Relational Model

Objectives

- origins of relational model
- terminology of relational model
- how tables are used to represent data
- connection between mathematical relations and relations in relational model
- properties of database relations

Objectives

- how to identify candidate, primary, alternate, and foreign keys
- meaning of entity integrity and referential integrity
- purpose and advantages of views in relational systems

Historical perspective

- Relational Database Management System (RDBMS) has become dominant data-processing software in use today, with an estimated total software revenue worldwide of US$ 20-30 billion
- software represents second generation of DBMSs based on proposed *relational data model* in 1970, Edgar Codd, at IBM's San Jose Research Laboratory

Historical perspective

- in relational model, all data is logically structured within relations (tables); each ***relation*** has name and is made up of named ***attributes*** (columns) of data; each ***tuple*** (row) contains one value per attribute
- great strength of relational model is this simple logical structure; behind this structure is sound theoretical foundation that is lacking in first generation of DBMSs (network & hierarchical)

Mathematics

- always strive for elegance and orthogonality - dislike exceptions
- system is said to be designed in an *orthogonal* way if its set of components that together make up whole system capability are *non-overlapping* and *mutually independent*
  - each capability should be implemented by only one component
  - one component should only implement one capability of system

Mathematics

- well-separated and independent components ensure that there are no side effects: using or even changing one component does not cause side effects in another area of system
- formal disciplines, most relevant ones in application of mathematics to field of databases
- Set theory

Mathematics

- *Everything should be made as simple as possible, but not simpler.*
    - Albert Einstein (1879–1955)

Codd relational model's objectives

- to allow high degree of data independence; application programs must not be affected by modifications to internal data representation, particularly by changes to file organizations, record orderings, or access paths
- to provide substantial grounds for dealing with data semantics, consistency, and redundancy problems; introduced concept of ***normalized*** relations that have no repeating groups
- to enable expansion of set-oriented data manipulation languages

1 - Prototype relational DBMS System R

- at IBM's San José Research Laboratory in California - prototype relational DBMS System R, which was developed during the late 1970s
- project was designed to prove practicality of relational model by providing an implementation of its data structures and operations
- proved to be excellent source of information about implementation concerns such as transaction management, concurrency control, recovery techniques, query optimization, data security and integrity, human factors, and user interfaces,

led to two major developments

- development of a structured query language called SQL, which has since become formal International Organization for Standardization (ISO) and de facto standard language for RDBMS
- production of various commercial relational DBMS products during late 1970s and 1980s: IBM DB2 and Oracle

2 - INGRES

- development of relational model INGRES (Interactive Graphics Retrieval System) project at University of California at Berkeley
- involved development of prototype RDBMS, with research concentrating on same overall objectives as System R project - led to an academic version of INGRES, which contributed to general appreciation of relational concepts, and spawned commercial products INGRES

Terminology and structural concepts of relational model

- relational model is based on mathematical concept of ***relation***, which is physically represented as ***table***
- we explain terminology and structural concepts of the relational model

Relation

- is a table with columns and rows
- RDBMS requires only that database be perceived by user as tables
- perception applies only to logical structure of database: external and conceptual levels of ANSI-SPARC architecture
- does not apply to physical structure of database, which can be implemented using variety of storage structures

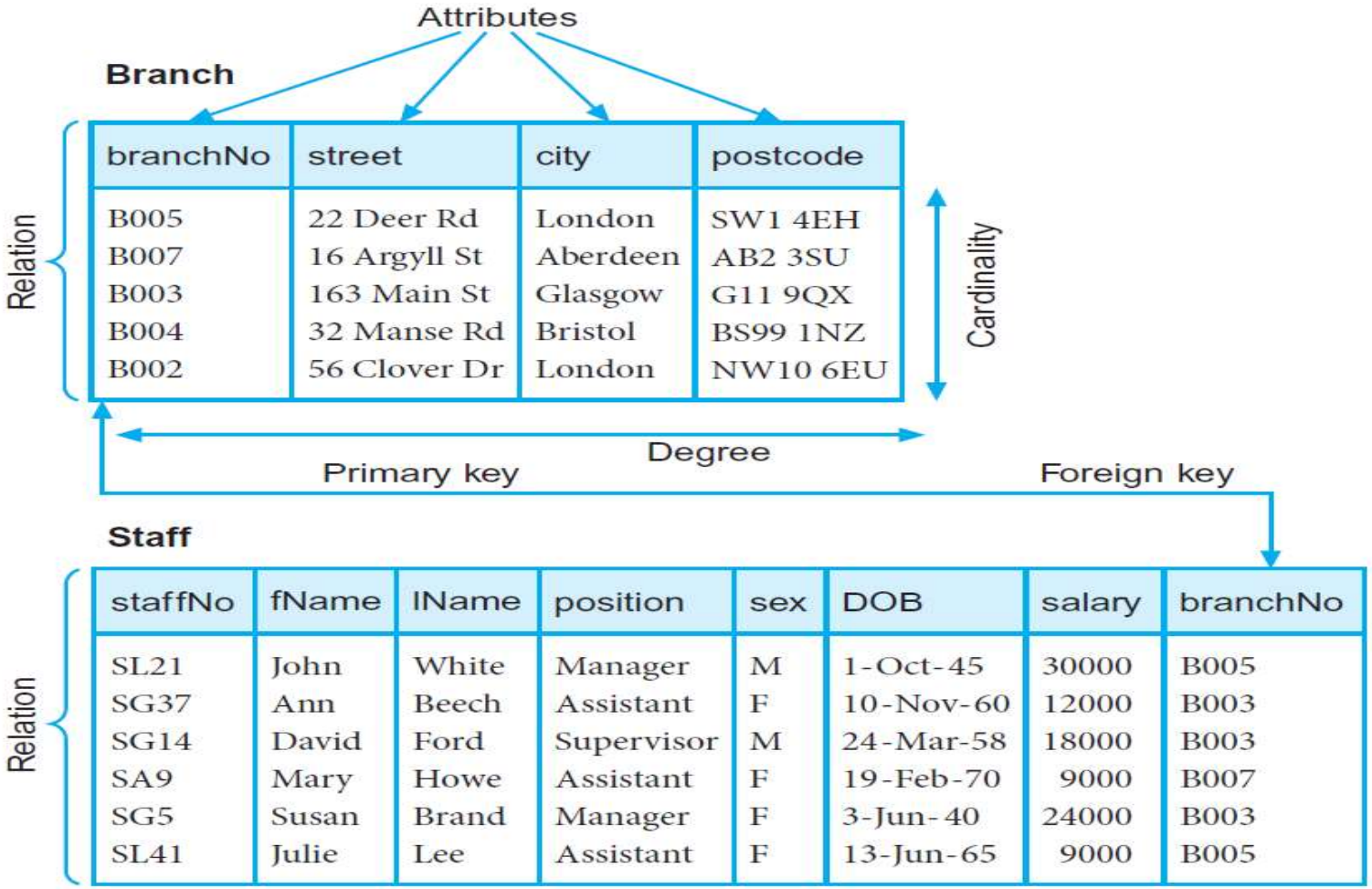
Attribute

- is named column of relation
- relations used to hold information about objects to be represented in database as a two-dimensional table in which rows correspond to individual records and columns correspond to attributes
- attributes can appear in any order and relation will still be the same - therefore will convey same meaning

