四川大學
SICHUAN UNIVERSITY

Database System Concepts

Relational Database Design

伍元凯
College of Computer Science (Software), Sichuan University
wuyk0@scu.edu.cn

2023/04/26

海纳百川 有容乃大

- Simplicity: A good database design should be simple and easy to understand.
- Minimization of data redundancy: Redundant data can lead to inconsistency and inefficiency in the database.
- Consistency: A good database design should maintain consistency of data throughout the database (foreign keys, check constraints, and triggers).
- Flexibility: A good database design should be flexible enough to accommodate future changes and modifications to the database schema.
- Normalization: A good database design should adhere to the rules of normalization, which is a process of breaking down data into smaller, more manageable units to minimize data redundancy and improve consistency.

- Integrity: A good database design should maintain the integrity of the data, ensuring that it is accurate, complete, and free from errors.
- Performance: A good database design should be optimized for performance.
- Security: A good database design should incorporate security measures to ensure that the data is protected from unauthorized access, modification, or deletion.

四川大學

A good design

```sql
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50),
    date_of_birth DATE
);

CREATE TABLE Course (
    course_id INT PRIMARY KEY,
    name VARCHAR(50),
    description VARCHAR(200),
    credit_value INT
);

CREATE TABLE Instructor (
    instructor_id INT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50),
    specialty VARCHAR(50)
);
```

A good design

```sql
CREATE TABLE Registration (
    registration_id INT PRIMARY KEY,
    registration_date DATE,
    grade INT,
    student_id INT,
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES Student(student_id),
    FOREIGN KEY (course_id) REFERENCES Course(course_id)
);

CREATE TABLE Instructor_Course (
    instructor_id INT,
    course_id INT,
    PRIMARY KEY (instructor_id, course_id),
    FOREIGN KEY (instructor_id) REFERENCES Instructor(instructor_id),
    FOREIGN KEY (course_id) REFERENCES Course(course_id)
);
```

Order ID	Customer Name	Product Name	Product Description	Quantity	Order Date
001	John Smith	iPhone 12	Apple iPhone 12	2	2022-03-15
002	Jane Doe	MacBook Pro	Apple MacBook Pro	1	2022-03-16
003	John Smith	iPad Air	Apple iPad Air	3	2022-03-17

Order Information Table

- Repeating Data: In the table above, we can see that the Customer Name and Product Description fields are repeated for each order.
- Data Update Anomalies: If we need to update a customer's name or a product's description in this table, we would have to update every row where that customer or product appears.

Order ID	Customer ID	Product ID	Quantity
001	001	001	2
002	002	002	1
003	001	003	3

Orders Table

Customer ID	Customer Name
001	John Smith
002	Jane Doe

Customers Table

Product ID	Product Name	Product Description
001	iPhone 12	Apple iPhone 12
002	MacBook Pro	Apple MacBook Pro
003	iPad Air	Apple iPad Air

(57766, Kim, Main, Perryridge, 75000)
(98776, Kim, North, Hampton, 67000)

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

目录

Let R be a relation schema and let R_1 and R_2 form a decomposition of R–that is, viewing R, R_1, and R_2 as sets of attributes, $R = R_1 \cup R_2$. We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with two relation schemas R_1 and R_2.

Math and Code

if we project r onto R_1 and R_2, and compute the natural join of the projection results, we get back exactly r.

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

tuples of relation r

```sql
select *
from (select R1 from r)
natural join
(select R2 from r);
```

Lossy decomposition

$$r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

The decomposed version is unable to represent the **absence of a connection, and absence of a connection is indeed information**.

A database models is a set of entities and relationships in the real world. There are usually a variety of constraints (rules) on the data in the real world.

- 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.

Summary of Notations

Notation	**Description**
Greek letters (e.g., α)	Sets of attributes
Uppercase Roman letter (R)	A relation schema
$r(R)$	the schema R for relation r
Roman letter	the set of attributes is definitely a schema
K	a superkey for a specific relation schema
Instance of r	The particular value at any given time.

目录

Superkey

Given $r(R)$, a subset K of R is a superkey of $r(R)$ if, in any legal instance of $r(R)$, for all pairs t_1 and t_2 of tuples in the instance of r, if $t_1 \neq t_2$, then $t_1[K] \neq t_2[K]$.

Uniquely identify the values of certain attributes

In 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 $t_1[\beta] = t_2[\beta]$.

In a given company, each department has a unique salary range, and all employees within a department receive the same salary.

employee_id	employee_name	department	salary
001	John Smith	Sales	50000
002	Jane Doe	IT	60000
003	Bob Johnson	Sales	50000
004	Sarah Lee	HR	55000

Employee Table

$$\text{department} \rightarrow \text{salary}$$

Using the functional-dependency notation, we say that K is a superkey for $r(R)$ if the functional dependency $K \rightarrow R$ holds on $r(R)$.

Trivial functional dependence

Some functional dependencies are said to be trivial because they are satisfied by all relations. For example, $A \to A$ is satisfied by all relations involving attribute A. Reading the definition of functional dependency literally, we see that, for all tuples t_1 and t_2 such that $t_1[A] = t_2[A]$, it is the case that $t_1[A] = t_2[A]$. Similarly, $AB \to A$ is satisfied by all relations involving attribute A. In general, a functional dependency of the form $\alpha \to \beta$ is trivial if $\beta \subseteq \alpha$.

F^+ denotes the closure of the set F, that is, the set of all functional dependencies that can be inferred given the set F. F^+ contains all of the functional dependencies in F.

Armstrong's Axioms property was developed by William Armstrong in 1974 to reason about functional dependencies.

The property suggests rules that hold true if the following are satisfied:

1. **Transitivity**: If $A \to B$ and $B \to C$, then $A \to C$, i.e., a transitive relation.
2. **Reflexivity**: $A \to B$ if B is a subset of A.
3. **Augmentation**: The last rule suggests: $AC \to BC$, if $A \to B$.

Armstrong, W. W. (1974, August). Dependency structures of data base relationships. In IFIP congress (Vol. 74, pp. 580-583).

目录

Functional dependencies and lossy decompositions

Let R, R_1, R_2, and F be as above. R_1 and R_2 form a lossless decomposition of R if at least one of the following functional dependencies is in F^+:

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

if $R_1 \cap R_2$ forms a superkey for either R_1 or R_2, the decomposition of R is a lossless decomposition.

例 1

Suppose we have a database of customers and their orders. The information is stored in a single relation $R(CUSTOMER_{ID}, CUSTOMER_{NAME}, ORDER_{ID}, ORDER_{DATE}, PRODUCT_{ID},$ PRODUCT$_{NAME})$, where $CUSTOMER_{ID}$ uniquely identifies a customer, $ORDER_{ID}$ uniquely identifies an order, and $PRODUCT_{ID}$ uniquely identifies a product.

