Contents

I Introduction

Database: a very large, integrated collection of data.
Models a real-world enterprise:

- Entities
- Relationship
- Active components

A Database Management System (DBMS) is a software system designed to store, manage and facilitate access to databases.

1. Purpose of Database Systems

1.1 Database Applications

Data processing and management.

1.2 Several Concepts

Database
Data Management System (DBMS): database + a set of program.
Characteristics of DBMS

- Efficiency and scalability in data access.
- Reduced application development time.
- Data independence (including physical data independence and logical data independence).
- Data integrity and security.
- Concurrent access and robustness.

1.3 File-Processing System

File-processing system is supported by a conventional Operating System (OS).
Drawbacks of FPS

- Data redundancy and inconsistency.
- Difficulty in accessing data.
- Data isolation — multiple files and multiple formats.
- Integrity problems.
- No atomicity of updates.
- Difficlut to concurrent access by multiple users.
- Security problems.

2. View of Data

2.1 Levels of Data Abstraction

- Physical level
- Logical level
- View level

2.2 Schemas and Instance

Schemas: the structure of the database on different level.

- Physical schema
- Logical schema
- Subschema

Instance: the actual content of the database at a particular point in time.

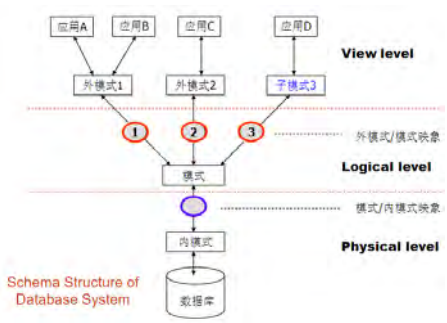


Figure 1: Schemas and Instance

2.3 Physical Independence vs Logical Independence

Ability to modify a schema definition at one level without affecting a schema definition at a higher level.

1) Physical data independence –– the ability to modify the physical schema without changing the logical schema.

2) Logical data independence –– protect application programs from changes in logical structure of data.

2.4 Data Models

Data model is a collection of conceptual for describing.

- data structure
- data relationships
- data semantics
- data constraints

Different level of data abstraction needs different data model to describe.

- Entity-Relationship model
- Relational model
- Other models

3. Database Language

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)

3.1 Data Definition Language

3.2 Data Manipulation Language

Two classes of DMLs

- Procedural DML
- Nonprocedural DML

3.3 SQL

SQL=DDL+DML+DCL

4. Database Design

4.1 Steps of Database Design

1) Requirement analysis
2) Conceptual database design
3) Logical database design
4) Schema refinement
5) Physical database design
6) Create and initialize the database & Security design

4.2 Entity-Relationship Model

E-R model

- Entities (objects)
- Relationships bewteen entities

4.3 Relational Model

Tuple, field

5. Database Users and Administrators

5.1 Database Users

5.2 Database Administrators

6. Transaction Management

Transaction requirement include atomicity, consistence, isolation, durability (acid).

7. Database Architecture

7.1 Storage Manager

include

- Transaction Manager
- Authorization and integrity manager
- File manager
- Buffer manager

7.2 Query Processor

include DDL interpreter, DML compiler, query processing.

Query Processing Optimization

7.3 Overall System Structure

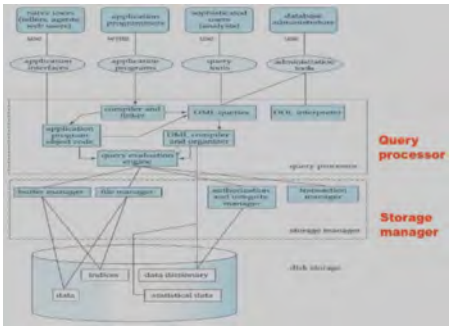


Figure 2: Database System Internals

7.4 Application Architecture

8. History of Database Systems

II Relation Model

A relational database is a collection of one or more relations, which are based on the relational model. A relation is a table with rows and columns. The major advantages of the relational model are its straightforward data representation and the ease with which even complex queries can be expressed.

relationship and relation

- A relationship is an association among several entities.
- A relation is the mathematical concept, referred to as a table.

1. Structure of Relational Databases

1.1 Basic Structure

Formally, given sets $D_1, D_2, \dots, D_n (D_i = a_{ij}|_{j=1,\dots,k})$, a relation r is a subset of $D_1 \times D_2 \times \cdots \times D_n$ — a Cartesian product of a list of domain D_i. Thus a relation is a set of n-tuples $(a_{1j}, a_{2j}, \dots, a_{nj})$, where each $a_{ij} \in D_i (i \in [1, n])$.

1.2 Attribute Types

Each attribute of a relation has a name. The set of allowed values for each attribute is called the domain of the attribute. Attribute values are (normally) required to be atomic, i.e., indivisible. (— 1st NF, 关系理论第一范式). The special value null is a member of every domain. The null value causes complications in the definition of many operations.

1.3 Concepts about Relation

A relation is concerned with two concepts: relation schema and relation instance.

- The relation schema describes the structure of the relation.
- The relation instance corresponds to the snapshot of the data in the relation at a given instant in time.

1.4 Relation Schema

Assume $A_1, A_2, \dots, A_n$ are attributes. Formally expressed: $R = (A_1, A_2, \dots, A_n)$ is a relation schema. $r(R)$ is a relation on the relation schema R.

1.5 Relation Instance

The current values (i.e., relation instance) of a relation are specified by a table. An element t of r is a tuple, represented by a row in a table.

Let a tuple variable t be a tuple, then t[name] denotes the value of t on the name attribute.

1.6 The Properties of Relation

1) The order of tuples is irrelevant.
2) No duplicated tuples in a relation.
3) Attribute values are atomic.

1.7 Database

A database consists of multiple relations. Information about an enterprise (e.g., university) is broken up into parts.

Normalization theory (Chapter 7) deals with how to design "good" relational schemas.

1.8 Key (码/键)

Let $K \subseteq R$.

1) K is a superkey (超码) of R if values for K are sufficient to identify a unique tuple of each possible relation r(R).
2) K is a candidate key (候选码) if K is minimal superkey.
3) K is a primary key (主码), if K is a candidate key and is defined by user explicitly. Primary key is usually marked by underline.

1.9 Foreign Key (外键/外码)

Assume there exists relations r and s: $r(\underline{A}, B, C)$, $s(\underline{B}, D)$, we can say that attribute B in relation r is foreign key referencing s, and r is a referencing relation (参照关系), and s is a referenced relation (被参照关系).

Primary key and foreign key are integrated constraints.

1.10 Schema Diagram

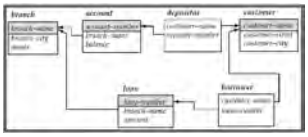


Figure 3: Schema Diagram

1.11 Query Languages

Relational Algebra

2. Fundamental Relational-Algebra Operations

1) Select 选择
2) Project 投影
3) Union 并
4) set difference 差 (集合差)
5) Cartesian product 笛卡儿积
6) Rename 改名 (重命名)

The operators take one or two relations as inputs, and return a new relation as a result.

2.1 Select Operation Formalization

Notation: $\sigma_p(r)$, p is called the selection predicate. Defined as:

$$\sigma_p(r) = \{t | \, t \in r \text{ and } p(t)\}$$

where p is a formula in propositional calculus consisting of terms connected by $\wedge, \vee, \neg$. Each term is one of

$$< attribute > op < attribute > or < constant >$$

where op is one of $=, \neq, >, \geq, <, \leq$.

2.2 Project Operation Formalization

Notation: $\prod_{A_1, A_2, \ldots, A_k}(r)$, where $A_1, A_2, \ldots, A_k$ are attribute names and r is a relation name. The result is defined as the relation of k columns obtained by erasing the columns that are not listed. Duplicate rows removed from result, since relations are sets.

2.3 Union Operation Formalization

Notation: $r \cup s$. Defined as:

$$r \cup s = \{t | \, t \in r \text{ ot } t \in s\}$$

For $r \cup s$ to be valid:

- r and s must have the same arity. (i.e., the same number of attributes)
- The attribute domains must be compatible.

2.4 Set Difference Operation Formalization

Notation: $r - s$. Defined as

$$r - s = \{t | \, t \in r \text{ and } t \notin s\}$$

Set differences must be taken between compatible relations.

- r and s must have the same arity.
- Attribute domains of r and s must be compatible.

2.5 Cartesian-Product Operation Formalization

Notation: $r \times s$. Defined as:

$$r \times s = \{\{t \, q\} \, | \, t \in r \text{ and } q \in s\}$$

Assume that attributes of $r(R)$ and $s(S)$ are disjoint (i.e., $R \cap S = \emptyset$). If attributes of $r(R)$ and $s(S)$ are not disjoint, then renaming for attributes must be used.

2.6 Composition of Operations

Can build expressions using multiple operations.

2.7 Rename Operation

Allows us to name, and therefore to refer to, the results of relational-algebra expressions. (procedural). Allows us to refer to a relation by more than one name.

E.g.

$$\rho_X(E)$$

returns the expression E under the name X.

If a relational-algebra expression E has arity n, then

$$\rho_{X(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression E (对 relation E 及其 attributes 都重命名).