Domain

- is set of allowable (possible) values for one or more attributes
- may be distinct for each attribute, or two or more attributes may be defined on same domain
- typically there will be values in domain that do not currently appear as values in corresponding attribute

Instances of Branch & Staff relation



Domains for some attributes

Attribute	Domain Name	Meaning	Domain Definition
branchNo	BranchNumbers	The set of all possible branch numbers	character: size 4, range B001–B999
street	StreetNames	The set of all street names in Britain	character: size 25
city	CityNames	The set of all city names in Britain	character: size 15
postcode	Postcodes	The set of all postcodes in Britain	character: size 8
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible values of staff birth dates	date, range from 1-Jan-20, format dd-mmm-yy
salary	Salaries	Possible values of staff salaries	monetary: 7 digits, range 6000.00–40000.00

tuple

- is row of relation
- elements of relation are rows or tuples in table
- tuples can appear in any order and relation will still be the same - therefore convey same meaning
- structure of relation, together with specification of domains and any other restrictions on possible values, is called its intension; usually fixed - tuples are called extension (or state) of relation, which changes over time

Degree

- is number of attributes it contains
- the Branch relation has four attributes or degree four – means that each row of table is a four-tuple, containing four values
- degree of a relation is property of *intension* of relation

Cardinality

- is number of tuples it contains
- changes as tuples are added or deleted; is property of *extension* of relation and is determined from particular instance of relation at any given moment

Relational database

- collection of normalized relations with distinct relation names
- consists of relations that are appropriately structured - refer to this as *normalization*

Alternative terminology

- third set of terms is sometimes used; stems from fact that, physically, RDBMS may store each relation in a file

Alternative terminology

Formal term	Logical	Physical
Relation	Table	File
Attribute	Column	Field
Tuple	Row	Record

MATHEMATICAL RELATIONS

Sets and Elements

- set is a collection of objects, and those objects are called the elements of the set

- set is fully characterized by its distinct elements, and nothing else but these elements

Sets and Elements

- elements of set don't have any ordering
- sets don't contain duplicate elements

- two sets A and B, are the same if each element of A is also an element of B and each element of B is also an element of A

- ∅ the empty set

Sets and Elements

- set theory has a mathematical symbol $\in$ that is used to state that some object is an element of some set
- $x \in S$
- $x \notin S \leftrightarrow \neg(x \in S)$

Methods to specify sets

- enumerative method
  - S1 := {1, 2, 3, 5, 7, 11, 13, 17, 19, 23}
- predicative method
  - S2 := {$x \in S \mid P(x)$ }
    - let P(x) be a given predicate with exactly one parameter named x of type S
- substitutive method
  - S3 := { $x^2 \mid x \in N$}

Cardinality

- sets can be finite and infinite
- when dealing with databases all your sets are finite by definition
- for every finite set S, cardinality is defined as number of elements of S

Subsets, Supersets

- A is a subset of B (B is a superset of A) if every element of A is an element of B
- $A \subseteq B$
- A is proper subset of B (B is proper superset of A) if A is subset of B and $A \neq B$
- $A \subset B$

Union, Intersection, Difference

- union $A \cup B = \{ x \mid x \in A \vee x \in B\}$
- intersection $A \cap B = \{ x \mid x \in A \wedge x \in B\}$
- difference $A - B = \{ x \mid x \in A \wedge x \notin B\}$

(Binary) Relation

- two sets, D_1 and D_2, where $D_1 = \{2, 4\}$ and $D_2 = \{1, 3, 5\}$; Cartesian product of these two sets, written $D_1 \times D_2$, is set of all ordered pairs such that first element is a member of D_1 and second element is a member of D_2 - $D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$
- any subset of this Cartesian product is a relation

Example of relation R

- $R = \{(2, 1), (4, 1)\}$
- $R = \{(x, y) \mid x \in D_1, y \in D_2, \text{ and } y = 1\}$
- easily extend notion to three sets; let D_1, D_2, and D_3 be sets; Cartesian product $D_1 \times D_2 \times D_3$ of these three sets is set of all ordered triples such that first element is from D_1, second is from D_2 and third element is from D_3
- any subset of Cartesian product is relation

N-ary Relation

- order of coordinates of ordered pair is important
- order of tuples it is not important – it is subset of Cartesian – subset of a set is a set – order of element in a set it is not important
- can extend three sets to a more general concept of n-ary relation

Relation

- define general relation on *n* domains
- let $D_1, D_2, \ldots, D_n$ be *n* sets
- Cartesian product is defined as:
- $D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$ and is usually written as $\prod_{i=1}^{n} D_i$
- any set of *n*-tuples from this Cartesian product is relation on *n* sets

DataBase Relations

- note that in defining these relations we have to specify sets, or domains, from which we choose values
- Relation schema: named relation defined by set of attribute and domain name pairs
- let $A_1, A_2, \ldots, A_n$ be attributes with domains $D_1, D_2, \ldots, D_n$
- then set $\{A_1{:}D_1, A_2{:}D_2, \ldots, A_n{:}D_n\}$ is relation schema

DataBase Relations

- relation *R* defined by relation schema is set of mappings from attribute names to their corresponding domains
- relation *R* is set of *n*-tuples:
- $(A_1{:}d_1, A_2{:}d_2, \ldots, A_n{:}d_n)$ such that $d_1 \in D_1$, $d_2 \in D_2, \ldots, d_n \in D_n$
- each element in *n*-tuple consists of attribute and value for that attribute

DataBase Relations

- normally we list attribute names as column headings and write out tuples as rows having form $(d_1, d_2, \ldots, d_n)$, where each value is taken from appropriate domain
- Relation is any subset of Cartesian product of domains of attributes
- table is simply physical representation of such relation

Branch and Staff relations

Branch

branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

- it shows for example that employee John White is manager with salary of 30,000, who works at branch (branchNo) B005, which, from first table, is at 22 Deer Rd in London
- there is relationship between Staff and Branch: branch office *has* staff
- there is no explicit link between these two tables; it is only by knowing that attribute branchNo in Staff have same value as branchNo of Branch

Branch relation

- has attributes branchNo, street, city, and postcode, each with its domain
- is any subset of Cartesian product of domains, or any set of four-tuples in which first element is from domain BranchNumbers, second is from domain StreetNames, …

Branch relation instance

- one of tuples is: or more correctly
- {(B005, 22 Deer Rd, London, SW1 4EH)}
- {(branchNo: B005, street: 22 Deer Rd, city: London, postcode: SW1 4EH)}
- we refer to this as *relation instance*
- Branch table is convenient way of writing out all tuples that form relation at specific moment in time

Relational database schema

- set of relation schemas, each with distinct name
- in same way that relation has schema, so too does relational database
- if R_1, R_2, . . . , R_n are set of relation schemas, then we can write *relational database schema,* or simply *relational schema, R,* as: $R = \{R_1, R_2, \ldots, R_n\}$

Properties of Relations

- relation has name that is distinct from all other relation names in relational schema
- each cell of relation contains exactly one atomic (single) value; relations do not contain repeating groups
- relation that satisfies this property is said to be *normalized* in *first normal form*