Functional Dependencies

$$CUSTOMER_{ID} \to CUSTOMER_{NAME}$$

$$ORDER_{ID} \to ORDER_{DATE}, CUSTOMER_{ID}$$

$$PRODUCT_{ID} \to PRODUCT_{NAME}$$

Check lossless

To ensure that the decomposition of R into R_1 and R_2 is lossless, we need to check whether at least one of the following functional dependencies is in F^+:

$R_1 \cap R_2 \to R_1$

$R_1 \cap R_2 \to R_2$

例 2

$$(CUSTOMER_{ID}, CUSTOMER_{NAME}, ORDER_{ID}, ORDER_{DATE}) \cap (ORDER_{ID}, PRODUCT_{ID}, PRODUCT_{NAME}) \to (CUSTOMER_{ID}, CUSTOMER_{NAME}, ORDER_{ID}, ORDER_{DATE})$$

CUSTOMER_ID	CUSTOMER_NAME	ORDER_ID	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
1	Alice	101	2023-04-20	501	Widget A
1	Alice	102	2023-04-21	502	Widget B
2	Bob	103	2023-04-21	501	Widget A
3	Charlie	104	2023-04-22	503	Widget C
3	Charlie	105	2023-04-23	502	Widget B

ORDER_ID	PRODUCT_ID	PRODUCT_NAME
101	501	Widget A
102	502	Widget B
103	501	Widget A
104	503	Widget C
105	502	Widget B

Suppose we decompose a relation schema $r(R)$ into $r_1(R_1)$ and $r_2(R_2)$, where $R_1 \cap R_2 \rightarrow R_1$. Then the following SQL constraints must be imposed on the decomposed schema to ensure their contents are consistent with the original schema:

- $R_1 \cap R_2$ is the primary key of r_1.
- $R_1 \cap R_2$ is a foreign key from r_2 referencing r_1.

```sql
CREATE TABLE customer_order (
    customer_id INTEGER,
    customer_name VARCHAR(255),
    order_id INTEGER,
    order_date DATE,
    product_id INTEGER,
    product_name VARCHAR(255),
    PRIMARY KEY (customer_id, order_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
CREATE TABLE order_product (
    order_id INTEGER,
    product_id INTEGER,
    product_name VARCHAR(255),
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES customer_order(order_id)
);
```

目录

definition

A relation schema R is in BCNF (Boyce-Codd Normal Form) with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \to \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \to \beta$ is a trivial functional dependency (i.e., $\beta \subseteq \alpha$).
- α is a superkey for schema R.

Suppose we have a table called **Orders** with the following columns:

- OrderID (**primary key**)
- CustomerID
- CustomerName
- ProductID
- ProductName
- Quantity

And suppose we have the following functional dependencies:

- OrderID $\to$ CustomerID, CustomerName
- ProductID $\to$ ProductName

This table violates BCNF because the functional dependency ProductID → ProductName does not satisfy the BCNF condition that the left side (ProductID) must be a superkey of the table.

General decomposition rule

Let R be a schema that is not in BCNF. Then there is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ such that α is not a superkey for R. We replace R in our design with two schemas:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

To fix this, we can decompose the Orders table into two tables:
OrderDetails

- OrderID (**primary key**)
- CustomerID
- CustomerName
- ProductID
- Quantity

Products

- ProductID (**primary key**)
- ProductName

Sales

- OrderID (primary key)
- CustomerID
- CustomerName
- EmployeeID
- EmployeeName
- ProductID
- ProductName
- Quantity
- Price

- OrderID $\rightarrow$ CustomerID, CustomerName
- OrderID $\rightarrow$ EmployeeID, EmployeeName
- ProductID $\rightarrow$ ProductName
- OrderID, ProductID $\rightarrow$ Quantity
- ProductID $\rightarrow$ Price

First decomposition

Sales

- OrderID (primary key)
- CustomerID
- CustomerName
- EmployeeID
- EmployeeName
- ProductID
- ProductName
- Price

OrderDetails

- OrderID (primary key)
- ProductID (primary key)
- Quantity

Second decomposition

Sales

- OrderID (primary key)
- CustomerID
- CustomerName
- EmployeeID
- EmployeeName
- ProductID

ProductNames

- ProductID (primary key)
- ProductName

ProductDetails

- ProductID (primary key)
- Price

s_ID	i_ID	dept name
111	333	Biology
222	444	Chemistry
111	555	Biology

The `dept advisor` table has the following functional dependencies:

- i_ID → dept name
- s_ID, dept name → i ID

we can decompose `dept advisor` into the following two schemas:

- (s_ID, i_ID)
- (i_ID, dept name)

it may allow an instructor to advise multiple departments, which violates the requirement that an instructor can act as advisor for only one department. Because **our design does not permit the enforcement of this functional dependency without a join**, we say that our design is not dependency preserving.

目录

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

s_ID	i_ID	dept name
111	333	Biology
222	444	Chemistry
111	555	Biology

The `dept advisor` table has the following functional dependencies:

- i_ID $\rightarrow$ dept name
- s_ID, dept name $\rightarrow$ i ID

Here $\alpha =$ i_ID, $\beta =$ dept name, and $\beta - \alpha =$ dept name. Since the functional dependency s_ID, dept name $\rightarrow$ i_ID holds on dept advisor, **the attribute dept name is contained in a candidate key and, therefore, dept advisor is in 3NF.**

目录

- BCNF is a stronger normal form than 3NF.
- It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation
- BCNF decomposition can result in more relations than 3NF decomposition, since BCNF requires that every non-trivial dependency in a relation be a dependency on a candidate key. This can lead to more tables in the database and potentially slower queries due to the need for more joins.
- 3NF is easier to achieve than BCNF, on the other hand, requires that every non-trivial dependency in a relation be a dependency on a candidate key, which can be more difficult to achieve in practice.

Our goals of database design with functional dependencies are:

- BCNF;
- Losslessness;
- Dependency preservation

Since it is not always possible to satisfy all three, we may be forced to choose between BCNF and dependency preservation with 3NF.

SQL does not provide a way of specifying functional dependencies, except for the special case of declaring superkeys by using the primary key or unique constraints. It is possible, although a little complicated, to write assertions that enforce a functional dependency.

