Chapter 2: Introduction to Relational Model[1]

Abu Raihan Mostofa Kamal

Professor, CSE Department
Islamic University of Technology (IUT)

November 13, 2025

[1] This is based on Textbook, its companion slide and other sources

Chapter Outline

Motivation

- The relational model remains the primary data model for commercial data-processing applications.
- It has the power of simplicity for designer and application programmer
- New features are regularly added such as Object Model, Complex Data-type, Stored Procedures, so on.
- The model is well-matured, it has been considered as the default standard for almost half a century.

Motivation

- The relational model remains the primary data model for commercial data-processing applications.
- It has the power of simplicity for designer and application programmer
- New features are regularly added such as Object Model, Complex Data-type, Stored Procedures, so on.
- The model is well-matured, it has been considered as the default standard for almost half a century.

Motivation

- The relational model remains the primary data model for commercial data-processing applications.
- It has the power of simplicity for designer and application programmer
- New features are regularly added such as Object Model, Complex Data-type, Stored Procedures, so on.
- The model is well-matured, it has been considered as the default standard for almost half a century.

Motivation

- The relational model remains the primary data model for commercial data-processing applications.
- It has the power of simplicity for designer and application programmer
- New features are regularly added such as Object Model, Complex Data-type, Stored Procedures, so on.
- The model is well-matured, it has been considered as the default standard for almost half a century.

Table/Relation, Column, Record

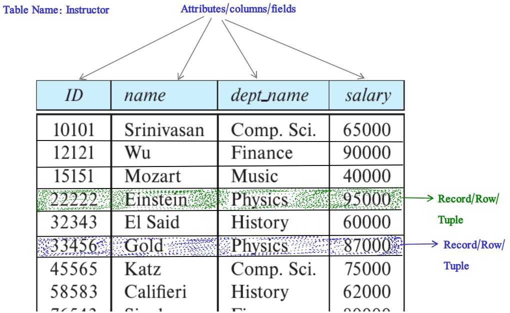
- A relational database consists of a collection of **inter-related** tables, each of which is assigned a unique name.
- In the relational model the term relation is used to refer to a table, while the term tuple is used to refer to a row/record. Similarly, the term attribute refers to a column of a table.

Table/Relation, Column, Record

- A relational database consists of a collection of **inter-related** 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/record. Similarly, the term attribute refers to a column of a table.

Table/Relation, Column, Record

- A relational database consists of a collection of **inter-related** 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/record. Similarly, the term attribute refers to a column of a table.



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

Figure: Relation, attribute and row

Relation and Relation Instance

We use the term **Relation Instance** to refer to a specific instance of a relation, that is, containing a specific set of rows. It is always tied to a specific time.

Example: The instance of department as shown here has 7 records/rows/tuples, corresponding to 7 departments. But after 2 years the records may be more or less or changed. That will be another instance at that time.

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

Figure: department Relation Instance

Attribute: Domain

- The set of allowed values for each attribute is called the **domain** of the attribute.

Example

- The domain of Program Type in RPS has a set of possibilities: Undergrad, PostGrad
- The domain of Shift has the set of all possible days: {Mon, Tue, Wed, Thur,Fri}.
- The domain of Name is the set of character strings that represents names of people.

Note: Domains can be set at the time of DDL using its basic data type and/or additional constraint.

Attribute: Atomic and Null Values

- Attribute values are (normally) required to be **atomic**; that is, indivisible
  Example: The domain of Name is the set of character strings that represents names of people. It has no sub-parts.
- 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

Attribute: Atomic and Null Values

- Attribute values are (normally) required to be **atomic**; that is, indivisible
  Example: The domain of Name is the set of character strings that represents names of people. It has no sub-parts.
- 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

Attribute: Atomic and Null Values

- Attribute values are (normally) required to be **atomic**; that is, indivisible
  Example: The domain of Name is the set of character strings that represents names of people. It has no sub-parts.
- 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

Relations are Unordered

- **Order** of tuples is **irrelevant** (tuples may be stored in an arbitrary order)
- **Example:** It does not have any logical consequence if any record is stored at the end or at the start. (response time may vary which is not connected to functionality)

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

Figure: department relation, order does not matter here

Schema and Instance

- Logical design of the database is the **Database Schema**, while the content of a database at a given time is called **Database Instance**. (In oracle technology it is called Snapshot).
- Similarly, Relation Schema and Relation Instance are defined.
- **Summary:** Schema is the structure (hardly changed), while Instance (i.e data) gets changed over time.

Schema and Instance

- Logical design of the database is the **Database Schema**, while the content of a database at a given time is called **Database Instance**. (In oracle technology it is called Snapshot).
- Similarly, Relation Schema and Relation Instance are defined.
- **Summary:** Schema is the structure (hardly changed), while Instance (i.e data) gets changed over time.

Schema and Instance

- Logical design of the database is the **Database Schema**, while the content of a database at a given time is called **Database Instance**. (In oracle technology it is called Snapshot).
- Similarly, Relation Schema and Relation Instance are defined.
- **Summary:** Schema is the structure (hardly changed), while Instance (i.e data) gets changed over time.

Keys

- In a relational database there must be a way to **distinctly or uniqely identify each record/tuple** of a given relation/table.
- Keys are used to **uniquely identify** each record.
- Evolution of Concepts of Keys :

$$\boxed{\text{Superkey} \Rightarrow \text{Candidate Keys} \Rightarrow \text{Primary Key}}$$

- **Foreign Key** is defined based on **Primary Key**.
- Notations: **R** for relation, **K** for keys, Relation Instance **r(R)**.

