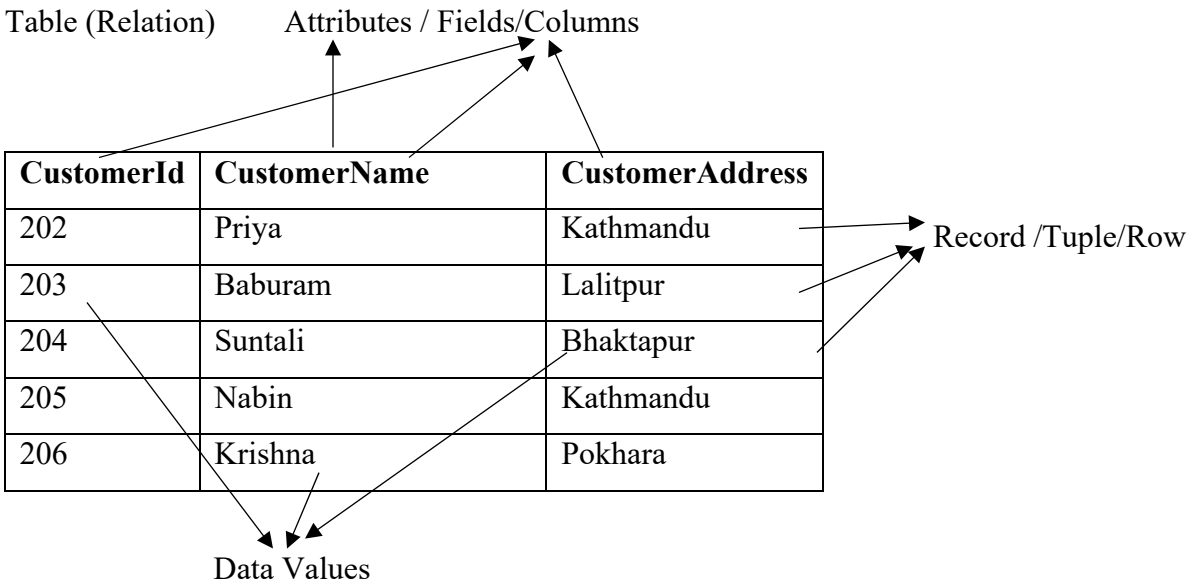
Unit -3: Relational Database Model

Structure of RDBMS and Terminology

- RDBMS = Relational Database Management System
- The relational model is today the primary data model for commercial data processing applications. It attained its primary position because of its simplicity. It provides simple but powerful way of representing data.
- Relational model is simply a collection of one or more relations, where each relation is represented by table with rows and columns.
- It is the most widely used data model. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are known as relations. Each table corresponds to an entity set or relationship set, and each row represents an instance of that entity set or relationship set. Relationships link rows from two tables by embedding row identifiers (keys) from one table as attribute values in the other table. Structured Query Language (SQL) is used to manipulate data stored in tables.



- A relational database consists of a collection of tables, each of which is assigned a unique name.
- In the relational model the term **relation** is used to refer to a table, while the term **tuple** is used to refer to a row. Similarly, the term attribute refers to a column of a table.
- We use the term relation instance to refer to a specific instance of a relation, i.e., containing a specific set of rows.
- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- For each attribute of a relation, there is a *set of permitted values*, called the **domain** of that attribute. Thus, the domain of the *salary* attribute of the *instructor* relation is the set of all possible salary values, while the domain of the *name* attribute is the set of all possible instructor names.
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- A domain is said to be **atomic** if elements of the domain are considered to be indivisible units. Domains of multivalued and composite attributes are **non atomic.**
- The **null** value is a special value that signifies that the value is unknown or does not exist.
- The special value ***null*** is a member of every domain

