Databases (BDD) — Chapter 2

The Relational Model

Fatiha Benabderrahmane

Department of Computer Science, Mila University

February 9, 2026

Outline

Relational Model Foundations

The Relational Model

The **relational model** represents data using a single concept: the **relation**.

In this model:

- Data is stored in tables

The Relational Model

The **relational model** represents data using a single concept: the **relation**.

In this model:

- Data is stored in tables
- Each table has a fixed structure

The Relational Model

The **relational model** represents data using a single concept: the **relation**.

In this model:

- Data is stored in tables
- Each table has a fixed structure
- The model is based on set theory

The Relational Model

The **relational model** represents data using a single concept: the **relation**.

In this model:

- Data is stored in tables
- Each table has a fixed structure
- The model is based on set theory

The Relational Model

The **relational model** represents data using a single concept: the **relation**.

In this model:

- Data is stored in tables
- Each table has a fixed structure
- The model is based on set theory

The relational model is simple, rigorous, and widely used in modern DBMS.

Relation

A **relation** is a two-dimensional table composed of:

- **Attributes** (columns)

Relation

A **relation** is a two-dimensional table composed of:

- **Attributes** (columns)
- **Tuples** (rows)

Relation

A **relation** is a two-dimensional table composed of:

- **Attributes** (columns)
- **Tuples** (rows)

Relation

A **relation** is a two-dimensional table composed of:

- **Attributes** (columns)
- **Tuples** (rows)

RELATION: STUDENT

StudentID	LastName	FirstName	Age
125	Amara	Leila	19
203	Ghichi	Faiza	20
65	Bey	Sami	20

Relation

A **relation** is a two-dimensional table composed of:

- **Attributes** (columns)
- **Tuples** (rows)

RELATION: STUDENT

StudentID	LastName	FirstName	Age
125	Amara	Leila	19
203	Ghichi	Faiza	20
65	Bey	Sami	20

Each relation has an **identifier** (key) that uniquely identifies its tuples.

Example: STUDENT Relation

STUDENT

Example: STUDENT Relation

STUDENT

StudentID	LastName	FirstName	Age
125	Amara	Leila	19
203	Ghichi	Faiza	20
65	Bey	Sami	20

Example: STUDENT Relation

STUDENT

StudentID	LastName	FirstName	Age
125	Amara	Leila	19
203	Ghichi	Faiza	20
65	Bey	Sami	20

Example tuple: $\langle 203,\ \textit{Ghichi},\ \textit{Faiza},\ 20 \rangle$

Attribute

An **attribute** represents a property of a relation.

Attribute

An **attribute** represents a property of a relation.

In the STUDENT relation, attributes are:

- StudentID

Attribute

An **attribute** represents a property of a relation.

In the STUDENT relation, attributes are:

- StudentID
- LastName

Attribute

An **attribute** represents a property of a relation.

In the STUDENT relation, attributes are:

- StudentID
- LastName
- FirstName

Attribute

An **attribute** represents a property of a relation.

In the STUDENT relation, attributes are:

- StudentID
- LastName
- FirstName
- Age

Attribute

An **attribute** represents a property of a relation.

In the STUDENT relation, attributes are:

- StudentID
- LastName
- FirstName
- Age

Attribute

An **attribute** represents a property of a relation.

In the STUDENT relation, attributes are:

- StudentID
- LastName
- FirstName
- Age

Each attribute has exactly one value per tuple.

Domain

A **domain** defines the set of allowed values for an attribute.

Domain

A **domain** defines the set of allowed values for an attribute.

Attribute	Domain
StudentID	integer in [0..300]
Name	string (max length 20)
Age	integer in [16..60]

Domain

A **domain** defines the set of allowed values for an attribute.

Attribute	Domain
StudentID	integer in [0..300]
Name	string (max length 20)
Age	integer in [16..60]

Domains enforce **domain integrity**.

Identifier (Key)

An **identifier** (or key) is an attribute or set of attributes that uniquely identifies a tuple.

Identifier (Key)

An **identifier** (or key) is an attribute or set of attributes that uniquely identifies a tuple.

In the STUDENT relation:

- StudentID is the identifier

Identifier (Key)

An **identifier** (or key) is an attribute or set of attributes that uniquely identifies a tuple.

In the STUDENT relation:

- StudentID is the identifier
- No two students share the same StudentID

Associations as Relations

Associations between entities are also represented as relations.

Associations as Relations

Associations between entities are also represented as relations.

For a many-to-many association, a separate relation is created.

Associations as Relations

Associations between entities are also represented as relations.

For a many-to-many association, a separate relation is created.

TAKES

StudentID	CourseName
365	Algo
65	OS
125	Algo
125	OS

Associations as Relations

Associations between entities are also represented as relations.

For a many-to-many association, a separate relation is created.

TAKES

StudentID	CourseName
365	Algo
65	OS
125	Algo
125	OS

Identifier: (StudentID, CourseName)

Relation Schema

The **relation schema** describes the structure of a relation.

Relation Schema

The **relation schema** describes the structure of a relation.

Example schema:

$$\textsc{Student}(\mathit{StudentID}{:}\mathit{int},\ \mathit{LastName}{:}\mathit{string},\ \mathit{FirstName}{:}\mathit{string},\ \mathit{Age}{:}\mathit{int})$$

Relation Schema

The **relation schema** describes the structure of a relation.

Example schema:

$$\textsc{Student}(\mathit{StudentID}{:}\mathit{int},\ \mathit{LastName}{:}\mathit{string},\ \mathit{FirstName}{:}\mathit{string},\ \mathit{Age}{:}\mathit{int})$$

The schema is also called the **intension**.

Relation Population

The **population** of a relation is the set of tuples currently stored.

Relation Population

The **population** of a relation is the set of tuples currently stored.

Properties:

- No duplicate tuples

Relation Population

The **population** of a relation is the set of tuples currently stored.

Properties:

- No duplicate tuples
- No order between tuples

Relation Population

The **population** of a relation is the set of tuples currently stored.

Properties:

- No duplicate tuples
- No order between tuples

Relation Population

The **population** of a relation is the set of tuples currently stored.

Properties:

- No duplicate tuples
- No order between tuples

The population is also called the **extension**.

Schema vs Population

A **schema** describes the structure of a relation (columns). A **population** is the set of tuples currently stored (rows).

Schema vs Population

A **schema** describes the structure of a relation (columns). A **population** is the set of tuples currently stored (rows).

Schema (structure only)

StudentID	LastName	FirstName

Population (current tuples)

StudentID	LastName	FirstName
125	Amara	Leila
203	Ghichi	Faiza
65	Bey	Sami

Database Schema

The **database schema** is the set of all relation schemas in the database.

Database Schema

The **database schema** is the set of all relation schemas in the database.

Example:

- STUDENT

Database Schema

The **database schema** is the set of all relation schemas in the database.

Example:

- STUDENT
- COURSE

Database Schema

The **database schema** is the set of all relation schemas in the database.

Example:

- STUDENT
- COURSE
- TAKES

Attribute Modeling Rule

In the relational model, attributes must be:

- Atomic

Attribute Modeling Rule

In the relational model, attributes must be:

- Atomic
- Single-valued

Attribute Modeling Rule

In the relational model, attributes must be:

- Atomic
- Single-valued

Attribute Modeling Rule

In the relational model, attributes must be:

- Atomic
- Single-valued

Complex or multivalued attributes must be transformed.

