ISE719

IDEF1X Modeling & Syntax

Modeling Databases

Robert E. Young

Department of Industrial & Systems Engineering

North Carolina State University

Raleigh, NC 27695

February 2010

Table of Contents

BASIC STRUCTURE OF A RELATIONAL DATABASE

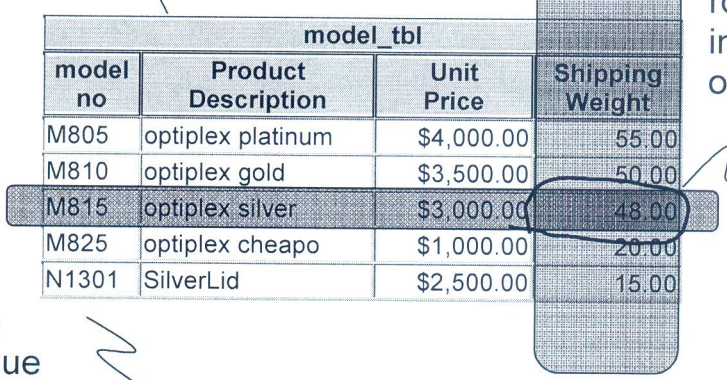
- In a *relational* database the tables contain data organized into rows and columns.
- Columns group data with the same meaning.
- Columns are given a name that provides a context within which to interpret the column values.
- Rows group data with different column values that have meaning when viewed together.
- Each row is unique in that no two rows are exactly the same.
- There is a combination of columns whose values uniquely identify each row.

STRUCTURE OF A DATABASE TABLE

Table
A relational database has at least one table. The table name provides a label for the table's contents.

Columns
Each database table has at least one column. The column name provides a context within which to interpret the column values.

Column Values
Data values in a column. Each row/column intersection has only one value.



model_tbl			
model no	Product Description	Unit Price	Shipping Weight
M805	optiplex platinum	$4,000.00	55.00
M810	optiplex gold	$3,500.00	50.00
M815	optiplex silver	$3,000.00	48.00
M825	optiplex cheapo	$1,000.00	20.00
N1301	SilverLid	$2,500.00	15.00

Rows in a table
Each row is unique in that no two rows are the same. Sometimes also called a *record*.

Primary Key
The combination of columns that uniquely identifies a row. Must be at least one column and can be all the columns.

RELATIONSHIP BETWEEN TABLES

model_tbl			
model no	Product Description	Unit Price	Shipping Weight
M805	optiplex platinum	$4,000.00	55.00
M810	optiplex gold	$3,500.00	50.00
M815	optiplex silver	$3,000.00	48.00
M825	optiplex cheapo	$1,000.00	20.00
N1301	SilverLid	$2,500.00	15.00

model no appears in both tables. Consequently, the features_tbl is related to the model_tbl through the shared column values of *model no*

features_tbl						
model no	processor	hdsize	cdtype	memsize	USB	network
M805	two P4-1800	120	CD-RW/DVD	1024	4	Yes
M810	P4-1500	80	CD-RW/DVD	512	2	Yes
M815	P4-1200	60	CD-RW	256	2	no
M825	Celeron-1000	30	CD-R	128	1	no
N1301	Dotheon	80	CD-RW/DVD	512	2	Yes

Tables are related to one another through *shared column values*. One or more columns and their values may be shared. In this example, only one column is shared.

IDEF1X Modeling Objectives

The IDEF1X database modeling objectives are to:

- Represent the columns in the database table.
- Specify the columns that uniquely identify a row in each database table.
- Specify the relationships between tables.

IDEF1X model of the model_tbl, the features_tbl, and the relationship between them.

We will find out shortly how to interpret this model.

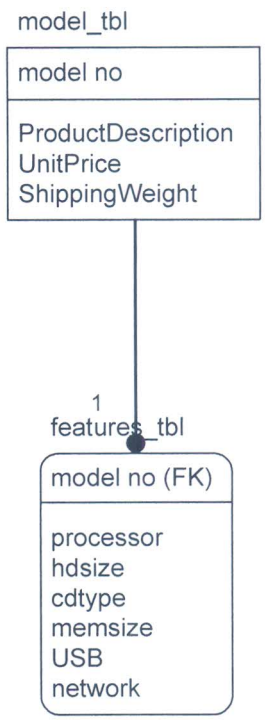


TABLE DEFINITIONS

Table

A table is a set of real or abstract things that have common characteristics. These characteristics are organized into columns with each characteristic represented by a separate column. Specific data values define a unique set of characteristics and each set is a row in the table.

Table name

The table name is an identifier for the table. It provides the context with which to interpret the contents of the table. The table is referenced using its table name. Consequently, the table name must be unique and descriptive of the table's contents

Column

A column defines a characteristic for a table. In this way, columns are a set of characteristics that describe a table. Each column is common to all rows in a table so we can say that every row in a table has the same columns.

Column name

The column name is an identifier for the column. It provides the context with which to interpret the column values. The column is referenced using its column name. Consequently, the column name must be unique and descriptive of the column's contents.

Column Value

A member of a column in a row of a relational database table (i.e., a "cell" in the relational table) is a column value. Typically, these are called the *field values* of an explicit *record*.

TABLE DEFINITIONS -- CONTINUED

Row

A row in a table is an explicit set of column values and is sometimes called a record. Where a column identifies a characteristic, a row identifies a unique set of values that have meaning when interpreted in the context of the column names. A row is identified by the column values of a subset of the table's columns. Since each row is unique and therefore different, No two rows in a database table are the same. Within each table the subset of columns whose column values uniquely identify each row are called the "primary key".

Primary Key columns

The primary key is the collection of one or more columns that uniquely identify a specific row in a table. This collection must be at least one column and can be all the columns in a table.

Non-key Columns

The non-key columns are the collection of columns that are not part of the primary key. This collection can be none of the columns if all the columns are part of the primary key.

RELATIONSHIP DEFINITIONS

Relationship

A relationship is the manner in which members of one table are associated with (related to) members of another table. If a row from one relational table relates to one or more rows in another table the two tables have a relationship. A relationship exists ONLY between two tables.

Relationships are *directed* in that in one table is independent and the other table is dependent. This means that the dependent table receives column values from the independent table.

Cardinality

The cardinality is the specification of the relationship. It explicitly defines how many rows in the dependent table may exist for each row in the independent table. The cardinality is identified in the model by the type of arc connecting the two tables and by the annotation on the arc.

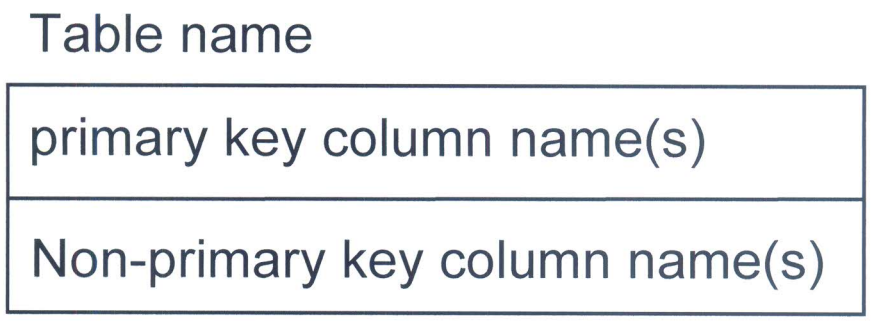
IDEF1X SYNTAX AND SEMANTICS

AN IDEF1X MODEL GRAPHICALLY SHOWS

- Tables and relationships
- Table names
- Column names
- Primary key columns
- Non-key columns
- Explicit type of relationships and cardinalities

BASIC TABLE REPRESENTATION

Syntax



Example

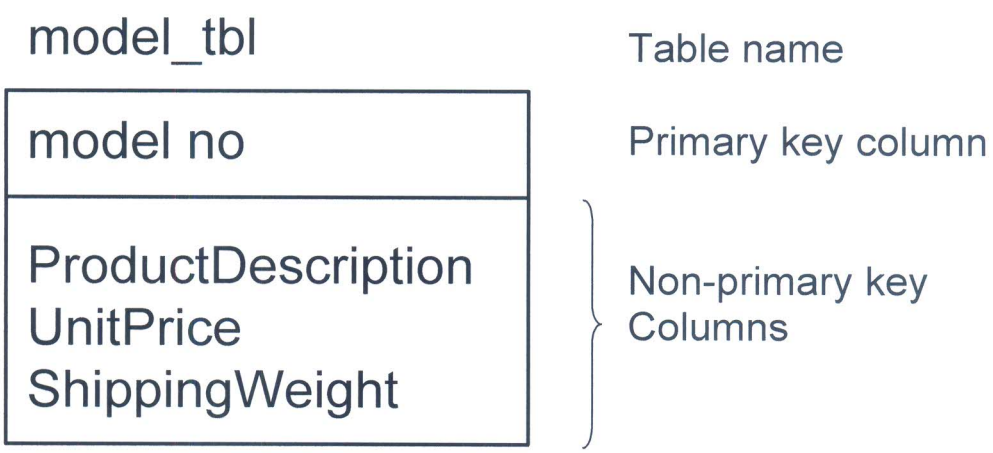
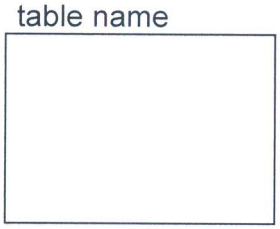


Table Syntax

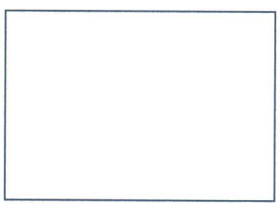
- Independent tables are indicated with a square box.
- Dependent tables are indicated with a rounded-corner box