Keys

- In a relational database there must be a way to **distinctly or uniqely identify each record/tuple** of a given relation/table.
- Keys are used to **uniquely identify** each record.
- Evolution of Concepts of Keys :

$$\boxed{\text{Superkey} \Rightarrow \text{Candidate Keys} \Rightarrow \text{Primary Key}}$$

- **Foreign Key** is defined based on **Primary Key**.
- Notations: **R** for relation, **K** for keys, Relation Instance **r(R)**.

Keys

- In a relational database there must be a way to **distinctly or uniqely identify each record/tuple** of a given relation/table.
- Keys are used to **uniquely identify** each record.
- Evolution of Concepts of Keys :

$\boxed{\text{Superkey} \Rightarrow \text{Candidate Keys} \Rightarrow \text{Primary Key}}$

- **Foreign Key** is defined based on **Primary Key**.
- Notations: **R** for relation, **K** for keys, Relation Instance **r(R)**.

Keys

- In a relational database there must be a way to **distinctly or uniqely identify each record/tuple** of a given relation/table.
- Keys are used to **uniquely identify** each record.
- Evolution of Concepts of Keys :

Superkey $\Rightarrow$ Candidate Keys $\Rightarrow$ Primary Key

- **Foreign Key** is defined based on **Primary Key**.
- Notations: **R** for relation, **K** for keys, Relation Instance **r(R)**.

Keys

- In a relational database there must be a way to **distinctly or uniqely identify each record/tuple** of a given relation/table.
- Keys are used to **uniquely identify** each record.
- Evolution of Concepts of Keys :

Superkey $\Rightarrow$ Candidate Keys $\Rightarrow$ Primary Key

- **Foreign Key** is defined based on **Primary Key**.
- Notations: **R** for relation, **K** for keys, Relation Instance **r(R)**.

Super Keys and Candidate Keys

- A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let $K \subseteq R$. K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
- In simple language "no two distinct tuples have the same values on all attributes in K"
- That is, if $t1$ and $t2$ are in r and $t1 \neq t2$, then $t1.K \neq t2.K$.
- A relation R may have a number of superkeys.
- If K is a superkey, then any superset of K must be superkey. (i.e. apriori property)
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **Candidate Keys**.
- Among the candidate keys, the one selected by the database designer is called **Primary Key**

Super Keys and Candidate Keys

- A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let $K \subseteq R$. K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
- In simple language "no two distinct tuples have the same values on all attributes in K"
- That is, if $t1$ and $t2$ are in r and $t1 \neq t2$, then $t1.K \neq t2.K$.
- A relation R may have a number of superkeys.
- If K is a superkey, then any superset of K must be superkey. (i.e. apriori property)
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **Candidate Keys**.
- Among the candidate keys, the one selected by the database designer is called **Primary Key**

Super Keys and Candidate Keys

- A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let $K \subseteq R$. K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
- In simple language "no two distinct tuples have the same values on all attributes in K"
- That is, if $t1$ and $t2$ are in r and $t1 \neq t2$, then $t1.K \neq t2.K$.
- A relation R may have a number of superkeys.
- If K is a superkey, then any superset of K must be superkey. (i.e. apriori property)
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **Candidate Keys**.
- Among the candidate keys, the one selected by the database designer is called **Primary Key**

Super Keys and Candidate Keys

- A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let $K \subseteq R$. K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
- In simple language "no two distinct tuples have the same values on all attributes in K"
- That is, if $t1$ and $t2$ are in r and $t1 \neq t2$, then $t1.K \neq t2.K$.
- A relation R may have a number of superkeys.
- If K is a superkey, then any superset of K must be superkey. (i.e. apriori property)
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **Candidate Keys**.
- Among the candidate keys, the one selected by the database designer is called **Primary Key**

Super Keys and Candidate Keys

- A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let $K \subseteq R$. K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
- In simple language "no two distinct tuples have the same values on all attributes in K"
- That is, if $t1$ and $t2$ are in r and $t1 \neq t2$, then $t1.K \neq t2.K$.
- A relation R may have a number of superkeys.
- If K is a superkey, then any superset of K must be superkey. (i.e. apriori property)
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **Candidate Keys**.
- Among the candidate keys, the one selected by the database designer is called **Primary Key**

Super Keys and Candidate Keys

- A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let $K \subseteq R$. K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
- In simple language "no two distinct tuples have the same values on all attributes in K"
- That is, if $t1$ and $t2$ are in r and $t1 \neq t2$, then $t1.K \neq t2.K$.
- A relation R may have a number of superkeys.
- If K is a superkey, then any superset of K must be superkey. (i.e. apriori property)
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **Candidate Keys**.
- Among the candidate keys, the one selected by the database designer is called **Primary Key**

Super Keys and Candidate Keys

- A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let $K \subseteq R$. K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
- In simple language "no two distinct tuples have the same values on all attributes in K"
- That is, if $t1$ and $t2$ are in r and $t1 \neq t2$, then $t1.K \neq t2.K$.
- A relation R may have a number of superkeys.
- If K is a superkey, then any superset of K must be superkey. (i.e. apriori property)
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **Candidate Keys**.
- Among the candidate keys, the one selected by the database designer is called **Primary Key**

Super Keys and Candidate Keys

- A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let $K \subseteq R$. K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
- In simple language "no two distinct tuples have the same values on all attributes in K"
- That is, if $t1$ and $t2$ are in r and $t1 \neq t2$, then $t1.K \neq t2.K$.
- A relation R may have a number of superkeys.
- If K is a superkey, then any superset of K must be superkey. (i.e. apriori property)
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **Candidate Keys**.
- Among the candidate keys, the one selected by the database designer is called **Primary Key**

Superkey and Candidate Key: Example

Name	Prog	DOB	CGPA
Kim	CSE	1-1-84	3.75
John	EEE	1-2-85	3.75
Kim	SWE	3-6-79	3.60
John	EEE	1-1-84	3.50

Table: Results Relation

Lets think of some possible formation of K

- $K_1 = Name, K_2 = \{Prog\}, K_3 = \{CGPA\}$ NOT superkey
- $K_4 = \{Name, Prog\}$ NOT a superkey (since [John,EEE] are not unique)
- $K_5 = \{Name, DOB\}$ is a superkey
- $K_6 = \{Name, DOB, CGPA\}$ is a superkey
- $K_7 = \{Name, CGPA\}$ is a superkey, other possibilities exist
- The last option is $K_n = \{allattributes\}$ must be a superkey, otherwise duplicate record exists.