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Example:

$$Address = Number + Street + City + PostalCode + Country$$

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Example:

$$Address = Number + Street + City + PostalCode + Country$$

Option 1: store the complex attribute as a single value

- STUDENT(StudentID, LastName, Address)

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Example:

$$Address = Number + Street + City + PostalCode + Country$$

Option 1: store the complex attribute as a single value

- STUDENT(StudentID, LastName, Address)
- Address stored as a single string

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Example:

$$Address = Number + Street + City + PostalCode + Country$$

Option 1: store the complex attribute as a single value

- STUDENT(StudentID, LastName, Address)
- Address stored as a single string
- Simple design, but components cannot be queried separately

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Example:

$$Address = Number + Street + City + PostalCode + Country$$

Option 1: store the complex attribute as a single value

- STUDENT(StudentID, LastName, Address)
- Address stored as a single string
- Simple design, but components cannot be queried separately

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Example:

$$Address = Number + Street + City + PostalCode + Country$$

Option 1: store the complex attribute as a single value

- STUDENT(StudentID, LastName, Address)
- Address stored as a single string
- Simple design, but components cannot be queried separately

Option 2: decompose into atomic attributes (recommended)

- STUDENT(StudentID, LastName, Number, Street, City, PostalCode, Country)

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Example:

$$Address = Number + Street + City + PostalCode + Country$$

Option 1: store the complex attribute as a single value

- STUDENT(StudentID, LastName, Address)
- Address stored as a single string
- Simple design, but components cannot be queried separately

Option 2: decompose into atomic attributes (recommended)

- STUDENT(StudentID, LastName, Number, Street, City, PostalCode, Country)
- Each component becomes an atomic attribute

Complex Attribute

A **complex attribute** is composed of several sub-parts.

Example:

$$Address = Number + Street + City + PostalCode + Country$$

Option 1: store the complex attribute as a single value

- STUDENT(StudentID, LastName, Address)
- Address stored as a single string
- Simple design, but components cannot be queried separately

Option 2: decompose into atomic attributes (recommended)

- STUDENT(StudentID, LastName, Number, Street, City, PostalCode, Country)
- Each component becomes an atomic attribute
- Enables queries on City, Country, etc.

Multivalued Attribute

A **multivalued attribute** can take several values for the same tuple.

Multivalued Attribute

A **multivalued attribute** can take several values for the same tuple.

Example: a student may have several first names.

Multivalued Attribute

A **multivalued attribute** can take several values for the same tuple.

Example: a student may have several first names.

Option 1 (NOT recommended): multiple columns in the same relation

- STUDENT(StudentID, LastName, FirstName1, FirstName2, . . .)

Multivalued Attribute

A **multivalued attribute** can take several values for the same tuple.

Example: a student may have several first names.

Option 1 (NOT recommended): multiple columns in the same relation

- STUDENT(StudentID, LastName, FirstName1, FirstName2, . . .)
- Problems: unknown maximum number, many NULLs, wasted space, update anomalies

Multivalued Attribute

A **multivalued attribute** can take several values for the same tuple.

Example: a student may have several first names.

Option 1 (NOT recommended): multiple columns in the same relation

- STUDENT(StudentID, LastName, FirstName1, FirstName2, . . .)
- Problems: unknown maximum number, many NULLs, wasted space, update anomalies

Multivalued Attribute

A **multivalued attribute** can take several values for the same tuple.

Example: a student may have several first names.

Option 1 (NOT recommended): multiple columns in the same relation

- STUDENT(StudentID, LastName, FirstName1, FirstName2, ...)
- Problems: unknown maximum number, many NULLs, wasted space, update anomalies

Option 2 (Recommended): create a separate relation

$$\textsc{Student\_FirstName}(\textit{StudentID},\ \textit{FirstName})$$

Multivalued Attribute

A **multivalued attribute** can take several values for the same tuple.

Example: a student may have several first names.

Option 1 (NOT recommended): multiple columns in the same relation

- STUDENT(StudentID, LastName, FirstName1, FirstName2, ...)
- Problems: unknown maximum number, many NULLs, wasted space, update anomalies

Option 2 (Recommended): create a separate relation

STUDENT_FIRSTNAME(*StudentID*, *FirstName*)

StudentID	FirstName
INF100	Mohammed
INF100	Redha
INF120	Yasmine

Multivalued Attribute

A **multivalued attribute** can take several values for the same tuple.

Example: a student may have several first names.

Option 1 (NOT recommended): multiple columns in the same relation

- STUDENT(StudentID, LastName, FirstName1, FirstName2, ...)
- Problems: unknown maximum number, many NULLs, wasted space, update anomalies

Option 2 (Recommended): create a separate relation

STUDENT_FIRSTNAME(*StudentID*, *FirstName*)

StudentID	FirstName
INF100	Mohammed
INF100	Redha
INF120	Yasmine

Optional Attribute (NULL)

Some attributes may have no value for certain tuples.

Optional Attribute (NULL)

Some attributes may have no value for certain tuples.

StudentID	LastName	FirstName	Age
INF200	Mohammed	Ahmed	NULL

Optional Attribute (NULL)

Some attributes may have no value for certain tuples.

StudentID	LastName	FirstName	Age
INF200	Mohammed	Ahmed	NULL

NULL represents an unknown or missing value.

Section Summary

- Data is represented using relations (tables)

Section Summary

- Data is represented using relations (tables)
- Relations are defined by attributes, domains, and keys

Section Summary

- Data is represented using relations (tables)
- Relations are defined by attributes, domains, and keys
- Schemas describe structure; populations describe data

Section Summary

- Data is represented using relations (tables)
- Relations are defined by attributes, domains, and keys
- Schemas describe structure; populations describe data
- Attributes must be atomic and single-valued

Normalization and Integrity

Why Normalization?

A poorly designed relational schema leads to data anomalies.

Why Normalization?

A poorly designed relational schema leads to data anomalies.
Typical problems:

- Data redundancy

Why Normalization?

A poorly designed relational schema leads to data anomalies.
Typical problems:

- Data redundancy
- Update anomalies

Why Normalization?

A poorly designed relational schema leads to data anomalies.
Typical problems:

- Data redundancy
- Update anomalies
- Insertion anomalies

Why Normalization?

A poorly designed relational schema leads to data anomalies.
Typical problems:

- Data redundancy
- Update anomalies
- Insertion anomalies
- Deletion anomalies

Why Normalization?

A poorly designed relational schema leads to data anomalies.
Typical problems:

- Data redundancy
- Update anomalies
- Insertion anomalies
- Deletion anomalies

Why Normalization?

A poorly designed relational schema leads to data anomalies.
Typical problems:

- Data redundancy
- Update anomalies
- Insertion anomalies
- Deletion anomalies

Normalization is the process that produces **non-redundant** and **consistent** relational schemas.

Motivating Example

A poorly designed relational schema leads to data problems.

Motivating Example

A poorly designed relational schema leads to data problems.

Example:

$$\text{EMP\_PROJECT}(\textit{EmpID}, \textit{EmpName}, \textit{Project}, \textit{Dept})$$

Motivating Example

A poorly designed relational schema leads to data problems.
Example:

$$\mathrm{EMP\_PROJECT}(\textit{EmpID}, \textit{EmpName}, \textit{Project}, \textit{Dept})$$

Problems observed:

- EmpName and Dept are repeated

Motivating Example

A poorly designed relational schema leads to data problems.

Example:

$$\text{EMP\_PROJECT}(\textit{EmpID}, \textit{EmpName}, \textit{Project}, \textit{Dept})$$

Problems observed:

- EmpName and Dept are repeated
- Updating a department requires multiple updates

Running Example: EMPLOYEE_PROJECT

We start with a single relation that mixes several concepts.

