CH8Summary

First normal form (1NF) a relation schema *R* is in **first normal form** (1NF) if the domains of all

attributes of *R* are atomic.

A domain is **atomic** if elements of the domain are considered to be indivisible units

[Integers are assumed to be atomic, so the set of integers is an atomic domain; however, the set of all sets of integers is a non-atomic domain]

An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation; a legal instance of a database is one where all the relation instances are legal instances

A **super key** is a set of attributes that uniquely identifies an entire tuple, a **functional dependency** allows us to express constraints that uniquely identify the values of certain attributes

Consider a relation schema *r* (*R*), and let $\alpha \subseteq R$ and $\beta \subseteq R$.

- Given an instance of *r* (*R*), we say that the instance **satisfies** the **functional dependency** $\alpha \rightarrow \beta$ if for all pairs of tuples *t*1 and *t*2 in the instance such that $t_1[\alpha] = t_2[\alpha]$, it is also the case that $t_1[\beta] = t_2[\beta]$.

- We say that the functional dependency $\alpha \rightarrow \beta$ **holds** on schema *r* (*R*) if, in every legal instance of *r* (*R*) it satisfies the functional dependency.

Functional dependencies allow us to express constraints that we cannot express with super keys We

shall use functional dependencies in two ways:

1. To test instances of relations to see whether they satisfy a given set *F* of functional dependencies.

2. To specify constraints on the set of legal relations.

Some functional dependencies are said to be **trivial** because they are satisfied by all relations A

functional dependency of the form $\alpha \rightarrow \beta$ is **trivial** if $\beta \subseteq \alpha$

the notation F^+ to denote the **closure** of the set *F*, that is, the set of all functional dependencies that can be inferred given the set *F* **Boyce–Codd normal form (BCNF)**

it eliminates all redundancy that can be discovered based on functional dependencies. A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$).

- α is a super key for schema *R*.

Database consistency constraints:

- primary-key constraints
- functional dependencies
- check constraints
- assertions
- triggers

Dependency preserving

Dependency preservation is usually considered desirable; we consider another normal form, weaker than BCNF that will allow us to preserve dependencies

A decomposition having the property $F'^{+} = F^{+}$ is a **dependency-preserving decomposition**

Third Normal Form

BCNF requires that all nontrivial dependencies be of the form$\alpha \rightarrow \beta$ where α is a super key. Third normal form (3NF) relaxes this constraint slightly by allowing certain nontrivial functional dependencies whose left side is not a super key.

A relation schema *R* is in **third normal form** with respect to a set *F* of functional dependencies if, for all functional dependencies in F^{+} of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

• $\alpha \rightarrow \beta$ is a trivial functional dependency.

• α is a super key for *R*.

• Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R*.

Any schema that satisfies BCNF also satisfies 3NF, since each of its functional dependencies would satisfy one of the first two alternatives. BCNF is therefore a more restrictive normal form than is 3NF.

Armstrong's axioms

Axioms, or rules of inference, provide a simpler technique for reasoning about functional dependencies

• **Reflexivity rule**. If α is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.

• **Augmentation rule**. If $\alpha \rightarrow \beta$ holds and Υ is a set of attributes, then $\Upsilon\alpha \rightarrow \Upsilon\beta$holds.

• **Transitivity rule**. If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \Upsilon$ holds, then $\alpha \rightarrow \Upsilon$ holds.

Armstrong's axioms are **sound**, because they do not generate any incorrect functional dependencies. They are **complete**, because, for a given set *F* of functional dependencies, they allow us to generate all F^{+}.

Additional rules

• Union rule. If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \Upsilon$ holds, then $\alpha \rightarrow \beta\Upsilon$ holds.

• Decomposition rule. If $\alpha \rightarrow \beta\Upsilon$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \Upsilon$ holds.

• Pseudo transitivity rule. If $\alpha \rightarrow \beta$ holds and $\Upsilon\beta \rightarrow \sigma$ holds, then $\alpha\Upsilon \rightarrow \sigma$ holds

Canonical cover

A canonical cover of F is a “minimal” set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies

A canonical cover for F is a set of dependencies Fc such that

* F logically implies all dependencies in Fc, and

* Fc logically implies all dependencies in F, and

* No functional dependency in Fc contains an extraneous attribute, and

* Each left side of functional dependency in Fc is unique.

Extraneous

An attribute of a functional dependency is said to be **extraneous** if we can remove it without changing the closure of the set of functional dependencies.

Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.

• Attribute *A* is extraneous in α if $A \in \alpha$, and *F* logically implies $(F - \{\alpha \rightarrow \beta\})$

$\cup \{(\alpha - A) \rightarrow \beta\}$.

• Attribute *A* is extraneous in β if $A \in \beta$, and the set of functional dependencies

$(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies *F*.

The decomposition is a **lossless decomposition** if there is no loss of information by replacing $r(R)$ with two relation schemas $r_1(R_1)$ and $r_2(R_2)$.
If we project r onto R_1 and R_2, and compute the natural join of the projection results, we get back exactly r. A decomposition that is not a lossless decomposition is called a lossy decomposition. The terms lossless-join decomposition and lossy-join decomposition are sometimes used in place of lossless decomposition and lossy decomposition.