Identifier *Independent* Table

table name

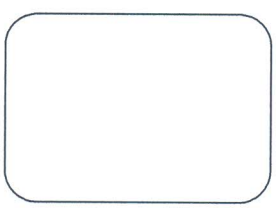


employee

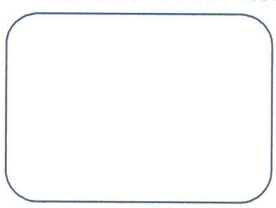


Identifier *Dependent* Table

table name



Purchase Order item



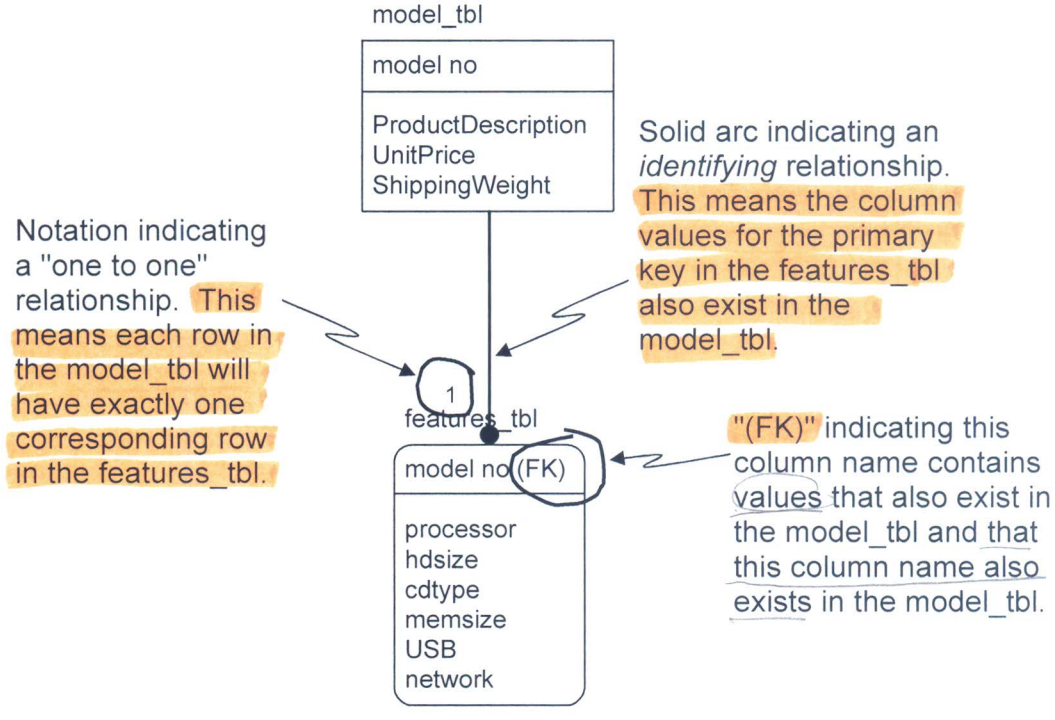
RELATIONSHIPS

Relationship

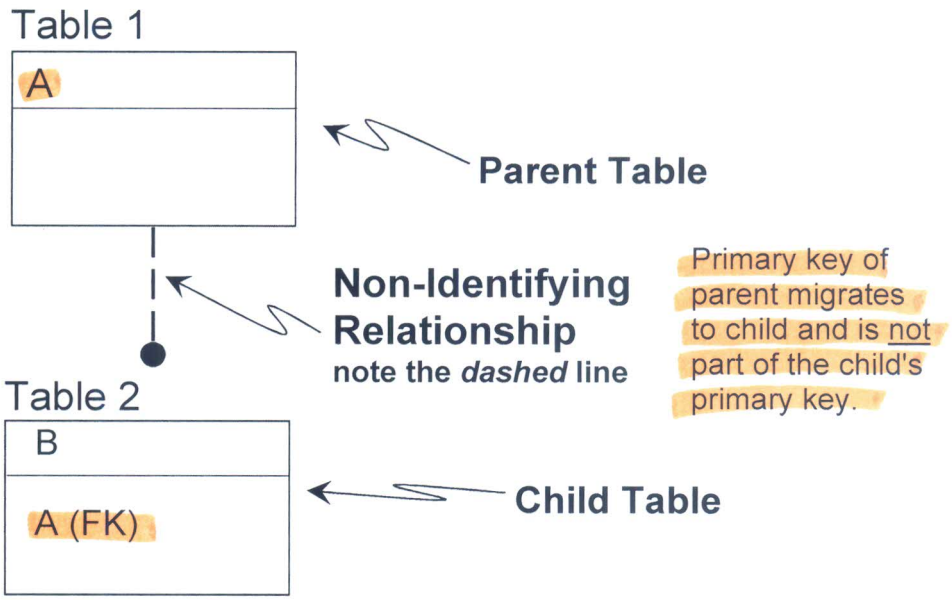
The manner in which members of one table are associated with members of another table. Represented by either a solid arc (*identifying* relationship) or a dashed arc (*non-identifying* relationship).

Cardinality

The explicit specification of the relationship between two tables. Defined by the type of arc (solid or dashed) and notation.



IDENTIFIER *INDEPENDENT* RELATIONSHIP SYNTAX

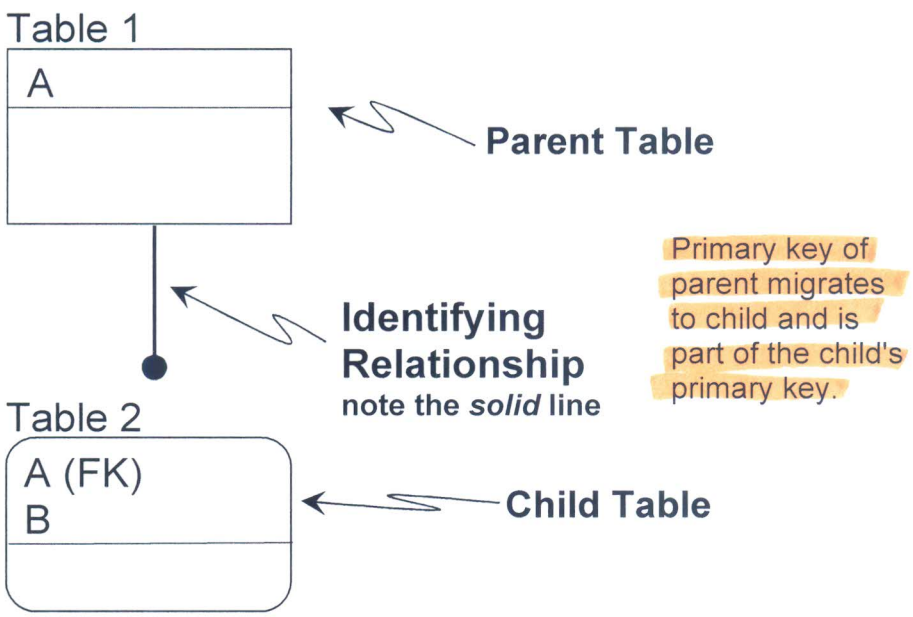


Comments

The child table in a "non-identifying relationship" is always an *identifier independent* table unless the table is also a child table in some other identifying relationship.

The parent table in a "non-identifying relationship" may be either an identifier independent table (as above) or an identifier dependent table based upon relationships with other tables.

IDENTIFIER *DEPENDENT* RELATIONSHIP SYNTAX



Comments

The child table in an "identifying relationship" is always an *identifier dependent* table.

The parent table in an "identifying relationship" may be either an identifier independent table (as above) or an identifier dependent table, based upon relationships with other tables.

CARDINALITY

TYPES OF CARDINALITY

Tables can have the following types of explicit relationships between them:

- **One relates to zero, one or more** ≥ 0 # Values

  One value in a table relates to zero, one or more rows in another table.

- **One relates to at least one or more** > 0 # Values

  One value in a table relates to one or more rows in another table.
  This is an existence condition.

- **One relates to zero, one or up to *n* (fixed upper bound)** $\geq 0, \leq n$

  One value in a table relates to zero, one or up to *n* rows in another table.

- **One relates to one or up to *n* (fixed upper bound)** $> 0, \leq n$

  One value in a table relates to one or up to *n* rows in another table. This is an existence condition.

CARDINALITY
(CONTINUED)

- **One relates to zero or one**

  One value in a table relates to zero or one row in another table.

- **One relates to only one**

  One value in a table relates to exactly one row in another table. This is an existence condition.

- **Zero** or one to …

  In certain situations it may be possible for the independent table to have a zero value when the dependent table has a non-zero row entry. This is an example of the child being *existence-dependent* on a parent, but not *identification-dependent* on the parent (Thomas A. Bruce, *Designing Quality Databases with IDEF1X Information Models,* Dorset House Publishing, New York, NY, 1992, pg. 96-97).

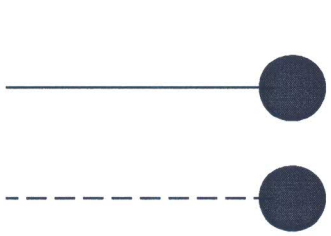
  In this case, the inherited primary key of the dependent table must be a non-primary key column or non-primary columns in the dependent table, making the relationship a *non-identifying* relationship. This situation is indicated by a diamond "◇" at the head of the *dashed-line* relationship arc.

CARDINALITY NOTATION
-- ONE TO MANY --

identifying as inherited attribute is part of the primary key.

non-identifying as inherited attribute is NOT part of primary key

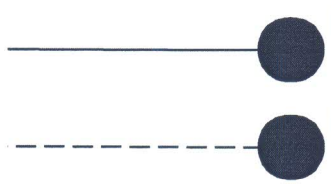
One *to* zero, one or many -- 1: (0,1,m)



identifying as inherited attribute is part of the primary key.

non-identifying as inherited attribute is NOT part of primary key

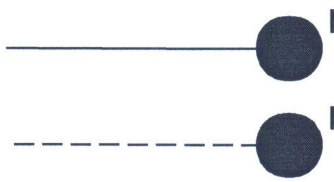
One *to* one or many -- 1: (1,m)



identifying as inherited attribute is part of the primary key.

non-identifying as inherited attribute is NOT part of primary key

One *to* at least n -- 1:n

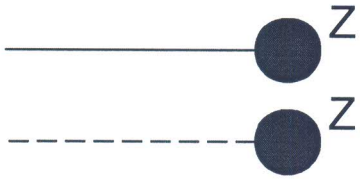


identifying as inherited attribute is part of the primary key.

non-identifying as inherited attribute is NOT part of primary key

One *to* at least m but no more than n -- 1:(m,n)

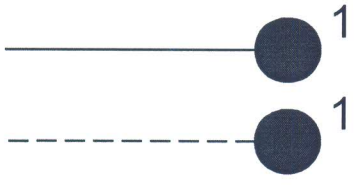
CARDINALITY NOTATION
-- ONE TO ONE --



identifying as inherited attribute is part of the primary key.

non-identifying as inherited attribute is NOT part of primary key

One *to* zero or one -- 1: (0, 1)



identifying as inherited attribute is part of the primary key.

non-identifying as inherited attribute is NOT part of primary key

One *to* one -- 1:1

CARDINALITY NOTATION
ZERO OR ONE TO ...

In certain situations it may be possible for the independent table to have a zero value when the dependent table has a non-zero value. This is an example of the child being *existence-dependent* on a parent, but not *identification-dependent* on the parent (Thomas A. Bruce, *Designing Quality Databases with IDEF1X Information Models,* Dorset House Publishing, New York, NY, 1992, pg. 96-97).

In this case, the inherited primary key of the dependent table must be non-primary key columns making the relationship a *non-identifying* relationship. This situation is indicated by a diamond "◇" at the head of the relationship arc.

independence can have a zero value.

Zero or one *to* zero, one or many -- 1: (0,1,m)

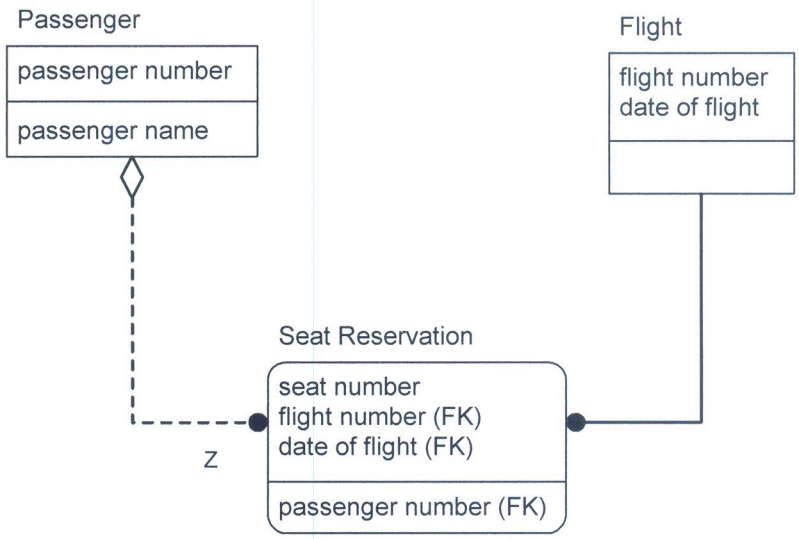
Zero or one *to* zero or one -- 1: (0, 1)

Zero or one *to* one or many -- 1: (1,m)

Zero or one *to* exactly n -- 1:n

Zero or one *to* at least m but at most n -- 1:(m,n)

CARDINALITY EXAMPLE



It is possible to have a seat reservation without having a passenger assigned to it. This makes sense because it is possible to have a flight with empty seats set aside. For example, seats in an isle with an emergency exit cannot be reserved by a passenger. They are assigned when you check-in. This is to assure that people sitting in these isles can remove the emergency doors.