Properties of Relations

- each attribute has distinct name
- value of attribute are all from same domain
- each tuple is distinct; there are no duplicate tuples
- order of attributes has no significance
- order of tuples has no significance
- most of properties specified result from properties of mathematical relations

Relational Keys

- … there are no duplicate tuples ...
- therefore, we need to identify one or more attributes (***relational keys***) that uniquely identifies each tuple in relation

Superkey

- attribute, or set of attributes, that uniquely identifies tuple within relation
- may contain additional attributes that are not necessary for unique identification
- interested in identifying superkeys that contain only minimum number of attributes necessary for unique identification

Candidate Key

- superkey such that no proper subset is superkey within relation
- *Uniqueness* – in each tuple of *R*, values of *K* uniquely identify that tuple
- *Irreducibility* - no proper subset of *K* has uniqueness property
- when key consists of more than one attribute, we call it *composite key*
- there may be several candidate keys for relation

Candidate Key

- consider Branch relation
- given value of city, we can determine several branch offices (London has two branch offices)
- attribute cannot be a candidate key
- company allocates each branch office unique branch number we can determine at most one tuple, so that *branchNo* is candidate key
- similarly, postcode is also candidate key

Candidate Key

- consider relation *Viewing*, which contains information relating to properties viewed by clients
- relation comprises client number (*clientNo*), property number (*propertyNo*), date of viewing (*viewDate*) and, optionally, comment (*comment*)

Candidate Key

- combination of *clientNo* and *propertyNo* identifies at most one tuple, so for *Viewing* together form (composite) candidate key
- if we take into account possibility that client may view property more than once, then we could add *viewDate* to composite key

Candidate Key

- instance of relation cannot be used to prove that attribute or combination of attributes is CK
- fact that there are no duplicates for values that appear at particular moment in time does not guarantee that duplicates are not possible
- presence of duplicates in instance can be used to show that attribute combination is not CK
- identifying candidate key requires that we know the “real-world” meaning of attribute(s) involved; can decide whether duplicates are possible

Primary Key

- candidate key that is selected to identify tuples uniquely within relation (by database designer)
- relation has no duplicate tuples; always possible to identify each row uniquely
- relation always has primary key
- in worst case, entire set of attributes could serve as primary key; usually some smaller subset is sufficient to distinguish tuples

Alternate Key

- candidate keys that are not selected to be primary key
- for *Branch* relation, if we choose *branchNo* as primary key, *postcode* would then be an alternate key
- for *Viewing* relation, there is only one candidate key, comprising *clientNo* and *propertyNo*, so these attributes would automatically form primary key

Foreign Key

- attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation
- when attribute appears in more than one relation, its appearance usually represents relationship between tuples of two relations
- for example, inclusion of *branchNo* in both *Branch* and *Staff* relations is quite deliberate and links each branch to details of staff working at that branch

Foreign Key

- in *Branch* relation, *branchNo* is primary key
- in *Staff* relation, *branchNo* attribute exists to match staff to branch office they work in
- in *Staff* relation, *branchNo* is *foreign key*
- we say that attribute *branchNo* in *Staff* relation *targets* primary key attribute *branchNo* in home relation, *Branch*

Representing Relational Database Schemas

- *Branch (branchNo, street, city, postcode)*
- *Staff (staffNo, fName, IName, position, sex, DOB, salary, branchNo)*
- *PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)*
- *Client (clientNo, fName, IName, telNo, prefType, maxRent, eMail)*

Representing Relational Database Schemas

- *PrivateOwner (ownerNo, fName, lName, address, telNo, eMail, password)*
- *Viewing (clientNo, propertyNo, viewDate, comment)*
- *Registration (clientNo, branchNo, staffNo, dateJoined)*

Representing Relational Database Schemas

- common convention for representing relation schema is to give name of relation followed by attribute names in parentheses; primary key is underlined
- *conceptual model,* or *conceptual schema,* is set of all such schemas for database

Instance of rental database

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Instance of rental database

Client

clientNo	fName	lName	telNo	prefType	maxRent	eMail
CR76	John	Kay	0207-774-5632	Flat	425	john.kay@gmail.com
CR56	Aline	Stewart	0141-848-1825	Flat	350	astewart@hotmail.com
CR74	Mike	Ritchie	01475-392178	House	750	mritchie01@yahoo.co.uk
CR62	Mary	Tregear	01224-196720	Flat	600	maryt@hotmail.co.uk

PrivateOwner

ownerNo	fName	lName	address	telNo	eMail	password
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	jkeogh@lhh.com	********
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419	cfarrel@gmail.com	********
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728	tinam@hotmail.com	********
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	tony.shaw@ark.com	********

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-13	too small
CR76	PG4	20-Apr-13	too remote
CR56	PG4	26-May-13	
CR62	PA14	14-May-13	no dining room
CR56	PG36	28-Apr-13	

Registration

clientNo	branchNo	staffNo	dateJoined
CR76	B005	SL41	2-Jan-13
CR56	B003	SG37	11-Apr-12
CR74	B003	SG37	16-Nov-11
CR62	B007	SA9	7-Mar-12

Null

- represents value for attribute that is currently unknown or is not applicable for this tuple
- mean logical value “unknown”, value is not applicable to particular tuple, or that no value has yet been supplied
- way to deal with incomplete or exceptional data
- not same as zero or empty text string
- represents absence of value

Null

- for example, in Viewing relation *comment* attribute may be undefined until potential renter has visited property and returned comment to agency
- without nulls, it becomes necessary to introduce false data to represent this state

Null

- can cause implementation problems, arising from fact that relational model is based on first-order predicate calculus, which is two-valued or Boolean logic; allowing nulls means that we have to work with higher-valued logic, such as three-valued logic (true, false, null)
- Codd regarded nulls as an integral part of relational model

Integrity Constraints

- relational integrity constraints ensure that data is accurate
- because every attribute has an associated domain, there are ***domain constraints*** that form restrictions on set of values allowed for attributes of relations
- ***integrity rules*** - constraints that apply to all instances of database: ***entity integrity*** and ***referential integrity***

Integrity Constraints

- because every attribute has an associated domain, there are ***domain constraints*** that form restrictions on set of values allowed for attributes of relations
- ***integrity rules*** - constraints that apply to all instances of database: ***entity integrity*** and ***referential integrity***

Entity integrity

- in base relation, no attribute of primary key can be null
- PK is minimal identifier used to identify tuples uniquely; if we allow null for any part of PK we are implying that not all attributes are needed
- for example, *branchNo* is PK of *Branch*, we should not be able to insert tuple with null
- Viewing (clientNo, propertyNo, …) we should not be able to insert tuple into Viewing with null for clientNo or null for propertyNo or nulls for both

Entity integrity

- using data of *Viewing* relation consider query "List all comments from viewings"
- produce unary relation consisting of attribute comment
- this attribute must be PK, but it contains nulls
- this relation is not base relation (derived relation), model allows primary key to contain nulls

Referential integrity

- if foreign key exists in relation, either foreign key value must match a candidate key value of some tuple in its home relation or foreign key value must be wholly null

Referential integrity

- for example, *branchNo* in *Staff* is FK targeting *branchNo* attribute in home relation, *Branch*
- possible to create *Staff* record with *branchNo* B025, if there is already *branchNo* B025 in *Branch* relation
- we should be able to create *Staff* record with null *branchNo* to allow for situation where member of staff has joined company but has not yet been assigned to particular branch office