CustomerID → CustomerName

```sql
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT NOT NULL,
    CustomerName VARCHAR(255) NOT NULL,
    ProductID INT NOT NULL,
    ProductName VARCHAR(255) NOT NULL,
    Quantity INT NOT NULL,
    CONSTRAINT orders_fk1 FOREIGN KEY (CustomerID) REFERENCES
        Customers(CustomerID)
);

CREATE ASSERTION orders_fd_check
    CHECK (NOT EXISTS (
        SELECT 1 FROM Orders o1, Orders o2
        WHERE o1.OrderID <> o2.OrderID
            AND o1.CustomerID = o2.CustomerID
            AND o1.CustomerName <> o2.CustomerName
    ));
```

(advisor.s_ID, advisor.dept_name) → (instructor.ID, instructor.dept_name)

```sql
-- Create tables
CREATE TABLE dept (
  dept_name VARCHAR(255) PRIMARY KEY,
  dept_location VARCHAR(255)
);

CREATE TABLE instructor (
  ID INT PRIMARY KEY,
  name VARCHAR(255),
  dept_name VARCHAR(255),
  FOREIGN KEY (dept_name) REFERENCES dept(dept_name)
);

CREATE TABLE student (
  ID INT PRIMARY KEY,
  name VARCHAR(255)
);
```

```sql

CREATE TABLE advisor (
  s_ID INT,
  i_ID INT,
  dept_name VARCHAR(255),
  PRIMARY KEY (s_ID, dept_name),
  FOREIGN KEY (s_ID) REFERENCES student(ID),
  FOREIGN KEY (i_ID, dept_name) REFERENCES instructor(ID, dept_name)
);

-- Create materialized view
CREATE MATERIALIZED VIEW advisor_info AS
  SELECT a.s_ID, a.dept_name, i.name AS instructor_name, i.dept_name AS instructor_dept_name
  FROM advisor a
  JOIN instructor i ON a.i_ID = i.ID AND a.dept_name = i.dept_name;

-- Query the materialized view
SELECT *
FROM advisor_info
WHERE instructor_dept_name = 'Computer Science';
```

目录

Unnecessary Repetition

We record with each instructor a set of children’s names and a set of landline phone numbers that may be shared by multiple people:

- (ID, phone number)
- (ID, child name)

If we were to combine these schemas to get (ID, child name, phone number). For example, let the instructor with ID 99999 have two children named “David” and “William” and two phone numbers, 512-555-1234 and 512-555-4321.

- (99999, David, 512-555-1234)
- (99999, David, 512-555-4321)
- (99999, William, 512-555-1234)
- (99999, William, 512-555-4321)

目录

Given a relation schema $r(R)$, a functional dependency f on R is logically implied by a set of functional dependencies F on R if every instance of a relation $r(R)$ that satisfies F also satisfies f.

例 3

Given a relation schema $r(A, B, C, G, H, I)$ and the set of functional dependencies:

$A \to B$
$A \to C$
$CG \to H$
$CG \to I$
$B \to H$

The functional dependency $A \to H$ is logically implied.

Let F be a set of functional dependencies. The closure of F, denoted by $F+$, is the set of all functional dependencies logically implied by F

1. **Transitivity**: If $A \to B$ and $B \to C$, then $A \to C$, i.e., a transitive relation.
2. **Reflexivity**: $A \to B$ if B is a subset of A.
3. **Augmentation**: The last rule suggests: $AC \to BC$, if $A \to B$.

Additional rules

1. **Union rule**: If $A \to B$ and $A \to C$ holds, then $A \to BC$.
2. **Decomposition rule**: $A \to BC$ if then $A \to B$ and $A \to C$.
3. **Pseudotransitivity rule**: If $A \to B$ holds and $CB \to D$ holds, then $AC \to D$ holds.

例 4

Proof of Union rule

1. $A \rightarrow B$ and $A \rightarrow C$ given
2. $A \rightarrow AC$, $AC \rightarrow BC$ using **Augmentation** rule
3. $A \rightarrow BC$ using **Transitivity** rule

例 5

Proof of Decomposition rule

1. $A \rightarrow BC$ given
2. $BC \rightarrow B$ using **Reflexivity** rule
3. $A \rightarrow B$ using **Transitivity** rule

Similar proof for $A \rightarrow C$

例 6

Proof of **Pseudotransitivity rule**

1. $A \to B$ and $CB \to D$ given
2. $CA \to CB$ using **Augmentation** rule
3. $CA \to D$ using **Transitivity** rule

Similar proof for $A \to C$

Finding the closure of a set of functional dependencies using Armstrong's axioms

$F^+ \leftarrow F$
apply the reflexivity rule /* Generates all trivial dependencies */
repeat
    for each functional dependency f in F^+ **do**
        apply the augmentation rule on f
        add the resulting functional dependencies to F^+
    end for
    for each pair of functional dependencies f_1 and f_2 in F^+ **do**
        if f_1 and f_2 can be combined using transitivity **then**
            add the resulting functional dependency to F^+
        end if
    end for
until F^+ does not change any further $=0$

目录

海纳百川 有容乃大

We say that an attribute B is functionally determined by α if $\alpha \rightarrow B$. To test whether a set α is a superkey, we must devise an algorithm for computing the set of attributes functionally determined by α. One way of doing this is to compute F^+, take all functional dependencies with α as the left-hand side, and take the union of the right-hand sides of all such dependencies.

An algorithm to compute $\alpha+$, the closure of α under F

```
result ← α
repeat
    for each functional dependency β → γ in F do
        if β ⊆ result then
            result ← result ∪ γ
        end if
    end for
until result does not change =0
```

test if A is a superkey

Let's consider a table T with the relation schema $R(A, B, C, D, E)$ and the following data:

A	B	C	D	E
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8

$A \rightarrow B$
$B \rightarrow C$
$BC \rightarrow D$
$D \rightarrow E$

check if a functional dependency $A \to E$ holds

1. Initialize the result set to A, i.e., $result \leftarrow A$ Compute the closure of A using the given functional dependencies:
   - Since $A \to B$, we can add B to the result set: $result \leftarrow A, B$
   - Since $B \to C$, we can add C to the result set: $result \leftarrow A, B, C$
   - Since $BC \to D$, we can add D to the result set: $result \leftarrow A, B, C, D$
   - Since $D \to E$, we can add E to the result set: $result \leftarrow A, B, C, D, E$
2. Since A, B, C, D, E contains all attributes in R, we can conclude that A is a superkey for R.

It gives us an alternative way to compute F^+: For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \to S$

例 7

$\gamma = A$, $\gamma^+ = \{A, B, C, D, E\}$, we obtain

- $A \to A$, $A \to B$, $A \to C$, $A \to D$, $A \to E$
- $A \to AB$, $A \to AC$, $A \to AD$, $A \to AE$, $A \to BC$, $A \to BD$, $A \to BE$, $A \to CD$, $A \to CE$, $A \to DE$
- $A \to ABC$, $A \to BCD$, $A \to CDE$, $A \to ABD$, $A \to BCE$, $A \to ACE$, $A \to BDE$
- $A \to ABCD$, $A \to BCDE$, $A \to ABCE$
- $A \to ABCDE$

Suppose that we have a set of functional dependencies F on a relation schema. Whenever a user performs an update on the relation, the database system must ensure that the update does not violate any functional dependencies, that is, all the functional dependencies in F are satisfied in the new database state. The system must roll back the update if it violates any functional dependencies in the set F.