3. Additional Relational-Algebra Operations

1) Set intersection 交
2) Natural join 自然连接
3) Division 除
4) Assignment 赋值

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

3.1 Set-Intersection Operation Formalization

Notation: $r \cap s$. Defined as:

$$r \cap s = \{t | \, t \in r \text{ and } t \in s\}$$

Assume:

- r and s have the same arity.
- attributes of r and s are compatible.

Note: $r \cap s = r - (r - s)$.

3.2 Natural Join Operation Formalization

Notation: $r \bowtie s$.
e.g. R = (A, B, C, D), S = (B, D, E)

- Result schema of the natural-join of r and s = (A, B, C, D, E)
- $r \bowtie s = \prod_{r.A,r.B,r.C,r.D,s.E} (\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$

Let r and s be relations on schemas R and S, respectively. Then, $r \bowtie s$ is a relation on schema $R \cup S$ obtained as follows: Consider each pair of tuples t_r from r and t_s from s. If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where

- t has the same value as t_r on r.
- t has the same value as t_s on s.

3.3 Division Operation Formalization

Notation: $r \div s$. Let r and s be relations on schemas R and S, respectively, where $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $S = (B_1, \ldots, B_n)$. Then, the result of $r \div s$ is a relation on the schema $R–S = (A_1, \ldots, A_m)$ and

$$r \div s = \left\{ t \,\middle|\, t \in \prod_{R-S}(r) \wedge \forall u \in s(tu \in r) \right\}$$

Note that $\prod_{R-S}(r)$ encloses the result of $r \div s$, and meanwhile, the union of the tuple(s) t and all the tuples in s is covered by r (i.e., 商来自于 $\prod_{R-S}(r)$, 并且其元组 t 与 s 所有元组的拼接被 r 覆盖).

Division Operation Characteristic: Let $q = r \div s$, then q is the largest relation satisfying $q \times s \subseteq r$.

3.4 Assignment Operation

The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.

1) Write query as a sequential program consisting of
   - A series of assignments.
   - Followed by an expression whose value is displayed as a result of the query.
2) Assignment must always be made to a temporary relation variable.

e.g. Write $r \div s$ as:

$$temp1 \leftarrow \prod_{R-S}(r)$$

$$temp2 \leftarrow \prod_{R-S}\left((temp1 \times s) - \prod_{R-S}(r)\right)$$

$$result = temp1 - temp2$$

3.5 Summary

- Union, set difference, Set intersection 为双目、等元运算
- Cartesian product, Natural join, Division 为双目运算
- Project, select 为单运算对象 (i.e., 单目运算)

The priority of operations is as follows:

1) Project
2) Select
3) Cartesian Product (times)
4) Join, division
5) Intersection
6) Union, difference

4. Extended Relational-Algebra Operations

1) Generalized Projection 泛化投影
2) Aggregate Functions 聚合处理
3) Outer Join 外链接

4.1 Generalized Projection

Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\prod_{F_1,F_2,\ldots,F_n}(E)$$

where E is any relational-algebra expression, and each of $F_1, F_2, \ldots, F_n$ are arithmetic expressions involving constants and attributes in the schema of E.

4.2 Aggregate Functions and Operations

Aggregation function takes a collection of values and returns a single value as a result.

- avg: average value
- min: minimum value
- max: maximum value
- sum: sum of values
- count: number of values

Aggregate operation in relational algebra

$$_{G_1,G_2,...,G_n}g_{F_1(A_1),F_2(A_2),...,F_n(A_n)}(E)$$

where E is any relational-algebra expression, $G_1, \ldots, G_n$ is a list of attributes on which to group (can be empty), each F_i is an aggregate function, and each A_i is an attribute name.

Result of aggregation does not have a name. For convenience, we permit renaming as part of aggregate operation.

$$_{A}g_{sum(C) \text{ as } \mathrm{abc}}(r)$$

4.3 Outer Join

An extension of the join operation that avoids loss of information. Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join. Uses null values:

1) null signifies that the value is unknown or does not exist.
2) All comparisons involving null are false by definition (roughly speaking). We shall study precise meaning of comparisons with nulls later.

$\supset\!\!\bowtie\!\!\subset$

4.4 Null Values

The result of any arithmetic expression involving null is null.

Aggregate functions simply ignore null values.

For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same.

Comparisons with null values return the special truth value: unknown. Three-valued logic using the truth value unknown:

- OR:
  - (unknown or true) = true,
  - (unknown or false) = unknown
  - (unknown or unknown) = unknown
- AND:
  - (true and unknown) = unknown,
  - (false and unknown) = false,
  - (unknown and unknown) = unknown
- NOT: (not unknown) = unknown
- In SQL "P is unknown" evaluates to true if predicate P evaluates to unknown.

Result of select predicate is treated as false if it evaluates to unknown.

5. Modification of the Database

The content of the database may be modified using the following operations:

1) Deletion
2) Insertion
3) Updating

All these operations are expressed using the assignment operator.

5.1 Deletion

A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

5.2 Insertion

In relational algebra, an insertion is expressed by:

$$r \leftarrow r \cup E$$

where r is a relation and E is a relational algebra expression.

5.3 Update

Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F_1,F_2,...,F_n}(r)$$

where each F_i is either the i-th attribute of r, if the i-th attribute is not updated, or, if the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute.

III SQL

1. Introduction

Structured Query Language (SQL, 结构化查询语言), called Structured English QUEry Language (SEQUEL).

1.1 SQL Conformance Level

SQL Conformance levels (标准符合度) can be classified into 4 categories:

1) Entry level SQL (入门级)
2) Transitional SQL (过渡级)
3) Intermediate SQL (中间级)
4) Full SQL (完全级)

1.2 SQL Operations

SQL includes several parts:

- Data-Definition Language (DDL)
- Data-Manipulation Language (DML)
- Data-Control Language (DCL)

2. Data Definition Language (DDL)

e.g.

```sql
CREATE TABLE branch(
    branch_name char(15) not null,
    branch_city varchar(30),
    assets numeric(8,2),
    primary key (branch_name)
);
```

The main functions of DDL contain:

- Define the schema for each relation
- Define the domain of values associated with each attribute
- Define the integrity constraints
- Define the physical storage structure of each relation on disk
- Define the indices to be maintained for each relations
- Define the view on relations

2.1 Domain types in SQL

- char(n): Fixed length character string, with user-specified length.
- varchar(n): Variable length character strings, with user-specified maximum length n.
- int: Integer (a finite subset of the integers that is machine- dependent).
- smallint: Small integer (a machine-dependent subset of the integer domain type).
- numeric(p, d): Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point.
- real, double precision: Floating point and double-precision floating point numbers,
with machine-dependent precision.
- float(n): Floating point number,
with user-specified precision of at least n digits.
- Null values are allowed in all the domain types. Declaring an attribute to be not null prohibits null values for that attribute.

- date: Dates, containing a (4 digits) year, month and date.
- Time: Time of day, in hours, minutes and seconds.
- timestamp: date plus time of day.

SQL 中有许多函数用于处理各种类型的数据及其类型转换, 但各数据库系统中函数的标准化程度不高.

2.2 Create table

An SQL relation is defined using the create table command:

```sql
CREATE TABLE r(
    A1 D1, ..., An Dn,
    (integrity constraint1),
    ...,
    (integrity constraintk)
)
```

Integrity Constraints in Create Table

- Not null
- Primary key $(A_1, \dots, A_n)$
- Check(P), where P is a predicate

e.g.

```sql
CREATE TABLE branch(
    branch_name char(20) not null,
    branch_city char(30),
    assets integer,
    primary key (branch_name),
    check (assets >= 0)
```

```
);

CREATE TABLE branch2(
    branch_name char(20)
    primary key,
    branch_city char(30),
    assets integer,
    check (assets >= 0)
);
```

2.3 *Drop and alter table*

Drop:

The drop table command deletes all information about the dropped relation from the database.

```
DROP TABLE r;
```

Be careful to use the DROP command.

Alter:

The alter table command is used to add attributes to an existing relation.

```
ALTER TABLE r ADD A D;
ALTER TABLE r ADD (A1 D1, ..., An Dn);
```

where A is the name of the attribute to be added to relation r, and D is the domain of A.

The alter table command can also be used to drop attributes of a relation.

```
ALTER TABLE r DROP A;
```

where A is the name of an attribute in relation r.

The alter table command can also be used to modify the attributes of a relation. e.g.

```
ALTER TABLE branch MODIFY (
    branch_name char(30),
    assets not null
);
```

2.4 *Create index*

```
CREATE INDEX <i-name>
ON <_table-name> (<attribute-list>);

CREATE UNIQUE INDEX <i-name>
ON <_table-name> (<attribute-list>);

DROP INDEX <i-name>;
```

3. Basic Structure

A typical SQL query has the form:

```
SELECT A1, A2, ..., An
FROM r1, r2, ..., rm
WHERE P;
```

where A_i: attribute, r_i: relations, and P: predicate.

This query is equivalent

$$\prod_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m))$$

3.1 *The select clause*

WHERE 可选, 命名用 _, -报错, 关键词不分大小写.