Relation schema and instance

- Database **schema**, is the logical design of the database, and the database **instance** is a snapshot of the data in the database at a given instant in time.
- In general, a relation schema consists of a list of attributes and their corresponding domains.
- $A_1, A_2, \ldots, A_n$ 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
- Relational Schemas for university database:
  - *instructor* (*ID, name, dept_name, salary*)
  - *department* (*dept _name*, *building*, *budget*)
  - *section* (*course _id*, *sec _id*, *semester*, *year*, *building*, *room_ number*, *time _slot_ id*)
  - *prereq* (*course _id*, *prereq_ id*)
  - *teaches* (*ID*, *course_ id*, *sec _id*, *semester*, *year*)
  - *student* (*ID*, *name*, *dept _name*, *tot _cred*)
  - *advisor* (*s_ id*, *i_ id*)
  - *takes* (*ID*, *course _id*, *sec_ id*, *semester*, *year*, *grade)*
  - *classroom* (*building*, *room number*, *capacity*)
  - *time _slot* (*time _slot _id*, *day*, *start _time*, *end _time*)

Keys (super key, candidate key, primary key, foreign key, composite key, alternative key)

- Keys are used to uniquely identify the tuples in a relations.
- No two tuples in a relation are allowed to have exactly the same value for all attributes.
- 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)*
  - A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **candidate keys**
  - A super key with no repeated attribute is called candidate key.
  - The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key.
  - Superkey *K* is a **candidate key** if *K* is minimal
    Example: {*ID*} is a candidate key for *instructor*

    Properties of Candidate key:
  - *It must contain unique values*
  - *Candidate key may have multiple attributes*

- *Must not contain null values*
- *It should contain minimum fields to ensure uniqueness*
- *Uniquely identify each record in a table*

- One of the candidate keys is selected to be the **primary key**.
  - The **primary key** is candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation

    Rules for defining Primary key:

  - Two rows can't have the same primary key value
  - It must for every row to have a primary key value.
  - The primary key field cannot be null.
  - The value in a primary key column can never be modified or updated if any foreign key
  - refers to that primary key.
- A relation, say r1, may include among its attributes the primary key of another relation, say r2. This attribute is called a **foreign key** from r1, referencing r2. The relation r1 is

also called the **referencing relation** of the foreign key dependency, and r2 is called the **referenced relation** of the foreign key

- A foreign key (FK) is an attribute or combination of attributes that is used to establish and enforce relationship between two relations (table). A set of attributes that references primary key of another table is called foreign key. For example, if a student enrolls in program then program-id (primary key of relation program) can be used as foreign key in student relation.