We can reduce the effort spent in checking for violations by **testing a simplified set of functional dependencies** that has the same closure as the given set

A canonical cover F_c for F is a set of dependencies such that F logically implies all dependencies in F_c, and F_c logically implies all dependencies in F. Furthermore, F_c must have the following properties:

- No functional dependency in F_c contains an extraneous attribute.
- Each left side of a functional dependency in F_c is unique. That is, there are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in F_c such that $\alpha_1 = \alpha_2$

The formal definition of extraneous attributes is as follows: Consider a set F of functional dependencies and the functional dependency $\alpha \to \beta$ in F.

- Removal from the left side: Attribute A is extraneous in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \to \beta\}) \cup (\alpha - A) \to \beta$. (Removing an attribute from the left side of a functional dependency could make it a **stronger constraint**. For example, if we have $AB \to C$ and remove B, we get the possibly stronger result $A \to C$.)
- Removal from the right side: Attribute A is extraneous in β if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \to \beta\}) \cup \alpha \to (\beta - A)$ logically implies F. (Removing an attribute from the right side of a functional dependency could make it a **weaker constraint**. For example, if we have $AB \to CD$ and remove C, we get the possibly weaker result $AB \to D$. It may be weaker because using just $AB \to D$, we can no longer infer $AB \to C$.)

Computing canonical cover

$F_c \leftarrow F$
repeat
  for each functional dependency $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ in F_c **do**
    Replace with $\alpha_1 \rightarrow \beta_1\beta_2$
  end for
  for each functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute **do**
    if an extraneous attribute is found in α **then**
      Remove it from $\alpha \rightarrow \beta$ in F_c
    else
      Remove it from β in $\alpha \rightarrow \beta$ in F_c
    end if
  end for
until F_c does not change =0

Assume we are given the following set F of functional dependencies on schema (A, B, C):

$$A \rightarrow BC$$
$$B \rightarrow C$$
$$A \rightarrow B$$
$$AB \rightarrow C$$

Let us compute a canonical cover for F.

- There are two functional dependencies with the same set of attributes on the left side of the arrow:

$$A \rightarrow BC$$
$$A \rightarrow B$$

We combine these functional dependencies into $A \rightarrow BC$.

- A is extraneous in $AB \rightarrow C$ because F logically implies $(F - \{AB \rightarrow C\}) \cup B \rightarrow C$. This assertion is true because $B \rightarrow C$ is already in our set of functional dependencies.
- C is extraneous in $A \rightarrow BC$, since $A \rightarrow BC$ is logically implied by $A \rightarrow B$ and $B \rightarrow C$.

Therefore, the canonical cover of F is:

$$A \rightarrow B$$
$$B \rightarrow C$$

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

If we apply the test for extraneous attributes to $A \rightarrow BC$ using the set F, we can see that both B and C are extraneous.

例 8

To prove that B is extraneous, we need to show that $F - \{A \rightarrow BC\} \cup \{A \rightarrow B\}$ logically implies $A \rightarrow BC$. We can use the **transitivity rule** of functional dependencies to derive $A \rightarrow C$ from $B \rightarrow AC$ and $A \rightarrow B$, and then use these two new dependencies to derive $A \rightarrow BC$ by applying the union rule. Therefore, B is extraneous in $A \rightarrow BC$.

例 9

To prove that C is extraneous, we need to show that $F - \{A \rightarrow BC\} \cup \{A \rightarrow C\}$ logically implies $A \rightarrow BC$. We can use the transitivity rule of functional dependencies to derive $A \rightarrow B$ from $C \rightarrow AB$ and $A \rightarrow C$, and then use these two new dependencies to derive $A \rightarrow BC$ by applying the union rule. Therefore, C is also extraneous in $A \rightarrow BC$ under F.

However, it is incorrect to delete both! The algorithm for finding the canonical cover picks one of the two and deletes it.
Exercise: find all canonical cover for F?

目录

Let F be a set of functional dependencies on a schema R, and let $R_1, R_2, \cdots, R_n$ be a decomposition of R. The restriction of F to R_i is the set F_i of all functional dependencies in F^+ that include only attributes of R_i.

例 10

suppose $F = \{A \rightarrow B, B \rightarrow C\}$, and we have a decomposition into AC and AB. The restriction of F to AC includes $A \rightarrow C$, since $A \rightarrow C$ is in F^+, even though it is not in F.

The set of restrictions $F_1, F_2, \cdots, F_n$ is the set of dependencies that can be checked efficiently. Let $F' = F_1 \cup F_2 \cup \cdots \cup F_n$. If $F'^+ = F^+$, then every dependency in F is logically implied by F', and, if we verify that F' is satisfied, we have verified that F is satisfied. We say that a decomposition having the property $F'^+ = F^+$ is a dependency-preserving decomposition.

computing F^+ is expensive.

Testing for dependency preservation

compute F^+
for all $R_i \in D$ **do**
    $F_i \leftarrow$ the restriction of F^+ to R_i
end for
$F' := \emptyset$
for all F_i **do**
    $F' = F' \cup F_i$
end for
$F'^+ \leftarrow$ the closure of F'
if $F'^+ = F^+$ **then**
    return true
else
    return false
end if=0

The test applies the following procedure to each $\alpha \rightarrow \beta$ in F.

Testing without F^+

```
result = α
repeat
    for all R_i in the decomposition do
        t = (result ∩ R_i)^+ ∩ R_i
        result = result ∪ t
    end for
until result does not change =0
```

The attribute closure here is under the set of functional dependencies F.

If result contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved.

The two key ideas behind the preceding test are as follows:

- The first idea is to test each functional dependency $\alpha \rightarrow \beta$in F to see if it is preserved in F'
- The second idea is to use a modified form of the attribute-closure algorithm to compute closure under F', without actually first computing F'. We wish to avoid computing F' since computing it is quite expensive. Note that F' is the union of all F_i, where F_i is the restriction of F on R_i.

目录

Testing of a relation schema R to see if it satisfies BCNF can be simplified in some cases:

- To check if a nontrivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, compute $\alpha+$ (the attribute closure of α), and verify that it includes all attributes of R; that is, it is a superkey for R.
- To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than check all dependencies in $F+$.

例 11

Consider relation schema (A, B, C, D, E), with functional dependencies F containing $A \rightarrow B$ and $BC \rightarrow D$. Suppose this were decomposed into (A, B) and (A, C, D, E). Now, neither of the dependencies in F contains only attributes from (A, C, D, E), so we might be misled into thinking that it is in BCNF. In fact, there is a dependency $AC \rightarrow D$ in F^+ (which can be inferred using the pseudotransitivity rule from the two dependencies in F) that shows that (A, C, D, E) is not in BCNF. Thus, we may need a dependency that is in F^+, but is not in F, to show that a decomposed relation is not in BCNF.