SQL allows duplicates. To force the elimination of duplicates

```
SELECT distinct branch-name
FROM loan;
```

By default, duplicates are allowed, i.e., all is the default.

```
SELECT all branch-name
FROM loan;
```

An asterisk * in the select clause denotes all attributes.

```
SELECT * FROM loan;
```

The select clause can contain arithmetic expressions

```
SELECT loan_number, branch_name, amount * 100
FROM loan;
```

3.2 *The where clause*

The WHERE clause specifies conditions that the result must satisfy. e.g.

```
SELECT loan_number
FROM loan
WHERE branch_name = 'Perryridge' and amount
    > 1200;
```

In WHERE clause, comparison results can be combined using the logical connectives including AND, OR, and NOT, as well as a BETWEEN comparison operator. e.g.

```sql
SELECT loan_number
FROM loan
WHERE amount BETWEEN 90000 AND 100000;
```

3.3 The from clause

Find the Cartesian product. e.g.

```sql
SELECT customer_name, borrower.loan_number,
    amount
FROM borrower, loan
WHERE borrower.loan_number = loan.loan_number
     and
branch_name = 'Perryridge' ;
```

loan(loan-number, branch-name, amount)
borrower(customer-name, loan-number)

3.4 The rename operation

Column Rename:

```sql
old_name as new_name

SELECT  customer_name,
        borrower.loan_number as loan_id,
        amount
FROM borrower, loan;
```

Tuple Variables: for simplification.

```sql
SELECT  customer_name,
        T.loan_number,
        S.amount
FROM borrower as T, loan as S
WHERE T.loan_number = S.loan_number;
```

or for discrimination

```sql
SELECT distinct T.branch_name
FROM branch as T, branch as S
WHERE T.assets > S.assets and S.branch_city =
     'Brooklyn'
```

3.5 String operations

SQL includes a string-matching operator.

- % — matches any substring (likes * in the file system).
- _ — matches any character (like ? in the file system).

e.g.

```sql
SELECT customer_name
FROM customer
WHERE customer_name LIKE '%泽%' ;
```

Match the name "Main%"

```sql
LIKE 'Main\%' escape '\' ;
```

3.6 Ordering the display of tuples

We may specify desc for descending order or asc for ascending order, and for each attribute, ascending order is the default.

```sql
select bra_name from loan
order by bra_name desc;
```

3.7 Duplicates

Select statement in SQL also supports the multiset operators.

4. Set Operations

Each of the operations including UNION, INTERSECT, and EXCEPT automatically eliminates duplicates. To retain all duplicates, we can use the corresponding multiset versions including UNION ALL, INTERSECT ALL, and EXCEPT ALL.

5. Aggregate Functions

These functions (see below) operate on the multiset values of a relation' s column, and return a value.

- avg(col): average value
- min(col): minimum value
- max(col): maximum value
- sum(col): sum of values
- count(col): number of values

e.g. Find the average account balance for each branch.

```sql
SELECT branch_name, avg(balance) avg_bal
FROM account
GROUP BY branch_name;
```

e.g. Find the names of all branches located in city Brooklyn where the average account balance is more than $1,200.

```
SELECT A.branch_name, avg(balance)
FROM account A, branch B
WHERE A.branch_name = B.branch_name and
    branch_city = 'Brooklyn'
GROUP BY A.branch_name
HAVING avg(balance) > 1200;
```

Attributes in select clause outside of aggregate functions must appear in group by list.

6. Summary

The format of SELECT statement:

```
SELECT <[DISTINCT] c1, c2, ...>
FROM <r1, ...>
[WHERE <condition>]
[GROUP BY <c1, c2, ...> [HAVING <cond2>]]
[ORDER BY <c1 [DESC] [, c2 [DESC|ASC],
    ...]>];
```

The execution order of SELECT:

From → where → group (aggregate) → having → select → distinct → order by.

Aggregate functions cannot be used in where clause directly.

7. Null Values

Null is a special marker used in SQL. The result of any arithmetic expression involving 'null' is null. Any comparison with null returns "unknown".

Result of where clause predicate is treated as false if it evaluates to unknown.

e.g. Find all loan number which appears in the loan relation with null values for amount.

```
SELECT loan_number
FROM loan
WHERE amount is null;
```

All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes.

8. Nested Subqueries (嵌套子查询)

A subquery is a select_from_where expression that is nested within another query.

e.g. Find all customers who have both an account and a loan at the bank.

```
SELECT distinct customer_name
FROM borrower
WHERE customer_name in (
    SELECT customer_name
    FROM depositor
);
```

e.g. Find all customers who have loans at a bank but do not have an account at the bank.

```
SELECT distinct customer_name
FROM borrower
WHERE customer_name not in (
    SELECT customer_name
    FROM depositor
);
```

8.1 Set comparison

Definition of Some Clause:

$$C\ \langle comp\rangle \text{ some } r \Leftrightarrow \exists\ t \in r, C\ \langle comp\rangle\ t \text{ holds}$$

where $\langle comp\rangle$ could be $<, \leq, >, \geq, =$ and $\neq$.

(=some) equiv in, ($\neq$ some) not equiv not in.

Definition of All Clause:

$$C\ \langle comp\rangle \text{ all } r \Leftrightarrow \forall\ t \in r, C\ \langle comp\rangle\ t \text{ holds}$$

(=all) not equiv in, ($\neq$ all) equiv not in.

8.2 Test for empty relations

The exists construct returns the value true if the argument subquery is non-empty.

$$\text{exists } r \Leftrightarrow r \neq \emptyset$$
$$\text{not exists } r \Leftrightarrow r = \emptyset$$

8.3 Test for absence of duplicate tuples

The unique construct tests whether a subquery has any duplicate tuples in its result.

unique and not unique

9. Views

Provide a mechanism to hide certain data from the view of certain users. To create a view we use the command:

```
CREATE VIEW <v_name> AS
SELECT c1, c2, ... FROM ...;
CREATE VIEW <v_name> (c1, c2, ...) AS
SELECT e1, e2, ... FROM ...;
```

Benefits of using views: Security, Easy to use, support logical independent.

To drop view:

```
DROP VIEW <v_name>;
```

View and Logical Data Independence.

10. Derived Relations (派生表)

e.g.

```
SELECT branch_name, avg_bal
FROM (
    SELECT branch_name, avg(balance)
    FROM account
    GROUP BY branch_name
) as result (branch_name, avg_bal)
WHERE avg_bal > 500;
```

"as" is must. 没用到也需要别名。

10.1 With Clause

```
WITH max_balance(_value) as
    SELECT max(balance)
    FROM account
SELECT account_number
FROM account, max_balance
WHERE account.balance = max_balance._value;
```

"WITH" defines a local view and in "WHERE" use the local view.

10.2 Complex Query Using With Clause

Use two "WITH".

11. Modification of the Database

11.1 Deletion

```
DELETE FROM <_table|_view>
[WHERE <condition>]
```

在同一 SQL 语句内, 除非外层查询的元组变量引入内层查询, 否则层查询只进行一次.

11.2 Insertion

```
INSERT INTO <_table|_view> [(c1,c2,...)]
VALUES (e1,e2,..)

INSERT INTO <_table|_view> [(c1,c2,...)]
SELECT e1,e2,...
FROM ...
```

The "select from where" statement is fully evaluated before any of its results are inserted into the relation.

11.3 Updates

```
UPDATE <_table|_view>
SET <c1=e1[, c2=e2, ...]>
[WHERE <condition>]
```

Case Statement for Conditional Updates

e.g.

```
UPDATE account
SET balance = case
    when balance <= 10000
    then balance * 1.05
    else balance * 1.06
end
```

Update of a View

建立在单个基本表上的视图, 且视图的列对应表的列, 称为 "行列视图"

- View 是虚表, 对其进行的所有操作都将转化为对基表的操作.
- 查询操作时, VIEW 与基表没有区别, 但对 VIEW 的更新操作有严格限制, 如只有行列视图, 可更新数据.

11.4 Indexes

```
create table student(
    ID varchar (5),
    name varchar (20) not null,
    dept_name varchar (20),
    tot_cred numeric (3,0) default 0,
    primary key (ID)
)

create index studentID_index on student(ID)
```

Indices are data structures used to speed up access to records with specified values for index attributes.

11.5 Transactions

A transaction is a sequence of queries and data update statements executed as a single logical unit. Transactions are started implicitly and terminated by one of

- COMMIT WORK: makes all updates of the transaction permanent in the database.
- ROLLBACK WORK: undoes all updates performed by the transaction

The four properties of transaction are required: atomicity, isolation, consistency, durability.

12. Joined Relations

Join operations take as input two relations and return as a result another relation.

- Join condition — defines which tuples in the two relations match, and what attributes are present in the result of the join.

Join Conditions

```
Natural on <predicate>
using (A1,A2,...,An)
```

- Join type — defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join Types
inner join
left outer join
right outer join
full outer join

12.1 Combination of Join Type and Join Condition

- Natural join:

```
R natural {inner join, left join, right
    join, full join} S
```

- Unnatural join:

```
R {inner join, left join, right join, full
     join} S
ON <predicate>
USING (A1,A2,...,An)
```

Key word Inner, outer is optional.