General constraints

- additional rules specified by users of database that define or constrain some aspect of organization
- for example, if an upper limit of 20 has been placed upon number of staff that may work at branch office
- level of support for general constraints varies from system to system

General constraints

- *declarative integrity constraint* is statement about database that is always true
- *trigger* is procedural code that is automatically executed in response to certain events on particular table or view in database; mostly used for maintaining integrity of information

Views

- in relational model has slightly different meaning then view from architecture
- rather than being entire external model of user's view, view is *virtual* or *derived relation* that does not necessarily exist in its own right, but may be dynamically derived from one or more base relations
- external model can consist of both base (conceptual-level) relations and views derived from base relations

Views

- relation that appears to user to exist, can be manipulated as if it were base relation, but does not necessarily exist in storage (although its definition is stored in system catalog)
- contents of view are defined as query on one or more base relations
- operations on view are automatically translated into operations on relations from which it is derived

Views

- dynamic – changes made to base relations that affect view are immediately reflected in view
- when users make permitted changes to (updating) view, these changes are made to underlying relations

Views

- provides powerful and flexible security mechanism by hiding parts of database from certain users; users are not aware of existence of any attributes or tuples that are missing
- it permits users to access data in way that is customized to their needs, same data can be seen by different users in different ways
- it can simplify complex operations on base relations

Views

- for example, if view defined as combination (join) of two relations users may now perform more simple operations on view, which will be translated by DBMS into equivalent operations
- some members of staff should see *Staff* tuples without salary attribute
- attributes may be renamed or order changed; user accustomed to calling *branchNo* attribute of branches by full name

Views

- provides *logical data independence;* supports reorganization of conceptual schema
- for example, if new attribute is added to relation, existing users can be unaware of its existence if their views are defined to exclude it; if existing relation is rearranged or split up, view may be defined so that users can continue to see their original views

Codd's rules

- Codd came up with rules database must obey in order to be regarded as relational
- hardly any commercial product follows all

Rule 0: Foundation

- system must qualify as relational, as a database, and as a management system
- system must use its relational facilities (exclusively) to manage database
- foundation rule, which acts as base for all other 12 rules derive from this

Rule 1: Information

- all information in database is to be represented in one and only one way, namely by values in column positions within rows of tables
- data stored in database, may it be user data or metadata, must be value of some table cell; everything in database must be stored in a table format

Rule 2: Guaranteed Access

- all data must be accessible; essentially restatement of fundamental requirement for primary keys; every individual scalar value in database must be logically addressable by specifying name of table, name of column and primary key value
- every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row), and attribute-name (column); no other means, such as pointers, can be used to access data

Rule 3: Systematic Treatment of NULL Values

- DBMS must allow each field to remain null (or empty); must support representation of "missing information and inapplicable information" that is systematic, distinct from all regular values (0, “”, …), and independent of data type; manipulated by DBMS in systematic way
- NULL values in database must be given systematic and uniform treatment; interpreted as − data is missing, data is not known, or data is not applicable

Rule 4: Active Online Catalog

- system must support online relational catalog that is accessible to authorized users by means of their regular query language; users must be able to access database's structure (catalog) using same query language that they use to access database's data
- structure description of entire database must be stored in online catalog, known as *data dictionary*, which can be accessed by authorized users; users can use same query language to access catalog which they use to access database itself

Rule 5: Comprehensive Data Sub-Language

- system must support at least one relational language that
  - has linear syntax
  - can be used both interactively and within application programs
  - supports data definition operations (including view definitions), data manipulation operations (update and retrieval), security and integrity constraints, transaction management operations (begin, commit, and rollback)

Rule 5: Comprehensive Data Sub-Language

- database can only be accessed using language having linear syntax that supports data definition, data manipulation, and transaction management operations; language can be used directly or by means of some application
- if database allows access to data without any help of this language, then it is considered violation

Rule 6: View Updating

- all views those can be updated theoretically, must be updated by system.
- all views of database, which can theoretically be updated, must also be updatable by system

Rule 7: High-Level Insert, Update, and Delete

- system must support set-at-a-time insert, update, and delete operators; data can be retrieved from relational database in sets constructed of data from multiple rows and/or multiple tables; insert, update, and delete operations should be supported for any retrievable set rather than just for single row in single table

Rule 7: High-Level Insert, Update, and Delete

- database must support high-level insertion, updation, and deletion; this must not be limited to single row, that is, it must also support union, intersection and minus operations to yield sets of data records

Rule 8: Physical Data Independence

- **c**hanges to physical level (how data is stored, whether in arrays or linked lists etc.) must not require change to application based on structure
- data stored in database must be independent of applications that access database; any change in physical structure of database must not have any impact on how data is being accessed by external applications

Rule 9: Logical Data Independence

- changes to logical level (tables, columns, rows, …) must not require change to application based on structure; more difficult to achieve
- logical data in database must be independent of its user's view (application); any change in logical data must not affect applications using it

Rule 10: Integrity Independence

- integrity constraints must be specified separately from application programs and stored in catalog; it must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications
- database must be independent of application that uses it; all its integrity constraints can be independently modified without need of any change in application; makes database independent of front-end application & interface

Rule 11: Distribution Independence

- distribution of portions of database to various locations should be invisible to users; existing applications should continue to operate
  - when distributed version is first introduced
  - when existing distributed data are redistributed
- end-user must not be able to see that data is distributed over various locations; users should always get impression that data is located at one site only;
- regarded as foundation of distributed database

Rule 12: Non-Subversion Rule

- if system provides low-level (record-at-a-time) interface, then that interface cannot be used to subvert system; bypassing relational security or integrity constraint
- if system has an interface that provides access to low-level records, then interface must not be able to subvert system and bypass security and integrity constraints

Review Questions

- discuss each of following concepts in context of relational data model: relation; attribute; domain; tuple; intension and extension; degree and cardinality
- describe term “normalized reaction”
- why are constraints so important in relational database
- discuss properties of relation

Review Questions

- differences between CK & PK of relation
- explain what is meant by FK; how do FK relate to CK? give examples
- define two principal integrity rules for relational model; why it is desirable to enforce these rules
- define views; why are they important in database approach

Entity–Relationship Modeling

Objectives

- ***use Entity–Relationship (ER) modeling in database design***
- basic concepts associated with ER model: entities, relationships, and attributes
- identify and resolve problems with ER models (connection traps)

Objectives

- **Specialization/Generalization**
- special types of entities known as superclasses and subclasses, and attribute inheritance
- present two main types of constraints on superclass/subclass relationships called participation and disjoint constraints
- show how to represent specialization / generalization in ER diagram