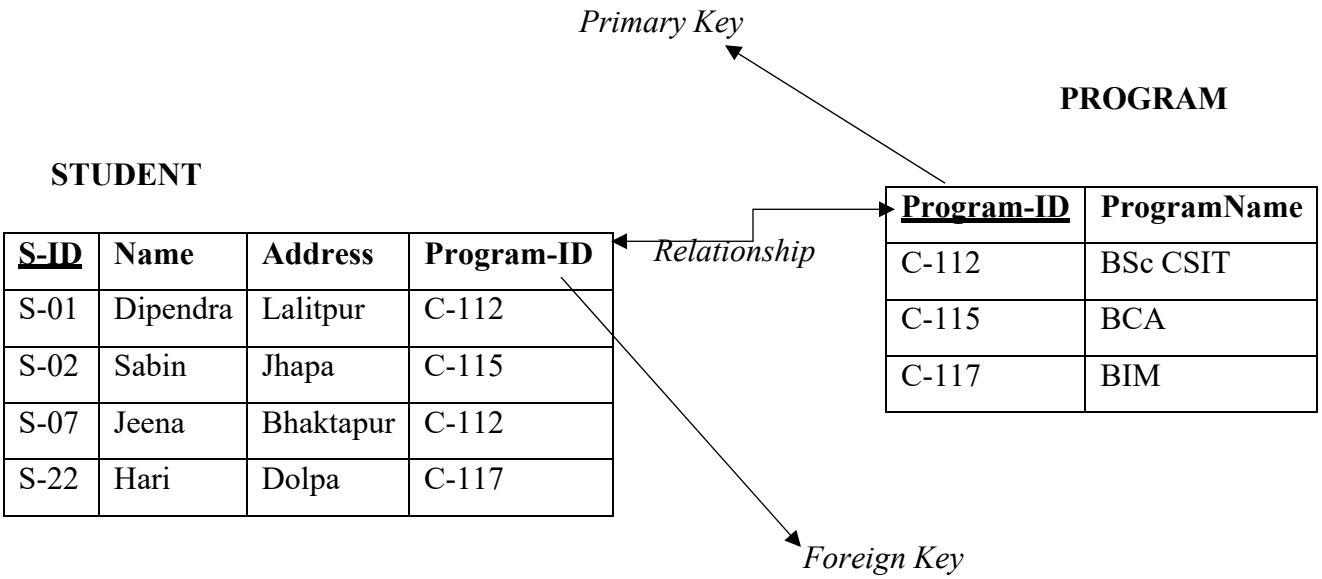
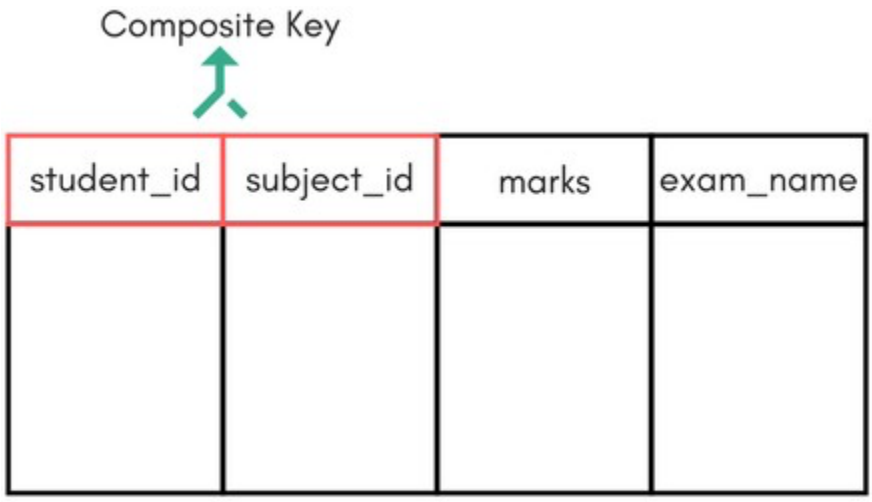


Figure: Primary Key and Foreign Key

Here, STUDENT is referencing relation and PROIGRAM is referenced relation

- The Key that consists of two or more columns/attributes that uniquely identify any record in a table/relation is called **composite key**. But the attributes which together form the composite key are not a key independently or individually.
  - In the figure below, we have a Score table which stores the marks scored by a student in a particular subject.
  - In this table *student_id* and *subject_id* together will form the primary key, hence it is a composite key.



Score Table - To save scores of the student for various subjects.

Alternate key

- ✓ All the keys which are not primary key are called an alternate key. It is a key which is currently not the primary key.
- ✓ However, A table may have single or multiple choices for the primary key.

StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

Schema Diagram

A database schema, along with primary key and foreign key dependencies, can be depicted by **schema diagrams.**

- A **schema diagram** is a graphical representation of database schema, along with primary key and foreign key dependencies
- In a schema diagram:
  - Each relation is represented by box
  - Attributes are listed inside box
  - Relation name is specified at top of the box.
  - Primary key in relation is underlined.

- Foreign key dependencies in schema diagram appear as arrow from the foreign key attributes of the referencing relation to the primary key of referenced relation.
- Schema diagram for university database is shown below:

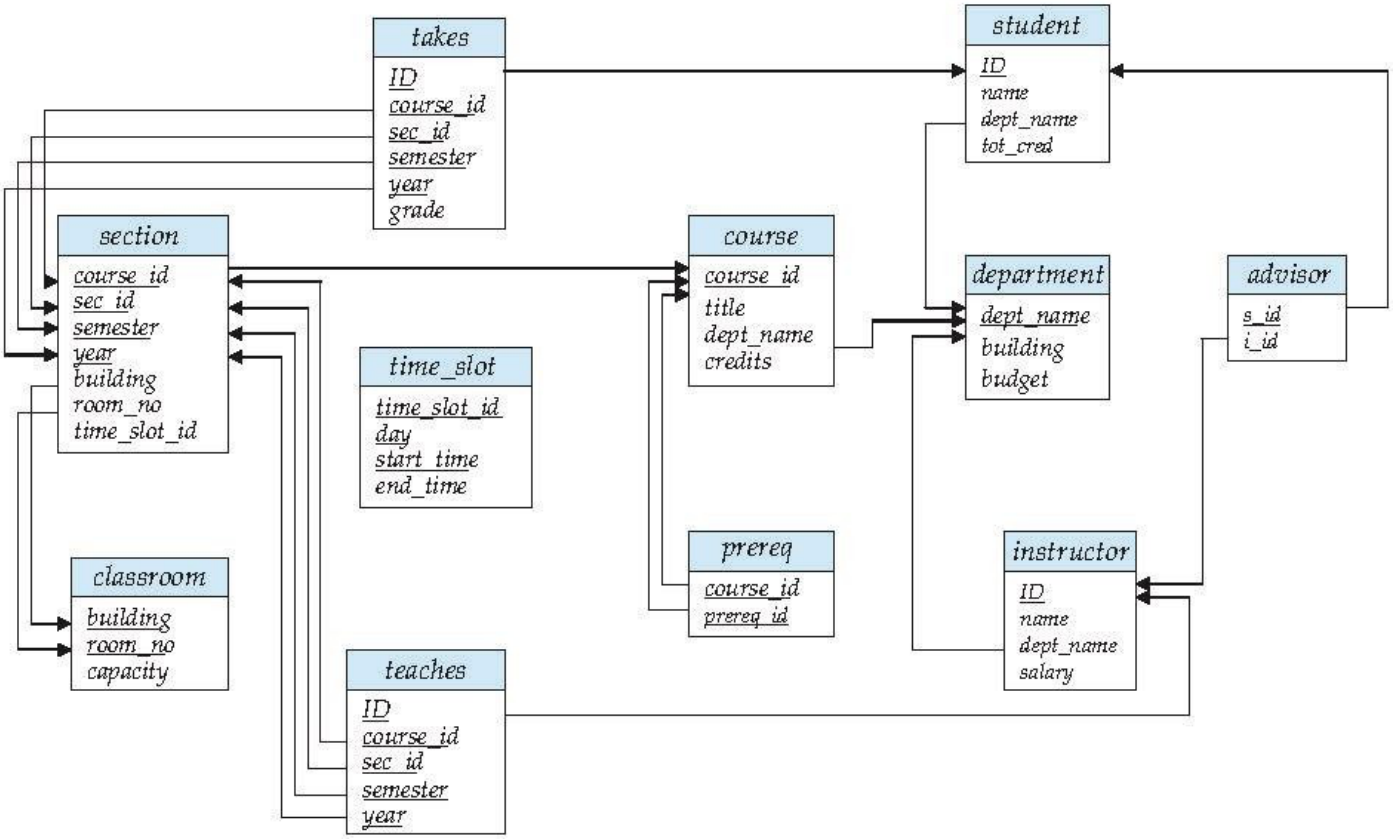


Figure: Schema diagram for university database

Introduction to Relational Algebra

- 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 their result.
- The fundamental operations in the relational algebra are *select, project, union, set difference, Cartesian product*, and *rename*.
- In addition to the fundamental operations, there are several other operations—namely, *set intersection, natural join*, and *assignment.* We shall define these operations in terms of the fundamental operations.

Fundamental Operations in Relational Algebra (RA):

- There are six fundamental (basic) operations /operators as follows**:**
  - **select: σ**

- **project: ∏**
- **union: ∪**
- **set difference: –**
- **Cartesian product: x**
- **rename: ρ**

- The fundamental operations can be either unary or binary.
  - The **select, project**, and **rename** operations are called *unary* operations, because they operate on one relation.
  - The other three operations (**union, set difference, Cartesian product**) operate on pairs of relations and are, therefore, called *binary* operations.