- Natural join: 以同名属性相等作为连接条件
- Inner join: 只输出匹配成功的元组
- Outer join: 还要考虑不能匹配的元组

12.2 Joined Relations in SQL

- 非自然连接, 容许不同名属性的比较, 且结果关系中不消去重名属性.
- 使用 using 的连接类似于 natural 连接, 但仅以 using 列出的公共属性为连接条件.

IV Advanced SQL

1. SQL Data Types and Schemas

Some build-in data types in sql.

1.1 User-defined types

- Structured data types
- Distinct types

```
CREATE  TYPE person_name as varchar(20);
CREATE student(
    sno char(10) primary key,
    sname person_name
);
Drop TYPE person_name;
```

1.2 Create new domain

```
CREATE DOMAIN Dollars as numeric(12,2) not
    null;
CREATE DOMAIN Pounds as numeric(12,2);
```

Domain: Constraints, not strongly types.

1.3 Large-object types

Large objects (e.g., photos, videos, CAD files, etc.) are stored as a large object:

- blob: binary large object — object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system).
- clob: character large object – object is a large collection of character data.

When a query returns a large object, a pointer is returned rather than the large object itself.

e.g.

```
CREATE TABLE students(
    sid char(10) primary key,
    photo blob(20MB),
    cv clob(10KB)
);
```

1.4 Catalogs, schemas and environments

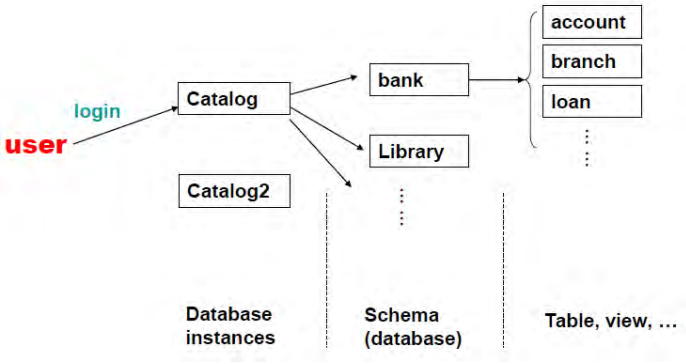


Figure 4: Catalogs, schemas and environments

2. Integrity Constraints (完整性约束)

Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.

- 实体完整性、参照完整性和用户定义的完整性约束
- 完整性约束是数据库实例 (Instance) 必须遵循的
- 完整性约束由 DBMS 维护

Constraints on a single relation

- Not null
- Primary key
- Unique
- Check(P), where P is a predicate

2.1 Domain Constraints

The check clause in SQL-92 permits domains to be restricted. The clause constraint value-test is optional; useful to indicate which constraint an update violated.

2.2 Referential Integrity

Definition IV.1 *Let $r_1(R_1)$ and $r_2(R_2)$ be the relations with primary keys K_1 and K_2, respectively. The subset α of R_2 is a foreign key referential K_1 in relation r_1, if for every t_2 in r_2 there must be a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$. Referential integrity constraints also called subset dependency, since its can be written as*

$$\prod_{\alpha}(r_2) \subseteq \prod_{K_1}(r_1)$$

Assume there exists relations r and s: r(A, B, C), s(B, D), we say attribute B in r is a foreign key from relation r, and r is called $referencingrelation$ (参照关系), and s is called $referencedrelation$ (被参照关系).

参照关系中外码的值必须在被参照关系中实际存在, 或为 null.

2.3 Checking Referential Integrity

The following tests must be made in order to preserve the following referential integrity constraint:

$$\prod_{\alpha}(r_2) \subseteq \prod_{K}(r_1)$$

α in r_2 is a Foreign Key.

Insert: If a tuple t_2 is inserted into r_2, the system must ensure that there is a tuple t_1 in r_1 such that $t_1[K] = t_2[\alpha]$, i.e.

$$t_2[\alpha] \in \prod_{K}(r_1)$$

Delete: If a tuple t_1 is deleted from r_1, the system must compute the set of tuples in r_2 that reference t_1:

$$\sigma_{\alpha=t_1[K]}(r_2)$$

If this set is not empty, then either the delete command is rejected or the tuple in t_2 that references t_1 must themseleves be deleted (cascading deletions are possible).

Update: If a tuple t_2 is updated in relation r_2 and the update modifies values for foreign key α, then a test similar to the insert case is made. Let t_2' denote the new value of tuple t_2, the system must ensure that

$$t_2'[\alpha] \in \prod_{K}(r_1)$$

2.4 Referential Integrity in SQL

Primary, candidate keys, and foreign keys can be specified as part of the SQL create table statement:

- The primary key clause lists attributes that comprise the primary key.
- The unique key clause lists attributes that comprise a candidate key.
- The foreign key clause lists the attributes that comprise the foreign key, and the name of the relation referenced by the foreign key.

```
Create table depositor(
    customer_name char(20),
    account_number char(10),
    primary key (customer_name,
        account_number),
    foreign key (account_number) references
        account,
    foreign key (customer_name) references
        customer
);
```

2.5 Cascading Actions in SQL

```
Create table account (
    . . .
    foreign key (branch-name) references
        branch
    [ on delete cascade]
    [ on update cascade ]
    . . .
);
```

2.6 Assertions

An assertion is a predicate expressing a condition that we wish the database always to satisfy. — for complex check condition on several relations. An assertion in SQL takes the form

```
    CREATE ASSERTION <assertion_name> CHECK <
        predicate>;
```

When an assertion is made, the system tests it for validity on every update that may violate the assertion.

But SQL does not provide a construct for asserting:

$$\text{for all } X, P(X)$$

So it is achieved in a round-about fashion, using:

$$\text{not exists } X, \text{ such that not } P(X).$$

$$\forall x P(x) = \neg \exists x \neg P(x)$$

e.g. 对每一笔借款, 至少有一个借款人有存款 $ 1000 以上.

```
CREATE ASSERTION balance_constraint CHECK(
    not exists (
```

```
        select * from loan L
        where not exists (
            select *
            from borrower B, depositor D,
                account A
            where L.loan_number = B.
                loan_number
            and B.customer_name = D.
                customer_name
            and D.account_number = A.
                account_number
            and A.balance >= 1000
        )
    )
);
```

2.7 *Triggers*

A trigger is a statement that is executed automatically by the system as a side-effect of a modification to the database.

Triggering event can be insert, delete or update. Triggers on update can be restricted to specific attributes. Values of attributes before and after an update can be referenced:

- Referencing old row as: for deletes and updates
- Referencing new row as: for inserts and updates

Statement Level Triggers: Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction.

1) Use for each statement instead of for each row
2) Use referencing old table or referencing new table to refer to temporary tables (called transition tables) containing the affected rows
3) Can be more efficient when dealing with SQL statements that update a large number of rows

Triggers were used for tasks:

1) Maintaining summary data
2) Replicating databases by recording changes to special relations (called change or delta relations) and having a separate process that applies the changes over to a replica.

3. Authorization

Security

Forms of authorization on parts of the database:

1) Read authorization - allows reading, but not modification of data.
2) Insert authorization - allows insertion of new data, but not modification of existing data.
3) Update authorization - allows modification, but not deletion of data.
4) Delete authorization - allows deletion of data.

Forms of authorization to modify the database schema:

1) Index authorization - allows creation and deletion of indices.
2) Resources authorization - allows creation of new relations.
3) Alteration authorization - allows addition or modifying of attributes in a relation.
4) Drop authorization - allows deletion of relations.

3.1 *Authorization and Views*

A combination of relational-level security and view-level security can be used to limit a user' s access to precisely the data that user needs.

Creation of view does not require resources authorization since no real relation is being created.

3.2 *Granting of Privileges*

P46

3.3 *Security Specification in SQL*

```
GRANT <privilege list> ON <_table | _view>
TO <user list>
```

<user list> is:

- user-ids
- public, which allows all valid users the privilege granted
- A role (more details about this later)

3.4 *Privileges in SQL*

1) Select: allows read access to relation, or the ability to query using the view.
2) Insert: the ability to insert tuples.
3) Update: the ability to update using the SQL update statement.

4) Delete: the ability to delete tuples.

5) References: ability to declare foreign keys when creating relations.

6) All privileges: used as a short form for all the allowable privileges.

7) All

8) With grant option: Allows a user who is granted a privilege to pass the privilege on to other users.

3.5 Roles

Roles permiting common privileges for a class of users can be specified just once, by creating a corresponding "role".

3.6 Revoking Authorization in SQL

The revoke statement is used to revoke authorization.

```
REVOKE <privilege list> ON <_table | -view>
FROM <user list> [_restrict | _cascade]
```

3.7 Limitations of SQL Authorization

1) SQL does not support authorization at a tuple level.

2) All end-users of an application (such as a web application) may be mapped to a single database user.

3) The task of authorization in above cases falls on the application program, with no support from SQL.

3.8 Audit Trails (审计)

An audit trail is a log of all changes (inserts/deletes /updates) to the database along with information such as which user performed the change, and when the change was performed.