Superkey and Candidate Key: Example

Name	Prog	DOB	CGPA
Kim	CSE	1-1-84	3.75
John	EEE	1-2-85	3.75
Kim	SWE	3-6-79	3.60
John	EEE	1-1-84	3.50

Table: Results Relation

Lets think of some possible formation of K

- $K_1 = Name, K_2 = \{Prog\}, K_3 = \{CGPA\}$ NOT superkey
- $K_4 = \{Name, Prog\}$ NOT a superkey (since [John,EEE] are not unique)
- $K_5 = \{Name, DOB\}$ is a superkey
- $K_6 = \{Name, DOB, CGPA\}$ is a superkey
- $K_7 = \{Name, CGPA\}$ is a superkey, other possibilities exist
- The last option is $K_n = \{allattributes\}$ must be a superkey, otherwise duplicate record exists.

Superkey and Candidate Key: Example

Name	Prog	DOB	CGPA
Kim	CSE	1-1-84	3.75
John	EEE	1-2-85	3.75
Kim	SWE	3-6-79	3.60
John	EEE	1-1-84	3.50

Table: Results Relation

Lets think of some possible formation of K

- $K_1 = Name, K_2 = \{Prog\}, K_3 = \{CGPA\}$ NOT superkey
- $K_4 = \{Name, Prog\}$ NOT a superkey (since [John,EEE] are not unique)
- $K_5 = \{Name, DOB\}$ is a superkey
- $K_6 = \{Name, DOB, CGPA\}$ is a superkey
- $K_7 = \{Name, CGPA\}$ is a superkey, other possibilities exist
- The last option is $K_n = \{allattributes\}$ must be a superkey, otherwise duplicate record exists.

Superkey and Candidate Key: Example

Name	Prog	DOB	CGPA
Kim	CSE	1-1-84	3.75
John	EEE	1-2-85	3.75
Kim	SWE	3-6-79	3.60
John	EEE	1-1-84	3.50

Table: Results Relation

Lets think of some possible formation of K

- $K_1 = Name, K_2 = \{Prog\}, K_3 = \{CGPA\}$ NOT superkey
- $K_4 = \{Name, Prog\}$ NOT a superkey (since [John,EEE] are not unique)
- $K_5 = \{Name, DOB\}$ is a superkey
- $K_6 = \{Name, DOB, CGPA\}$ is a superkey
- $K_7 = \{Name, CGPA\}$ is a superkey, other possibilities exist
- The last option is $K_n = \{allattributes\}$ must be a superkey, otherwise duplicate record exists.

Superkey and Candidate Key: Example

Name	Prog	DOB	CGPA
Kim	CSE	1-1-84	3.75
John	EEE	1-2-85	3.75
Kim	SWE	3-6-79	3.60
John	EEE	1-1-84	3.50

Table: Results Relation

Lets think of some possible formation of K

- $K_1 = Name, K_2 = \{Prog\}, K_3 = \{CGPA\}$ NOT superkey
- $K_4 = \{Name, Prog\}$ NOT a superkey (since [John,EEE] are not unique)
- $K_5 = \{Name, DOB\}$ is a superkey
- $K_6 = \{Name, DOB, CGPA\}$ is a superkey
- $K_7 = \{Name, CGPA\}$ is a superkey, other possibilities exist
- The last option is $K_n = \{allattributes\}$ must be a superkey, otherwise duplicate record exists.

Superkey and Candidate Key: Example

Name	Prog	DOB	CGPA
Kim	CSE	1-1-84	3.75
John	EEE	1-2-85	3.75
Kim	SWE	3-6-79	3.60
John	EEE	1-1-84	3.50

Table: Results Relation

Lets think of some possible formation of K

- $K_1 = Name, K_2 = \{Prog\}, K_3 = \{CGPA\}$ NOT superkey
- $K_4 = \{Name, Prog\}$ NOT a superkey (since [John,EEE] are not unique)
- $K_5 = \{Name, DOB\}$ is a superkey
- $K_6 = \{Name, DOB, CGPA\}$ is a superkey
- $K_7 = \{Name, CGPA\}$ is a superkey, other possibilities exist
- The last option is $K_n = \{allattributes\}$ must be a superkey, otherwise duplicate record exists.

Superkey and Candidate Key: Example

Name	Prog	DOB	CGPA
Kim	CSE	1-1-84	3.75
John	EEE	1-2-85	3.75
Kim	SWE	3-6-79	3.60
John	EEE	1-1-84	3.50

Table: Results Relation

So,

- K_5 (size is 2), K_6 (size is 3),K_7 (size is 2) are the set of superkeys
- Among them, K_5 and K_7 are the **candidate keys** since they have the minimum size (i.e. no of attributes).

Recall:
$K_5 = \{Name, DOB\}$
$K_7 = \{Name, CGPA\}$

Superkey and Candidate Key: Example

Name	Prog	DOB	CGPA
Kim	CSE	1-1-84	3.75
John	EEE	1-2-85	3.75
Kim	SWE	3-6-79	3.60
John	EEE	1-1-84	3.50

Table: Results Relation

So,

- K_5 (size is 2), K_6 (size is 3),K_7 (size is 2) are the set of superkeys
- Among them, K_5 and K_7 are the **candidate keys** since they have the minimum size (i.e. no of attributes).