Running Example: EMPLOYEE_PROJECT

We start with a single relation that mixes several concepts.

EMPLOYEE_PROJECT

EmpID	EmpName	Project	StartDate	EndDate	Department
E01	Mohamed	Alpha	01/02/24	30/06/24	IT
E01	Mohamed	Beta	15/03/24	31/12/24	IT
E02	Salima	Alpha	01/02/24	30/06/24	Finance
E03	Yacine	Gamma	10/01/24	30/09/24	IT

Running Example: EMPLOYEE_PROJECT

We start with a single relation that mixes several concepts.

EMPLOYEE_PROJECT

EmpID	EmpName	Project	StartDate	EndDate	Department
E01	Mohamed	Alpha	01/02/24	30/06/24	IT
E01	Mohamed	Beta	15/03/24	31/12/24	IT
E02	Salima	Alpha	01/02/24	30/06/24	Finance
E03	Yacine	Gamma	10/01/24	30/09/24	IT

This relation stores employee data, project data, and assignments together.

Data Redundancy

In the EMPLOYEE_PROJECT relation:

Data Redundancy

In the EMPLOYEE_PROJECT relation:

- Employee name and department are repeated for each project

Data Redundancy

In the EMPLOYEE_PROJECT relation:

- Employee name and department are repeated for each project
- Project information is repeated for each employee

Data Redundancy

In the EMPLOYEE_PROJECT relation:

- Employee name and department are repeated for each project
- Project information is repeated for each employee

Data Redundancy

In the EMPLOYEE_PROJECT relation:

- Employee name and department are repeated for each project
- Project information is repeated for each employee

Example:

- EmpName = Mohamed appears several times

Data Redundancy

In the EMPLOYEE_PROJECT relation:

- Employee name and department are repeated for each project
- Project information is repeated for each employee

Example:

- EmpName = Mohamed appears several times
- Department = IT is duplicated

Update Anomaly

An **update anomaly** occurs when the same information must be updated in several places.

Update Anomaly

An **update anomaly** occurs when the same information must be updated in several places.

Example:

- If employee Martin changes department

Update Anomaly

An **update anomaly** occurs when the same information must be updated in several places.

Example:

- If employee Martin changes department
- We must update several rows

Update Anomaly

An **update anomaly** occurs when the same information must be updated in several places.

Example:

- If employee Martin changes department
- We must update several rows
- Forgetting one row leads to inconsistency

Insertion Anomaly

An **insertion anomaly** occurs when we cannot insert data without other data.

Insertion Anomaly

An **insertion anomaly** occurs when we cannot insert data without other data.

Example:

- A new employee with no project

Insertion Anomaly

An **insertion anomaly** occurs when we cannot insert data without other data.

Example:

- A new employee with no project
- Cannot be inserted into EMPLOYEE_PROJECT

Deletion Anomaly

A **deletion anomaly** occurs when deleting data causes loss of other data.

Deletion Anomaly

A **deletion anomaly** occurs when deleting data causes loss of other data.

Example:

- If project Gamma is deleted

Deletion Anomaly

A **deletion anomaly** occurs when deleting data causes loss of other data.

Example:

- If project Gamma is deleted
- Employee Yacine information is lost

Functional Dependency

A **functional dependency** $X \rightarrow Y$ means:

Functional Dependency

A **functional dependency** $X \rightarrow Y$ means:

- if two tuples agree on X

Functional Dependency

A **functional dependency** $X \rightarrow Y$ means:

- if two tuples agree on X
- then they must agree on Y

Functional Dependency

A **functional dependency** $X \rightarrow Y$ means:

- if two tuples agree on X
- then they must agree on Y

Functional Dependency

A **functional dependency** $X \rightarrow Y$ means:

- if two tuples agree on X
- then they must agree on Y

Example:

$$EmpID \rightarrow EmpName$$

An employee identifier determines the employee name.

Functional Dependencies: Example

Relation:

$$\text{EMPLOYEE}(\textit{EmpID}, \textit{EmpName}, \textit{Dept}, \textit{DeptPhone})$$

Functional Dependencies: Example

Relation:

$$\mathrm{EMPLOYEE}(\mathit{EmpID}, \mathit{EmpName}, \mathit{Dept}, \mathit{DeptPhone})$$

Valid functional dependencies:

- $\mathit{EmpID} \rightarrow \mathit{EmpName}, \mathit{Dept}$

Functional Dependencies: Example

Relation:

$$\text{EMPLOYEE}(\textit{EmpID}, \textit{EmpName}, \textit{Dept}, \textit{DeptPhone})$$

Valid functional dependencies:

- $\textit{EmpID} \rightarrow \textit{EmpName}, \textit{Dept}$
- $\textit{Dept} \rightarrow \textit{DeptPhone}$

Functional Dependencies: Example

Relation:

$$\text{EMPLOYEE}(\textit{EmpID}, \textit{EmpName}, \textit{Dept}, \textit{DeptPhone})$$

Valid functional dependencies:

- $\textit{EmpID} \rightarrow \textit{EmpName}, \textit{Dept}$
- $\textit{Dept} \rightarrow \textit{DeptPhone}$

Functional Dependencies: Example

Relation:

$$\mathrm{EMPLOYEE}(\mathit{EmpID}, \mathit{EmpName}, \mathit{Dept}, \mathit{DeptPhone})$$

Valid functional dependencies:

- $\mathit{EmpID} \rightarrow \mathit{EmpName}, \mathit{Dept}$
- $\mathit{Dept} \rightarrow \mathit{DeptPhone}$

Invalid dependency:

$$\mathit{EmpName} \nrightarrow \mathit{Dept}$$

Functional Dependency

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema.

Functional Dependency

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema.
A **functional dependency** (FD) is written:

$$A_1, A_2, \ldots, A_n \rightarrow B$$

Functional Dependency

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema.
A **functional dependency** (FD) is written:

$$A_1, A_2, \ldots, A_n \rightarrow B$$

Meaning:

- If two tuples have the same values for $A_1, \ldots, A_n$

Functional Dependency

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema.
A **functional dependency** (FD) is written:

$$A_1, A_2, \ldots, A_n \rightarrow B$$

Meaning:

- If two tuples have the same values for $A_1, \ldots, A_n$
- Then they must have the same value for B

Functional Dependency

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema.
A **functional dependency** (FD) is written:

$$A_1, A_2, \ldots, A_n \rightarrow B$$

Meaning:

- If two tuples have the same values for $A_1, \ldots, A_n$
- Then they must have the same value for B

Functional Dependency

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema.
A **functional dependency** (FD) is written:

$$A_1, A_2, \ldots, A_n \rightarrow B$$

Meaning:

- If two tuples have the same values for $A_1, \ldots, A_n$
- Then they must have the same value for B

Example:

$$Num\_Employe \rightarrow Nom,\ Prenom$$

Armstrong's Axioms

Let X, Y, Z, W be sets of attributes.

Armstrong's Axioms

Let X, Y, Z, W be sets of attributes.

Reflexivity:

$$Y \subseteq X \Rightarrow X \rightarrow Y$$

Armstrong's Axioms

Let X, Y, Z, W be sets of attributes.

Reflexivity:

$$Y \subseteq X \Rightarrow X \rightarrow Y$$

Augmentation:

$$X \rightarrow Y \Rightarrow XZ \rightarrow YZ$$

Armstrong's Axioms

Let X, Y, Z, W be sets of attributes.