An alternative BCNF test is sometimes easier than computing every dependency in $F+$. To check if a relation schema R_i in a decomposition of R is in BCNF, we apply this test:

- For every subset α of attributes in R_i, check that $\alpha+$ (the attribute closure of α under F) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i.

If the condition is violated by some set of attributes α in R_i, consider the following functional dependency, which can be shown to be present in F^+:

- $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$.

This dependency shows that Ri violates BCNF.

BCNF decomposition algorithm.

```
result := {R}
done := False
while not done do
    if there is a schema R_i in result that is not in BCNF then
        let α → β be a nontrivial functional dependency that holds on R_i
    such that α⁺ does not contain R_i and α ∩ β = ∅
        result := (result − R_i) ∪ (R_i − β) ∪ (α, β)
else
    done:= True
end if=0
```

Suppose we have a database design using the class relation, whose schema is as shown below:
$class(courseid, title, deptname, credits, secid, semester, year, building,$ room number, capacity, time slot id)
The set of functional dependencies that we need to hold on this schema are:

$$courseid \rightarrow title, deptname, credits$$

$$building, roomnumber \rightarrow capacity$$

$$courseid, secid, semester, year \rightarrow building, roomnumber, timeslotid$$

A candidate key for this schema is $\{courseid, secid, semester, year\}$.

The functional dependency: $courseid \rightarrow title, deptname, credits$ holds, but $courseid$ is not a superkey. Thus, class is not in BCNF.
We replace class with two relations with the following schemas:

$$course(courseid, title, deptname, credits)$$

$class{-}1(courseid, secid, semester, year, building, roomnumbercapacity,$
time slot id)

The functional dependency: $building, roomnumber \rightarrow capacity$ holds on class-1, but $\{building, roomnumber\}$ is not a superkey for class-1.
We replace class-1 two relations with the following schemas:

$$classroom(building, roomnumber, capacity)$$

$$section(courseid, secid, semester, year, building, roomnumber,$$
time slot id)

The "inst" relation

ID	dept_name	name	street	city
1	Computer Science	University of XYZ	123 Main St.	Anytown, USA
2	Information	University of ABC	456 Elm St.	Somewhere, USA
3	Biology	University of LMN	789 Oak St.	Nowhere, USA
3	Biology	University of LMN	777 Oak St.	Nowhere, USA

r_1

ID	name
1	University of XYZ
2	University of ABC
3	University of LMN

r_2 (**being in BCNF, there is redundancy.**)

ID	dept_name	street	city
1	Computer Science	123 Main St.	Anytown, USA
2	Information	456 Elm St.	Somewhere, USA
3	Biology	789 Oak St.	Nowhere, USA
3	Biology	777 Oak St.	Nowhere, USA

r_21

dept_name	ID
Computer Science	1
Information	2
Biology	3

r_22

ID	street	city
1	123 Main St.	Anytown, USA
2	456 Elm St.	Somewhere, USA
3	789 Oak St.	Nowhere, USA
3	777 Oak St.	Nowhere, USA

To deal with this problem, we must define a new form of constraint, called a **multivalued dependency**. As we did for functional dependencies, we shall use **multivalued dependencies** to define a normal

目录

Definition of Multivalued Dependency

Let $r(R)$ be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency $\alpha \twoheadrightarrow \beta$ holds on R if, in any legal instance of relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that

$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
$t_3[\beta] = t_1[\beta]$
$t_3[R - \beta] = t_2[R - \beta]$
$t_4[\beta] = t_2[\beta]$
$t_4[R - \beta] = t_1[R - \beta]$

	α	β	$R-\alpha-\beta$
t_1	$a_1 \cdots a_i$	$a_{i+1} \cdots a_j$	$a_{j+1} \cdots a_n$
t_2	$a_1 \cdots a_i$	$b_{i+1} \cdots b_j$	$b_{j+1} \cdots b_n$
t_3	$a_1 \cdots a_i$	$a_{i+1} \cdots a_j$	$b_{j+1} \cdots b_n$
t_4	$a_1 \cdots a_i$	$b_{i+1} \cdots b_j$	$a_{j+1} \cdots a_n$

ID	dept	street	city
22222	Physics	North	Rye
22222	Physics	Main	Manchester
12121	Finance	Lake	Horseneck

$$\text{ID} \twoheadrightarrow \text{street, city}$$

$$\text{ID} \twoheadrightarrow \text{dept}$$

例 12

From the definition of multivalued dependency, we can derive the following rules for $\alpha, \beta \subseteq R$:

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

例 13

$\alpha \twoheadrightarrow \beta$ is satisfied by all relations on schema R, then $\alpha \twoheadrightarrow \beta$ is a trivial multivalued dependency on schema R. Thus, $\alpha \twoheadrightarrow \beta$ is trivial if $\beta \subseteq \alpha$ or $\beta \cup \alpha = R$.

Functional dependencies

if we know the value of α in a tuple, **we can determine the value of** β **without any ambiguity**. For example, in a table of employee information, if we have the employee ID (which is part of α) we can determine the employee's name (which is part of β).

Multivalued dependencies

if we know the value of α in a tuple, **we cannot determine the value of** β **without ambiguity**. Instead, we need to look at additional tuples that have the same value for α to determine the possible values of β. For example, in a table of employee information, if we have the department (which is part of α), we may need to look at multiple tuples to determine the possible values of manager names (which is part of β) for that department.

Course	Book	Lecturer
DBS	Database System Concepts	Wu
DBS	Fundamentals of Database Systems	Wu
DBS	Database System Concepts	Chen
DBS	Fundamentals of Database Systems	Chen
DBS	Database System Concepts	Li
DBS	Fundamentals of Database Systems	Li
OS	Modern Operating Systems	Liu
OS	Modern Operating Systems	Yi

Because the lecturers attached to the course and the books attached to the course are independent of each other, this database design has a multivalued dependency; if we were to add a new book to the CS course, we would have to add one record for each of the lecturers on that course, and vice versa.

Suppose we want to add the book "Operating System Concepts" by Abraham Silberschatz to OS course.

Add DBS book

```sql
INSERT INTO courses (Course, Book, Lecturer)
VALUES ('IS', 'Operating System Concepts', 'Liu');

INSERT INTO courses (Course, Book, Lecturer)
VALUES ('OS', 'Operating System Concepts', 'Yi');
```

Similarly, if we were to add a new lecturer who teaches the CS course

Add DBS book

```sql
INSERT INTO courses (Course, Book, Lecturer)
VALUES ('OS', 'Modern Operating Systems', 'Zhang');

INSERT INTO courses (Course, Book, Lecturer)
VALUES ('DBS', 'Database System Concepts', 'Zhang');

INSERT INTO courses (Course, Book, Lecturer)
VALUES ('DBS', 'Fundamentals of Database Systems', 'Zhang');
```

Fourth Normal Form

A relation schema 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 $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \twoheadrightarrow \beta$ is a trivial multivalued dependency.
- α is a superkey for R.