UNDERSTANDING THE IDEF1X MODEL

TO UNDERSTAND AN IDEF1X MODEL WE MUST UNDERSTAND THE FOLLOWING:

- Column synonyms and aliases
- Single, compound and alternate primary keys
- Primary key migration
- Owned, inherited, and shared columns
- Foreign keys
- Role name for a foreign key
- Categorization Relationships
- Rules for Categorization Relationships
- Relationship restrictions
- Basic rules which must be observed
- Dual path relationships

COLUMN SYNONYMS AND ALIASES

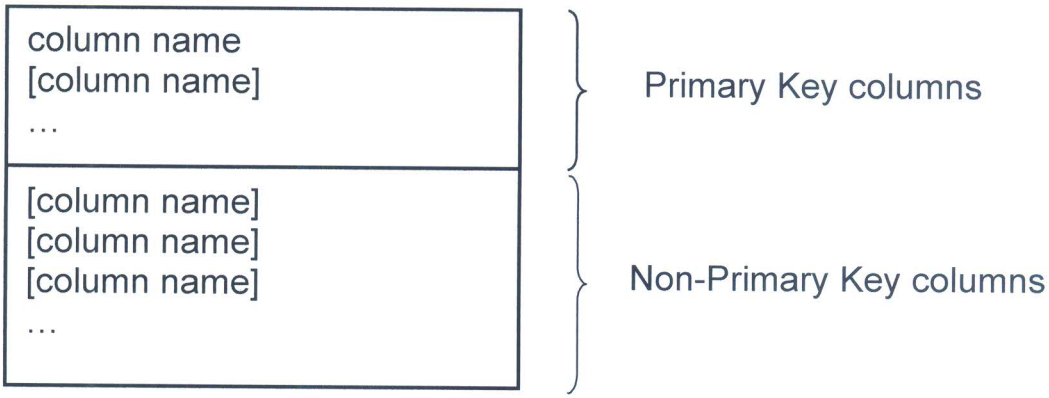
Other names by which the column is known.
Examples:

- student ID number and social security number
- part no. and item no.
- crescent wrench, adjustable wrench and spanner
- car and vehicle

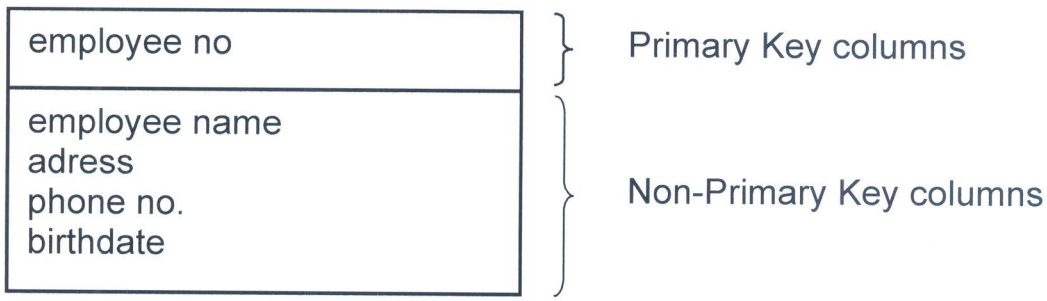
A synonym or an alias allows you to have more than one column name for a column. This means you can write queries using any of the names associated with the column.

PRIMARY KEY AND COLUMN SYNTAX

table name



employee



TYPES OF PRIMARY KEYS

Primary keys are shown in the top of the table box. They may be:

- Simple or Single
- Compound
- Alternate

TYPES OF PRIMARY KEYS

PRIMARY KEY CHARACTERISTICS

- May be single or compound.
- Its composition and structure is independent of the primary key. (e.g., if the primary key is a compound key the alternate primary key may be single or compound, the reverse is also true.)

ALTERNATE PRIMARY KEYS NEVER MIGRATE.

SIMPLE PRIMARY KEY

A single column whose value identifies a unique row in a table.

Note: In a relational table, a primary key that consists of only one column is a single column that identifies a unique row in a table.

COMPOUND PRIMARY KEY

Two or more columns whose values together uniquely identify a unique row in a table.

ALTERNATE (EQUIVALENT) PRIMARY KEY

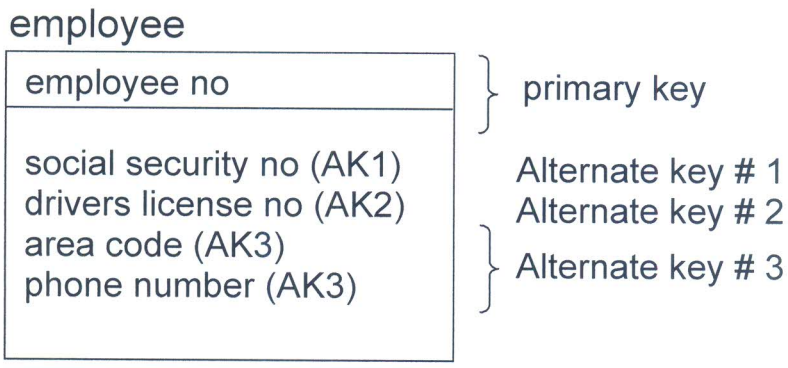
Another group of columns that can be used instead of the primary key to uniquely identify a row in a table.

ALTERNATE (EQUIVALENT) PRIMARY KEY SYNTAX

ALTERNATE KEY SYNTAX

column name (AK1)
[column name (AK2)]
. . .
[column name (AKn)]

Where the number uniquely identifies each alternate key. If multiple columns have the same number then they form a *compound alternate key*.

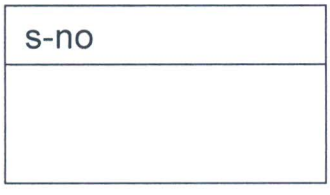


In the example,

- *employee no* is the primary key
- *social security no* is alternate primary key #1
- *drivers license no* is alternate primary key #2
- *area code* and *phone number* form a compound alternate primary key are are alternate primary key #3