Reflexivity:

$$Y \subseteq X \Rightarrow X \rightarrow Y$$

Augmentation:

$$X \rightarrow Y \Rightarrow XZ \rightarrow YZ$$

Transitivity:

$$X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$$

Armstrong's Axioms

Reflexivity:

Armstrong's Axioms

Reflexivity:

$$\{EmpID, EmpName\} \rightarrow EmpID$$

Armstrong's Axioms

Reflexivity:

$$\{EmpID, EmpName\} \rightarrow EmpID$$

Augmentation:

Armstrong's Axioms

Reflexivity:

$$\{EmpID, EmpName\} \rightarrow EmpID$$

Augmentation:

If:

$$EmpID \rightarrow Dept$$

Then:

$$EmpID, Project \rightarrow Dept$$

Armstrong's Axioms

Reflexivity:

$$\{EmpID, EmpName\} \rightarrow EmpID$$

Augmentation:
If:

$$EmpID \rightarrow Dept$$

Then:

$$EmpID, Project \rightarrow Dept$$

Transitivity:

Armstrong's Axioms

Reflexivity:

$$\{EmpID, EmpName\} \rightarrow EmpID$$

Augmentation:
If:

$$EmpID \rightarrow Dept$$

Then:

$$EmpID, Project \rightarrow Dept$$

Transitivity:

$$EmpID \rightarrow Dept \land Dept \rightarrow DeptPhone \Rightarrow EmpID \rightarrow DeptPhone$$

Derived Inference Rules

From Armstrong's axioms, we derive:

Derived Inference Rules

From Armstrong's axioms, we derive:

- **Union:** $\{X \to Y, X \to Z\} \Rightarrow X \to YZ$

Derived Inference Rules

From Armstrong's axioms, we derive:

- **Union:** $\{X \to Y, X \to Z\} \Rightarrow X \to YZ$
- **Decomposition:** $X \to YZ \Rightarrow X \to Y$ and $X \to Z$

Derived Inference Rules

From Armstrong's axioms, we derive:

- **Union:** $\{X \to Y, X \to Z\} \Rightarrow X \to YZ$
- **Decomposition:** $X \to YZ \Rightarrow X \to Y$ and $X \to Z$
- **Composition:** $X \to Y,\ Z \to W \Rightarrow XZ \to YW$

Derived Inference Rules

From Armstrong's axioms, we derive:

- **Union:** $\{X \to Y, X \to Z\} \Rightarrow X \to YZ$
- **Decomposition:** $X \to YZ \Rightarrow X \to Y$ and $X \to Z$
- **Composition:** $X \to Y,\ Z \to W \Rightarrow XZ \to YW$
- **Pseudo-transitivity:**

$$X \to Y,\ WY \to Z \Rightarrow XW \to Z$$

Inference Rules

Union:

Inference Rules

Union:
If:

$$EmpID \rightarrow EmpName \quad \text{and} \quad EmpID \rightarrow Dept$$

Then:

$$EmpID \rightarrow EmpName, Dept$$

Inference Rules

Union:
If:

$$EmpID \rightarrow EmpName \quad \text{and} \quad EmpID \rightarrow Dept$$

Then:

$$EmpID \rightarrow EmpName, Dept$$

Decomposition:

Inference Rules

Union:
If:

$$EmpID \rightarrow EmpName \quad \text{and} \quad EmpID \rightarrow Dept$$

Then:

$$EmpID \rightarrow EmpName, Dept$$

Decomposition:

$$EmpID \rightarrow EmpName, Dept \Rightarrow \begin{cases} EmpID \rightarrow EmpName \\ EmpID \rightarrow Dept \end{cases}$$

Types of Functional Dependencies

1) Trivial FD

Types of Functional Dependencies

1) Trivial FD

$X \to Y$ is trivial if $Y \subseteq X$.

Types of Functional Dependencies

1) Trivial FD

$X \rightarrow Y$ is trivial if $Y \subseteq X$.

Example: $(CodeProd, CodeDep) \rightarrow CodeProd$

Types of Functional Dependencies

1) Trivial FD

$X \to Y$ is trivial if $Y \subseteq X$.

Example: $(CodeProd, CodeDep) \to CodeProd$

2) Canonical FD

Types of Functional Dependencies

1) Trivial FD

$X \rightarrow Y$ is trivial if $Y \subseteq X$.

Example: $(CodeProd, CodeDep) \rightarrow CodeProd$

2) Canonical FD

An FD is canonical if its right-hand side has a single attribute: $X \rightarrow A$.

Types of Functional Dependencies

1) Trivial FD

$X \rightarrow Y$ is trivial if $Y \subseteq X$.

Example: $(CodeProd, CodeDep) \rightarrow CodeProd$

2) Canonical FD

An FD is canonical if its right-hand side has a single attribute: $X \rightarrow A$.

Example: $CodeProd \rightarrow (Label, Price)$ is not canonical.

Types of Functional Dependencies

1) Trivial FD

$X \to Y$ is trivial if $Y \subseteq X$.

Example: $(CodeProd, CodeDep) \to CodeProd$

2) Canonical FD

An FD is canonical if its right-hand side has a single attribute: $X \to A$.

Example: $CodeProd \to (Label, Price)$ is not canonical.

By decomposition:

$$CodeProd \to Label \qquad \text{and} \qquad CodeProd \to Price$$

Types of Functional Dependencies

Types of Functional Dependencies

3) Elementary FD A canonical FD $X \rightarrow A$ is **elementary** if no proper subset of X determines A.

Types of Functional Dependencies

3) Elementary FD A canonical FD $X \to A$ is **elementary** if no proper subset of X determines A.

Example (not elementary):

$$(CodeProd, CodeDep) \to Label$$

is not elementary if

$$CodeProd \to Label$$

holds (because $CodeProd \subset (CodeProd, CodeDep)$).

Direct Functional Dependency

An FD $X \to Y$ is **direct** if:

Direct Functional Dependency

An FD $X \to Y$ is **direct** if:

- it is elementary, and

Direct Functional Dependency

An FD $X \to Y$ is **direct** if:

- it is elementary, and
- Y does not depend on X by transitivity

Direct Functional Dependency

An FD $X \to Y$ is **direct** if:

- it is elementary, and
- Y does not depend on X by transitivity

Direct Functional Dependency

An FD $X \to Y$ is **direct** if:

- it is elementary, and
- Y does not depend on X by transitivity

Equivalently: there is **no** attribute set Z such that

$$X \to Z \quad \text{and} \quad Z \to Y.$$

Direct Functional Dependency

An FD $X \to Y$ is **direct** if:

- it is elementary, and
- Y does not depend on X by transitivity

Equivalently: there is **no** attribute set Z such that

$$X \to Z \quad \text{and} \quad Z \to Y.$$

Example:

Direct Functional Dependency

An FD $X \to Y$ is **direct** if:

- it is elementary, and
- Y does not depend on X by transitivity

Equivalently: there is **no** attribute set Z such that

$$X \to Z \quad \text{and} \quad Z \to Y.$$

Example:

If

$$EmpID \to Dept \qquad \text{and} \qquad Dept \to DeptPhone$$

then

$$EmpID \to DeptPhone$$

is **not direct** (it is obtained by transitivity).

Direct Functional Dependency

An FD $X \to Y$ is **direct** if:

- it is elementary, and
- Y does not depend on X by transitivity

Equivalently: there is **no** attribute set Z such that

$$X \to Z \quad \text{and} \quad Z \to Y.$$

Example:

If

$$EmpID \to Dept \qquad \text{and} \qquad Dept \to DeptPhone$$

then

$$EmpID \to DeptPhone$$

is **not direct** (it is obtained by transitivity).