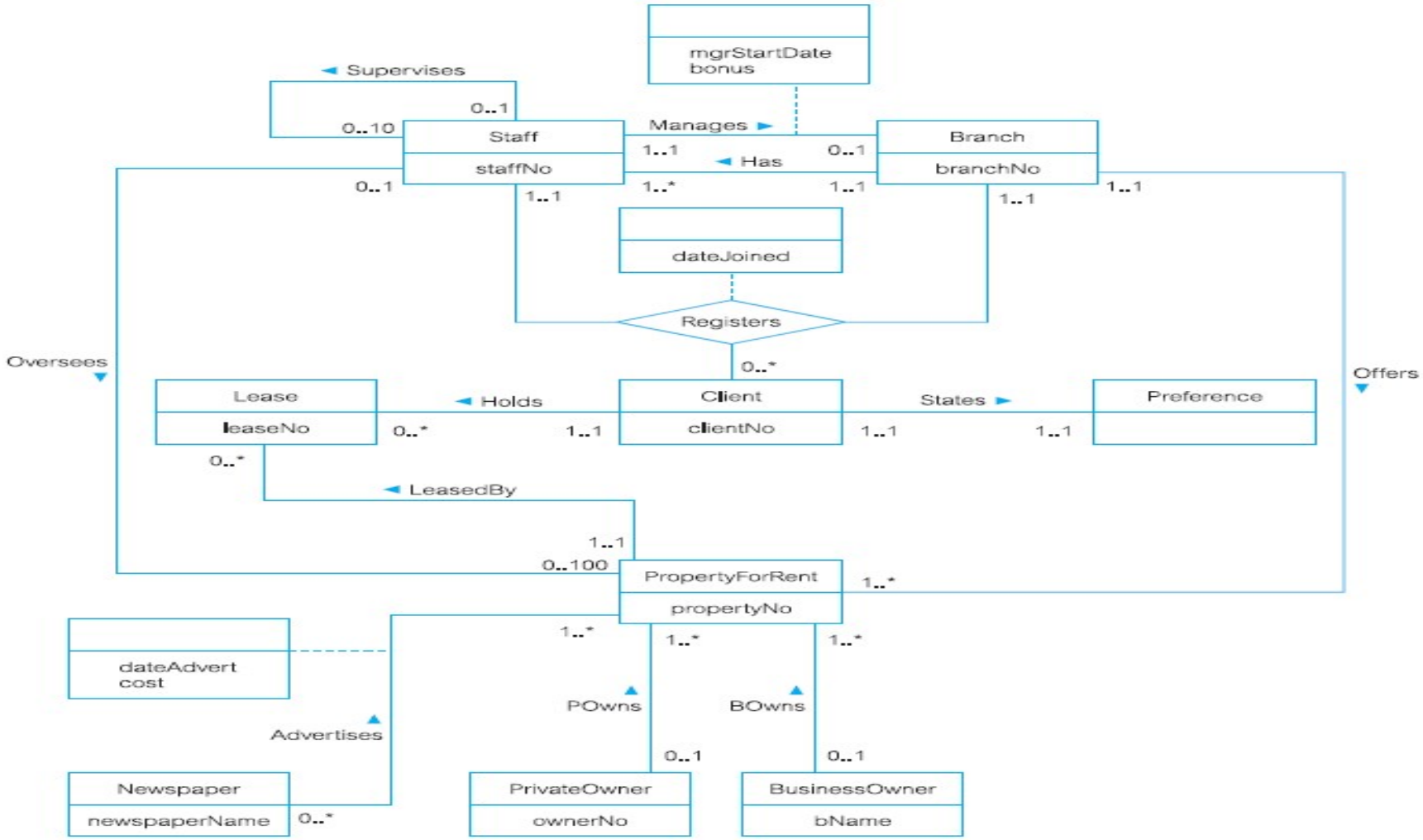
- designers, programmers, and end-users tend to view data & its use in different way
- common understanding that reflects how enterprise operates
- need model for communication - nontechnical and free of ambiguities
- Entity–Relationship (ER) model is one such example

- Entity–Relationship (ER) modeling is top-down approach to database design that begins by identifying important data called *entities* and *relationships;* add more details, such as information we want to hold about entities and relationships (*attributes)* and any *constraints*

Entity Set

- group of objects with same properties, which are identified by enterprise as having an independent existence

Entity–Relationship (ER) diagram of Branch view



Entity

- independent existence can be objects with physical (or “real”) existence
- objects with conceptual (or “abstract”) existence

Physical existence

- Staff
- Property
- Customer
- Part
- Supplier
- Product

Conceptual existence

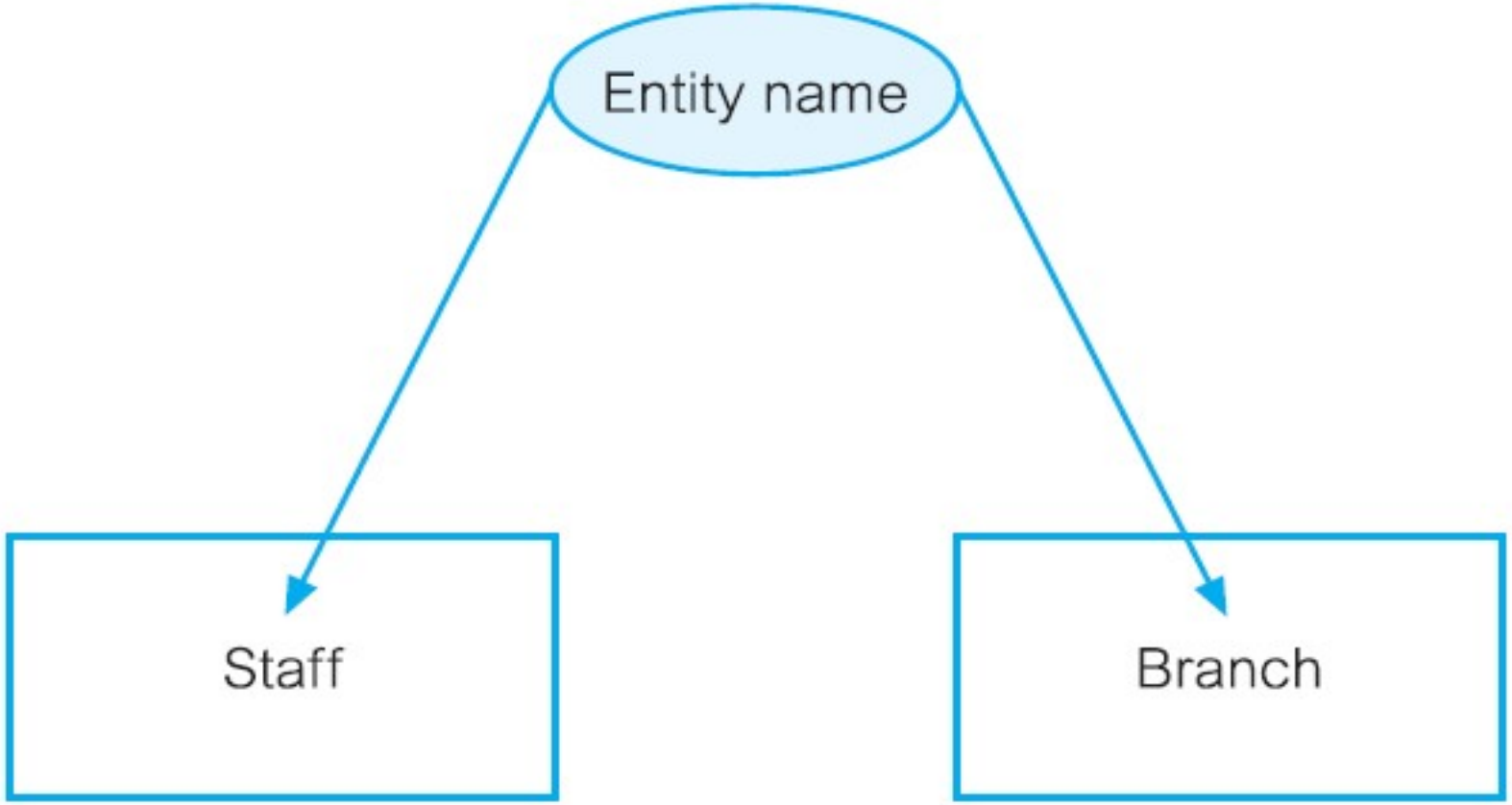
- Viewing
- Inspection
- Sale
- Work experience