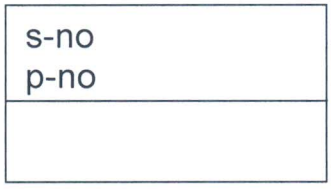
EXAMPLES OF DIFFERENT PRIMARY KEY FORMS

suppliers



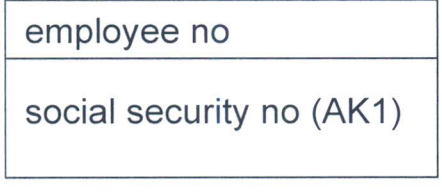
simple primary key

shipments



compound primary key

employee



simple primary key

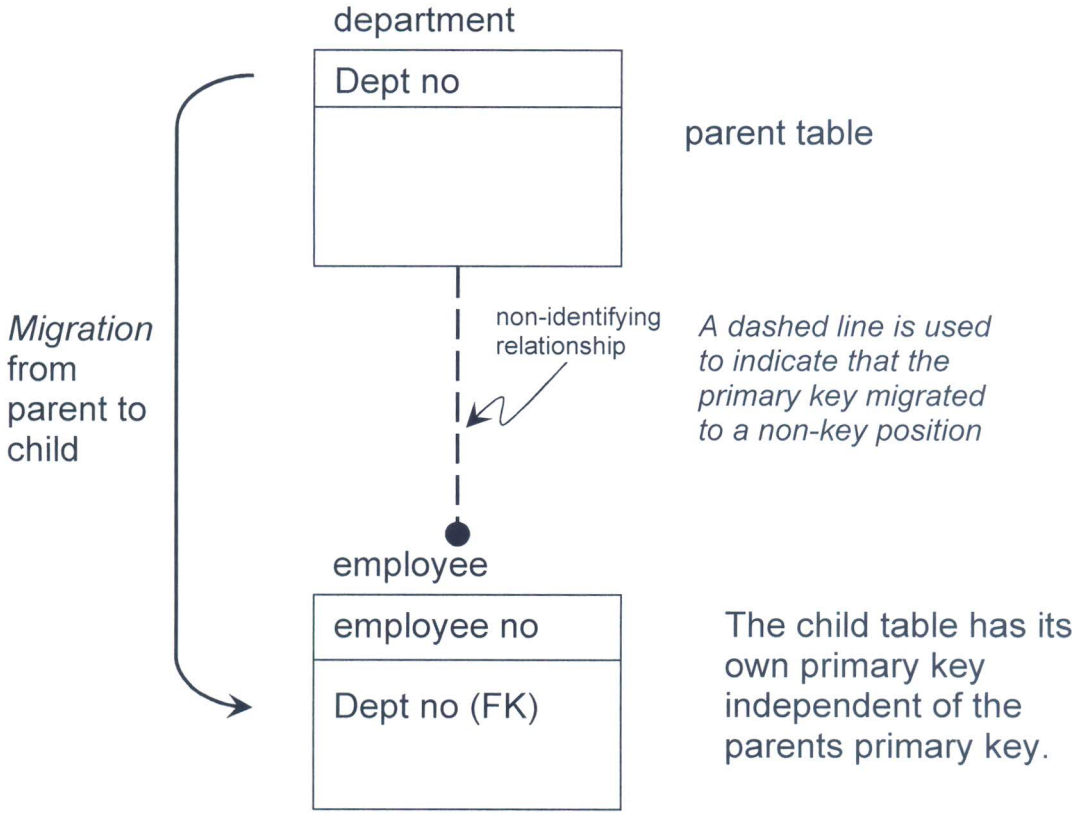
alternate (equivalent) primary key

PRIMARY KEY MIGRATION

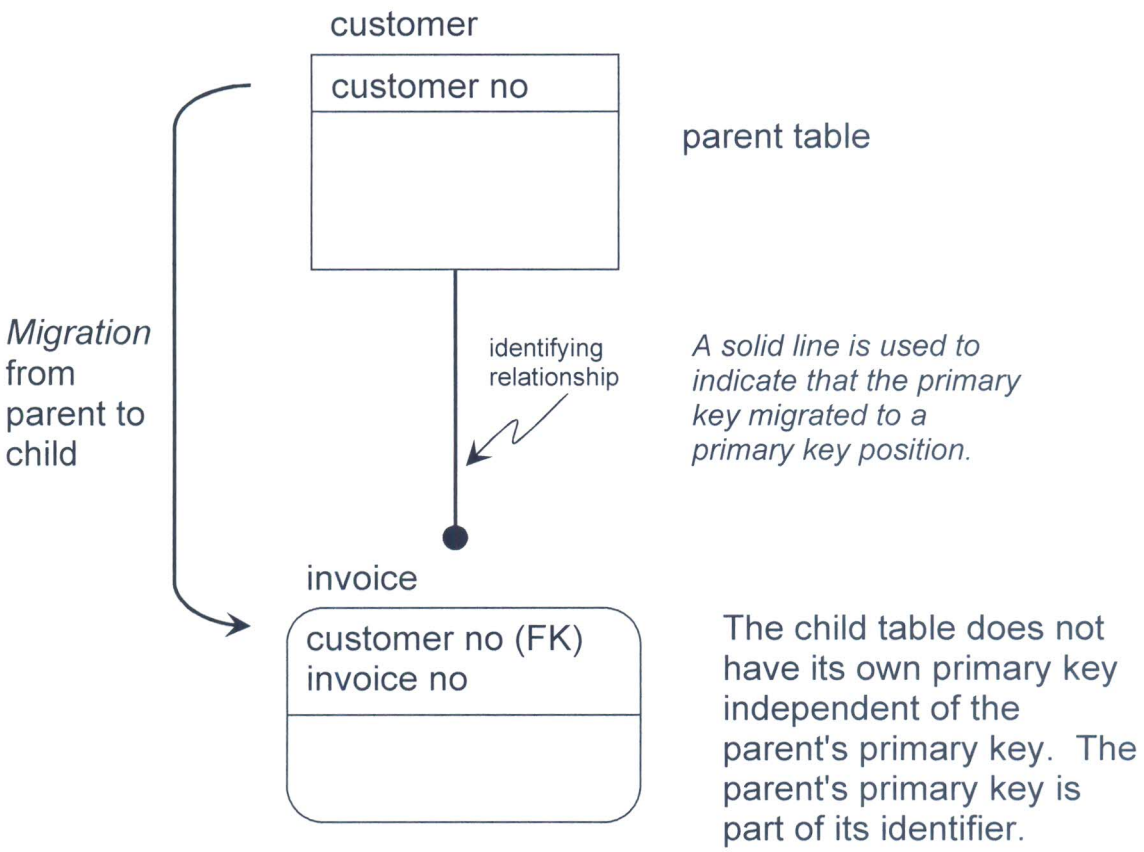
Through primary key migration columns become "shared" or "inherited" among tables.

In relational databases, two tables are related to one another through columns that appear in both tables and **share** values. Thus, a value in the column in the dependent table also exists as a value in the column in the independent table. These shared columns are shared or inherited columns among tables.

PRIMARY KEY MIGRATION TO AN IDENTIFIER INDEPENDENT TABLE



PRIMARY KEY MIGRATION TO AN IDENTIFIER DEPENDENT ENTITY



RULES FOR PRIMARY KEY MIGRATION

1. Migration always occurs from the independent to the dependent table in the related pair.

2. The entire primary key *must migrate* once for each relationship shared by the table pair.

   By the "entire primary key" we mean all columns that are members of the primary key.

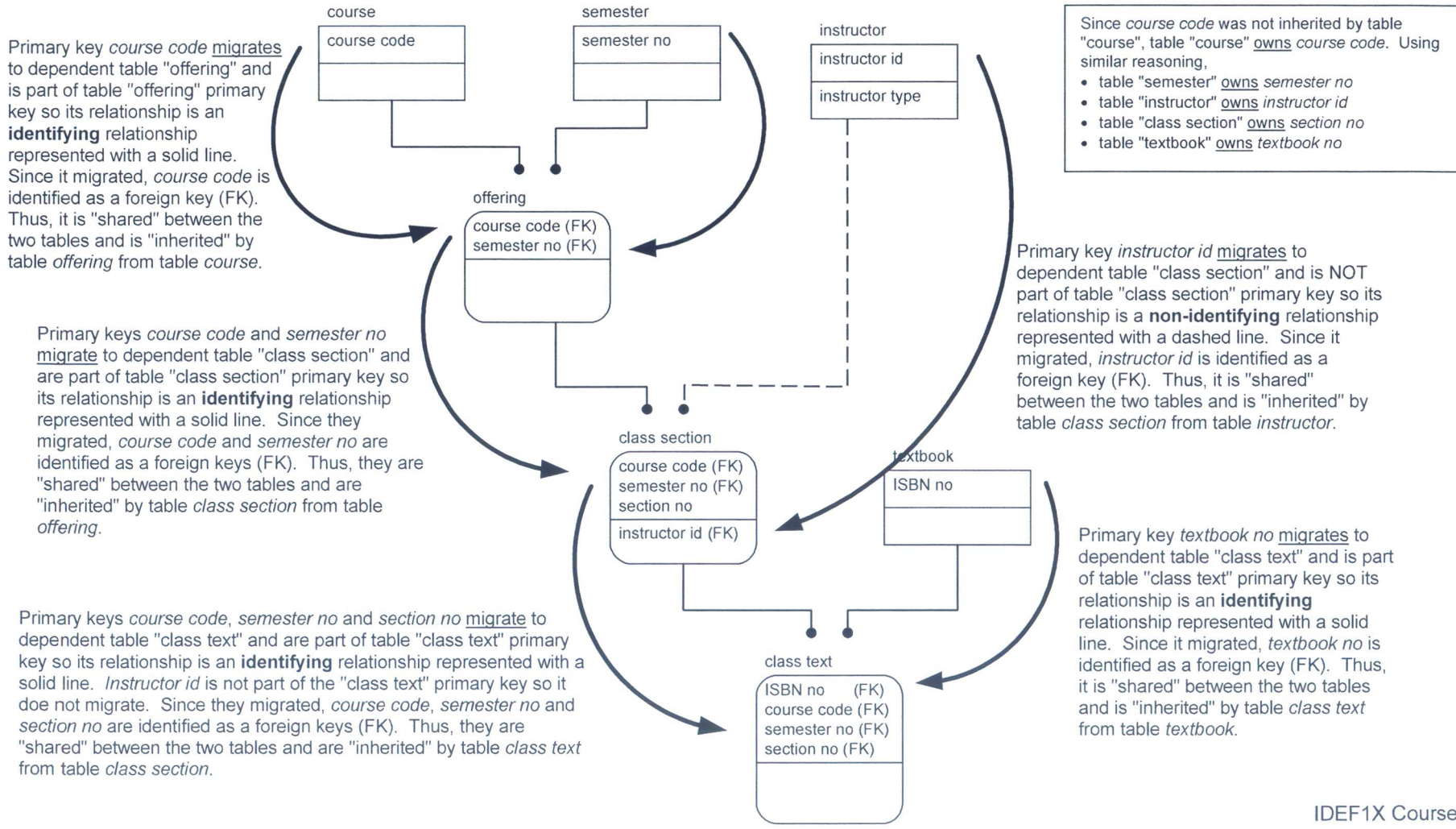
3. Alternate primary keys never migrate.

4. Non-key columns never migrate.

Primary key Migration Example

WHAT CONSTITUTES THE PRIMARY KEY IS DETERMINED INDEPENDENTLY FOR EACH TABLE BASED UPON WHAT IS NEEDED TO UNIQUELY IDENTIFY A SINGLE ROW IN THE TABLE. WHAT CONSTITUTES THE PRIMARY KEY IS NOT RELATED TO MIGRATION OR A COLUMN BEING A FOREIGN KEY.

For each dependent table, if one or more relationships are *identifying* then the dependent table has rounded corners.



NOTES ON PRIMARY KEY MIGRATION

- All inherited primary keys are also foreign keys in the table to which they have migrated.
- Inherited or Shared primary keys may be non-key columns in the tables to which they have migrated.
- Inherited primary keys may also be the primary key or form part of the primary key of the table to which they have migrated.
- If as an inherited column they form all or part of a primary key, then they must migrate to still another table.

WHY A ROUNDED BOX?

or

Which one?

If <u>any</u> foreign key is part of the primary key (i.e., above the line) then the box shape is **rounded**. Thus, we have:

versus

Owned, Inherited and Shared Columns

Owned Column

Although a column can migrate to many tables, there is only one table from which it did not migrate. This table is the starting point for the migration of the column and thus "owns" it.

Inherited Column

When a column has migrated from one table to another, it has been "inherited" by one table from another. Within the table that inherited it, the column is an "inherited column."

Shared Column

A column that belongs to more than one table.
Caution: It is possible for columns in different tables to have the same name but have no relationship to one another.

Note: Inherited columns are also shared columns.

NOTES ON OWNED COLUMNS

1. A column can be owned by only one table.
2. The "owner" table is the origin of all of the migrations of the shared column.
3. Owned columns can also be primary keys that are not shared with another table.

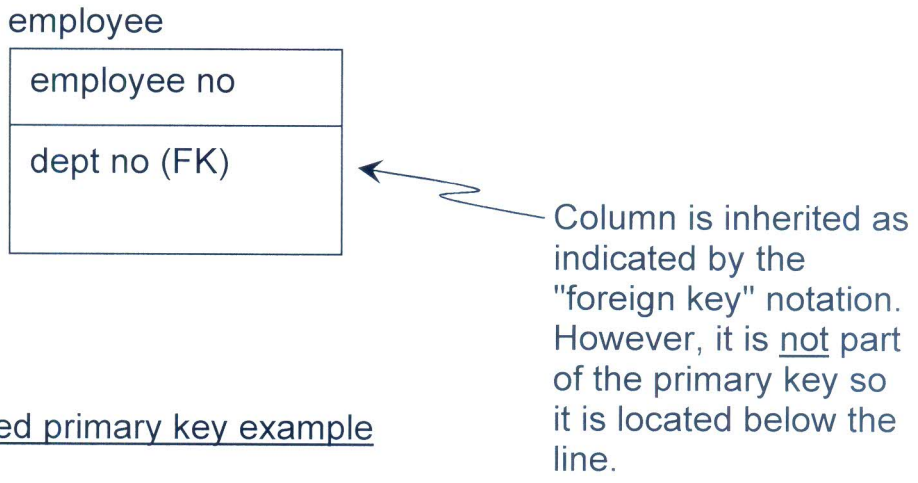
FOREIGN KEY

All inherited primary keys are also *foreign keys* in the table to which they have migrated. Thus, a foreign key can be a non-key column, a primary key, or be formed from columns in the primary key as well as non-key columns.

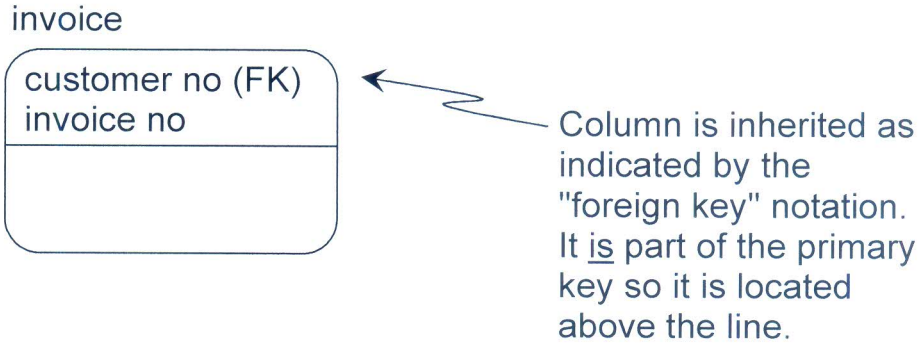
Foreign keys are identified by "(FK)" next to them in the table box.

EXAMPLES OF FOREIGN KEY SYNTAX

Inherited non-primary key example



Inherited primary key example



NOTES ON FOREIGN KEYS

The same column can generate more than one foreign key in the same child table when the column migrates through two or more relationships.