So the direct dependencies here are:

$$EmpID \to Dept \qquad \text{and} \qquad Dept \to DeptPhone.$$

Functional Dependency Graph

A **functional dependency graph** is a directed graph where:

Functional Dependency Graph

A **functional dependency graph** is a directed graph where:

- Nodes represent attributes

Functional Dependency Graph

A **functional dependency graph** is a directed graph where:

- Nodes represent attributes
- Arrows represent functional dependencies

Functional Dependency Graph

A **functional dependency graph** is a directed graph where:

- Nodes represent attributes
- Arrows represent functional dependencies

Functional Dependency Graph

A **functional dependency graph** is a directed graph where:

- Nodes represent attributes
- Arrows represent functional dependencies

Example:

$$F = \{A \to B,\ A \to D,\ AE \to C\}$$

Functional Dependency Graph

A **functional dependency graph** is a directed graph where:

- Nodes represent attributes
- Arrows represent functional dependencies

Example:

$$F = \{A \to B,\ A \to D,\ AE \to C\}$$

The graph below represents these dependencies (the FD $AE \to C$ is shown as a joint dependency from A and E).

Functional Dependency Graph

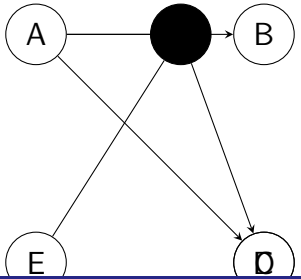
A **functional dependency graph** is a directed graph where:

- Nodes represent attributes
- Arrows represent functional dependencies

Example:

$$F = \{A \to B,\ A \to D,\ AE \to C\}$$

The graph below represents these dependencies (the FD $AE \to C$ is shown as a joint dependency from A and E).



FD Graph and Minimal FD Graph

Example: $F = \{X \to Y,\ Y \to Z,\ X \to Z\}$

FD Graph and Minimal FD Graph

Example: $F = \{X \to Y,\ Y \to Z,\ X \to Z\}$
Non-minimal graph contains a redundant edge $X \to Z$ (implied by transitivity).

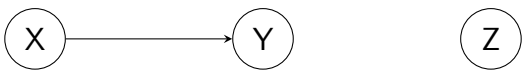
FD Graph and Minimal FD Graph

Example: $F = \{X \to Y,\ Y \to Z,\ X \to Z\}$
Non-minimal graph contains a redundant edge $X \to Z$ (implied by transitivity).

The dashed edge is removed in the **minimal** FD graph: keep only $X \to Y$ and $Y \to Z$.

FD Graph and Minimal FD Graph

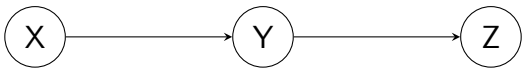
Example: $F = \{X \to Y,\ Y \to Z,\ X \to Z\}$
Non-minimal graph contains a redundant edge $X \to Z$ (implied by transitivity).



The dashed edge is removed in the **minimal** FD graph: keep only $X \to Y$ and $Y \to Z$.

FD Graph and Minimal FD Graph

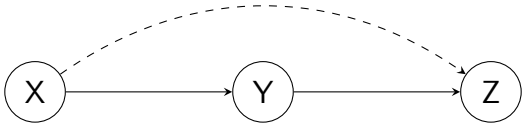
Example: $F = \{X \to Y,\ Y \to Z,\ X \to Z\}$
Non-minimal graph contains a redundant edge $X \to Z$ (implied by transitivity).



The dashed edge is removed in the **minimal** FD graph: keep only $X \to Y$ and $Y \to Z$.

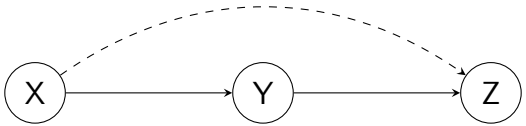
FD Graph and Minimal FD Graph

Example: $F = \{X \to Y,\ Y \to Z,\ X \to Z\}$
Non-minimal graph contains a redundant edge $X \to Z$ (implied by transitivity).



FD Graph and Minimal FD Graph

Example: $F = \{X \to Y,\ Y \to Z,\ X \to Z\}$
Non-minimal graph contains a redundant edge $X \to Z$ (implied by transitivity).



The dashed edge is removed in the **minimal** FD graph: keep only $X \to Y$ and $Y \to Z$.

Transitive Closure of Functional Dependencies

The **transitive closure** of a set of FDs F, noted F^+, is:

Transitive Closure of Functional Dependencies

The **transitive closure** of a set of FDs F, noted F^+, is:
The set of all functional dependencies that can be inferred from F using Armstrong's axioms.

Transitive Closure of Functional Dependencies

The **transitive closure** of a set of FDs F, noted F^+, is:
The set of all functional dependencies that can be inferred from F using Armstrong's axioms.

Example:

$$F = \{A \rightarrow B,\ B \rightarrow C\}$$

Transitive Closure of Functional Dependencies

The **transitive closure** of a set of FDs F, noted F^+, is:
The set of all functional dependencies that can be inferred from F using Armstrong's axioms.

Example:

$$F = \{A \rightarrow B,\ B \rightarrow C\}$$

$$F^+ = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C\}$$

Closure of Functional Dependencies

The closure F^+ contains all dependencies implied by F.

Closure of Functional Dependencies

The closure F^+ contains all dependencies implied by F.

Example:

Given:

$$F = \{EmpID \rightarrow Dept,\ Dept \rightarrow DeptPhone\}$$

Closure of Functional Dependencies

The closure F^+ contains all dependencies implied by F.

Example:

Given:

$$F = \{EmpID \rightarrow Dept,\ Dept \rightarrow DeptPhone\}$$

Then:

$$EmpID \rightarrow DeptPhone \in F^+$$

Attribute Closure

The closure of an attribute set X is denoted $[X]^+$.

Attribute Closure

The closure of an attribute set X is denoted $[X]^+$.

Example:

Given:

$$F = \{EmpID \rightarrow EmpName,\ EmpID \rightarrow Dept\}$$

Attribute Closure

The closure of an attribute set X is denoted $[X]^+$.

Example:

Given:

$$F = \{EmpID \rightarrow EmpName,\ EmpID \rightarrow Dept\}$$

$$[EmpID]^+ = \{EmpID, EmpName, Dept\}$$

Attribute Closure

The closure of an attribute set X with respect to F, noted $[X]^+$, is:

Attribute Closure

The closure of an attribute set X with respect to F, noted $[X]^+$, is:

$$[X]^+ = \{A \mid F \models X \rightarrow A\}$$

Attribute Closure

The closure of an attribute set X with respect to F, noted $[X]^+$, is:

$$[X]^+ = \{A \mid F \models X \rightarrow A\}$$

Algorithm:

- Initialize $X^+ := X$

Attribute Closure

The closure of an attribute set X with respect to F, noted $[X]^+$, is:

$$[X]^+ = \{A \mid F \models X \rightarrow A\}$$

Algorithm:

- Initialize $X^+ := X$
- Repeat:

Attribute Closure

The closure of an attribute set X with respect to F, noted $[X]^+$, is:

$$[X]^+ = \{A \mid F \models X \rightarrow A\}$$

Algorithm:

- Initialize $X^+ := X$
- Repeat:
  - For each $Y \rightarrow Z$ in F

Attribute Closure

The closure of an attribute set X with respect to F, noted $[X]^+$, is:

$$[X]^+ = \{A \mid F \models X \rightarrow A\}$$

Algorithm:

- Initialize $X^+ := X$
- Repeat:
  - For each $Y \rightarrow Z$ in F
  - If $Y \subseteq X^+$ then $X^+ := X^+ \cup Z$

Attribute Closure

The closure of an attribute set X with respect to F, noted $[X]^+$, is:

$$[X]^+ = \{A \mid F \models X \rightarrow A\}$$

Algorithm:

- Initialize $X^+ := X$
- Repeat:
  - For each $Y \rightarrow Z$ in F
  - If $Y \subseteq X^+$ then $X^+ := X^+ \cup Z$
- Until no change

Example: Attribute Closure

Let:

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CG \to B\}$$

Example: Attribute Closure

Let:

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CG \to B\}$$

Compute $[AB]^+$:

Example: Attribute Closure

Let:

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CG \to B\}$$

Compute $[AB]^+$:

- Start: $\{A, B\}$

Example: Attribute Closure

Let:

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CG \to B\}$$

Compute $[AB]^+$:

- Start: $\{A, B\}$
- Add C using $AB \to C$

Example: Attribute Closure

Let:

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CG \to B\}$$

Compute $[AB]^+$:

- Start: $\{A, B\}$
- Add C using $AB \to C$
- Add D using $BC \to D$

Example: Attribute Closure

Let:

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CG \to B\}$$

Compute $[AB]^+$:

- Start: $\{A, B\}$
- Add C using $AB \to C$
- Add D using $BC \to D$
- Add E using $D \to E$

Example: Attribute Closure

Let:

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CG \to B\}$$

Compute $[AB]^+$:

- Start: $\{A, B\}$
- Add C using $AB \to C$
- Add D using $BC \to D$
- Add E using $D \to E$

Example: Attribute Closure

Let:

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CG \to B\}$$

Compute $[AB]^+$:

- Start: $\{A, B\}$
- Add C using $AB \to C$
- Add D using $BC \to D$
- Add E using $D \to E$

$$[AB]^+ = \{A, B, C, D, E\}$$

Minimal Cover

A set of functional dependencies F is a **minimal cover** if:

Minimal Cover

A set of functional dependencies F is a **minimal cover** if:

- Each FD has a single attribute on the right-hand side

Minimal Cover

A set of functional dependencies F is a **minimal cover** if:

- Each FD has a single attribute on the right-hand side
- No FD can be removed without changing F^+

Minimal Cover

A set of functional dependencies F is a **minimal cover** if:

- Each FD has a single attribute on the right-hand side
- No FD can be removed without changing F^+
- No attribute can be removed from the left-hand side

Minimal Cover

A set of functional dependencies F is a **minimal cover** if:

- Each FD has a single attribute on the right-hand side
- No FD can be removed without changing F^+
- No attribute can be removed from the left-hand side

Minimal Cover

A set of functional dependencies F is a **minimal cover** if:

- Each FD has a single attribute on the right-hand side
- No FD can be removed without changing F^+
- No attribute can be removed from the left-hand side

Note: minimal covers are not unique.

Minimal Cover

A minimal cover removes all redundancies from F.

Minimal Cover

A minimal cover removes all redundancies from F.

Example:

Given:

$$F = \{EmpID \rightarrow EmpName, EmpID \rightarrow Dept, EmpID \rightarrow EmpName\}$$

Minimal Cover

A minimal cover removes all redundancies from F.

Example:

Given:

$$F = \{EmpID \rightarrow EmpName, EmpID \rightarrow Dept, EmpID \rightarrow EmpName\}$$

Minimal cover:

$$F_{min} = \{EmpID \rightarrow EmpName,\ \ EmpID \rightarrow Dept\}$$

Minimal Cover Algorithm

Step 1: Decompose right-hand sides

Minimal Cover Algorithm

Step 1: Decompose right-hand sides
Step 2: Remove redundant dependencies

Minimal Cover Algorithm

Step 1: Decompose right-hand sides
Step 2: Remove redundant dependencies
Step 3: Remove redundant attributes from left-hand sides

Minimal Cover Algorithm

Step 1: Decompose right-hand sides
Step 2: Remove redundant dependencies
Step 3: Remove redundant attributes from left-hand sides
The result is a minimal cover equivalent to F.

Minimal Cover — Example 1

Let the relation schema be:

$$R(A, B, C)$$

and the set of functional dependencies:

$$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$$

Minimal Cover — Example 1

Let the relation schema be:

$$R(A, B, C)$$

and the set of functional dependencies:

$$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$$

Step 1 — Canonical form

Minimal Cover — Example 1

Let the relation schema be:

$$R(A, B, C)$$

and the set of functional dependencies:

$$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$$

Step 1 — Canonical form

Rewrite each FD with a single attribute on the right-hand side:

$$F = \{A \to B,\ A \to C,\ B \to C,\ AB \to C\}$$

Minimal Cover — Example 1

Let the relation schema be:

$$R(A, B, C)$$

and the set of functional dependencies:

$$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$$

Step 1 — Canonical form

Rewrite each FD with a single attribute on the right-hand side:

$$F = \{A \to B,\ A \to C,\ B \to C,\ AB \to C\}$$

Step 2 — Remove redundant dependencies

Minimal Cover — Example 1

Let the relation schema be:

$$R(A, B, C)$$

and the set of functional dependencies:

$$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$$

Step 1 — Canonical form

Rewrite each FD with a single attribute on the right-hand side:

$$F = \{A \to B,\ A \to C,\ B \to C,\ AB \to C\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$

Minimal Cover — Example 1

Let the relation schema be:

$$R(A, B, C)$$

and the set of functional dependencies:

$$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$$

Step 1 — Canonical form

Rewrite each FD with a single attribute on the right-hand side:

$$F = \{A \to B,\ A \to C,\ B \to C,\ AB \to C\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$
- $AB \to C$ is redundant since $B \to C$ and $B \subset AB$

Minimal Cover — Example 1

Let the relation schema be:

$$R(A, B, C)$$

and the set of functional dependencies:

$$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$$

Step 1 — Canonical form

Rewrite each FD with a single attribute on the right-hand side:

$$F = \{A \to B,\ A \to C,\ B \to C,\ AB \to C\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$
- $AB \to C$ is redundant since $B \to C$ and $B \subset AB$

Minimal Cover — Example 1

Let the relation schema be:

$$R(A, B, C)$$

and the set of functional dependencies:

$$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$$

Step 1 — Canonical form

Rewrite each FD with a single attribute on the right-hand side:

$$F = \{A \to B,\ A \to C,\ B \to C,\ AB \to C\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$
- $AB \to C$ is redundant since $B \to C$ and $B \subset AB$

Minimal cover:

$$F_{min} = \{A \to B,\ B \to C\}$$

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Step 1 — Canonical form

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Step 1 — Canonical form

$$F = \{A \to B,\ A \to C,\ B \to C,\ C \to D\}$$

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Step 1 — Canonical form

$$F = \{A \to B,\ A \to C,\ B \to C,\ C \to D\}$$

Step 2 — Remove redundant dependencies

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Step 1 — Canonical form

$$F = \{A \to B,\ A \to C,\ B \to C,\ C \to D\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Step 1 — Canonical form

$$F = \{A \to B,\ A \to C,\ B \to C,\ C \to D\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Step 1 — Canonical form

$$F = \{A \to B,\ A \to C,\ B \to C,\ C \to D\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$

Step 3 — Check minimality of left-hand sides

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Step 1 — Canonical form

$$F = \{A \to B,\ A \to C,\ B \to C,\ C \to D\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$

Step 3 — Check minimality of left-hand sides

No attribute can be removed from any left-hand side.