Entity occurrence

- uniquely identifiable object of entity set

- use terms “entity set” and “entity” or “entity occurrence”; use more general term “entity” where meaning is obvious
- identify each entity type by name and list of properties
- examples of entity sets: Staff, Branch, PropertyForRent, and PrivateOwner

Diagram of Staff and Branch entity sets



Relationship

- set of meaningful associations among entity sets

Relationship occurrence

- uniquely identifiable association that includes one occurrence from each participating entity set

- relationship called *Has*, which represents association between Branch and Staff entities, that is Branch Has Staff
- each occurrence of *Has* relationship associates one Branch entity occurrence with one Staff entity occurrence

occurrences of *Has* relationship

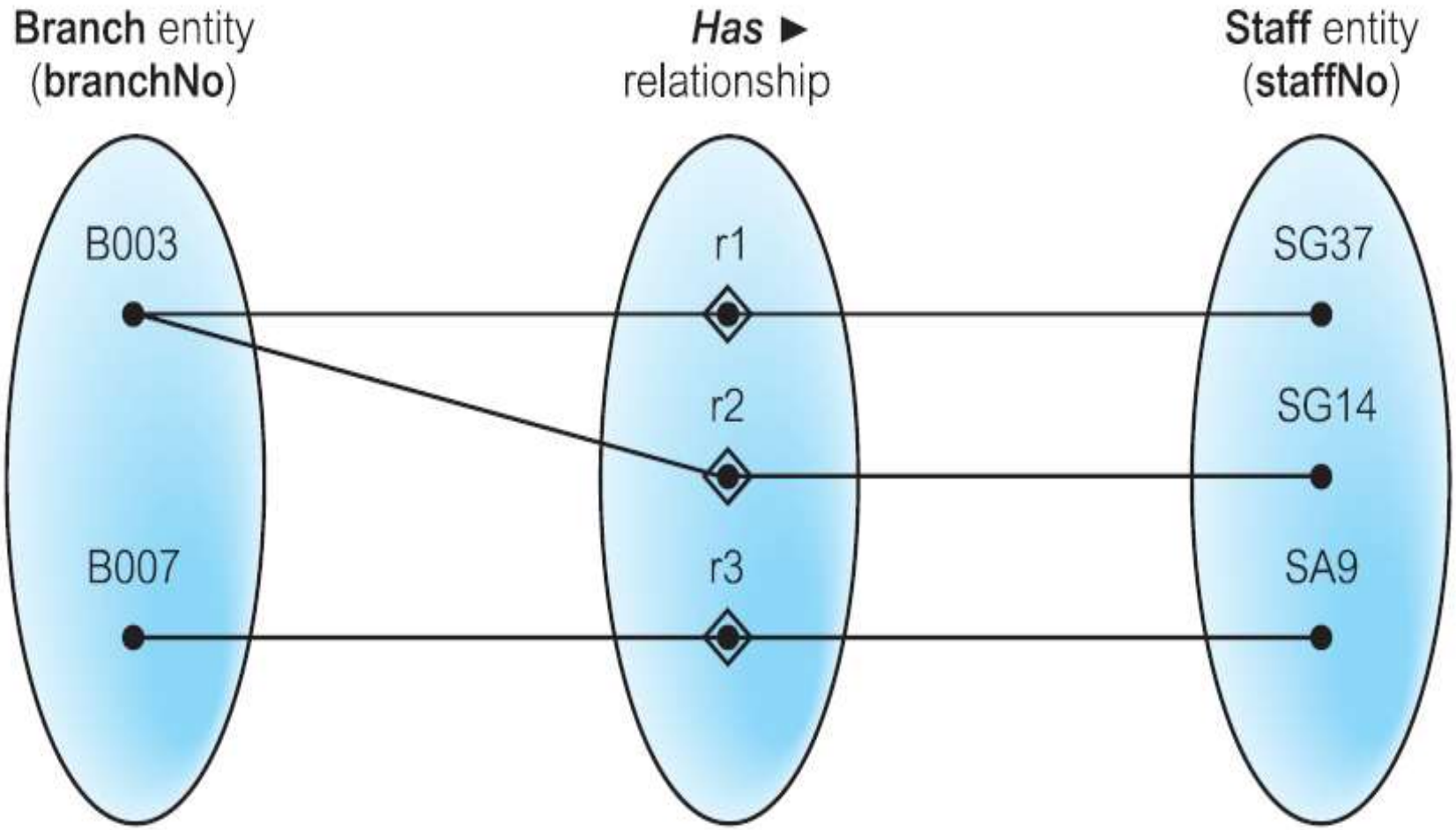
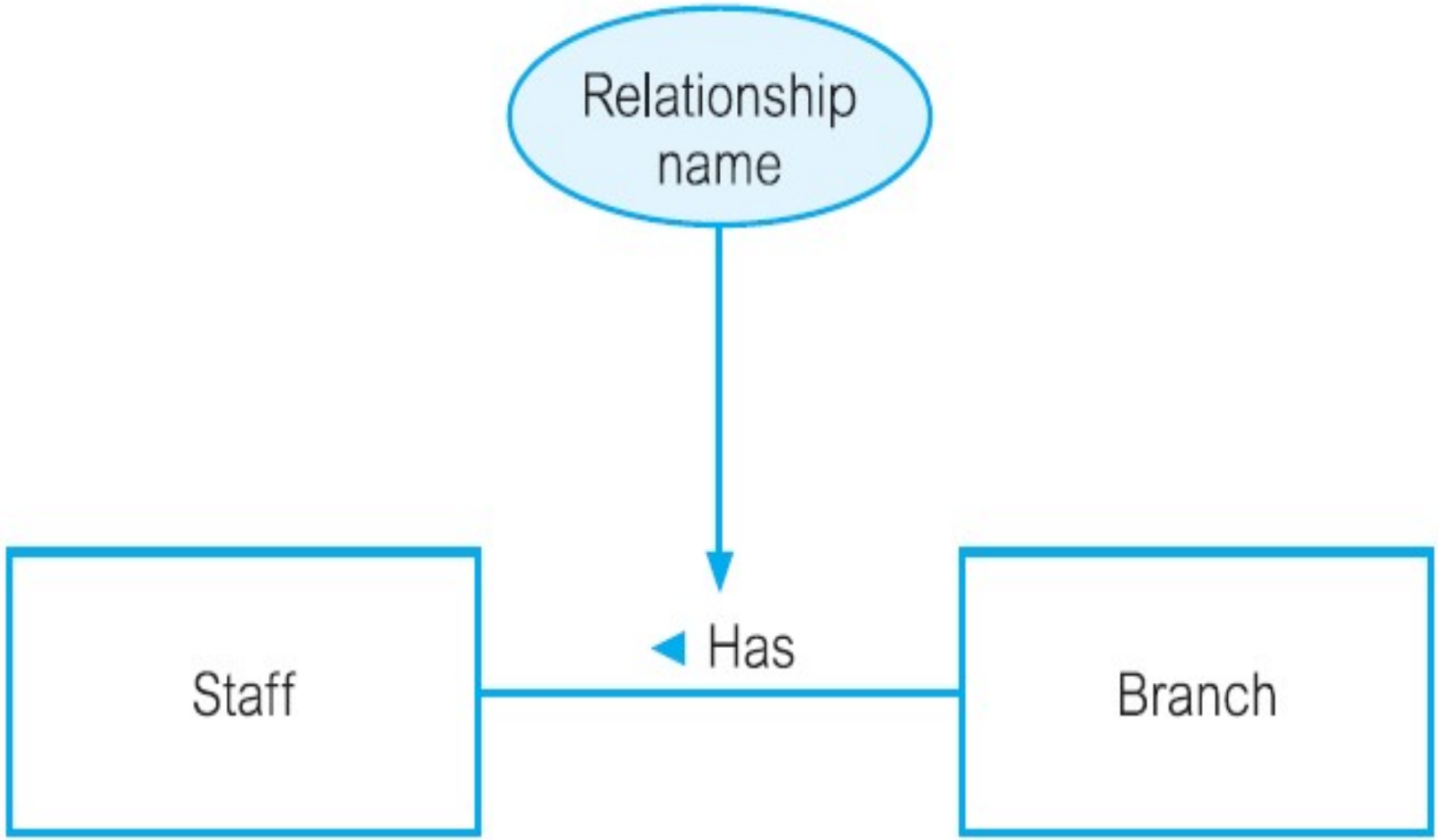