- When each child instance must have the same value for that column in the foreign keys then the column appears only once in the table and is identified as a foreign key.

- When each child instance can have different values for that column in the foreign keys then the column appears more than once in the table and each occurrence must be distinguished through the use of *role names*.

ROLE NAME FOR A FOREIGN KEY

A role name is used to distinguish duplicates of the same column that has migrated to a table through different paths and can have different values for that column. The role name distinguishes among them by identifying the context in which to interpret the column. The role name precedes the foreign key name and is separated from it by a period.

COLUMN ROLE NAMES

The *role name* distinguishes between two columns that have migrated from the same table by identifying the context in which to interpret them.

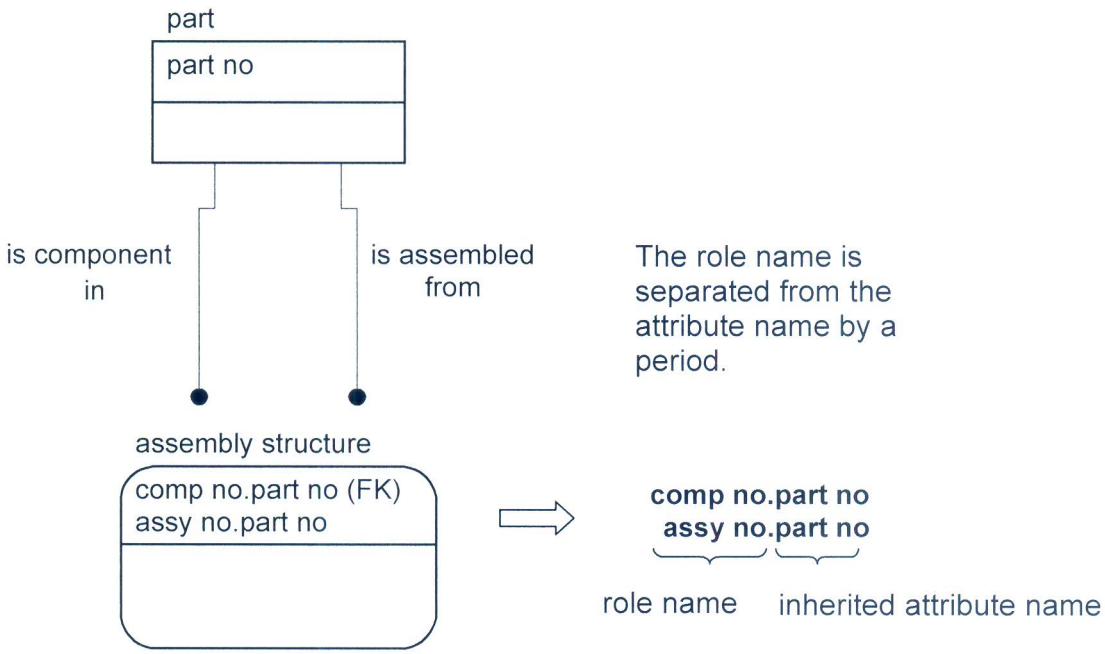


Fig_89.vsd

Each of the migrated "part no" primary keys in table *product structure* is given an additional role name specifying its function within the child table.

COLUMN ROLE NAME SYNTAX

The role name is separated from the column name by a period.

role name . column name (FK)

Example

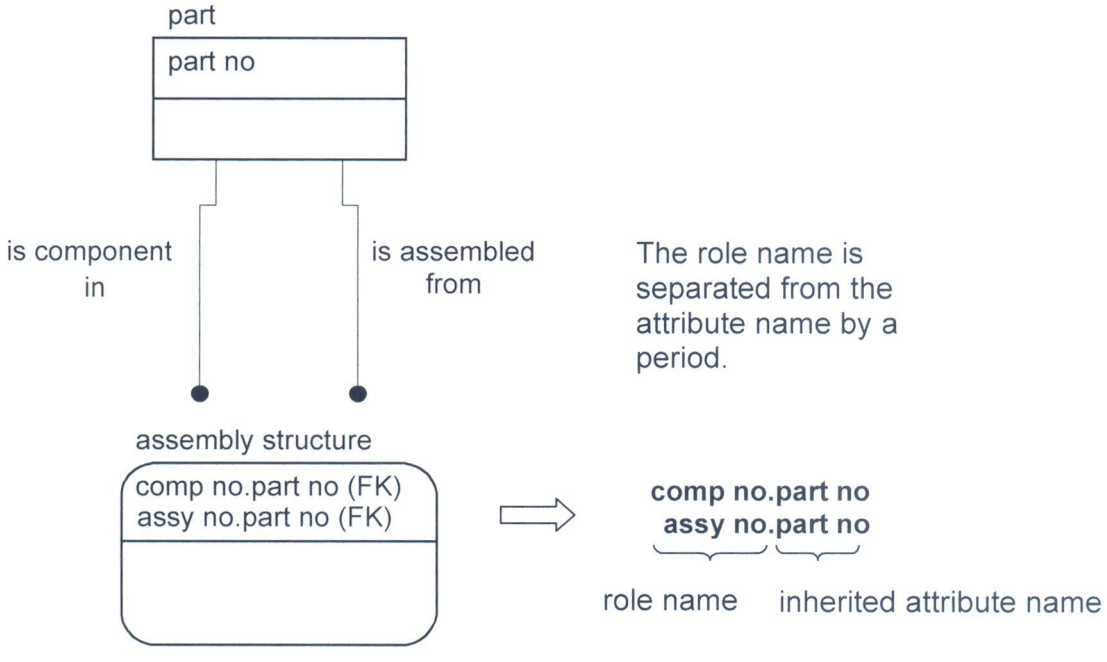


Fig_89.vsd

In the example, *comp no* is the role name for the first instance of part no being inherited while *assy no* is the role name for the second instance of part no been inherited.

CATEGORIZATION RELATIONSHIPS

Categorization relationships are used to represent mutually exclusive relationships between a parent table and a set of child tables. In a categorization relationship, the parent table is known as the *generic table* and each child in the set to which it is related is known as a *category table*.

Categorization Relationship Definitions

Incomplete Categorization Relationship

In an incomplete categorization relationship it is possible that an instance of the generic table is not associated with any of the category tables.

Complete Categorization Relationship

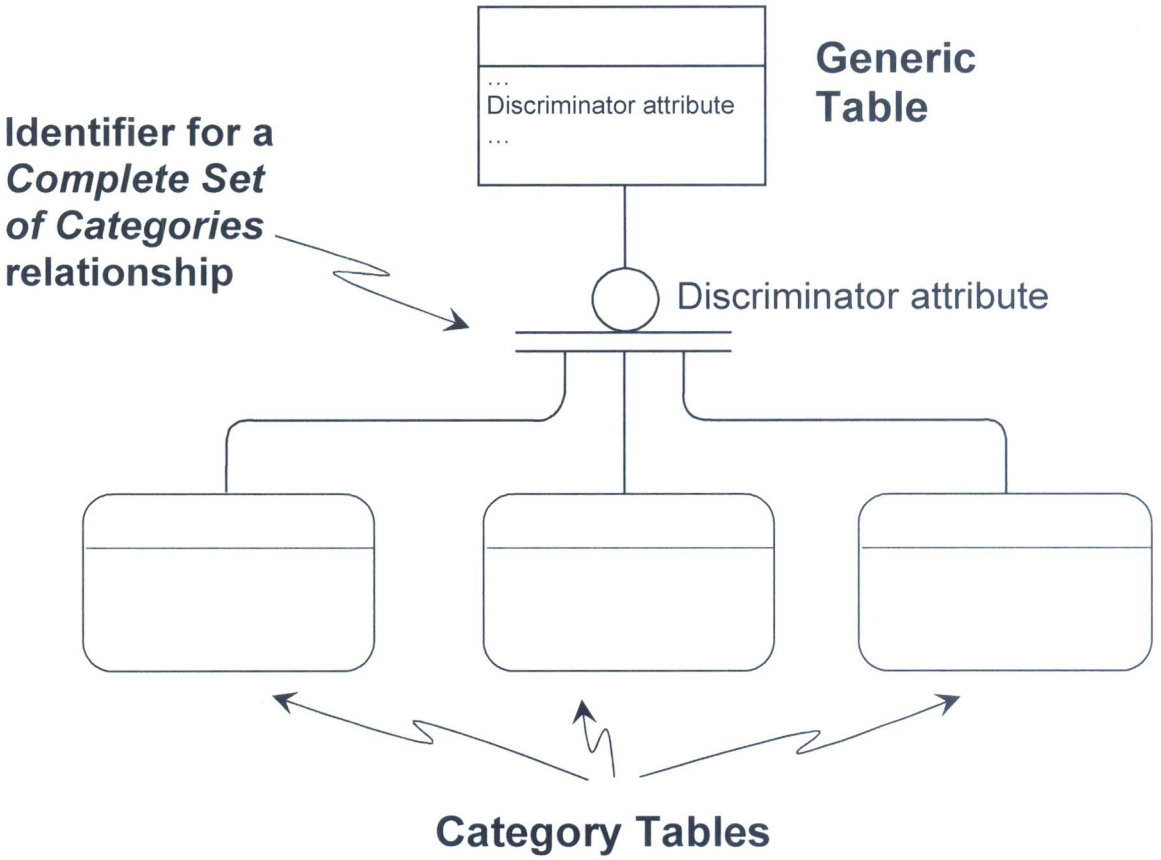
In a complete categorization relationship each row in the generic table must be associated with only one row of one of the category tables. This is an existence condition for the relationship.

Discriminator Column Name

The name of the column in the generic table whose value in a generic table row determines to which of the possible category tables the generic table instance is related.

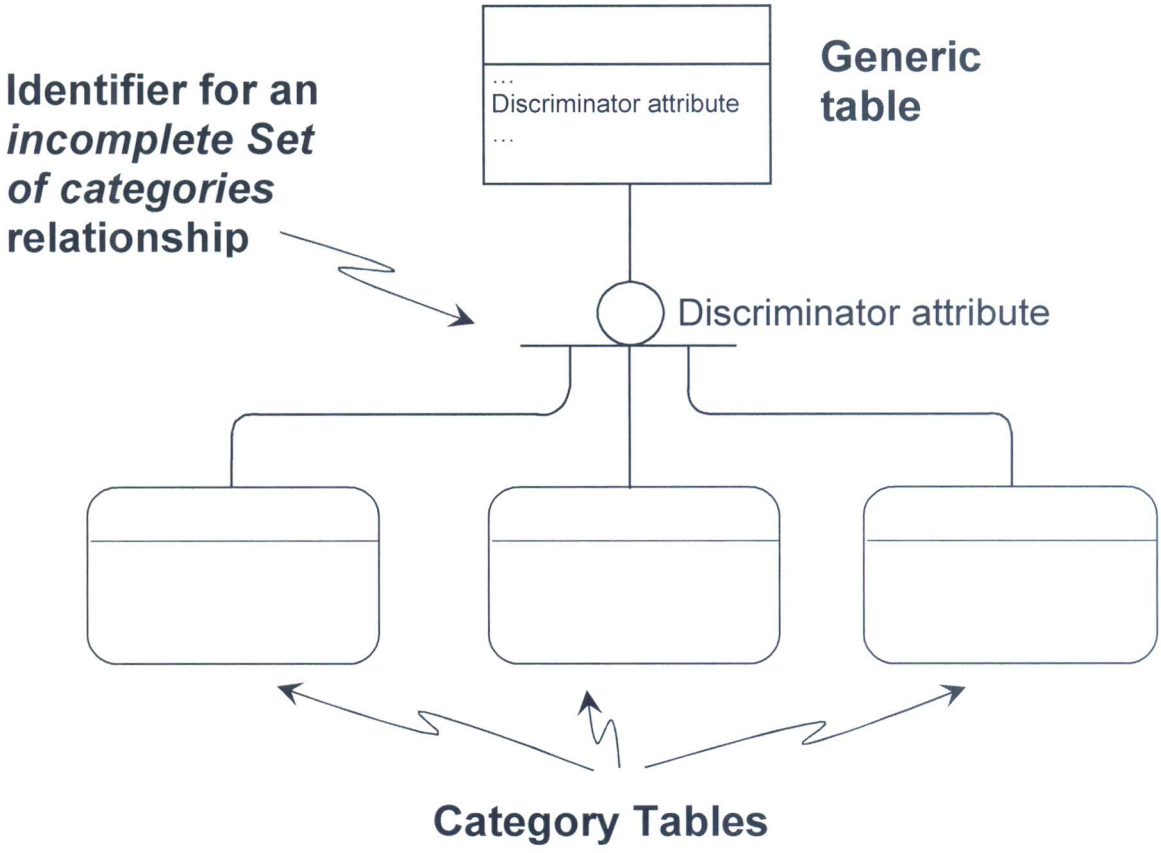
COMPLETE CATEGORIZATION RELATIONSHIP SYNTAX

In a *complete categorization relationship* each instance of the generic table must be associated with only one instance of one of the category tables. This is an existence condition for the relationship.