Minimal Cover — Example 2

Let:

$$R(A, B, C, D)$$

and:

$$F = \{A \to BC,\ B \to C,\ C \to D,\ A \to B\}$$

Step 1 — Canonical form

$$F = \{A \to B,\ A \to C,\ B \to C,\ C \to D\}$$

Step 2 — Remove redundant dependencies

- $A \to C$ is implied by $A \to B$ and $B \to C$

Step 3 — Check minimality of left-hand sides

No attribute can be removed from any left-hand side.

Minimal cover:

$$F_{min} = \{A \to B,\ B \to C,\ C \to D\}$$

Keys

An attribute set X is a **key** if:

Keys

An attribute set X is a **key** if:

$$[X]^+ = \text{all attributes of the relation}$$

Keys

An attribute set X is a **key** if:

$$[X]^+ = \text{all attributes of the relation}$$

Example:
Relation:

$$\text{EMPLOYEE}(\textit{EmpID}, \textit{EmpName}, \textit{Dept})$$

Keys

An attribute set X is a **key** if:

$$[X]^+ = \text{all attributes of the relation}$$

Example:
Relation:

$$\text{EMPLOYEE}(\textit{EmpID}, \textit{EmpName}, \textit{Dept})$$

$$[\textit{EmpID}]^+ = \{\textit{EmpID}, \textit{EmpName}, \textit{Dept}\} \Rightarrow \textit{EmpID} \text{ is a key}$$

Superkey and Key

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema and F a set of FDs.

Superkey and Key

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema and F a set of FDs.
An attribute set X is a **superkey** of R if:

$$[X]^+ = \{A_1, A_2, \ldots, A_n\}$$

Superkey and Key

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema and F a set of FDs.
An attribute set X is a **superkey** of R if:

$$[X]^+ = \{A_1, A_2, \ldots, A_n\}$$

A **key** is a **minimal superkey**.

Superkey and Key

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema and F a set of FDs.
An attribute set X is a **superkey** of R if:

$$[X]^+ = \{A_1, A_2, \ldots, A_n\}$$

A **key** is a **minimal superkey**.
Example:

Superkey and Key

Let $R(A_1, A_2, \ldots, A_n)$ be a relation schema and F a set of FDs.
An attribute set X is a **superkey** of R if:

$$[X]^+ = \{A_1, A_2, \ldots, A_n\}$$

A **key** is a **minimal superkey**.
Example:
If:

$$EmpID \rightarrow EmpName,\ Dept$$

then:

$$[EmpID]^+ = \{EmpID, EmpName, Dept\} \Rightarrow EmpID \text{ is a key.}$$

Candidate Keys and Primary Key

If a relation has several minimal keys, they are called **candidate keys**.

Candidate Keys and Primary Key

If a relation has several minimal keys, they are called **candidate keys**.
One candidate key is chosen as the **primary key**.

Candidate Keys and Primary Key

If a relation has several minimal keys, they are called **candidate keys**.
One candidate key is chosen as the **primary key**.

Example:

$$\textsc{Vehicle}(\textit{Registration},\ \textit{Brand},\ \textit{Type},\ \textit{ChassisNo})$$

Candidate Keys and Primary Key

If a relation has several minimal keys, they are called **candidate keys**.
One candidate key is chosen as the **primary key**.

Example:

$$\textsc{Vehicle}(\textit{Registration},\ \textit{Brand},\ \textit{Type},\ \textit{ChassisNo})$$

Candidate keys:

$$\textit{Registration},\ \textit{ChassisNo}$$

Candidate Keys and Primary Key

If a relation has several minimal keys, they are called **candidate keys**.
One candidate key is chosen as the **primary key**.

Example:

$$\textsc{Vehicle}(\textit{Registration},\ \textit{Brand},\ \textit{Type},\ \textit{ChassisNo})$$

Candidate keys:

$$\textit{Registration},\ \textit{ChassisNo}$$

Primary key chosen:

$$\textit{Registration}$$

From Functional Dependencies to Normal Forms

Functional dependencies are used to:

From Functional Dependencies to Normal Forms

Functional dependencies are used to:

- identify redundancy

From Functional Dependencies to Normal Forms

Functional dependencies are used to:

- identify redundancy
- detect anomalies

From Functional Dependencies to Normal Forms

Functional dependencies are used to:

- identify redundancy
- detect anomalies
- guide decomposition

From Functional Dependencies to Normal Forms

Functional dependencies are used to:

- identify redundancy
- detect anomalies
- guide decomposition

From Functional Dependencies to Normal Forms

Functional dependencies are used to:

- identify redundancy
- detect anomalies
- guide decomposition

Normal forms are based on functional dependencies.

Normal Forms

Normal forms use functional dependencies to improve schemas.

Normal Forms

Normal forms use functional dependencies to improve schemas.

- First Normal Form (1NF)

Normal Forms

Normal forms use functional dependencies to improve schemas.

- First Normal Form (1NF)
- Second Normal Form (2NF)

Normal Forms

Normal forms use functional dependencies to improve schemas.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

Normal Forms

Normal forms use functional dependencies to improve schemas.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

Normal Forms

Normal forms use functional dependencies to improve schemas.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

Goal: remove redundancy and anomalies.

Lossless-Join Decomposition (SPI)

A decomposition of a relation R into R_1 and R_2 is **lossless-join** if:

Lossless-Join Decomposition (SPI)

A decomposition of a relation R into R_1 and R_2 is **lossless-join** if:

$$R = R_1 \bowtie R_2$$

Lossless-Join Decomposition (SPI)

A decomposition of a relation R into R_1 and R_2 is **lossless-join** if:

$$R = R_1 \bowtie R_2$$

Heath's Theorem

Lossless-Join Decomposition (SPI)

A decomposition of a relation R into R_1 and R_2 is **lossless-join** if:

$$R = R_1 \bowtie R_2$$

Heath's Theorem
If:

$$X \to Y$$

then the decomposition:

$$R(X, Y, Z) \Rightarrow R_1(X, Y),\ R_2(X, Z)$$

is lossless.

Dependency Preservation (SPD)

A decomposition preserves dependencies if:

Dependency Preservation (SPD)

A decomposition preserves dependencies if:
All functional dependencies in F can be enforced without performing joins.