A database design is in 4NF if each member of the set of relation schemas that constitutes the design is in 4NF.

例 14

Every 4NF schema is in BCNF. To see this fact, we note that if a schema R is not in BCNF, then there is a nontrivial functional dependency $\alpha \to \beta$ holding on R, where α is not a superkey. Since $\alpha \to \beta$ implies $\alpha \twoheadrightarrow \beta$, R cannot be in 4NF.

The restriction of D to R_i

Let R be a relation schema, and let $R_1, R_2, \ldots, R_n$ be a decomposition of R. 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: $\alpha \twoheadrightarrow \beta \cap R_i$ where $\alpha \subseteq R_i$ and $\alpha \twoheadrightarrow \beta$ is in D^+.

Lossless Decomposition

Let $r(R)$ be a relation schema, and let D be a set of functional and multivalued dependencies on R. Let $r_1(R_1)$ and $r_2(R_2)$ form a decomposition of R. This decomposition of R is lossless if and only if at least one of the following multivalued dependencies is in D^+:

- $R_1 \cap R_2 \twoheadrightarrow R_1$
- $R_1 \cap R_2 \twoheadrightarrow R_2$

Decomposition into 4NF

$result := R$;
$done := false$;
Compute D^+;
for each schema R_i in $result$ **do**
    Let D_i denote the restriction of D^+ to R_i;
end for
while not $done$ **do**
    if there is a schema R_i in $result$ that is not in 4NF w.r.t. D_i **then**
        Let $\alpha \twoheadrightarrow \beta$ be a nontrivial multivalued dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \emptyset$;
        $result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    else
        $done := true$;
    end if
end while=0

fifth normal form

Traveling salesman	Brand	Product type
Jack Schneider	Acme	Vacuum cleaner
Jack Schneider	Acme	Breadbox
Mary Jones	Robusto	Pruning shears
Mary Jones	Robusto	Vacuum cleaner
Mary Jones	Robusto	Breadbox
Mary Jones	Robusto	Umbrella stand
Louis Ferguson	Robusto	Vacuum cleaner
Louis Ferguson	Robusto	Telescope
Louis Ferguson	Acme	Vacuum cleaner
Louis Ferguson	Acme	Lava lamp
Louis Ferguson	Nimbus	Tie rack

products of the type designated by product type, made by the brand designated by brand, are available from the traveling salesman designated by traveling salesman.

fifth normal form

A traveling salesman has certain brands and certain product types in their repertoire. If brand B1 and brand B2 are in their repertoire, and product type P is in their repertoire, then (assuming brand B1 and brand B2 both make product type P), the traveling salesman must offer products of product type P those made by brand B1 and those made by brand B2.

fifth normal form

Traveling salesman	Product type
Jack Schneider	Vacuum cleaner
Jack Schneider	Breadbox
Mary Jones	Pruning shears
Mary Jones	Vacuum cleaner
Mary Jones	Breadbox
Mary Jones	Umbrella stand
Louis Ferguson	Telescope
Louis Ferguson	Vacuum cleaner
Louis Ferguson	Lava lamp
Louis Ferguson	Tie rack

Traveling salesman	Brand
Jack Schneider	Acme
Mary Jones	Robusto
Louis Ferguson	Robusto
Louis Ferguson	Acme
Louis Ferguson	Nimbus

Brand	Product type
Acme	Vacuum cleaner
Acme	Breadbox
Acme	Lava lamp
Robusto	Pruning shears
Robusto	Vacuum cleaner
Robusto	Breadbox
Robusto	Umbrella stand
Robusto	Telescope
Nimbus	Tie rack

fifth normal form

Suppose that Jack Schneider starts selling Robusto's products breadboxes and vacuum cleaners.

Original

```sql
INSERT INTO Traveling-salesman (Traveling salesman, Brand, Product type)
VALUES ('Jack Schneider', 'Robusto', 'breadboxes');

INSERT INTO Traveling-salesman (Traveling salesman, Brand, Product type)
VALUES ('Jack Schneider', 'Robusto', 'vacuum cleaners');
```

Remove redundancy

Decomposition

```sql
INSERT INTO Brands(Traveling salesman, Brand)
VALUES ('Jack Schneider', 'Robusto');
```

Join Dependencies

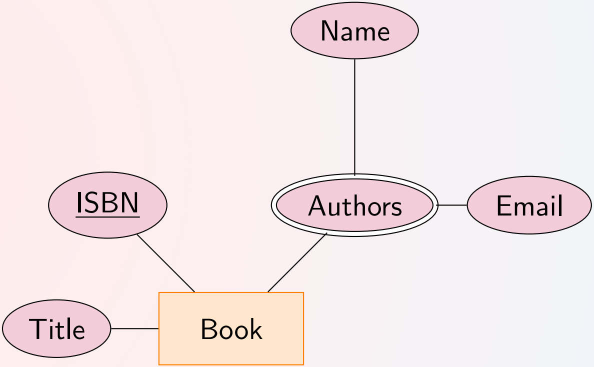
Let R be a relation schema and let $R_1, R_2, \dots, R_n$ be a decomposition of R. The relation $r(R)$ satisfies the join dependency $*(R_1, R_2, \dots, R_n)$ if $\bowtie_{i=1}^{n} \Pi_{R_i}(r) = r$. A join dependency is trivial if one of the R_i is R itself.

例 15

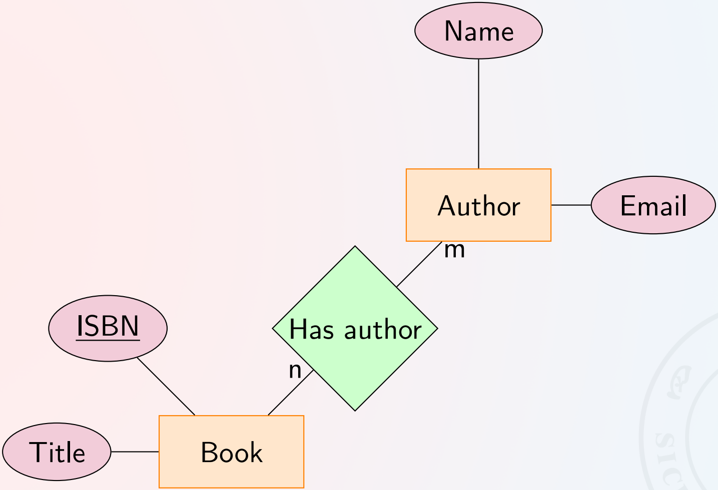
2-ary join dependencies are called multivalued dependency as a historical artifact of the fact that they were studied before the general case. More specifically if U is a set of attributes and R a relation over it, then R satisfies $X \twoheadrightarrow Y$ if and only if R satisfies $*(X \cup Y, X \cup (U - Y))$.

project-join normal form (PJNF/5NF)

A relation is said to be in 5NF if and only if it satisfies **4NF and no join dependency exists**.