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$

  Here, ***p*** is called the **selection predicate** and **r** is a **relation**

- We use the lowercase Greek letter sigma (**σ**) to denote selection. The predicate appears as a subscript to **σ** . The argument relation is in parentheses after the **σ**.
- Example: select those tuples of the *instructor* relation where the instructor is in the "Physics" department.

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: The *instructor* relation

- Query

  $\sigma_{dept\_name="Physics"}(instructor)$

- Result

ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

- In general, we allow comparisons using $=, \neq, >, \geq, <$ and $\leq$ in the selection predicate(p).
- We can combine several predicates into a larger predicate by using the connectives:

  $\wedge$ **(and),** $\vee$ **(or),** $\neg$ **(not)**

  - Example: Find the instructors in Physics with a salary greater $90,000, we write:

$$\sigma_{dept\_name="Physics" \wedge salary > 90,000}(instructor)$$

- Then select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:

$$\sigma_{dept\_name=building}(department)$$

Project Operation

- A unary operation that returns its argument relation, with certain attributes left out.
- It projects column(s) that satisfy a given predicate.
- Notation:

  $\Pi_{A1,A2,A3 \ldots Ak}(r)$

  where A_1, A_2 are attribute names and r is a relation name.

- Projection is denoted by the uppercase Greek letter pi (Π). We list those attributes that we wish to appear in the result as a subscript to Π. The argument relation follows in parentheses.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed

- Duplicate rows are removed from result, since relations are sets.
- Example: eliminate the *dept_name* attribute of *instructor*
- Query*:*

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

Result:

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Composition of Relational Operations

- The result of a relational-algebra operation is relation and therefore of relational-algebra operations can be composed together into a **relational-algebra expression**.
- Consider the query – Find the names of all instructors in the Physics department.

$$\Pi_{name}(\sigma_{\ dept\_name\ =\text{“}Physics\text{”}}\ (instructor))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

Union Operation

- The union operation allows us to combine two relations.
- Notation: $r \cup s$

  Here, r and s are two relations

- For ***r*** ∪ *s* to be valid, we require that two conditions hold:
  1. *r, s* must have the *same* arity (same number of attributes)
  2. The domains of the **i**th attribute of **r** and the **i**th attribute of **s** must be the same, for all **i.** Which indicates that the attribute domains must be compatible (example: 2nd column of ***r*** deals with the same type of values as does the 2nd column of ***s***)
- Note that r and s can be either database relations or temporary relations that are the result of relational-algebra expressions

Example

Consider the following tables.

Table A		Table B	
column 1	column 2	column 1	column 2
1	1	1	1
1	2	1	3

A ∪ B gives

Table A ∪ B	
column 1	column 2
1	1
1	2
1	3

- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
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: The *section* Relation

Query:

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2010}(section))$$

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

The query gives above table as a Result.

Set Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation: $r - s$

  Here, r and s are two relations

- Set differences must be taken between **compatible** relations.
- Therefore, for a set-difference operation **r − s** to be valid:
  - ***r*** and ***s*** must have the same arity
  - Attribute domains of r and s must be compatible (the domains of the **i**th attribute of **r** and the **i**th attribute of **s** be the same, for all **i**.)

Consider the following tables.

Table A		Table B	
column 1	column 2	column 1	column 2
1	1	1	1
1	2	1	3

Table A – B	
column 1	column 2
1	2

- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

  Query:

  $$\prod_{course\_id}(\sigma_{semester="Fall" \wedge year=2009}(section)) - \prod_{course\_id}(\sigma_{semester="Spring" \wedge year=2010}(section))$$

  Result:

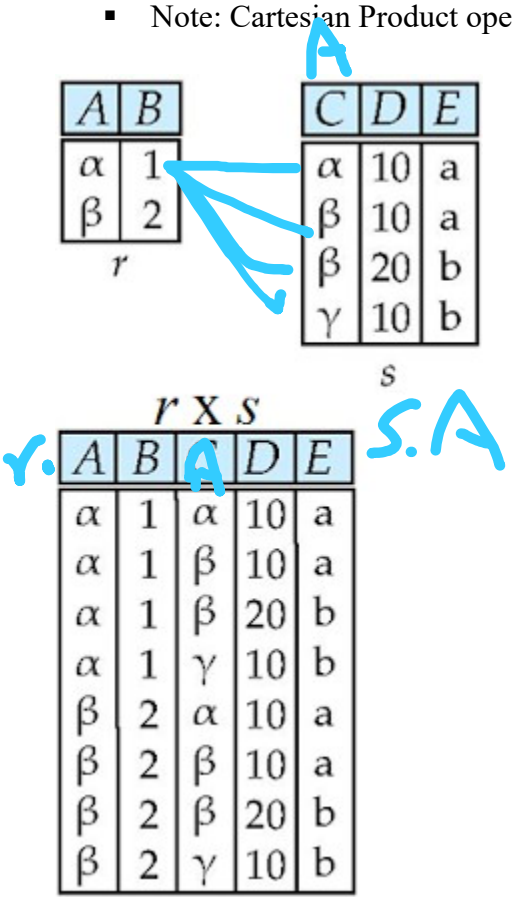
course_id
CS-347
PHY-101

Cartesian-Product Operation

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.

- We write the Cartesian product of relations **r1** and **r2** as r = **r1 × r2**.
- In general, if we have relations r1(R1) and r2(R2), then r1 × r2 is a relation whose schema is the concatenation of R1 and R2.

- Note: Cartesian Product operation is also known as cross join.



r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

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

- Example: the Cartesian product of the relations *instructor* and t*eaches* is written as:

  instructor X *teaches*

ID	*course_id*	*sec_id*	*semester*	*year*
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: The *teaches Relation*

- We construct a tuple of the result out of each possible pair of tuples: one from the *instructor* relation and one from the *teaches* relation as in table below.

Instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

Figure: *instructor* × *teaches* relations

- Since the instructor *ID* appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
    - *instructor.ID*
    - *teaches.ID*

The Rename Operation

- The results of relational-algebra expressions do not have a name that we can use to refer to them. The rename operator, denoted by the lowercase Greek letter rho (ρ), ρ , is provided for that purpose.

  The expression: $\rho_x(E)$

  returns the result of expression E under the name x

- Another form of the rename operation:

  $\rho_{x(A1,A2,..An)}(E)$

  returns the result of expression E under the name x, and with the attributes renamed to A1, A2, . . . , An

Types of JOIN

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Natural-Join Operation

- The ***natural join*** is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
- It is denoted by the join symbol ⋈.
- The natural-join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes
- Returning to the example of the relations *instructor* and *teaches*, computing ***instructor*** **natural join** ***teaches*** considers only those pairs of tuples where both the tuple from

instructor and the tuple from *teaches* have the same value on the common attribute ID.The result is shown in following figure.

- Notice that we do not repeat those attributes that appear in the schemas of both relations; rather they appear only once. Notice also the order in which the attributes are listed:
    - first the attributes common to the schemas of both relations,
    - second those attributes unique to the schema of the first relation,
    - and finally, those attributes unique to the schema of the second relation

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure : The natural join of the *instructor* relation with the *teaches* relation

- The natural join of r and s, denoted by $r \bowtie s$, is a relation on schema $R \cup S$ formally defined as follows:

$$r \bowtie s = \Pi_{R \cup S} (\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (r \times s))$$

Where $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

- If r (R) and s(S) are relations without any attributes in common, that is, $R \cap S = \varnothing$, then $r \bowtie s = r \times s$.

Let

R=(A,B,C,D)
S=(E,B,D)

Now,

Result schema=(A,B,C,D,E)
r ⋈ s is define as

$$\Pi_{r.A,r.B,r.C,r.D,s.E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$

- 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

- 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

Consider the following two tables

C

Num	Square
2	4
3	9

D

Num	Cube
2	8
3	27

C ⋈ D

C ⋈ D

Num	Square	Cube
2	4	8
3	9	27

- The **theta join** operation is a variant of the natural-join operation that allows us to combine a selection and a Cartesian product into a single operation. Consider relations r (R) and s(S), and let θ be a predicate on attributes in the schema R ∪ S. The theta join operation

  r $\bowtie_\theta$ **s** is defined as follows:

  $$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

  Thus, $\sigma_{instructor.id = teaches.id}$ (*instructor* x *teaches*)

  Can equivalently be written as:

  $$instructor \bowtie_{Instructor.id = teaches.id} teaches.$$

For example:

```
A ⋈ A.column 2 > B.column 2 (B)
```

A ⋈ A.column 2 > B.column 2 (B)	
column 1	column 2
1	2

Equi join operation

- When a theta join uses only equivalence condition, it becomes a equi join.