Recall:

$K_5 = \{Name, DOB\}$

$K_7 = \{Name, CGPA\}$

Primary Keys: Important Notes

- By definition a Primary Key must be unique and can not be null.
- Primary Key constraint creates the primary indexing to reduce search time. Index is created automatically at the time of DDL statement.
- Format of Primary Key should be informative, non-changeable over time and efficient to implement. Often an wise trade-off is made to select the boundary between information and efficiency.

Foreign Keys

Motivation

One of the major problems of a bad database design is that it incurs data redundancy and inconsistency.

Definition

A **Foreign Key** is an attribute (or collection of attributes) in one table/relation **(r1)**, that refers to the **Primary Key** in another table/relation **(r2)**.

Here two tables or relations are needed (Self-referencing is also possible!!)
r1 is called referencing relation while r2 is the referenced relation.

Foreign Keys

Motivation

One of the major problems of a bad database design is that it incurs data redundancy and inconsistency.

Definition

A **Foreign Key** is an attribute (or collection of attributes) in one table/relation **(r1)**, that refers to the **Primary Key** in another table/relation **(r2)**.

Here two tables or relations are needed (Self-referencing is also possible!!)
r1 is called referencing relation while r2 is the referenced relation.

Foreign Key: Motivating Example

Name	Dept	Dept Location	Dept Budget	Prog	DOB	CGPA
Kim	CSE	AB2	2.5	B.Sc. CSE	1-1-84	3.75
John	EEE	AB1	2.4	B.Sc.EEE	1-2-85	3.75
Kim	CSE	AB2	2.5	B.Sc. SWE	3-6-79	3.60
John	EEE	AB1	2.4	B.Sc. EEE	1-1-84	3.50

Table: Results Relation

- It has data redundancy
- It is difficult to maintain the consistency of data. Update must be propagated to all places. For example, CSE dept budget is now 3.2, it should be updated in both records here.

Foreign Key: Motivating Example

Name	Dept	Dept Location	Dept Budget	Prog	DOB	CGPA
Kim	CSE	AB2	2.5	B.Sc. CSE	1-1-84	3.75
John	EEE	AB1	2.4	B.Sc.EEE	1-2-85	3.75
Kim	CSE	AB2	2.5	B.Sc. SWE	3-6-79	3.60
John	EEE	AB1	2.4	B.Sc. EEE	1-1-84	3.50

Table: Results Relation

- It has data redundancy
- It is difficult to maintain the consistency of data. Update must be propagated to all places. For example, CSE dept budget is now 3.2, it should be updated in both records here.

Foreign Key: Motivating Example

Name	Dept	Dept Location	Dept Budget	Prog	DOB	CGPA
Kim	CSE	AB2	2.5	B.Sc. CSE	1-1-84	3.75
John	EEE	AB1	2.4	B.Sc.EEE	1-2-85	3.75
Kim	CSE	AB2	2.5	B.Sc. SWE	3-6-79	3.60
John	EEE	AB1	2.4	B.Sc. EEE	1-1-84	3.50

Table: Results Relation

- It has data redundancy
- It is difficult to maintain the consistency of data. Update must be propagated to all places. For example, CSE dept budget is now 3.2, it should be updated in both records here.

Foreign Key: Motivating Example (cont.)

Solution is to split one larger relation in two separate relations.

Dept	Dept Location	Dept Budget
CSE	AB2	2.5
EEE	AB1	2.4

Table: Dept Relation

Name	Prog	DOB	CGPA
Kim	B.Sc. CSE	1-1-84	3.75
John	B.Sc.EEE	1-2-85	3.75
Kim	B.Sc. SWE	3-6-79	3.60
John	B.Sc. EEE	1-1-84	3.50

Table: Results Relation

- Dept Relation has fewer records, one for each department. So, dept is the primary key here.
- Results has all records but not information about department, so a **link/ pointer** is needed here.
- The link or pointer is called Foreign Key referencing Dept Relation
- Foreign Key can be NULL and it may be repeated.

Foreign Key: Motivating Example (cont.)

Solution is to split one larger relation in two separate relations.

Dept	Dept Location	Dept Budget
CSE	AB2	2.5
EEE	AB1	2.4

Table: Dept Relation

Name	Prog	DOB	CGPA	Dept
Kim	B.Sc. CSE	1-1-84	3.75	CSE
John	B.Sc.EEE	1-2-85	3.75	EEE
Kim	B.Sc. SWE	3-6-79	3.60	CSE
John	B.Sc. EEE	1-1-84	3.50	EEE

Table: Results Relation

- Dept Relation has fewer records, one for each department. So, dept is the primary key here.
- Results has all records but not information about department, so a **link/ pointer** is needed here.
- The link or pointer is called Foreign Key referencing Dept Relation
- Foreign Key can be NULL and it may be repeated.

Foreign Key: Motivating Example (cont.)

Solution is to split one larger relation in two separate relations.

Dept	Dept Location	Dept Budget
CSE	AB2	2.5
EEE	AB1	2.4

Table: Dept Relation

Name	Prog	DOB	CGPA	Dept
Kim	B.Sc. CSE	1-1-84	3.75	CSE
John	B.Sc.EEE	1-2-85	3.75	EEE
Kim	B.Sc. SWE	3-6-79	3.60	CSE
John	B.Sc. EEE	1-1-84	3.50	EEE

Table: Results Relation

- Dept Relation has fewer records, one for each department. So, dept is the primary key here.
- Results has all records but not information about department, so a **link/ pointer** is needed here.
- The link or pointer is called Foreign Key referencing Dept Relation
- Foreign Key can be NULL and it may be repeated.

Foreign Key: Motivating Example (cont.)