Diagram of Branch *Has* Staff relationship



- each entity set is shown as rectangle, labeled with name of entity, which is normally singular noun
- relationship is named using verb
- entity name should be unique, whenever possible, relationship name should also be unique for given ER model
- relationship is labeled in one direction; name of relationship only makes sense in one direction

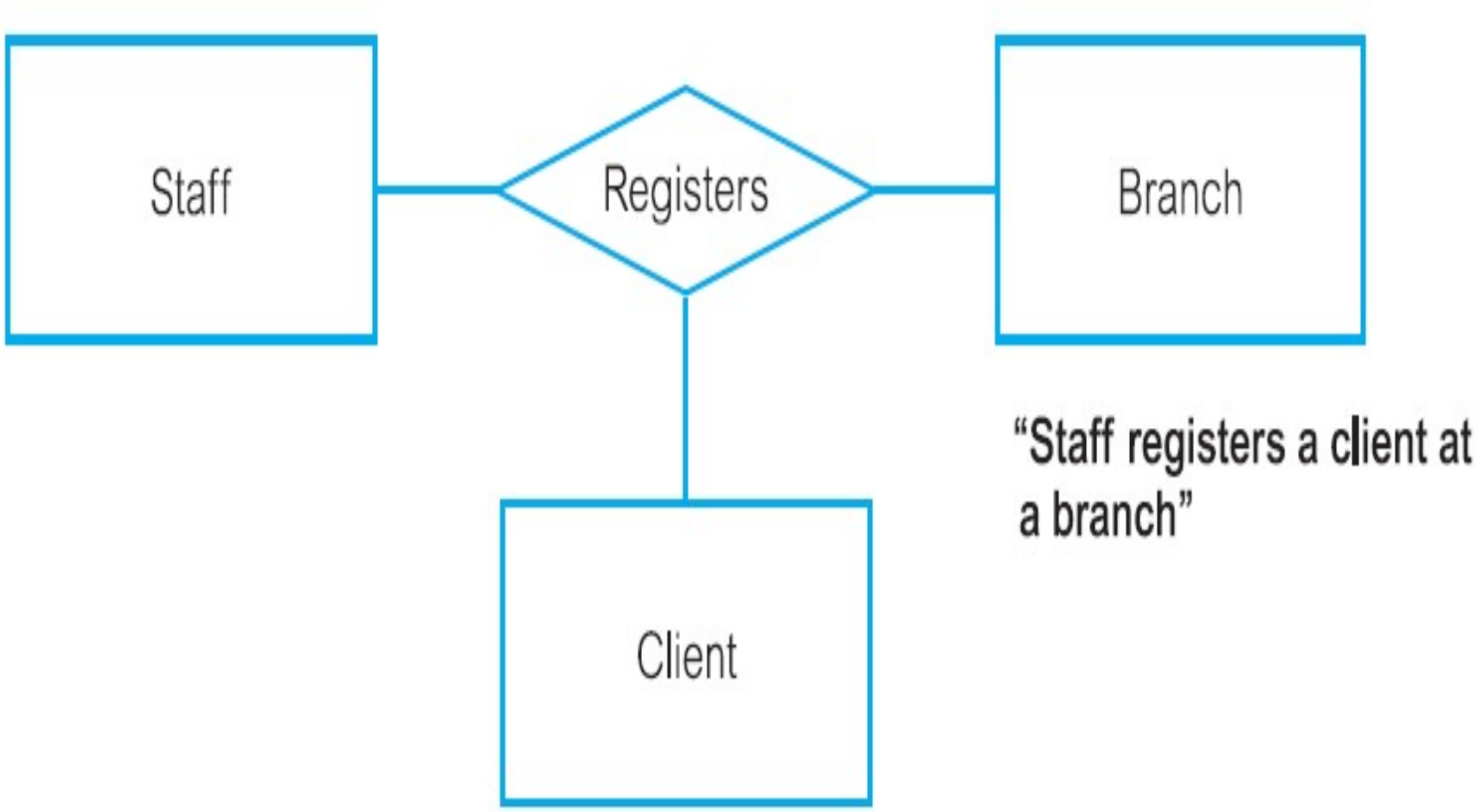
Degree of Relationship

- number of participating entity sets in relationship

- entities involved in relationship are referred to as ***participants***
- number of participants is called ***degree***
- relationship of degree two is called *binary* most common degree

- term “complex relationship” used to describe relationships with degrees higher than binary
- uses diamond to represent relationships with degrees higher than binary; name of relationship is displayed inside