语句审计:

```
AUDIT <st-opt> [BY <users>] [BY SESSION |
    ACCESS] [WHENEVER_SUCCESSFUL |
    WHENEVER_NOT_SUCCESSFUL]
```

- 当 BY <users> 缺省, 对所有用户审计.
- BY SESSION 每次会话期间, 相同类型的需审计的 SQL 语句仅记录一次.
- 常用的 <St-opt>: table, view, role, index, ...
- 取消审计: NOAUDIT ...(其余同 audit 语句)

对象 (实体) 审计:

```
AUDIT <obj-opt> ON <obj> | DEFAULT [BY
    SESSION | BY ACCESS] [WHENEVER_SUCCESSFUL
    | WHENEVER_NOT_SUCCESSFUL]
```

- obj-opt: insert, delete, update, select, grant, ...
- 实体审计对所有的用户起作用.
- ON <obj> 指出审计对象表、视图名.
- ON DEFAULT 对其后创建的所有对象起作用.
- 取消审计: NOAUDIT ...

4. Embedded SQL

SQL 的功能不完备性.

A language in which SQL queries are embedded is referred to as a Host language (宿主语言), and the SQL structures permitted in the host language comprise embedded SQL.

5. Dynamic SQL

Allows programs to construct and submit SQL queries at run time.

6. ODBC

Open DataBase Connectivity (ODBC, 开放数据库互连). ODBC 提供了一个公共的、与具体数据库无关的应用程序设计接口 API. 它为开发者提供单一的编程接口, 这样同一个应用程序就可以访问不同的数据库服务器.

6.1 Prepared Statement

解决 sql 注入

V Entity-Relationship Model

Steps of Database Design

1) Requirement analysis
   What data, applications, and operations needed.
2) Conceptual database design
   A high-level description of data, constraints using Entity-Relationship (E-R) model or a similar high level data model.
3) Logical database design
   Convert the conceptual design into a DB schema.
4) Schema refinement
   Normalization of relations: Check relational s- chema for redundancies and related anomalies.
5) Physical database design
   Indexing, query, clustering, and database tuning.
6) Create and initialize the database & Security design
   Load initial data, testing.
   Identify different user groups and their roles.

1. Entity Sets

1.1 Entity

The real world can be modeled as:

- A collection of entities (实体)
- Relationships (联系) among entities

An entity is an object that exists and is distinguishable from other objects. — An entity may be concrete, or abstract.

Entities have attributes (属性).

An entity set is a set of entities of the same type that share the same properties.

1.2 Attributes

An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set. Domain (域, value set) is the set of permitted values for each attribute.

Attribute types:

1) Simple and composite attributes (简单和复合属性，如 sex, name).
2) Single-valued and multi-valued attributes(单值和多值属性).
   E.g., multivalued attribute: phone-numbers (多个电话号码).
3) Derived attributes (派生属性).
   Can be computed from other attributes, e.g., age, given date of birth.

versus base attributes or stored attributes (基属性，存储属性).

1.3 Composite Attributes

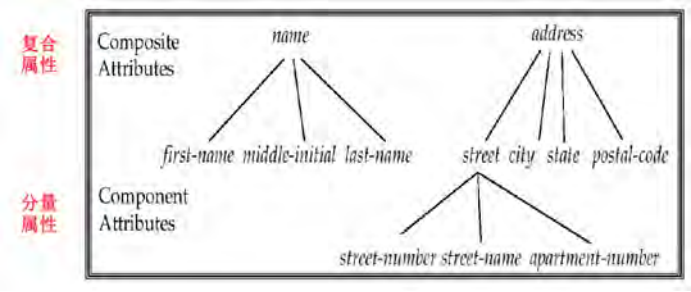


Figure 5: Composite Attributes

2. Relationship Sets

A relationship is an association among several entities (是二个或多个不同类实体之间的关联). A relationship set is a set of relationship of the same type.

Definition V.1 *A relationship set is a mathematical relation among $n \geq 2$ entities, each taken from entity sets*

$$\{(e_1, e_2, \dots, e_n) | e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

where $(e_1, e_2, \dots, e_n)$ is a relationship, E_i is an entity set.

2.1 Degree of a Relationship Set

Refers to the number of entity sets that participate in a relationship set.

1) Relationship sets that involve two entity sets are binary (or degree two).
2) Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)

2.2 Mapping Cardinalities

Express the number of entities to which another entity can be associated via a relationship set. (一个联系集中，一个实体可以与另一类实体相联系的实体数目。其中数目是指最多一个还是多个)

Most useful in describing binary relationship sets. For a binary relationship set the mapping cardinality must be one of the following types:

- One to one (1 : 1)
- One to many (1 : n)

- Many to one (n : 1)
- Many to many (n : m)

Note: Some elements in entity set A and B may not be mapped to any elements in the other set.

3. Keys

3.1 Keys for Entity Sets

1) A super key of an entity set is a set of one or more attributes whose values uniquely determine each entity.

2) A candidate key of an entity set is a minimal super key.

3) Although several candidate keys may exist, one of the candidate keys is selected to be the primary key.

3.2 Keys for Relationship Sets

The combination of primary keys of the participating entity sets forms a super key of a relationship set (参与一个联系集的各实体集的码的组合，构成该联系集的超码). Must consider the mapping cardinality of the relationship set when deciding what are the candidate keys (1:1, 1:n, m:n). Need to consider semantics of relationship set in selecting the primary key in case of more than one candidate key.

4. E-R Diagram

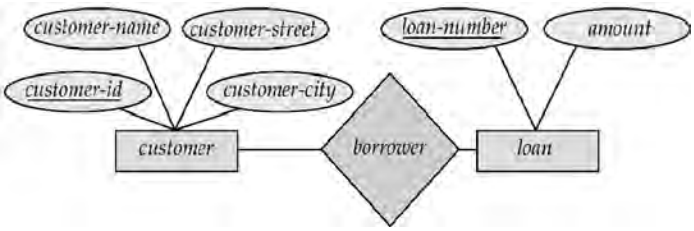


Figure 6: E-R Diagrams

- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Lines link attributes to entity sets and entity sets to relationship sets.
- Ellipses represent attributes.
  Double ellipses represent multivalued attributes.
  Dashed ellipses denote derived attributes.
- Underline indicates primary key attributes (will study later).

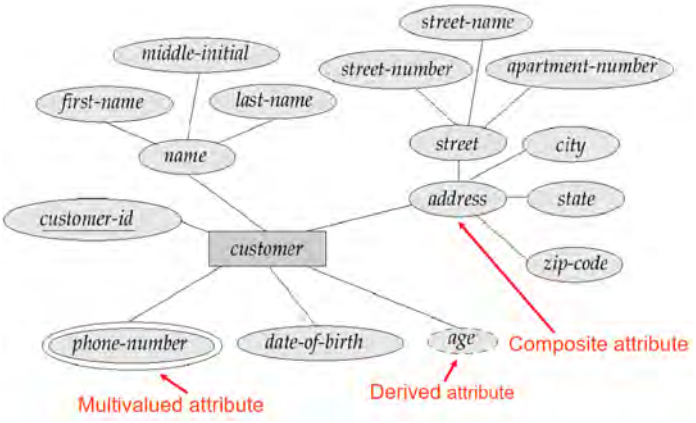


Figure 7: E-R Diagram With Composite, Multivalued and Derived Attributes

Role: the function that an entity plays in a relationship. Role labels are optional, and are used to clarify semantics of the relationship.

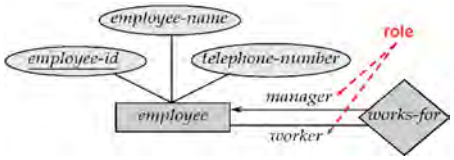


Figure 8: Role

4.1 Express the Cardinality Constraints

We express cardinality constraints by drawing either a directed line (→), signifying "one", or an undirected line (—), signifying "many", between the relationship set and the entity set.

4.2 Participation of an Entity Set in a Relationship Set

- Total participation (全参与) (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set.
- Partial participation (部分参与): some entities may not participate in any relationship in the relationship set.

映射基数约束 (Mapping cardinality constraints)，限定了一个实体与发生关联的另一端实体可能关联的数目上限。

全参与和部分参与约束，则反映了一个实体参与关联的数目下限: 0 次，还是至少 1 次。

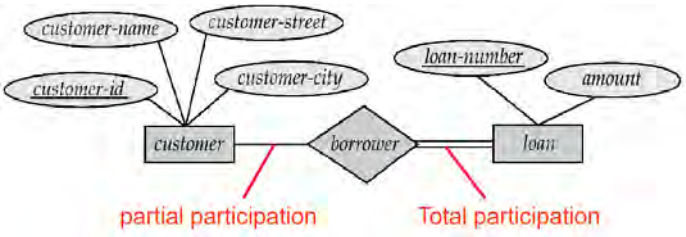


Figure 9: Participation

4.3 Alternative Notation for relationship Constraints

Alternative notation for cardinality constraints and participation constraints.

E.g., 一个客户可以借多笔或 0 笔贷款, 一笔贷款至少、至多属于一个客户.