Solution is to split one larger relation in two separate relations.

Dept	Dept Location	Dept Budget
CSE	AB2	2.5
EEE	AB1	2.4

Table: Dept Relation

Name	Prog	DOB	CGPA	Dept
Kim	B.Sc. CSE	1-1-84	3.75	CSE
John	B.Sc.EEE	1-2-85	3.75	EEE
Kim	B.Sc. SWE	3-6-79	3.60	CSE
John	B.Sc. EEE	1-1-84	3.50	EEE

Table: Results Relation

- Dept Relation has fewer records, one for each department. So, dept is the primary key here.
- Results has all records but not information about department, so a **link/ pointer** is needed here.
- The link or pointer is called Foreign Key referencing Dept Relation
- Foreign Key can be NULL and it may be repeated.

Primary Key and Foreign Key: Properties

Primary key:

- Primary key must be unique.
- It can not be null.

Foreign key:

- Foreign key must be the Primary Key of another relation.
- Foreign key can be null.
- Foreign key can be repeated.
- Foreign key ensures data may appear from a specific source only.

Primary Key and Foreign Key: Properties

Primary key:

- Primary key must be unique.
- It can not be null.

Foreign key:

- Foreign key must be the Primary Key of another relation.
- Foreign key can be null.
- Foreign key can be repeated.
- Foreign key ensures data may appear from a specific source only.

Primary Key and Foreign Key: Properties

Primary key:

- Primary key must be unique.
- It can not be null.

Foreign key:

- Foreign key must be the Primary Key of another relation.
- Foreign key can be null.
- Foreign key can be repeated.
- Foreign key ensures data may appear from a specific source only.

Primary Key and Foreign Key: Properties

Primary key:

- Primary key must be unique.
- It can not be null.

Foreign key:

- Foreign key must be the Primary Key of another relation.
- Foreign key can be null.
- Foreign key can be repeated.
- Foreign key ensures data may appear from a specific source only.

Primary Key and Foreign Key: Properties

Primary key:

- Primary key must be unique.
- It can not be null.

Foreign key:

- Foreign key must be the Primary Key of another relation.
- Foreign key can be null.
- Foreign key can be repeated.
- Foreign key ensures data may appear from a specific source only.

Primary Key and Foreign Key: Properties

Primary key:

- Primary key must be unique.
- It can not be null.

Foreign key:

- Foreign key must be the Primary Key of another relation.
- Foreign key can be null.
- Foreign key can be repeated.
- Foreign key ensures data may appear from a specific source only.

Primary Key and Foreign Key: Final Points

- These are generally termed as Constraints, which is true for the entire life-time of the relation.
- These 2 keys are the fundamental tool to make relationship among relations/tables.

Primary Key and Foreign Key: Final Points

- These are generally termed as Constraints, which is true for the entire life-time of the relation.
- These 2 keys are the fundamental tool to make relationship among relations/tables.

Relational Algebra

Relational Algebra

- The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- There are both Unary and Binary operations.
- Although the relational algebra operations form the basis for the widely used SQL query language, database systems do not allow users to write queries in relational algebra.

Relational Algebra

Relational Algebra

- The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- There are both Unary and Binary operations.
- Although the relational algebra operations form the basis for the widely used SQL query language, database systems do not allow users to write queries in relational algebra.

Relational Algebra

Relational Algebra

- The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- There are both Unary and Binary operations.
- Although the relational algebra operations form the basis for the widely used SQL query language, database systems do not allow users to write queries in relational algebra.

Relational Algebra: Operators

Six basic operators:

(i) select σ (sigma)

(ii) project Π

(iii) union $\cup$

(iv) set difference $-$

(v) Cartesian product $\times$

(vi) rename ρ

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- It works on entire record (horizontal direction), based on the p records are returned.
- p is called the selection predicate clause where we can mention any condition.
- **Example:** select those tuples of the instructor relation where the instructor is in the "Physics" department.

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- It works on entire record (horizontal direction), based on the p records are returned.
- p is called the selection predicate clause where we can mention any condition.
- **Example:** select those tuples of the instructor relation where the instructor is in the "Physics" department.

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- It works on entire record (horizontal direction), based on the p records are returned.
- p is called the selection predicate clause where we can mention any condition.
- **Example:** select those tuples of the instructor relation where the instructor is in the "Physics" department.

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- It works on entire record (horizontal direction), based on the p records are returned.
- p is called the selection predicate clause where we can mention any condition.
- **Example:** select those tuples of the instructor relation where the instructor is in the "Physics" department.

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- It works on entire record (horizontal direction), based on the p records are returned.
- p is called the selection predicate clause where we can mention any condition.
- **Example:** select those tuples of the instructor relation where the instructor is in the "Physics" department.

Select Operation

- **Example:** select those tuples of the instructor relation where the instructor is in the "Physics" department.
- In notation: $\sigma_{dept\_name="physics"}(instructor)$

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

Figure: instructor relation

Select Operation

- **Example:** select those tuples of the instructor relation where the instructor is in the "Physics" department.
- In notation: $\sigma_{dept\_name=\text{"}physics\text{"}}(instructor)$

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

Figure: instructor relation

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

Figure: instructor relation **result of Selection**

Select Operation: Predicate

- comparisons are allows:
  $=, \neq, <, >, \leqslant, \geqq$
- Combination of connectives are allowed:
  $\wedge$ (and), $\vee$ (or), $\neg$ (not)
- An Example of predicate:
  $\sigma_{dept\_name=\text{"}physics\text{"} \wedge salary>50000}(instructor)$

Select Operation: Predicate

- comparisons are allows:
  $=, \neq, <, >, \leqslant, \geqq$
- Combination of connectives are allowed:
  $\wedge$ (and), $\vee$ (or), $\neg$ (not)
- An Example of predicate:
  $\sigma_{dept\_name = "physics" \wedge salary > 50000}(instructor)$