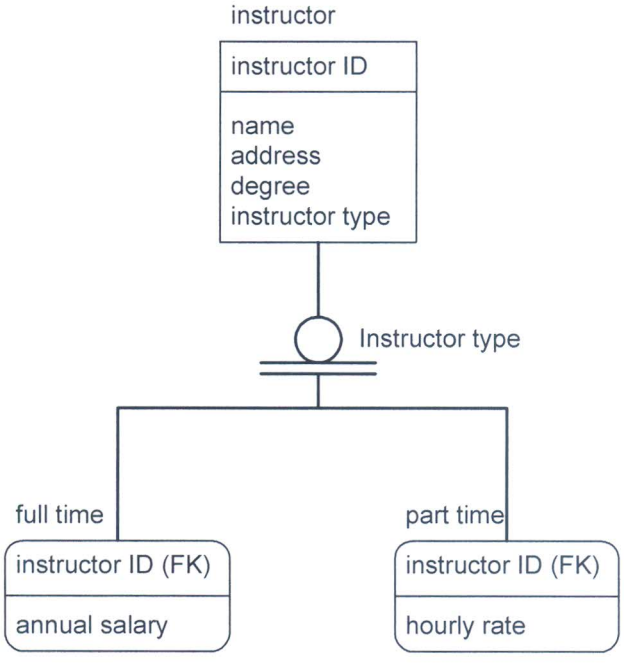


INCOMPLETE CATEGORIZATION RELATIONSHIP SYNTAX

In an *incomplete categorization relationship*, it is possible that an instance of the generic table is not associated with any of the category tables.

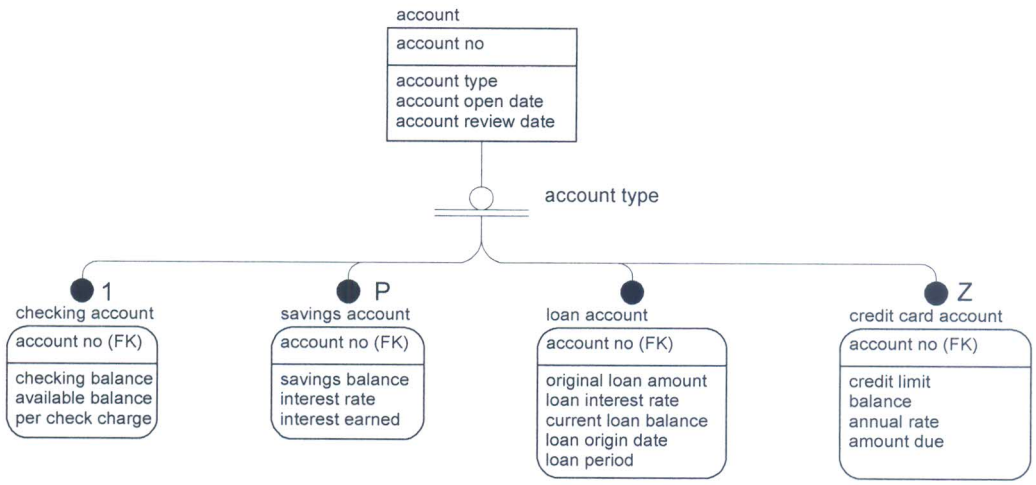


COMPLETE CATEGORIZATION EXAMPLE



An instructor must be either full time or part time but not both. Consequently, we model the relationship as a *complete categorization* since it is mutually exclusive.

COMPLETE CATEGORIZATION EXAMPLE



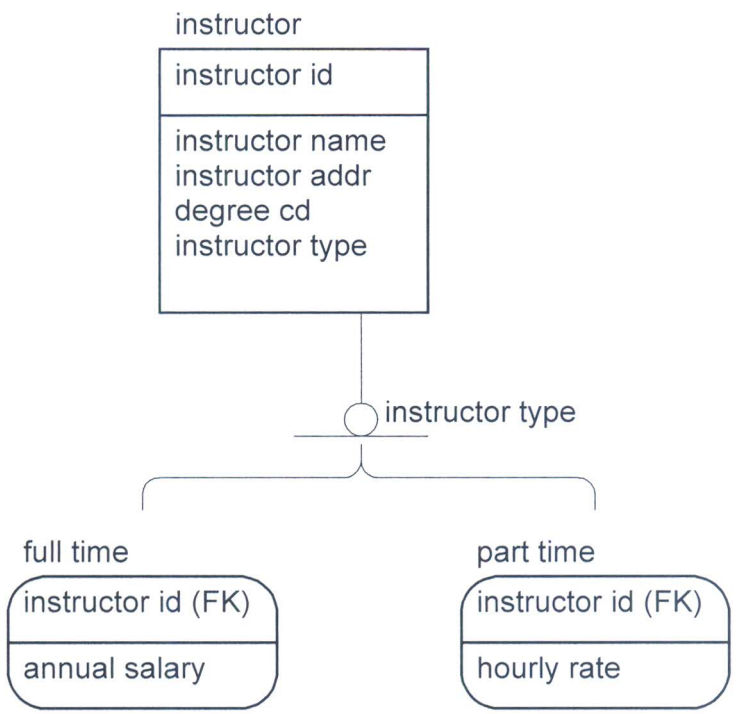
At a bank, we can have different types of accounts. These include checking, savings, loan, and credit card accounts. We use a categorization to represent the relationship because each type of account requires different information.

The relationship defines <u>all possible</u> account types so it is a *complete* list. Consequently, we model the relationship as a *complete categorization*. It is mutually exclusive because each relationship between the account table (parent) and the account type table (child) is mutually exclusive for an entry in the account table.

Note that each relationship can have a different cardinality.

- The *checking account* is 1:1.
- The *savings account* is 1:(1,M).
- The *loan account* is 1:(0,M).
- The *credit card* account is 1:(0,1)

INCOMPLETE CATEGORIZATION EXAMPLE



extracted from Fig_76.vsd

In this example, it is possible that an instructor is working free and is not paid. In this case, the instructor may not be associated with either the full time or part-time category. This means that there are categories not listed so the list is *incomplete*. Consequently, we model the relationship as an *incomplete categorization*.

RULES FOR CATEGORIZATION RELATIONSHIPS

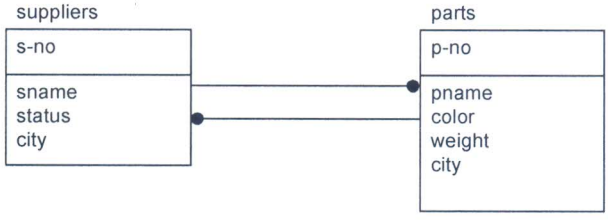
- In an Incomplete Categorization Relationship, an instance of a generic table must be associated with zero or one instance of a only one of the category tables.
- In a Complete Categorization Relationship, an instance of a generic table must be associated with one instance of a only one of the category tables.
- The generic table and each category table must have the same primary key.
- A generic table may be related to more than one group of category tables.
- Role names may be used in the category tables.
- A category table cannot be a child table in another relationship where the inherited columns are part its the primary key.
- A category table can be related to only one generic table.
- A category table may be a generic table in another categorization relationship. This supports modeling nested categorization relationships.

BASIC RULES FOR FINAL MODELS

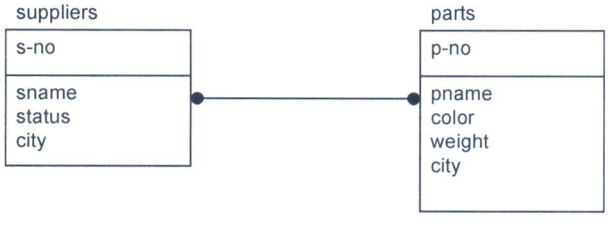
1. The use of "non-specific" relationships is prohibited.
2. Primary key migration from independent to dependent tables is mandatory.
3. Columns cannot have null values (no-null rule).
4. A column cannot have more than one value at one time (no-repeat rule).
5. Tables with compound primary keys cannot be split into multiple tables with simpler primary keys (smallest-key rule).
6. Dual path relationships must be resolved.

NON-SPECIFIC RELATIONSHIP REFINEMENT

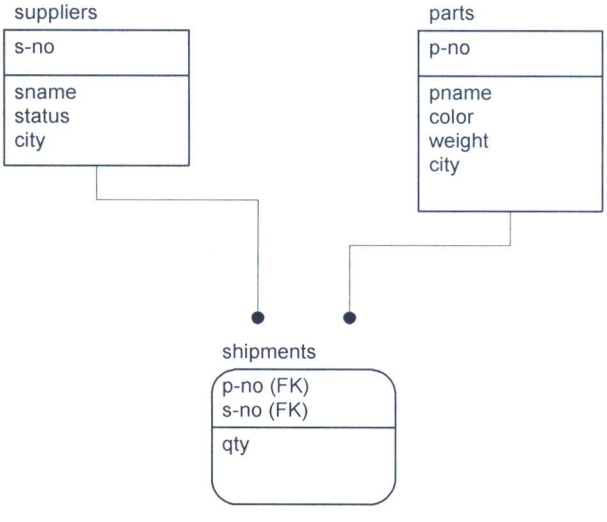
Non-specific relationships take the following forms



OR

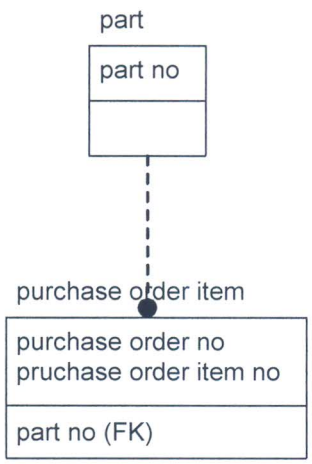


The non-specific relationship is always resolved by adding another table that explicitly specifies the relationship. In this example, the suppliers and parts are related through a shipments table that contains the supplier no, part no and quantity shipped.