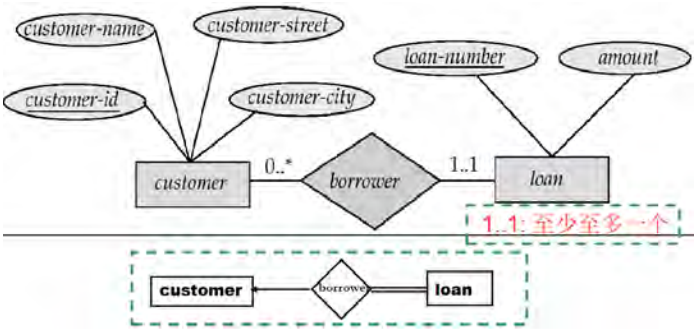


Figure 10: Alternative Notation

4.4 E-R Diagram with a Ternary Relationship

一个银行职员在多个支行兼职, 并承担不同类型的工作.

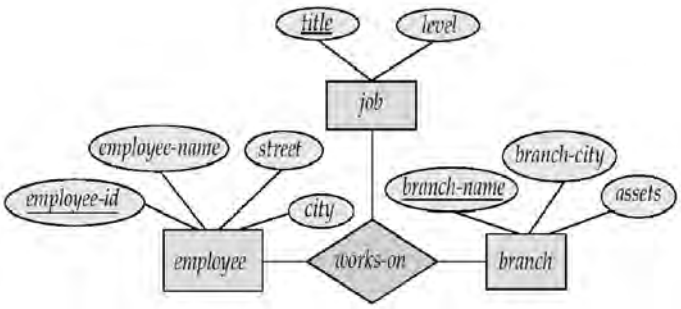


Figure 11: E-R Diagram with a Ternary Relationship

4.5 Binary vs. Non-Binary Relationships

Some relationships that appear to be non-binary may be better represented using binary relationships. Using two binary relationships allows partial information. But there are some relationships that are naturally non-binary.

4.6 Converting Non-Binary Relationships to Binary Form

In general, any non-binary relationship can be represented using binary relationships by creating an artificial entity set.

1) Replace non-binary relationship R between entity sets A, B, and C by an entity set E, and three new relationship sets.

2) Create a special identifying attribute for E.

3) Add any attributes of R to E.

4) For each relationship $(a_i,\ b_i,\ c_i)$ in R, create $R_A,\ R_B,\ R_C$.

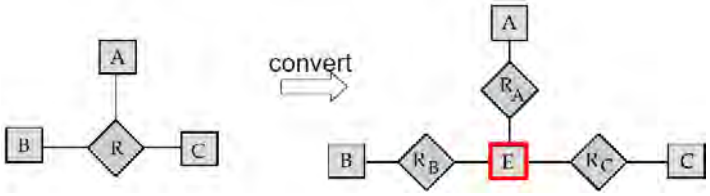


Figure 12: Converting

5. Weak Entity Sets

An entity set that does not have a primary key is referred to as a weak entity set (弱实体集).

The existence of a weak entity set depends on the existence of a identifying entity set or owner entity set (标识实体集或属主实体集).

The related relationship is called identifying relationship (标识性联系).

The discriminator or partial key (分辨符或部分码) of a weak entity set is the set of attributes that distinguishes among all those entities in a weak entity set that depend on one particular strong entity. The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set’s discriminator.

Note: the primary key of the strong entity set is not explicitly stored with the weak entity set, since it is implicit in the identifying relationship.

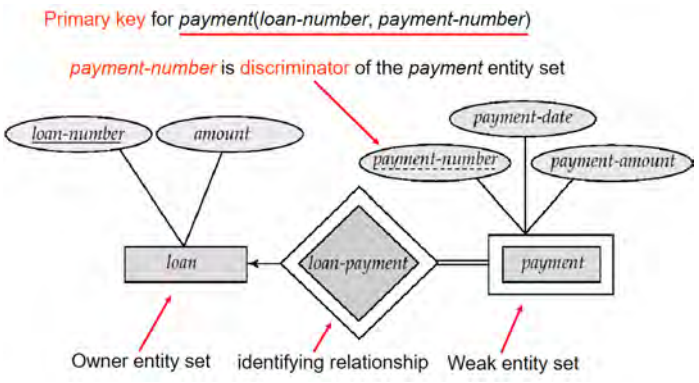


Figure 13: Weak Entity Sets

6. Extended E-R Features

Stratum of the entity set:

1) Specialization (特殊化、具体化): Top-down design process

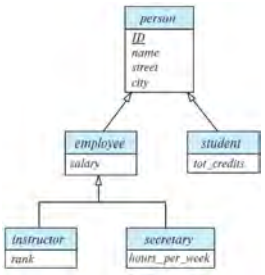


Figure 14: Specialization Example

2) Generalization (泛化、普遍化): Bottom-up design process

6.1 Design Constraints on a Specialization / Generalization

1) Constraint on which entities can be members of a given lower-level entity set.

- Condition-defined (条件定义的)
- User-defined

2) Constraint on whether or not entities may belong to more than one lower-level entity set within a single generalization.

- Disjoint (不相交)

  An entity can belong to only one lower-level entity set.

  Noted in E-R diagram by writing disjoint next to the ISA triangle.
- Overlapping (可重叠)

  An entity can belong to more than one lower-level entity set.

3) Completeness constraint (完全性约束) – specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.

- Total: an entity must belong to one of the lower-level entity sets.
- Partial: an entity need not belong to one of the lower-level entity sets.

6.2 Aggregation

Eliminate this redundancy via aggregation.

- Treat relationship as an abstract entity.
- Allows relationships between relationships.
- Abstraction of relationship into new entity.

6.3 Summary of Symbols Used in E-R Notation

15

7. Design of an E-R Database Schema

7.1 E-R Design Decisions

1) Use an attribute or entity set to represent an object?

若一个对象只对其名字及单值感兴趣，则可作为属性，如性别；若一个对象除名字外，本身还有其他属性需描述，则该对象应定义为实体集。如电话, 部门.

一个对象不能同时作为实体和属性.

一个实体集不能与另一实体集的属性相关联，只能实体与实体相联系.

2) Use it as an entity set or a relationship set?

Relationship set — to describe an action that occurs between entities (二个对象之间发生的动作 — 用“relationship set” 表示).

The mapping cardinality will effect the matter.

3) Use it as an attribute of an entity or a relationship?

要从对象的语义独立性和减少数据冗余方面考虑

4) The use of a ternary or n-ary relationship versus a pair of binary relationships.

5) The use of a strong or weak entity set.

6) The use of specialization/generalization – contributes to modularity in the design (有助于模块化).

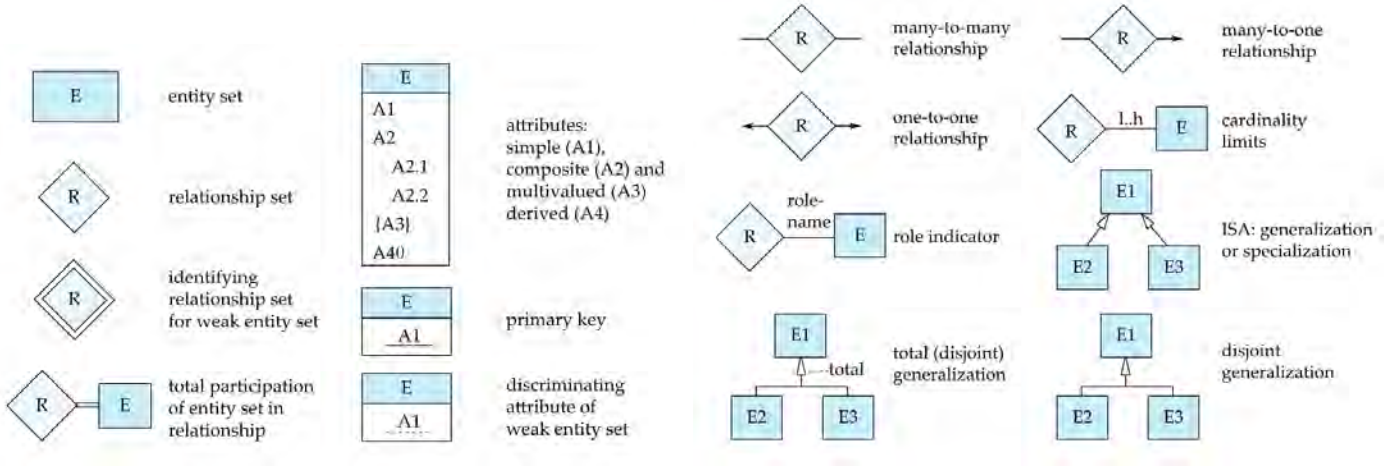


Figure 15: E-R Notation

7) The use of aggregation –– can group a part of E-R diagram into a single entity set, and treat it as a single unit without concern for the details of its internal structure.

8. Reduction of an E-R Schema to Tables

Converting an E-R diagram to a table format is the basis for deriving a relational database design from an E-R diagram.

8.1 Entity Sets

1) Entity Set with Composite Attributes: Composite attributes are flattened out by creating a separate attribute for each component attribute.

2) Entity Set with Multivalued Attributes: A multivalued attribute M of an entity E is represented by a separate table EM. Each value of the multivalued attribute maps to a separate row of the table EM.

3) Representing Weak Entity Sets: A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set.