Select Operation: Predicate

- comparisons are allows:
  $=, \neq, <, >, \leqslant, \geqq$
- Combination of connectives are allowed:
  $\wedge$ (and), $\vee$ (or), $\neg$ (not)
- An Example of predicate:
  $\sigma_{dept\_name = "physics" \wedge salary > 50000}(instructor)$

Projection Operation

- A unary operation that returns its argument relation, with certain attributes left out (normally).
- Notation: $\Pi_{A_1, A_2 \ldots A_k}(r)$
- where A_a, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed. (works vertically)
- Duplicate rows removed from result, since relations are sets

Projection Operation

- A unary operation that returns its argument relation, with certain attributes left out (normally).
- Notation: $\Pi_{A_1, A_2 \ldots A_k}(r)$
- where A_a, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed. (works vertically)
- Duplicate rows removed from result, since relations are sets

Projection Operation

- A unary operation that returns its argument relation, with certain attributes left out (normally).
- Notation: $\Pi_{A_1,A_2 \ldots A_k}(r)$
- where A_a, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed. (works vertically)
- Duplicate rows removed from result, since relations are sets

Projection Operation

- A unary operation that returns its argument relation, with certain attributes left out (normally).
- Notation: $\Pi_{A_1, A_2 \ldots A_k}(r)$
- where A_a, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed. (works vertically)
- Duplicate rows removed from result, since relations are sets

Projection Operation

- A unary operation that returns its argument relation, with certain attributes left out (normally).
- Notation: $\Pi_{A_1, A_2 \ldots A_k}(r)$
- where A_a, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed. (works vertically)
- Duplicate rows removed from result, since relations are sets

Projection Operation: Example

- **Example:** Select ID, Name and Salary from instructor relation (i.e. erase others).
- In notation: $\Pi_{ID,name,salary}(r)$

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

Figure: instructor relation

Projection Operation: Example

- **Example:** Select ID, Name and Salary from instructor relation (i.e. erase others).
- In notation: $\Pi_{ID,name,salary}(r)$

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

Figure: instructor relation

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

Figure: instructor relation **result of Projection**, ordered as per ID

Selection and Projection: Combined

- The result of a relational-algebra operation is relation
- Both Selection and Projection are uninary operations.
- They can be combined
- Order of data processing does not matter (verify it!!)
- 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. (each result is a relation) [this principal is the key of Nested Query]

Selection and Projection: Combined

- The result of a relational-algebra operation is relation
- Both Selection and Projection are uninary operations.
- They can be combined
- Order of data processing does not matter (verify it!!)
- Consider the query Find the names of all instructors in the Physics department.
- $\pi_{name}(\sigma_{dept\_name="physics"}(instructor))$
- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation. (each result is a relation) [this principal is the key of Nested Query]

Selection and Projection: Combined

- The result of a relational-algebra operation is relation
- Both Selection and Projection are uninary operations.
- They can be combined
- Order of data processing does not matter (verify it!!)
- 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. (each result is a relation) [this principal is the key of Nested Query]

Selection and Projection: Combined

- The result of a relational-algebra operation is relation
- Both Selection and Projection are uninary operations.
- They can be combined
- Order of data processing does not matter (verify it!!)
- Consider the query Find the names of all instructors in the Physics department.
- $\pi_{name}(\sigma_{dept\_name="physics"}(instructor))$
- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation. (each result is a relation) [this principal is the key of Nested Query]

Selection and Projection: Combined

- The result of a relational-algebra operation is relation
- Both Selection and Projection are uninary operations.
- They can be combined
- Order of data processing does not matter (verify it!!)
- Consider the query Find the names of all instructors in the Physics department.
- $\pi_{name}(\sigma_{dept\_name="physics"}(instructor))$
- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation. (each result is a relation) [this principal is the key of Nested Query]

Selection and Projection: Combined

- The result of a relational-algebra operation is relation
- Both Selection and Projection are uninary operations.
- They can be combined
- Order of data processing does not matter (verify it!!)
- 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. (each result is a relation) [this principal is the key of Nested Query]

Selection and Projection: Combined

- The result of a relational-algebra operation is relation
- Both Selection and Projection are uninary operations.
- They can be combined
- Order of data processing does not matter (verify it!!)
- Consider the query Find the names of all instructors in the Physics department.
- $\pi_{name}(\sigma_{dept\_name="physics"}(instructor))$
- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation. (each result is a relation) [this principal is the key of Nested Query]

Cartesian-Product Operation

- The Cartesian-product operation (denoted by $\times$) allows us to combine information from any two relations. (all possible combinations)
- Example: the Cartesian product of the relations instructor and teaches is written as:
  instructor $\times$ teaches
- Since it results in all possible combinations: total number of tuples in the operation will be $n \times m$ where n and m are the total number of tuples in relation $r1$ and $r2$ respectively.
- In the result datasets: both meaningful and meaningless records are found.

Cartesian-Product Operation

- The Cartesian-product operation (denoted by $\times$) allows us to combine information from any two relations. (all possible combinations)
- Example: the Cartesian product of the relations instructor and teaches is written as:
  instructor $\times$ teaches
- Since it results in all possible combinations: total number of tuples in the operation will be $n \times m$ where n and m are the total number of tuples in relation $r1$ and $r2$ respectively.
- In the result datasets: both meaningful and meaningless records are found.

Cartesian-Product Operation

- The Cartesian-product operation (denoted by $\times$) allows us to combine information from any two relations. (all possible combinations)
- Example: the Cartesian product of the relations instructor and teaches is written as:
  instructor $\times$ teaches
- Since it results in all possible combinations: total number of tuples in the operation will be $n \times m$ where n and m are the total number of tuples in relation $r1$ and $r2$ respectively.
- In the result datasets: both meaningful and meaningless records are found.