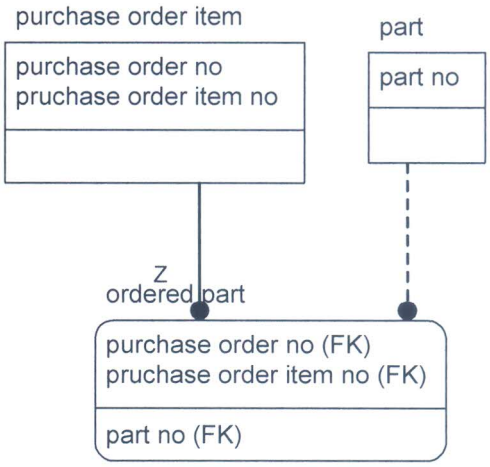
NO-NULL RULE EXAMPLE

Structure with no-null problem



This structure does not provide for purchase order items that may not be for parts such as services, supplies, etc.

Corrected structure



This structure provides the needed flexibility

Fig_86.vsd

NO-NULL RULE EXAMPLE WITH CATEGORIZATION

In this example, the no-null rule is applied to an non-key attribute.

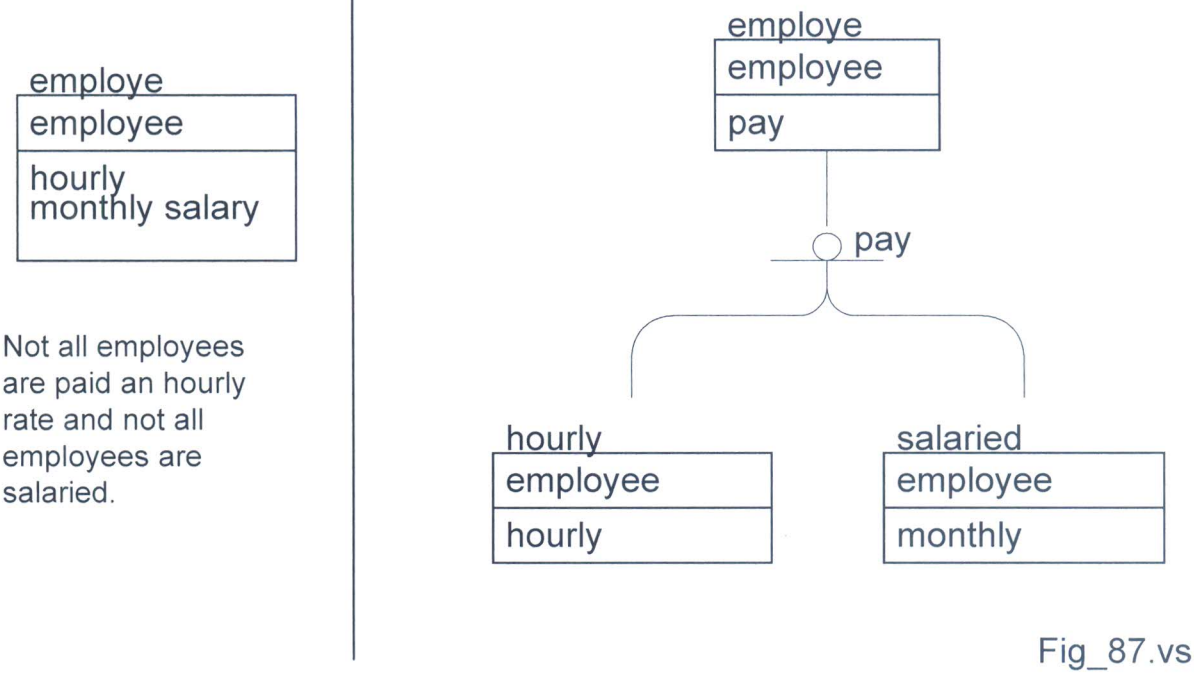


Fig_87.vs

NO-REPEAT RULE EXAMPLE

Typically, the *no-repeat rule* is violated when an item has an embedded or nested list. Examples are a purchase order or a work order. In the example of a purchase order, the document has a header section with the purchase order number, who sold to, account number, address, total, etc. This is followed by a section in which to list the items purchased. This embedded list creates the repeated items in any table used to model the purchase order. It must be extracted into a dependent table.

Purchase Order

Purchase order no. _____
Date_____
Account no. _____
sold to: _____
address: _____

qty	item no	description	unit price	total price
			Total Cost	

NO-REPEAT RULE EXAMPLE ...

Purchase Order

purchase order no. _____
date: _____
sold to: _____
address: _____

QTY	item no	Description	unit price	total price
			TOTAL COST	

Might be incorrectly modeled as a single table with much redundancy.

purchase order

purchase order no item no
date sold to address qty description unit price total price total cost

NO-REPEAT RULE EXAMPLE ...

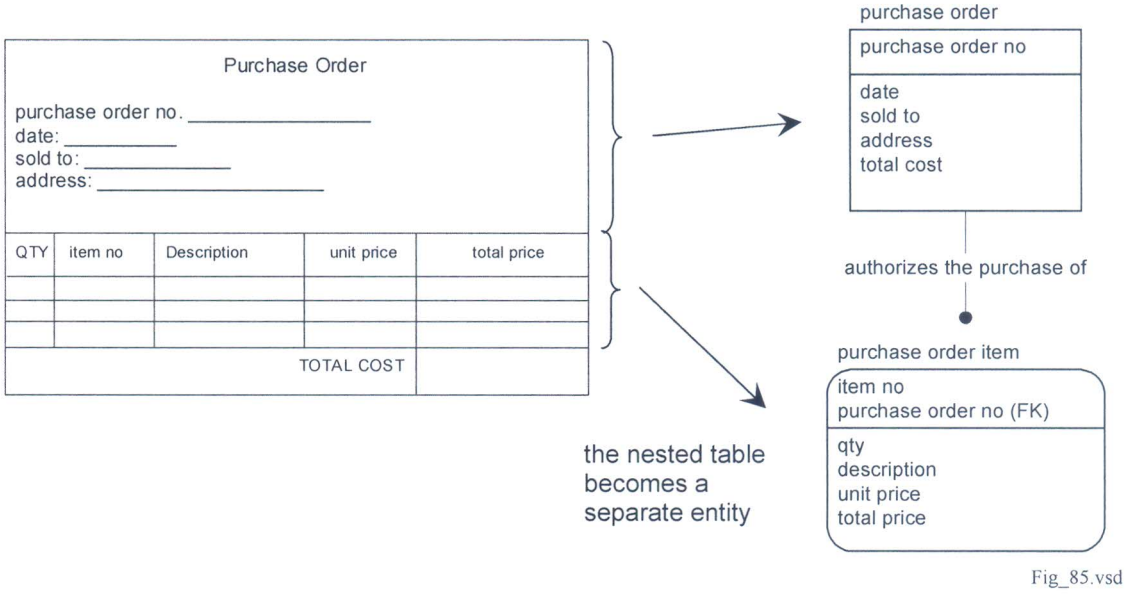
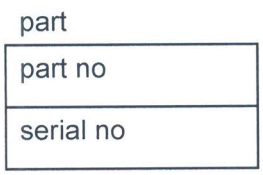


Fig_85.vsd

The header information, including the total cost, is held in the independent table. For each purchase number, there can be one or more purchased items. Consequently, the purchased items are held in an identifier dependent table whose primary key includes the purchase order number and the item no. In this way, an item in the purchased item table can be directly related to a specific purchase order.

NO-REPEAT RULE EXAMPLE FOR A NON-KEY COLUMN



many
numbers may exist
for the same
number

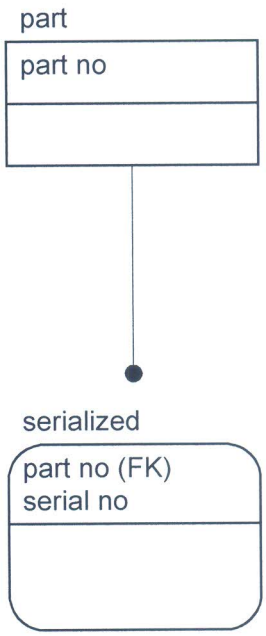


Fig_88.vsd

DUAL PATH RELATIONSHIPS

A dual path relationship exists anytime two tables can be related either directly or indirectly through two different sequences of relationships.

- Equal paths
- Unequal paths
- Indeterminate paths

A dual path relationship is defined by creating a *path assertion*.

DUAL PATH RELATIONSHIPS

EQUAL PATHS

Paths are equal if for each instance of the child table the relationship paths always lead to the same table instance in the root parent table.

UNEQUAL PATHS

Paths are unequal if for each instance of the child table the relationship paths always lead to different table instances in the root parent table.

INDETERMINATE PATHS

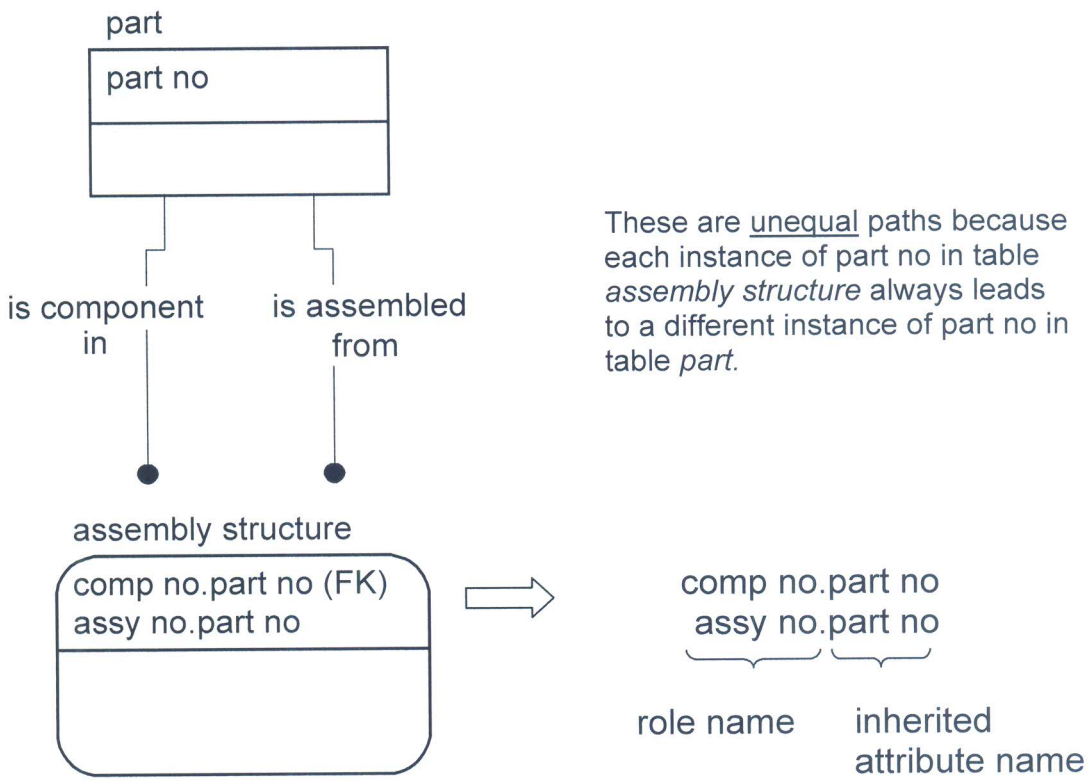
Paths are indeterminate if the paths are sometimes equal and sometimes unequal. This means that for each instance of the child table the relationship paths sometimes lead to the same table instance in the root parent table and sometimes lead to different table instances in the root parent table.