R_1 and R_2 form a lossless decomposition of R if at least one of the following functional dependencies is in F^+:

- $R_1 \cap R_2 \rightarrow R_1$

- $R_1 \cap R_2 \rightarrow R_2$

In other words, if $R_1 \cap R_2$ forms a super key of either R_1 or R_2, the decomposition of R is a lossless decomposition.

An interesting property of the 3NF algorithm. Sometimes, the result is not only in 3NF, but also in BCNF. This suggests an alternative method of generating a BCNF design. First use the 3NF algorithm. Then, for any schema in the 3NF design that is not in BCNF, decompose using the BCNF algorithm. If the result is not dependency-preserving, revert to the 3NF design.

The 3NF algorithm ensures the preservation of dependencies by explicitly building a schema for each dependency in a canonical cover. It ensures that the decomposition is a lossless decomposition by guaranteeing that at least one schema contains a candidate key for the schema being decomposed

This algorithm is also called the **3NF synthesis algorithm**, since it takes a set of dependencies and adds one schema at a time, instead of decomposing the initial schema repeatedly.

Comparison of BCNF and 3NF

Advantages to 3NF in that we know that it is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation. Nevertheless, there are disadvantages to 3NF: We may have to use null values to represent some of the possible meaningful relationships among data items, and there is the problem of repetition of information.

A materialized view is a view whose result is stored in the database and brought up to date when the relations used in the view are updated. The functional dependency can be tested easily on the materialized view, using one of the constraints **unique** (α) or **primary key** (α).

On the negative side, there is a space and time overhead due to the materialized view, but on the positive side, the application programmer need not worry about writing code to keep redundant data consistent on updates; it is the job of the database system to maintain the materialized view, that is, keep it up to date when the database is updated.

We shall use multivalued dependencies to define a normal form for relation schemas. This normal form, called **fourth normal form (4NF)**, is more restrictive than BCNF. We shall see that every 4NF schema is also in BCNF but there are BCNF schemas that are not in 4NF.

Functional dependencies rule out certain tuples from being in a relation If $A \rightarrow B$, then we cannot have two tuples with the same A value but different B values.

Multivalued dependencies, on the other hand, do not rule out the existence of certain tuples. Instead, they *require* that other tuples of a certain form be present in the relation. For this reason, functional dependencies sometimes are referred to as **equality-generating dependencies**, and multivalued dependencies are referred to as **tuple-generating dependencies**.

Let *r* (*R*) be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$\alpha \rightarrow\rightarrow \beta$

We shall use multivalued dependencies in two ways:

1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies
2. To specify constraints on the set of legal relations

Let *D* denote a set of functional and multivalued dependencies. The **closure** D^+ of *D* is the set of all functional and multivalued dependencies logically implied by *D*.

• If $\alpha \rightarrow \beta$, then $\alpha \rightarrow\rightarrow \beta$. In other words, every functional dependency is also a multivalued dependency.

• If $\alpha \rightarrow\rightarrow \beta$, then $\alpha \rightarrow\rightarrow R - \alpha - \beta$

A relation schema *r* (*R*) is in **fourth normal form** (4NF) with respect to a set *D* of functional and multivalued dependencies if, for all multivalued dependencies in D^+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

• $\alpha \rightarrow\rightarrow \beta$ is a trivial multivalued dependency.

• α is a super key for *R*.

Every 4NF schema is in BCNF

The **restriction** of *D* to R_i is the set D_i consisting of:

1. All functional dependencies in D^+ that include only attributes of R_i .

2. All multivalued dependencies of the form:

$\alpha \rightarrow\rightarrow \beta \cap R_i$ where $\alpha \subseteq R_i$ and $\alpha \rightarrow\rightarrow \beta$ is in D+.

Decomposition is lossless of *R* if and only if at least one of the following multivalued dependencies is in D^+:

$R_1 \cap R_2 \rightarrow\rightarrow R_1$

$R_1 \cap R_2 \rightarrow\rightarrow R_2$

The fourth normal form is by no means the “ultimate” normal form, there are types of constraints called **join dependencies** that generalize multivalued dependencies, and lead to another normal form called **Project-join normal form (PJNF)** (PJNF is called **fifth normal form**) There is a class of even more general constraints that leads to a normal form called **domain-key normal form (DKNF)**.

They are not only hard to reason with, but there is also no set of sound and complete inference rules for reasoning about the constraints.

The process of creating an E-R design tends to generate 4NF designs. If a multivalued dependency holds and is not implied by the corresponding functional dependency, it usually arises from one of the following sources:

• A many-to-many relationship set.

• A multivalued attribute of an entity set.

Unique-role assumption, which means that each attribute name has a unique meaning in the database, it is a good idea to keep names for incompatible attributes distinct, if attributes of different relations have the same meaning, it may be a good idea to use the same attribute name.

Denormalization

The process of taking a normalized schema and making it nonnormalized is called denormalization, and designers use it to tune performance of systems to support time-critical operations.

Like denormalization, using materialized views does have space and time overhead; however, it has the advantage that keeping the view up to date is the job of the database system, not the application programmer.

Temporal data are data that have an associated time interval during which they are **valid**. We use the term **snapshot** of data to mean the value of the data at a particular point in time.

Functional dependencies that hold at a particular point in time are called **temporal functional dependencies.**