Cartesian-Product Operation

- The Cartesian-product operation (denoted by $\times$) allows us to combine information from any two relations. (all possible combinations)
- Example: the Cartesian product of the relations instructor and teaches is written as:
  instructor $\times$ teaches
- Since it results in all possible combinations: total number of tuples in the operation will be $n \times m$ where n and m are the total number of tuples in relation $r1$ and $r2$ respectively.
- In the result datasets: both meaningful and meaningless records are found.

Cartesian-Product: Example

Consider the previous example:

Dept	Dept Location	Dept Budget
CSE	AB2	2.5
EEE	AB1	2.4

Table: Dept Relation

Name	Prog	DOB	CGPA	Dept
Kim	B.Sc. CSE	1-1-84	3.75	CSE
John	B.Sc.EEE	1-2-85	3.75	EEE
Kim	B.Sc. SWE	3-6-79	3.60	CSE
John	B.Sc. EEE	1-1-84	3.50	EEE

Table: Results Relation

Cartesian-Product: Example

Consider the previous example:

Dept	Dept Location	Dept Budget
CSE	AB2	2.5
EEE	AB1	2.4

Table: Dept Relation

Name	Prog	DOB	CGPA	Dept
Kim	B.Sc. CSE	1-1-84	3.75	CSE
John	B.Sc.EEE	1-2-85	3.75	EEE
Kim	B.Sc. SWE	3-6-79	3.60	CSE
John	B.Sc. EEE	1-1-84	3.50	EEE

Table: Results Relation

Name	Prog	DOB	CGPA	Dept	Dept	Dept Location	Dept Budget
Kim	B.Sc. CSE	1-1-84	3.75	CSE	CSE	AB2	2.5
Kim	B.Sc. CSE	1-1-84	3.75	CSE	EEE	AB1	2.4
John	B.Sc.EEE	1-2-85	3.75	EEE	CSE	AB2	2.5
John	B.Sc.EEE	1-2-85	3.75	EEE	EEE	AB1	2.4
Kim	B.Sc. SWE	3-6-79	3.60	CSE	CSE	AB2	2.5
Kim	B.Sc. SWE	3-6-79	3.60	CSE	EEE	AB1	2.4
John	B.Sc. EEE	1-1-84	3.50	EEE	CSE	AB2	2.5
John	B.Sc. EEE	1-1-84	3.50	EEE	EEE	AB1	2.4

Table: Resultant Tuples of *results* $\times$ *dept*

Cartesian-Product: Example

Consider the previous example:

Dept	Dept Location	Dept Budget
CSE	AB2	2.5
EEE	AB1	2.4

Table: Dept Relation

Name	Prog	DOB	CGPA	Dept
Kim	B.Sc. CSE	1-1-84	3.75	CSE
John	B.Sc.EEE	1-2-85	3.75	EEE
Kim	B.Sc. SWE	3-6-79	3.60	CSE
John	B.Sc. EEE	1-1-84	3.50	EEE

Table: Results Relation

Name	Prog	DOB	CGPA	Dept	Dept	Dept Location	Dept Budget
Kim	B.Sc. CSE	1-1-84	3.75	CSE	CSE	AB2	2.5
Kim	B.Sc. CSE	1-1-84	3.75	CSE	EEE	AB1	2.4
John	B.Sc.EEE	1-2-85	3.75	EEE	CSE	AB2	2.5
John	B.Sc.EEE	1-2-85	3.75	EEE	EEE	AB1	2.4
Kim	B.Sc. SWE	3-6-79	3.60	CSE	CSE	AB2	2.5
Kim	B.Sc. SWE	3-6-79	3.60	CSE	EEE	AB1	2.4
John	B.Sc. EEE	1-1-84	3.50	EEE	CSE	AB2	2.5
John	B.Sc. EEE	1-1-84	3.50	EEE	EEE	AB1	2.4

Table: Resultant Tuples of $results \times dept$

There are both meaningful and meaningless tuples.

Cartesian-Product: Meaningful Tuples Only

Name	Prog	DOB	CGPA	Dept	Dept	Dept Location	Dept Budget
Kim	B.Sc. CSE	1-1-84	3.75	CSE	CSE	AB2	2.5
Kim	B.Sc. CSE	1-1-84	3.75	CSE	EEE	AB1	2.4
John	B.Sc.EEE	1-2-85	3.75	EEE	CSE	AB2	2.5
John	B.Sc.EEE	1-2-85	3.75	EEE	EEE	AB1	2.4
Kim	B.Sc. SWE	3-6-79	3.60	CSE	CSE	AB2	2.5
Kim	B.Sc. SWE	3-6-79	3.60	CSE	EEE	AB1	2.4
John	B.Sc. EEE	1-1-84	3.50	EEE	CSE	AB2	2.5
John	B.Sc. EEE	1-1-84	3.50	EEE	EEE	AB1	2.4

Table: Resultant Tuples of $results \times dept$

- Notation for all tuples: dept × results
- Notation for meaningful tuples: $\sigma_{dept.dept=results.dept}(dept \times results)$
  This is the basis of Natural Join (will be covered soon)

Union Operation

- The union operation allows us to combine two relation. Selected tuples are concatenated/added back to back.
- Notation: $R \cup S$
- 2 relations are referred to as compatible relations if following 2 conditions are met:
  1. We must ensure that the input relations to the union operation have the same number of attributes; the number of attributes of a relation is referred to as its arity.
  2. When the attributes have associated types, the types of the i_{th} attributes of both input relations must be the same, for each i.

Note: Even if these 2 conditions are met, you might get erroneous result (?).