One with multivalue attributes



first normal form

1NF

A domain is said to be atomic if its elements are considered indivisible units. A relation schema R is in first normal form (1NF) if all attributes of R have atomic domains.

Book

ISBN	Title	Authors
123456789	"Database Systems"	"Hector Garcia-Molina", "hector@cs.stanford.edu"
234567890	"Introduction to Algorithms"	"Thomas H. Cormen", "cormen@mit.edu"
345678901	"Operating System Concepts"	"Abraham Silberschatz","silberschatz@yale.edu"

Book

ISBN	Title
123456789	"Database Systems"
234567890	"Introduction to Algorithms"
345678901	"Operating System Concepts"

Author

Name	Email
"Hector Garcia-Molina"	"hector@cs.stanford.edu"
"Thomas H. Cormen"	"cormen@mit.edu"
"Abraham Silberschatz"	"silberschatz@yale.edu"

HasAuthor

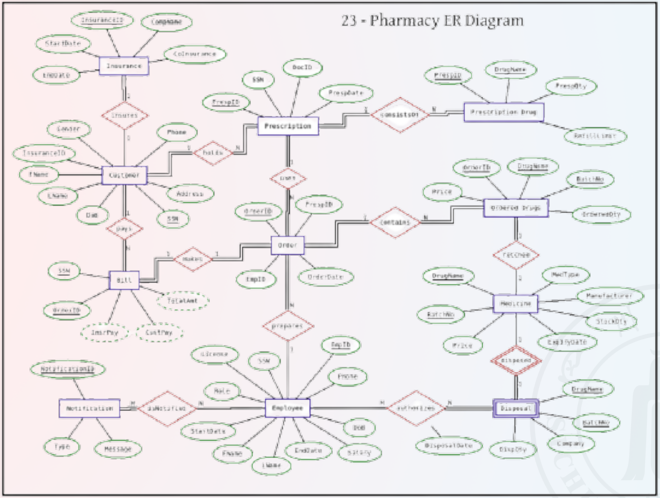
ISBN	Email
123456789	"hector@cs.stanford.edu"
234567890	"cormen@mit.edu"
345678901	"silberschatz@yale.edu"

There are several ways in which we could have come up with the schema $r(R)$:

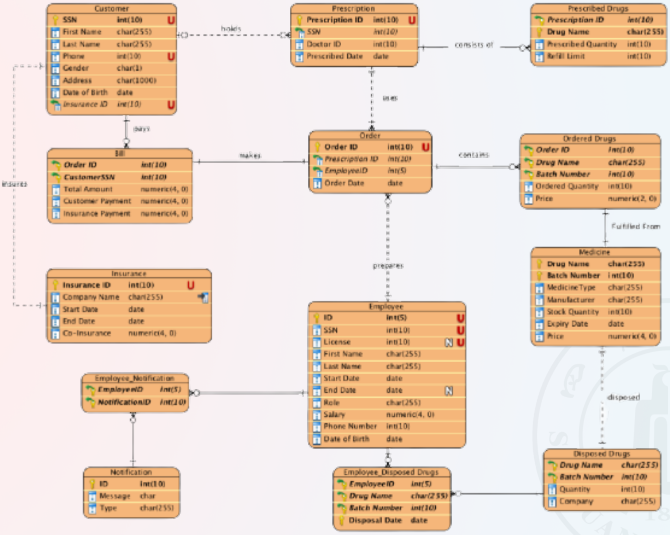
1. $r(R)$ could have been generated in converting an E-R diagram to a set of relation schemas
2. $r(R)$ could have been a single relation schema containing all attributes that are of interest. The normalization process then breaks up $r(R)$ into smaller schemas.
3. $r(R)$ could have been the result of an ad hoc design of relations that we then test to verify that it satisfies a desired normal form.

目录

ER m



https://github.com/rahul1947/Database-Design-Pharmacy-Management-System/blob/master/CS6360.002-Team23-PharmacyManagementSystem.pdf


Customer
SSN int(10) U
First Name char(255)
Last Name char(255)
Phone int(10) U
Gender char(1)
Address char(1000)
Date of Birth date
Insurance ID int(10) U
holds
Prescription
Prescription ID int(10) U
SSN int(10)
Doctor ID int(10)
Prescribed Date date
consists of
Prescribed Drugs
Prescription ID int(10)
Drug Name char(255)
Prescribed Quantity int(10)
Refill Limit int(10)
pays
uses
Bill
Order ID int(10)
CustomerSSN int(10)
Total Amount numeric(4, 0)
Customer Payment numeric(4, 0)
Insurance Payment numeric(4, 0)
makes
Order
Order ID int(10) U
Prescription ID int(10)
EmployeeID int(5)
Order Date date
contains
Ordered Drugs
Order ID int(10)
Drug Name char(255)
Batch Number int(10)
Ordered Quantity int(10)
Price numeric(2, 0)
insures
Fulfilled From
Insurance
Insurance ID int(10) U
Company Name char(255)
Start Date date
End Date date
Co-Insurance numeric(4, 0)
prepares
Medicine
Drug Name char(255)
Batch Number int(10)
MedicineType char(255)
Manufacturer char(255)
Stock Quantity int(10)
Expiry Date date
Price numeric(4, 0)
Employee
ID int(5) U
SSN int(10) U
License int(10) N U
First Name char(255)
Last Name char(255)
Start Date date
End Date date N
Role char(255)
Salary numeric(4, 0)
Phone Number int(10)
Date of Birth date
Employee_Notification
EmployeeID int(5)
NotificationID int(10)
disposed
Notification
ID int(10)
Message char
Type char(255)
Employee_Disposed Drugs
EmployeeID int(5)
Drug Name char(255)
Batch Number int(10)
Disposal Date date
Disposed Drugs
Drug Name char(255)
Batch Number int(10)
Quantity int(10)
Company char(255)

Customer

SSN	First Name	Last Name	Phone	Gender	Address	Date of Birth

Primary Key: SSN, *Foreign Key:* Customer(Insurance ID) $\rightarrow$ Insurance(Insurance ID)

Customer(SSN, First Name, Last Name, Phone, Gender, Address, Date of Birth, Insurance ID)

SSN $\rightarrow$ First Name, Last Name, Phone, Gender, Address, Date of Birth, Insurance ID

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.