Dependency Preservation (SPD)

A decomposition preserves dependencies if:
All functional dependencies in F can be enforced without performing joins.
Formally:

$$(F_1 \cup F_2)^+ = F^+$$

Dependency Preservation (SPD)

A decomposition preserves dependencies if:
All functional dependencies in F can be enforced without performing joins.
Formally:

$$(F_1 \cup F_2)^+ = F^+$$

Goal:

- Maintain integrity constraints

Dependency Preservation (SPD)

A decomposition preserves dependencies if:
All functional dependencies in F can be enforced without performing joins.
Formally:

$$(F_1 \cup F_2)^+ = F^+$$

Goal:

- Maintain integrity constraints
- Avoid expensive join operations

First Normal Form (1NF)

A relation is in **First Normal Form** if:

First Normal Form (1NF)

A relation is in **First Normal Form** if:

- All attributes are atomic

First Normal Form (1NF)

A relation is in **First Normal Form** if:

- All attributes are atomic
- No multivalued attributes exist

First Normal Form (1NF)

A relation is in **First Normal Form** if:

- All attributes are atomic
- No multivalued attributes exist

First Normal Form (1NF)

A relation is in **First Normal Form** if:

- All attributes are atomic
- No multivalued attributes exist

Example (not in 1NF):

$$Phones = \{0551, 0662\}$$

First Normal Form (1NF)

A relation is in **First Normal Form** if:

- All attributes are atomic
- No multivalued attributes exist

Example (not in 1NF):

$$Phones = \{0551, 0662\}$$

1NF decomposition:

$$\text{EMP\_PHONE}(EmpID,\ Phone)$$

Second Normal Form (2NF)

A relation is in **Second Normal Form** if:

Second Normal Form (2NF)

A relation is in **Second Normal Form** if:

- It is in 1NF

Second Normal Form (2NF)

A relation is in **Second Normal Form** if:

- It is in 1NF
- No non-key attribute depends on a proper subset of a key

Second Normal Form (2NF)

A relation is in **Second Normal Form** if:

- It is in 1NF
- No non-key attribute depends on a proper subset of a key

Second Normal Form (2NF)

A relation is in **Second Normal Form** if:

- It is in 1NF
- No non-key attribute depends on a proper subset of a key

Example:

$$(EmpID, Project) \rightarrow StartDate$$

but:

$$EmpID \rightarrow EmpName$$

Third Normal Form (3NF)

A relation is in **Third Normal Form** if:

Third Normal Form (3NF)

A relation is in **Third Normal Form** if:

- It is in 2NF

Third Normal Form (3NF)

A relation is in **Third Normal Form** if:

- It is in 2NF
- No transitive dependency exists between non-key attributes

Third Normal Form (3NF)

A relation is in **Third Normal Form** if:

- It is in 2NF
- No transitive dependency exists between non-key attributes

Third Normal Form (3NF)

A relation is in **Third Normal Form** if:

- It is in 2NF
- No transitive dependency exists between non-key attributes

Example:

$$EmpID \rightarrow Dept \rightarrow DeptPhone$$

Normalization Process (Bernstein Algorithm)

The Bernstein algorithm produces a normalized schema in 3NF.

Normalization Process (Bernstein Algorithm)

The Bernstein algorithm produces a normalized schema in 3NF.

Input:

- Relation schema R

Normalization Process (Bernstein Algorithm)

The Bernstein algorithm produces a normalized schema in 3NF.
Input:

- Relation schema R
- Set of functional dependencies F

Normalization Process (Bernstein Algorithm)

The Bernstein algorithm produces a normalized schema in 3NF.
Input:

- Relation schema R
- Set of functional dependencies F

Normalization Process (Bernstein Algorithm)

The Bernstein algorithm produces a normalized schema in 3NF.

Input:

- Relation schema R
- Set of functional dependencies F

Output:

- A dependency-preserving

Normalization Process (Bernstein Algorithm)

The Bernstein algorithm produces a normalized schema in 3NF.

Input:

- Relation schema R
- Set of functional dependencies F

Output:

- A dependency-preserving
- Lossless-join decomposition in 3NF

Bernstein Algorithm — Steps

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}
2. Group dependencies with the same left-hand side

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}
2. Group dependencies with the same left-hand side

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}
2. Group dependencies with the same left-hand side
3. Create one relation for each group

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}
2. Group dependencies with the same left-hand side
3. Create one relation for each group

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}
2. Group dependencies with the same left-hand side
3. Create one relation for each group
4. Ensure that at least one relation contains a key of R

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}
2. Group dependencies with the same left-hand side
3. Create one relation for each group
4. Ensure that at least one relation contains a key of R

Bernstein Algorithm — Steps

1. Compute a minimal cover F_{min}
2. Group dependencies with the same left-hand side
3. Create one relation for each group
4. Ensure that at least one relation contains a key of R
5. Remove redundant relations if any

Bernstein Algorithm — Example

Given:

$$R(A, B, C, D) \quad \text{and} \quad F = \{A \to B,\ B \to C,\ C \to D\}$$

Bernstein Algorithm — Example

Given:

$$R(A, B, C, D) \quad \text{and} \quad F = \{A \to B,\ B \to C,\ C \to D\}$$

Minimal cover:

$$F_{min} = \{A \to B,\ B \to C,\ C \to D\}$$

Bernstein Algorithm — Example

Given:

$$R(A, B, C, D) \quad \text{and} \quad F = \{A \to B,\ B \to C,\ C \to D\}$$

Minimal cover:

$$F_{min} = \{A \to B,\ B \to C,\ C \to D\}$$

Generated relations:

$$R_1(A, B),\ R_2(B, C),\ R_3(C, D)$$

Bernstein Algorithm — Example

Given:

$$R(A, B, C, D) \quad \text{and} \quad F = \{A \to B,\ B \to C,\ C \to D\}$$

Minimal cover:

$$F_{min} = \{A \to B,\ B \to C,\ C \to D\}$$

Generated relations:

$$R_1(A, B),\ R_2(B, C),\ R_3(C, D)$$

This decomposition is lossless and dependency-preserving.

First Normal Form (1NF)

A relation is in 1NF if all attributes are atomic.

First Normal Form (1NF)

A relation is in 1NF if all attributes are atomic.

Example (NOT 1NF):

$$Phones = \{0551, 0662\}$$

First Normal Form (1NF)

A relation is in 1NF if all attributes are atomic.

Example (NOT 1NF):

$$Phones = \{0551, 0662\}$$

1NF solution:

$$\text{EMP\_PHONE}(EmpID, Phone)$$

Second Normal Form (2NF)

A relation is in 2NF if:

Second Normal Form (2NF)

A relation is in 2NF if:

- it is in 1NF

Second Normal Form (2NF)

A relation is in 2NF if:

- it is in 1NF
- no partial dependency exists

Second Normal Form (2NF)

A relation is in 2NF if:

- it is in 1NF
- no partial dependency exists

Second Normal Form (2NF)

A relation is in 2NF if:

- it is in 1NF
- no partial dependency exists

Example:

$$(EmpID, Project) \rightarrow StartDate$$

But:

$$EmpID \rightarrow EmpName$$

Third Normal Form (3NF)

A relation is in 3NF if:

Third Normal Form (3NF)

A relation is in 3NF if:

- it is in 2NF

Third Normal Form (3NF)

A relation is in 3NF if:

- it is in 2NF
- no transitive dependency exists

Third Normal Form (3NF)

A relation is in 3NF if:

- it is in 2NF
- no transitive dependency exists

Third Normal Form (3NF)

A relation is in 3NF if:

- it is in 2NF
- no transitive dependency exists

Example:

$$EmpID \rightarrow Dept \rightarrow DeptPhone$$

Integrity Constraints

Domain integrity:

Integrity Constraints

Domain integrity:
Age $\in [18, 60]$

Integrity Constraints

Domain integrity:
Age $\in [18, 60]$
Entity integrity:

Integrity Constraints

Domain integrity:
Age $\in [18, 60]$
Entity integrity:
EmpID cannot be NULL

Integrity Constraints

Domain integrity:
Age $\in [18, 60]$
Entity integrity:
EmpID cannot be NULL
Referential integrity:

Integrity Constraints

Domain integrity:
Age $\in [18, 60]$
Entity integrity:
EmpID cannot be NULL
Referential integrity:
EmpID in ASSIGNMENT must exist in EMPLOYEE

From the Relational Model to SQL

Wrap-up

Questions

Questions?