8.2 Representing Relationship Sets as Tables

A relationship set is represented as a table with columns for the primary keys of the two participating entity sets, (which are foreign keys here) and any descriptive attributes of the relationship set itself.

1) tables for many to many relationship sets: The two attributes corresponding to the two entity sets become the primary key of the table.

2) tables for many to one relationship sets: The attribute corresponding to the entity set on the “many” side becomes the primary key of the table.

3) tables for one to one relationship sets: like 2)

8.3 Redundancy of Tables

1) Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the “many” side, containing the primary key of the one side (对 1:n 联系，可将“联系”所对应的表，合并到对应“多”端实体的表中).

If participation is partial on the many side, replacing a table by an extra attribute in the relation corresponding to the “many” side could result in null values.

For one-to-one relationship sets, either side can be chosen to act as the “many”side. That is, extra attribute can be added to either of the tables corresponding to the two entity sets.

2) The table corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant (联系弱实体集及其标识性实体集的联系集对应的表是冗余的，即对应 identifying relationship 的表是多余的。).

8.4 Representing Specialization as Tables

- Method 1

  1) Form a table for the higher level entity.

  2) Form a table for each lower level entity set, including primary key of higher level entity set and local attributes.

  3) Drawback: getting information about.

- Method 2

1) Form a table for each entity set with all local and inherited attributes.

2) If specialization is total, table for generalized entity not required to store information.

3) Drawback: Attributes may be stored redundantly

VI Relational Database Design

1. First Normal Form

Domain is atomic if its elements are considered to be indivisible units.

Definition VI.1 *A relational schema R is in first normal form (1NF) if the domains of all attributes of R are atomic.*

For the relational database, it' s required that all relations are in 1NF.

How to deal with non-atomic values:

- For composite attributes: use a number of attributes.
- For multi-value attributes
  - Use multi fields.
  - Use a separate table.
  - Use a single field.

Drawbacks of non-atomic strategy:

1) Complicate storage
2) Encourage redundant storage of data
3) Complicated to query

Atomicity is actually a property of how the elements of the domain are used. E.g. Don't use string to store all information.

2. Pitfalls in Relational Database Design

Relational database design requires that we find a "good" collection of relation schemas.

A bad design may lead to: Redundant storage, insert / delete / update anomalies — inability to represent certain information.

2.1 Decomposition

Main refinement technique: decomposition.

1) All attributes of an original schema (R) must appear in the decomposition (R_1, R_2)

$$R = R_1 \cup R_2$$

2) Lossless-join decomposition (无损连接分解), i.e., for all possible relations r on schema R

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

2.2 Goal: Devise a Theory for the Following

1) Decide whether a particular relation R is in "good" form. — No redundant.

2) In the case that a relation R is not in "good" form, decompose it into a set of relations $R_1, R_2, \ldots, R_n$ such that

- Each relation is in good form.
- The decomposition is a lossless-join decomposition.

Our theory is based on:

- Functional dependencies (函数依赖)
- Multivalued dependencies (多值依赖)

3. Functional Dependencies

Definition VI.2 *Let R be a relation schema, α and β be attributes, i.e. $\alpha \subset R$, $\beta \subset R$.*

The functional dependency $\alpha \to \beta$ holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

β is functionally dependent on α, α functionally determines β. (β 函数依赖于 α, α 函数决定 β.)

Functional dependency — a kind of integrity constraints, which express the relationship of values on specific attributes, can be used to judge schema normalization and to suggest refinements.

Functional dependencies allow us to express constraints that cannot be expressed using keys.

3.1 The Use of Functional Dependencies

1) Test relations are legal under a set of functional dependencies F or not.

If a relation r is legal under a set F of functional dependencies, we say that r satisfies F.

2) Specify constraints (F) on the set of legal relations — schema.

We say that F holds on R (F 在 R 上成立) if all legal relations r on R satisfy the set of functional dependencies F.

3.2 Definition of Trivial and Non-Trivial Dependency

Definition VI.3 *A functional dependency is trivial (平凡的) if it is satisfied by all relations. In general, $\alpha \to \beta$ is*

trivial if $\beta \subseteq \alpha$, otherwise, is non-trivial, i.e.,

Trivial: $\alpha \to \beta$, if $\beta \subseteq \alpha$ (平凡的函数依赖)

Non-trivial: $\alpha \to \beta$, if $\beta \not\subseteq \alpha$ (非平凡的函数依赖)

3.3 Closure of a Set of Functional Dependencies

Definition VI.4 *The set of all functional dependencies logically implied by F is the colsure of F (函数依赖集 F 的闭包), denoted by F^+.*

Armstrong's Axioms provide inference rules to find F^+

1) If $\beta \subseteq \alpha$, then $\alpha \to \beta$ (reflexivity, 自反律) — trivial
2) If $\alpha \to \beta$, then $\gamma\alpha \to \gamma\beta$ or $\gamma\alpha \to \beta$ (augmentation, 增补律)
3) If $\alpha \to \beta$, and $\beta \to \gamma$, then $\alpha \to \gamma$ (transitivity, 传递律)

These rules are

Sound (保真的, generate only functional dependencies that actually hold)
Complete (完备的, generate all functional dependencies that hold)

Armstrong's Axioms additional rules

1) If $\alpha \to \beta$ and $\alpha \to \gamma$ holds, then $\alpha \to \beta\gamma$ holds (union, 合并律)
2) If $\alpha \to \beta\gamma$ holds, then $\alpha \to \beta$ and $\alpha \to \gamma$ holds (decomposition, 分解律)
3) If $\alpha \to \beta$ and $\gamma\beta \to \delta$ holds, then $\alpha\gamma \to \delta$ holds (pseudo-transitivity, 伪传递律)

The above rules can be inferred from Armstrong's axioms.

3.3.1 Procedure for Computing F^+ To compute the closure of a set of functional dependencies F

Algorithm 1 Procedure for Computing F^+

```
F+ = F
repeat
    for each functional dependency f in F+ do
        Apply reflexivity and augmentation rules on f
        Add the resulting to F+
    end for
    for each pair of f1 and f2 in F+ do
        if they can be combined using transitivity then
            add the resulting to F+
        end if
    end for
until F+ doesn't change any further
```

Note: The maximum number of possible Functional Dependencies (FDs) is $2^n \times 2^n$, for n attributes.

3.4 Closure of Attribute Sets

Definition VI.5 *Given a set of attributes α, the closure of α under F denoted by α^+, is the set of attributes that are functionally determined by α under F (在 F 下由 a 所直接和间接函数决定的属性的集合称为 α^+).*

Algorithm 2 To get α^+

```
result := α
while changes to result do
    for each β → γ in F do
        if β ⊆ result then
            result := result ∪ γ
        end if
    end for
    α+ := result
end while
```

3.4.1 Uses of Attribute Set Closure There are 3 kind uses of the attribute set closure algorithm:

1) To test $\alpha \to \beta$ is in $F^+ \Leftrightarrow \beta \subseteq \alpha^+$
2) To test α is a superkey, $\alpha \to R$ is in $F^+ \Leftrightarrow R \subseteq \alpha^+$
3) To compute F^+

Algorithm 3 compute F^+

```
for each γ ⊆ R do
    find γ+
    for each S ⊆ γ+ do
        output γ → S
    end for
end for
all γ → S form F+
```

3.5 Canonical Cover (正则覆盖)

Definition VI.6 *Intuitively, a canonical cover of F, denoted by F_c, is a **minimal** set of FDs equivalent to F, i.e. no FD in F_c contains an extraneous attribute. And each left side is unique.*

$$F_c \overset{Logically\ Imply}{\Longleftrightarrow} F$$

To get F_c, we should delete extraneous attributes. There are 3 cases for the extraneous attributes:

1) FDs can be inferred from the others
2) redundant left side
3) redundant right side

3.5.1 Extraneous Attributes (无关属性) Consider the FD $\alpha \to \beta$ in F:

1) Attributes A is extraneous in α, if $A \in \alpha$ and F logically implies $F' = (F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$
2) Attribute A is extraneous in β, if $A \in \beta$ and FDs $F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ logically implies F.

3.5.2 Testing if an Attribute is Extraneous

1) To test $A \in \alpha$ is extraneous in $\alpha \Leftrightarrow \alpha = \{A\alpha'\}$, $\{A\alpha'\} \to \beta$, whether $\beta \in (\alpha')^+$ $[\Rightarrow \alpha \to \beta \subseteq F]$
2) To test $A \in \beta$ is extraneous in $\beta \Leftrightarrow \beta = \{A\beta'\}$, $\alpha \to \{A\beta'\}$, whether $A \in (\alpha)^+$ $[\Rightarrow \alpha \to A \subseteq F'$, $F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}]$

To compute a canonical cover for F:

Algorithm 4 compute a canonical cover

repeat
Replace any FD in F like $\alpha_1 \to \beta_1$ and $\alpha_1 \to \beta_2$ with $\alpha_1 \to \beta_1\beta_2$
Find a FD $\alpha \to \beta$ with an extraneous attribute either in α or in β
if an extraneous attribute is found **then** delete it
end if
until F doesn't change

4. Decomposition

4.1 Goals of Normalization

Judge whether a relation R is in a **good** form (no redundant, no insert/delete/update anomalies). If not, decompose it into a set of relations $\{R_1, R_2, \dots, R_n\}$ such that:

1) a lossless-join decomposition (无损链接分解)
2) dependency preservation (依赖保持)
3) Each relation is in BCNF or 3NF

4.2 Desirable properties of decomposition

1) All attributes of original schema R must appear in the decomposition $(R_1, R_2) : R = R_1 \cup R_2$
2) Lossless-join decomposition: For all possible relations r on schema R

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

A decomposition of R into R_1 and R_2 is lossless-join iff at least one of the following dependencies are held in F^+:

- $\{R_1 \cap R_2\} \to R_1$
- $\{R_1 \cap R_2\} \to R_2$

(分解后的二个子模式的共同属性必须是 R_1 或 R_2 的码 (适用于一分为二的分解))

3) Dependency preservation: Restriction of F to R_i is : $F_i \subseteq F^+$, F_i includes only attributes of R_i.