Union Operation

- The union operation allows us to combine two relation. Selected tuples are concatenated/added back to back.
- Notation: $R \cup S$
- 2 relations are referred to as compatible relations if following 2 conditions are met:
    1. We must ensure that the input relations to the union operation have the same number of attributes; the number of attributes of a relation is referred to as its arity.
    2. When the attributes have associated types, the types of the i_{th} attributes of both input relations must be the same, for each i.

Note: Even if these 2 conditions are met, you might get erroneous result (?).

Union Operation

- The union operation allows us to combine two relation. Selected tuples are concatenated/added back to back.
- Notation: $R \cup S$
- 2 relations are referred to as compatible relations if following 2 conditions are met:
  1. We must ensure that the input relations to the union operation have the same number of attributes; the number of attributes of a relation is referred to as its arity.
  2. When the attributes have associated types, the types of the i_{th} attributes of both input relations must be the same, for each i.

Note: Even if these 2 conditions are met, you might get erroneous result (?).

Union Operation

- The union operation allows us to combine two relation. Selected tuples are concatenated/added back to back.
- Notation: $R \cup S$
- 2 relations are referred to as compatible relations if following 2 conditions are met:
    1. We must ensure that the input relations to the union operation have the same number of attributes; the number of attributes of a relation is referred to as its arity.
    2. When the attributes have associated types, the types of the i_{th} attributes of both input relations must be the same, for each i.

Note: Even if these 2 conditions are met, you might get erroneous result (?).

Other Operations

- The intersection operation, denoted by $\cap$, allows us to find tuples that are in both the input relations.
- The set-difference operation, denoted by $-$, allows us to find tuples that are in one relation but are not in another.
- It is useful in some cases to give them names; the rename operator, denoted by the lowercase Greek letter rho ρ, lets us do this.
  Notation: $\rho_X(E)$
  It returns the result of expression E under the name x.

Other Operations

- The intersection operation, denoted by $\cap$, allows us to find tuples that are in both the input relations.
- The set-difference operation, denoted by $-$, allows us to find tuples that are in one relation but are not in another.
- It is useful in some cases to give them names; the rename operator, denoted by the lowercase Greek letter rho ρ, lets us do this.
  Notation: $\rho_x(E)$
  It returns the result of expression E under the name x.

Other Operations

- The intersection operation, denoted by $\cap$, allows us to find tuples that are in both the input relations.
- The set-difference operation, denoted by $-$, allows us to find tuples that are in one relation but are not in another.
- It is useful in some cases to give them names; the rename operator, denoted by the lowercase Greek letter rho ρ, lets us do this.
  Notation: $\rho_x(E)$
  It returns the result of expression E under the name x.

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- **Example:** Find information about courses taught by instructors in the Physics department with salary greater than 70,000
- Query 1 : Apply both condition at the same-time
  $\sigma_{dept\_name=\text{"}Physics\text{"} \wedge salary>70000}(instructor)$
- Query 2 : Apply condition 1(salary) first and then apply condition 2(dept) on this result-set.
  $\sigma_{dept\_name=\text{"}Physics\text{"}}(\sigma_{salary>70000}(instructor))$
- The two queries are not identical; they are, however, equivalent they give the same result on any database

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- **Example:** Find information about courses taught by instructors in the Physics department with salary greater than 70,000
- Query 1 : Apply both condition at the same-time
  $\sigma_{dept\_name=\text{"}Physics\text{"} \wedge salary>70000}(instructor)$
- Query 2 : Apply condition 1(salary) first and then apply condition 2(dept) on this result-set.
  $\sigma_{dept\_name=\text{"}Physics\text{"}}(\sigma_{salary>70000}(instructor))$
- The two queries are not identical; they are, however, equivalent they give the same result on any database

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- **Example:** Find information about courses taught by instructors in the Physics department with salary greater than 70,000
- Query 1 : Apply both condition at the same-time
  $\sigma_{dept\_name=\text{"}Physics\text{"} \wedge salary>70000}(instructor)$
- Query 2 : Apply condition 1(salary) first and then apply condition 2(dept) on this result-set.
  $\sigma_{dept\_name=\text{"}Physics\text{"}}(\sigma_{salary>70000}(instructor))$
- The two queries are not identical; they are, however, equivalent they give the same result on any database

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- **Example:** Find information about courses taught by instructors in the Physics department with salary greater than 70,000
- Query 1 : Apply both condition at the same-time
  $\sigma_{dept\_name="Physics" \wedge salary>70000}(instructor)$
- Query 2 : Apply condition 1(salary) first and then apply condition 2(dept) on this result-set.
  $\sigma_{dept\_name="Physics"}(\sigma_{salary>70000}(instructor))$
- The two queries are not identical; they are, however, equivalent they give the same result on any database

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- **Example:** Find information about courses taught by instructors in the Physics department with salary greater than 70,000
- Query 1 : Apply both condition at the same-time
  $\sigma_{dept\_name=\text{"}Physics\text{"} \wedge salary>70000}(instructor)$
- Query 2 : Apply condition 1(salary) first and then apply condition 2(dept) on this result-set.
  $\sigma_{dept\_name=\text{"}Physics\text{"}}(\sigma_{salary>70000}(instructor))$
- The two queries are not identical; they are, however, equivalent they give the same result on any database

Example

```
// primary key clause is used

create table depts
(dept varchar2(20) primary key,
 budget number,
 location varchar2(20)
);

// here is how we can create foreign key

create table students(
name varchar2(30),
dob date,
cgpa number,
deptinfo varchar2(20) foreign key references depts[dept]
);
```

Example; Self Reference

```
create table emp(
ID number primary key,
name varchar2(30),
designation varchar2(20),
salary number,
IBID number foreign key references emp[ID]
);
```

End of Chapter 2

Thank You