PATH ASSERTIONS

Path Assertions are attached as notes to the model. They define the dual paths that have been identified.

DUAL PATH EXAMPLE

Each instance of table *assembly structure* may relate to two different instances of table part. This is because a part can be simultaneously both a part and a component in a product structure. In this case, no redundancy exists.



These are unequal paths because each instance of part no in table *assembly structure* always leads to a different instance of part no in table *part*.

Fig_89.vsd

DUAL PATH EXAMPLE -- UNEQUAL PATHS WITH MORE THAN ONE RELATIONSHIP

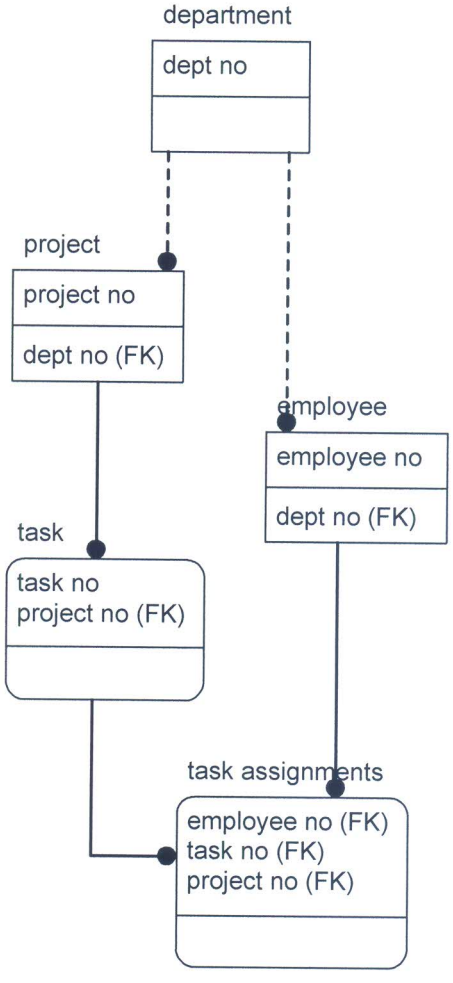


Fig_90.vsd

DUAL PATH EXAMPLE -- UNEQUAL PATHS WITH MORE THAN ONE RELATIONSHIP

Possible Path Assertions

EQUAL PATH ASSERTION

An employee can only be assigned to a project that is managed by their department.

UNEQUAL PATH ASSERTION

An employee can only be assigned to a project that is not managed by their department.

INDETERMINATE PATH ASSERTION

An employee is assigned to a project regardless of the managing department.

REDUNDANT PATHS

When there are equal paths from one table to another table, if one path consists of a single relationship then the single relationship is redundant and should be removed.

REDUNDANT PATH EXAMPLE -- A "TRIAD"

If the path assertion is such that the division to which an employee belongs is the same division as their department (i.e., equal paths) then the relationship between the table *division* and the table *employee* is redundant and should be removed.

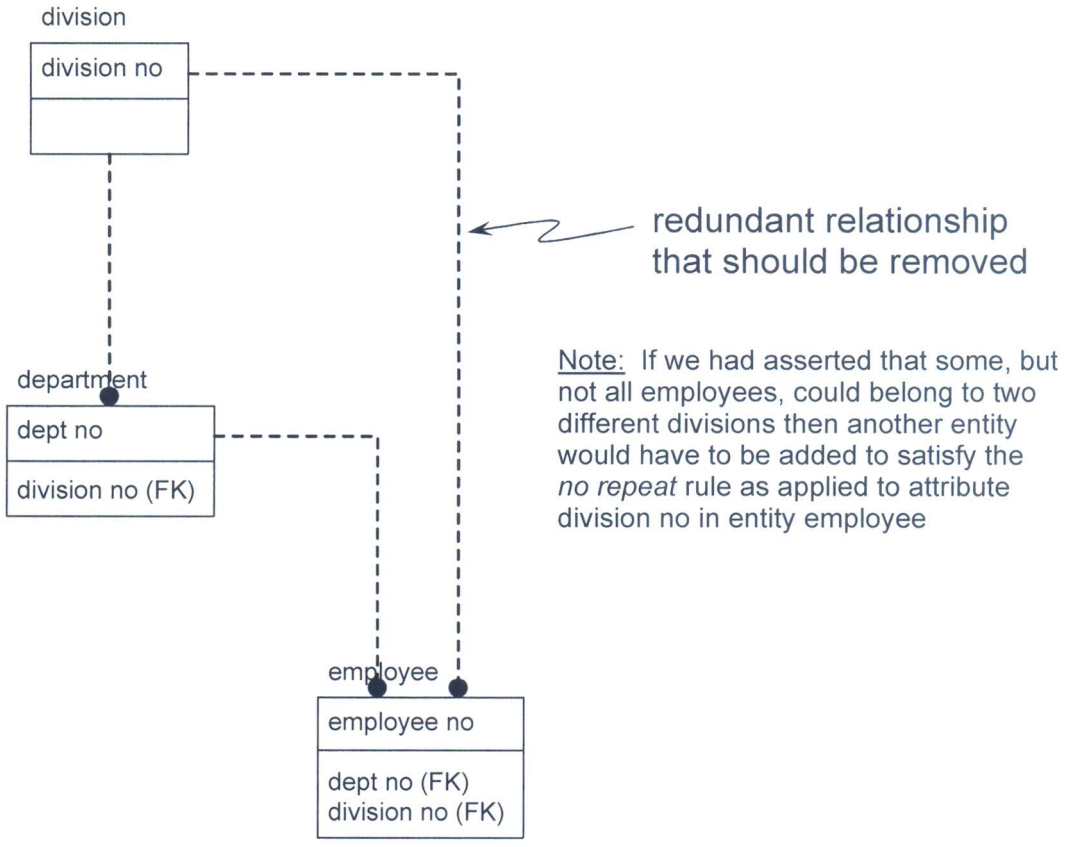
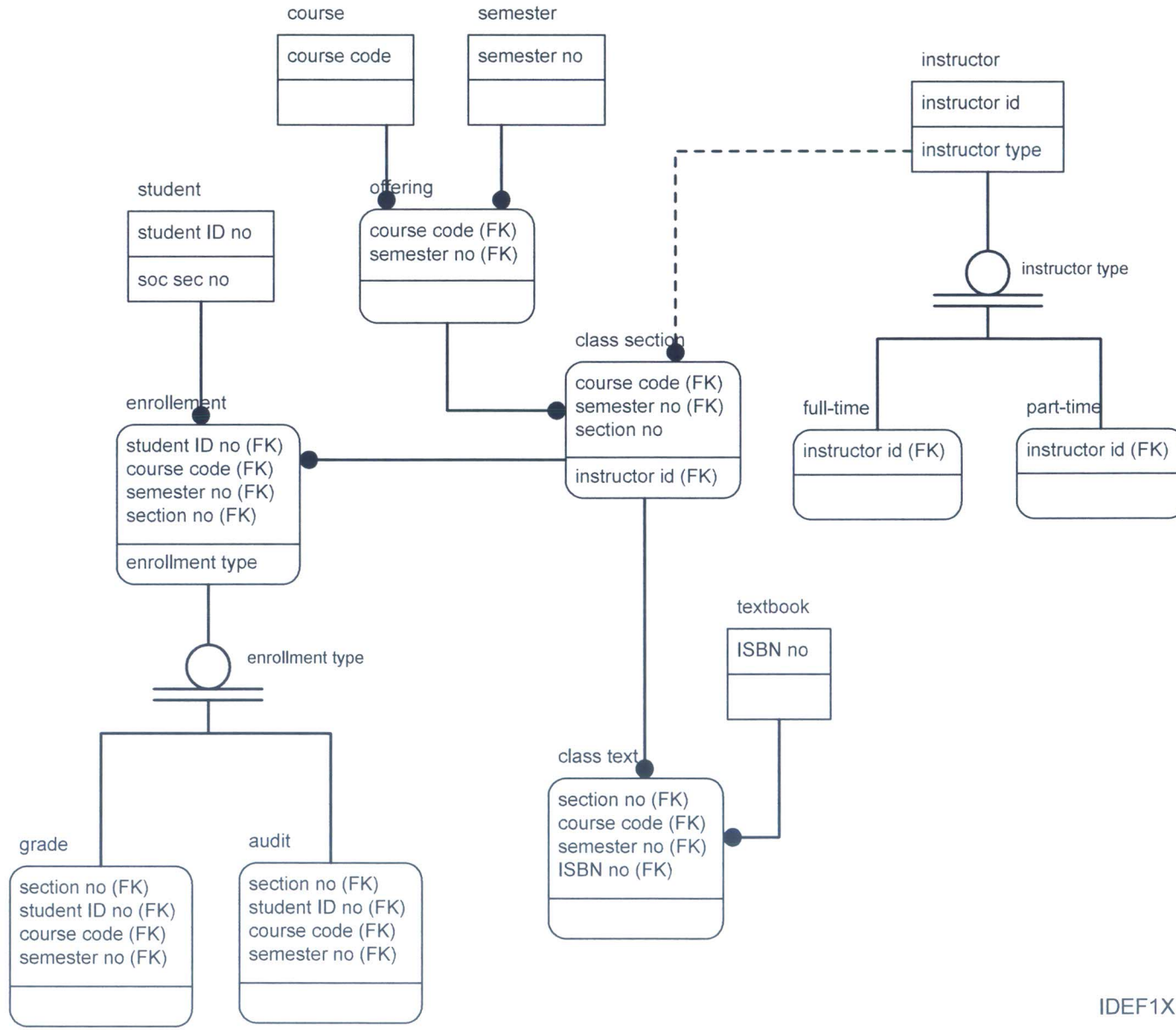


Fig_91.vsd

EXAMPLE FINAL IDEF1X MODEL



The IDEF1X model contains an explicit definition of the operation of the business. This can be used both to create the model and to define operations from an existing model. As an example, we can generate business rules from the *Example Final IDEF1X Model* on the previous page.

- A course is uniquely identified by its *course code.*
- A semester is uniquely identified by its *semester no.*
- An offering is uniquely identified by both its *course code* and an explicit semester (its *semester no*).
- In order for a course to be offered it must be associated with an explicit semester (its *semester no*) and have a unique course code.
- An instructor is uniquely identified by their *instructor ID*.
- An instructor can be one of two types, either full-time or part-time.
- A class section has a course code, a semester number, a section number and an assigned instructor. Each section is uniquely identified by its course code, semester number and section number.
- In order for a class section to exist it must be associated with an explicit offering and a specific instructor.
- A student is uniquely identified by their *student no*.
- An alternative unique identifier for a student is their social security number but this is not commonly used.
- The enrollment in a course is determined by the student ID number, the course code, the semester number and the section number. These items uniquely determine the enrollment.
- There are only two types of enrollment in a course, either for a grade or as an audit.
- A textbook is uniquely determined by its *ISBN no*.
- A class's textbook is uniquely determined by the section number, the course code, the semester number and the ISBN number.

COMMENTS ON BUSINESS RULE EXAMPLE

The business rules define how the organization operates because the database reflects the way information is organized, characterized and categorized. As a result, the structure of the database is an image of the information structure of the business.