For example:

```
A ⋈ A.column 2 = B.column 2 (B)
```

A ⋈ A.column 2 = B.column 2 (B)	
column 1	column 2
1	1

Outer join Operations

- The outer-join operation is an extension of the join operation to deal with missing information.
- The outer join operation works in a manner similar to the natural join operation, but preserves those tuples that would be lost in an join by creating tuples in the result containing null values
- We can use the outer-join operation to avoid this loss of information.
- There are actually three forms of the operation:
  - **left outer join** (denoted as $⟕$)
  - **right outer join** (denoted as $⟖$)

- **full outer join** (denoted as ⟗)

- The **left outer join** takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join.
  - *Takes all tuples in the left relation. If there are any tuples in right relation that does not match with tuple in left relation, simply pads these right relation tuples with null.*

- The **right outer join** is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join
  - *Takes all tuples in right relation. If there are any tuples in the left relation that does not match with tuples in right relation, simply pads relation tuples with null.*

- The **full outer join** does both the left and right outer join operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.
  - *Pads tuples from the left relation that did not match any from the right relation.*
  - *Pads tuples from the left relation that did not match any from the left relation*

Consider the following 2 Tables

A	
Num	Square
2	4
3	9
4	16

B	
Num	Cube
2	8
3	18
5	75

Left outer join

$A \bowtie B$

A ⋈ B		
Num	Square	Cube
2	4	8
3	9	18
4	16	–

Right outer join

$A \bowtie B$

A ⋈ B		
Num	Cube	Square
2	8	4
3	18	9
5	75	–

Full outer join

$A \bowtie B$

A ⋈ B		
Num	Cube	Square
2	4	8
3	9	18
4	16	–
5	–	75

Example: Consider two relations loan and borrow. Different join operations on these relations are shown in figures below.

loan_number	branch_name	amount
L01	B1	500
L02	B2	600
L05	B1	700

Relation loan

customer_name	loan_number
X	L01
Y	L02
Z	L07

Relation borrower

Loan ⋈ borrower

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y

loan⟕ borrower

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y
L05	B1	700	null

Loan ⟕ borrower

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y
L07	null	null	Z

Loan ⟖ borrower

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y
L05	B1	700	null
L07	null	null	Z

Summary

Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	– Symbol denotes it. The result of A – B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.

Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

Exercise :

Player relation

Player Id	Team Id	Country	Age	Runs	Wickets
1001	101	India	25	10000	300
1004	101	India	28	20000	200
1006	101	India	22	15000	150
1005	101	India	21	12000	400
1008	101	India	22	15000	150
1009	103	England	24	6000	90
1010	104	Australia	35	1300	0
1011	104	Australia	29	3530	10
1012	105	Pakistan	28	1421	166
1014	105	Pakistan	21	3599	205

Deposit relation

Acc. No.	Cust-name
A 231	Rahul
A 432	Omkar
R 321	Sachin
S 231	Raj
T 239	Sumit

Borrower relation

Loan No.	Cust-name
P-3261	Sachin
Q-6934	Raj
S-4321	Ramesh
T-6281	Anil

1. Find all tuples from player relation for which country is India.
2. Select all the tuples for which runs are greater than or equal to 15000.
3. Select all the players whose runs are greater than or equal to 6000 and age is less than 25
4. List all the countries in Player relation.
5. List all the team ids and countries in Player Relation.
6. Find all the customers having an account but not the loan.
7. Find all the customers having a loan but not the account.
8. Rename Customer relation to CustomerList.