目录

Unique-role assumption

Each attribute name has a unique meaning in the database. This prevents us from using the same attribute to mean different things in different schemas

ID	name	number
101	Smith	555-1234
102	Johnson	555-9876
103	Davis	555-5555
104	Lee	555-4321
105	Rodriguez	555-8888

Example table for instructor schema

building	room_number	number	capacity
Packard	101	350	50
Painter	514	453	30
Taylor	3128	438	75
Watson	100	120	35
Watson	120	452	25

Example table for classroom schema

If we try to join the instructor and classroom relations on the number attribute, we may get unexpected or meaningless results.

Meaningless query

```sql
SELECT DISTINCT instructor.name
FROM instructor, classroom
WHERE instructor.number = classroom.number
  AND classroom.capacity > 50;
```

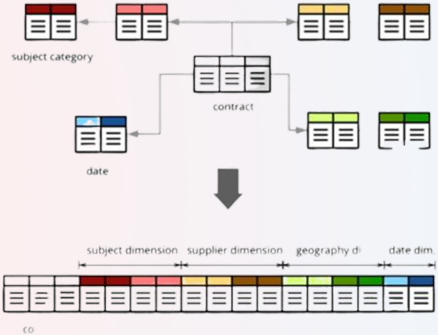
instructor_number, classroom_number

目录

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

Cons of normalization

Normalization eliminates duplicate data, applies changes in one place and **creates more tables and relationships while requiring more joins and indexes** to access the data and imposing a rigid and formal structure.



In the case of BigQuery, and especially using BQ as a backend for a BI platform, denormalized data provides for a quicker user experience because it doesn't have to do the joins when a user hits 'run'.

Temporal data are data that have an associated time interval during which they are valid.

course_id	title	dept_name	credits	start	end
BIO-101	Intro. to Biology	Biology	4	1985-01-01	9999-12-31
CS-201	Intro. to C	Comp. Sci.	4	1985-01-01	1999-01-01
CS-201	Intro. to Java	Comp. Sci.	4	1999-01-01	2010-01-01
CS-201	Intro. to Python	Comp. Sci.	4	2010-01-01	9999-12-31

As per the SQL:2011 standard, the interval is closed on the left-hand side, that is, **the tuple is valid at time start, but is open on the right-hand side, that is, the tuple is valid until just before time end, but is invalid at time end.**

$course\_id \rightarrow title, dept\_name, credits$

Instead, the following constraint would hold: "A course $course_id$ has only one $title$ and $dept\_name$ value at any given time t."

We use the term snapshot of data to mean the value of the data at a particular point in time.

Formally, a temporal functional dependency $\alpha \xrightarrow{\tau} \beta$ holds on a relation schema $r(R)$ if, for all legal instances of $r(R)$, all snapshots of r satisfy the functional dependency $\alpha \rightarrow \beta$.

Formally, if r.A is a temporal primary key of relation r, then whenever two tuples t1 and t2 in r are such that t1.A = t2.A, their valid time intervals of t1 and t2 must not overlap.

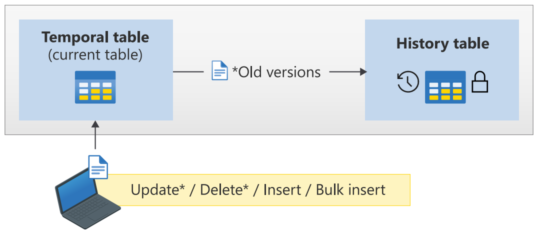
A temporal foreign-key constraint from r.A to s.B ensures the following: for each tuple t in r, with valid time interval (l, u), there is a subset st of one or more tuples in s such that each tuple $s_i \in st$ has $s_i.B = t.A$, and further the union of the temporal intervals of all the s_i contains (l, u).

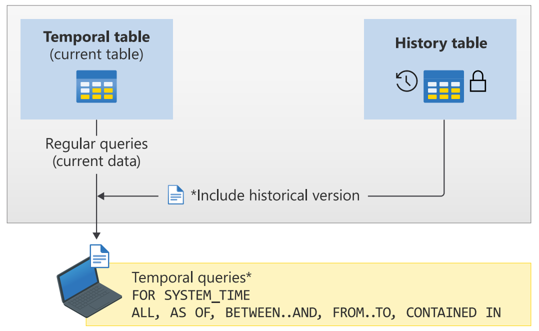
temporal join

the valid time of a tuple in the join result is defined as the intersection of the valid times of the tuples from which it is derived. If the valid times do not intersect, the tuple is discarded from the result.

specify that the tuple is valid in the interval

```sql
CREATE TABLE dbo.Employee
(
  [EmployeeID] int NOT NULL PRIMARY KEY CLUSTERED
  , [Name] nvarchar(100) NOT NULL
  , [Position] varchar(100) NOT NULL
  , [Department] varchar(100) NOT NULL
  , [Address] nvarchar(1024) NOT NULL
  , [AnnualSalary] decimal (10,2) NOT NULL
  , [ValidFrom] datetime2 GENERATED ALWAYS AS ROW START
  , [ValidTo] datetime2 GENERATED ALWAYS AS ROW END
  , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
 )
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.EmployeeHistory))
    ;
```


Temporal table
(current table)
*Old versions
History table
Update* / Delete* / Insert / Bulk insert


Temporal table
(current table)
History table
Regular queries
(current data)
*Include historical version
Temporal queries*
FOR SYSTEM_TIME
ALL, AS OF, BETWEEN..AND, FROM..TO, CONTAINED IN

- **Functional dependencies** are consistency constraints that are used to define two widely used normal forms, **Boyce–Codd normal form (BCNF) and third normal form (3NF)**.
- If the decomposition is **dependency preserving**, all functional dependencies can be inferred logically by considering only those dependencies that apply to one relation.
- A **canonical cover** is a set of functional dependencies equivalent to a given set of functional dependencies, that is minimized in a specific manner to **eliminate extraneous attributes.**
- **Multivalued dependencies** specify certain constraints that cannot be specified with functional dependencies alone. **Fourth normal form (4NF)** is defined using the concept of multivalued dependencies.

- Relational designs typically are based on simple **atomic domains** for each attribute. This is called **first normal form**.
- **Time** plays an important role in database systems. Databases are models of the real world. Whereas most databases model the state of the real world at a point in time (at the current time), temporal databases model the states of the real world across time.
- The reason we could define rigorous approaches to relational database design is that the **relational data model rests on a firm mathematical foundation**.

A functional dependency $\alpha \rightarrow \beta$ is called a partial dependency if there is a proper subset γ of α such that $\gamma \rightarrow \beta$; we say that β is partially dependent on α. A relation schema R is in second normal form (2NF) if each attribute A in R meets one of the following criteria:

- It appears in a candidate key.
- It is not partially dependent on a candidate key.

Show that every 3NF schema is in 2NF.

Thanks
End of Chapter 7