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

where F_i be the set of dependencies in F^+ that include only attributes in R_i.

4) No redundancy: R_i should be in BCNF or 3NF.

4.3 Testing for Dependency Preservation

Algorithm 5 check $\alpha \to \beta$ is preserved in a decomposition

$result = \alpha$
while changes to $result$ **do**
for R_i in the decomposition **do**
$t = (result \cap R_i)^+ \cap R_i$
end for
$result = result \cup t$
if $result$ contains all attributes in β **then**
$\alpha \to \beta$ is preserved
end if
end while

Apply the test on all dependencies in F to check if a decomposition is dependency preserving.

对于 F 中的某个 $\alpha \to \beta$, 投影到各个 R_i 中, 判别是否有某个 R_i 能保持函数依赖 $\alpha \to \beta$. 若对 F 中的每个 $\alpha \to \beta$ 都能有一个 R_i 满足函数依赖, 则该分解保持依赖.

5. Boyce-Codd Normal Form

Definition VI.7 *A relation schema R is in BCNF, with respect to a set F of FDs, if $\forall$ FDs in F^+ of the form $\alpha \to \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds: $\forall \alpha \to \beta$ in F^+,*

$$\begin{cases} & \alpha \to \beta \text{ is trivial (i.e. } \beta \subseteq \alpha) \\ or & \alpha \text{ is a superkey for } R (i.e.\ R \subseteq \alpha^+,\ \alpha \to R) \end{cases}$$

5.1 Testing for BCNF

To check if a non-trivial dependency $\alpha \to \beta$ causes a violation of BCNF

1) Compute α^+
2) Verify that α^+ includes all attributes of R (α os a superkey of 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. However, using only F to test for BCNF may be incorrect when testing a relation R_i in a decomposition of R.

可在 F 下判别 R 是否违反 BCNF, 但必须在 F^+ 下判别 R 的分解式是否违反 BCNF.

5.2 BCNF Decomposition Algorithm

Algorithm 6 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 $\alpha \to \beta$ be a non-trivial FD that holds on R_i such that $\alpha \to R_i$ is not in F^+, and $\alpha \cap \beta = \emptyset$
    $result := (result - R_i) \cup \underbrace{(\alpha, \beta)}_{R_{i1}} \cup \underbrace{(R_i - \beta)}_{R_{i2}}$ ▷ 将 R_i 分解为二个子模式: R_{i1}, R_{i2}, α 是它们的共同属性.
  else
    $done := true$
  end if
end while

Finally, every sub-schema is in BCNF, and the decomposition is lossless-join.

5.3 BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving. Therefore, we cannot always satisfy all three design goals:

1) Lossless-join
2) BCNF
3) Dependency preservation

6. Third Normal Form

A weaker normal form:

1) Allows some redundancy (with resultant problems).
2) But FDs can be checked on individual relations without computing a join — dependency preserving.
3) There is always a lossless-join, dependency preserving decomposition into 3NF.

6.1 Third Normal Form (3NF)

Definition VI.8 *A relation schema R is in third normal form (3NF) if $\forall\ \alpha \to \beta$ in F^+, at least one of the following conditions holds:*

- *$\alpha \to \beta$ is trivial (i.e., $\beta \in \alpha$)*
- *α is a superkey for R*
- *Each attribute A in $\beta - \alpha$ is contained in a candidate key for R (即 $A \in \beta - \alpha$ 是主属性, 若 $\alpha \cap \beta = \emptyset$, 则 $A = \beta$ 是主属性)*

Note: each attribute may be in a different candidate key.

If a relation is in BCNF, it is in 3NF. Third condition is a minimal relaxation of BCNF to ensure dependency preservation.

6.1.1 Redundancy of 3NF A schema that is in 3NF but not in BCNF has the problems of repetition of information, and may need to use null values.

6.2 Testing for 3NF

1) Need to check only FDs in F, need not check all FDs in F^+.
2) Use attribute closure to check for each dependency $\alpha \to \beta$, to see if α is a superkey.
3) If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of R. (This test is rather more expensive, since it involve finding all candidate keys)

6.3 Comparison of BCNF and 3NF

It is always possible to decompose a relation into relations in 3NF and

- The decomposition is lossless.
- The dependencies are preserved.

It is always possible to decompose a relation into relations in BCNF and

- The decomposition is lossless.
- But it may not be possible to preserve dependencies.

6.4 3NF Decomposition Algorithm

Algorithm 7 3NF Decomposition Algorithm

Let F_c be a canonical cover for F
$i := 0$
for each FD $\alpha \to \beta$ in F_c **do**
  if none of the schemas R_j, $1 \le j \le i$ contains $\alpha\beta$ **then**
    $i := i + 1$
    $R_i = (\alpha\beta)$ ▷ 将 F_c 中的每个 $\alpha \to \beta$ 分解为子模式 $R_i := (\alpha, \beta)$, 从而保证 dependency-preserving.
  end if
end for
if none of the schemas R_j, $1 \le j \le i$ contains a candidate key for R **then**
  $i := i + 1$
  $R_i :=$ any candidate key for R ▷ 保证至少在一个 R_i 中存在 R 的候选码, 从而保证 lossless-join.
end if
return $(R_1, R_2, \ldots, R_i)$

6.5 Design Goals

Goal for a relational database design is:

- BCNF.
- Lossless join.
- Dependency preservation.

If we cannot achieve this, we accept one of

- Lack of dependency preservation.
- Redundancy due to use of 3NF.

7. Multivalued Dependencies

7.1 Multivalued Dependencies (MVDs)

Definition VI.9 *Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$, the multivalued dependency*

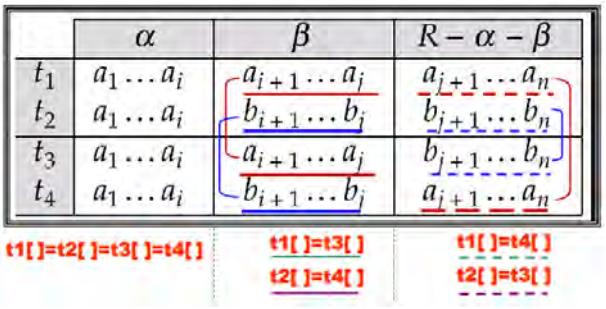
$$\alpha \twoheadrightarrow \beta$$

holds on R, if in any legal relation $r(R)$, $\forall$ 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:

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

If $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$, then $\alpha \twoheadrightarrow \beta$ is trivial.

Table 1: Tabular representation of $\alpha \twoheadrightarrow \beta$



7.2 Theory of MVDs

If $\alpha \to \beta$, then $\alpha \twoheadrightarrow \beta$ (If $R - \alpha - \beta = \emptyset$, i.e., $\alpha \cup \beta = R$). That is, every FD is also a multivalued dependency.

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

- We can compute D^+ from D, using the formal definitions of FDs and multivalued dependencies.
- We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice.
- For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules.

8. Fourth Normal Form

Definition VI.10 *A relation schema R is in 4NF with respect to a set D of functional and multivalued dependencies if $\forall$ multivalued dependencies in D^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$,at least one of the following hold:*

- *$\alpha \twoheadrightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)*
- *α is a superkey for schema R.*

If a relation is in 4NF, it is in BCNF.

8.1 Requirement for decomposition — Restriction of Multivalued Dependencies

Assume R is decomposed into $R_1, R_2, \ldots, R_n$, each R_i is required to conform to 4NF. The restriction of D to R_i is the set D_i consisting of

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

8.2 4NF Decomposition Algorithm

Algorithm 8 4NF Decomposition Algorithm

result $:= \{R\}$
done $= false$
compute D^+
Let D_i denote the restriction of D^+ to R_i.
while not *done* **do**
  if there is a schema R_i in *result* that is not in 4NF **then**
    Let $\alpha \rightarrow\rightarrow \beta$ be a non-trivial multivalued dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \emptyset$.
    $result := (result - R_i) \cup \underbrace{(\alpha, \beta)}_{R_{i1}} \cup \underbrace{(R_i - \beta)}_{R_{i2}}$
  end if
end while

Note: each R_i is in 4NF, and decomposition is lossless-join.

8.3 Other Design Issues

Some aspects of database design are not caught by normalization.