example of ternary relationship called *Registers*

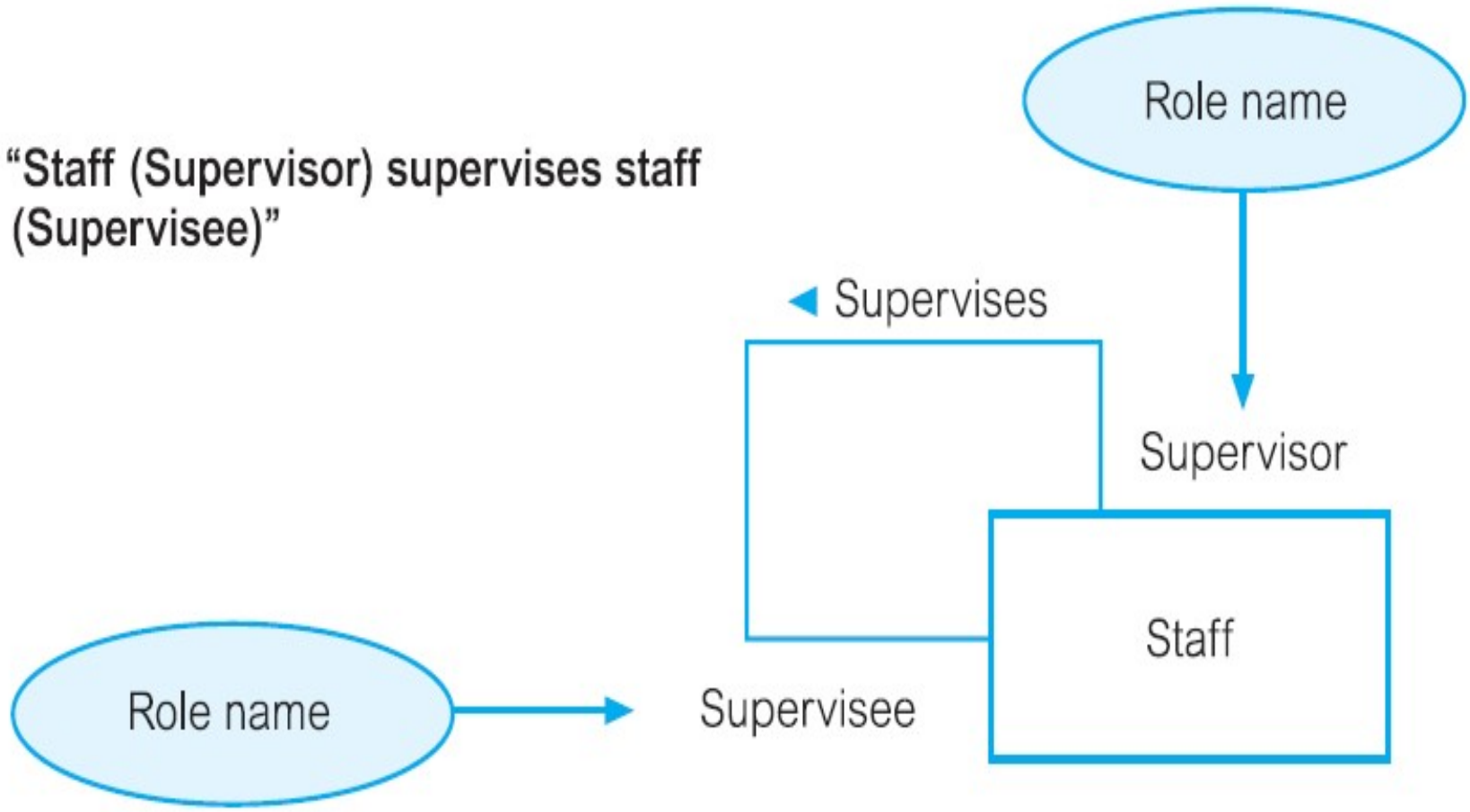


Recursive Relationship

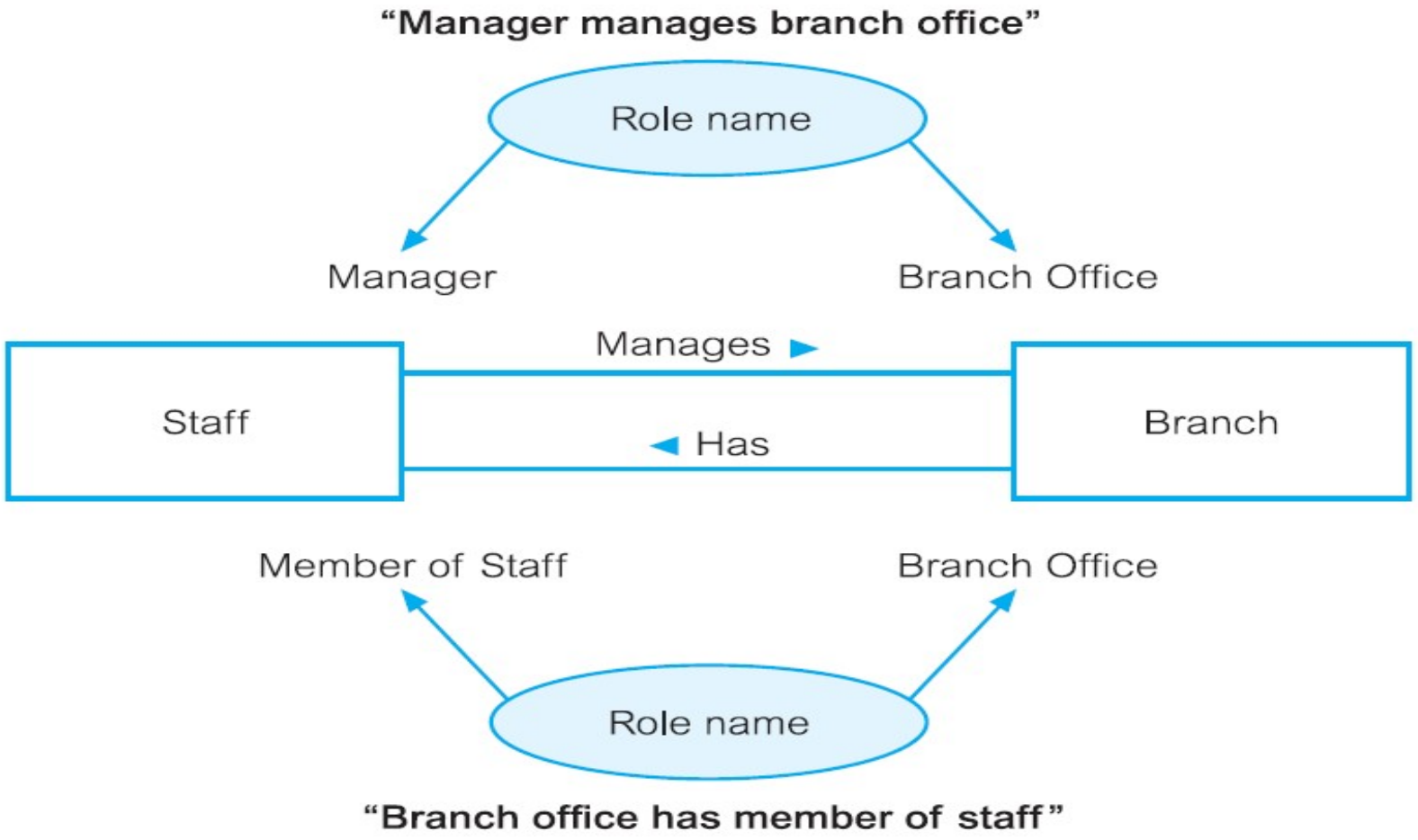
- relationship in which *same* entity type participates more than once in *different roles*

- role names important for recursive relationships to determine function of each participant; usually not required if function of participating entities is unambiguous

example of recursive relationship *Supervises*



example of two relationships with role names



Attribute

- property of an entity or relationship

Attribute domain

- set of allowable values for one or more attributes

Derived attributes

- attribute that represents value that is derivable from value of related attribute or set of attributes, not necessarily in same entity set

Candidate Key

- minimal set of attributes that uniquely identifies each occurrence of an entity set

Primary Key

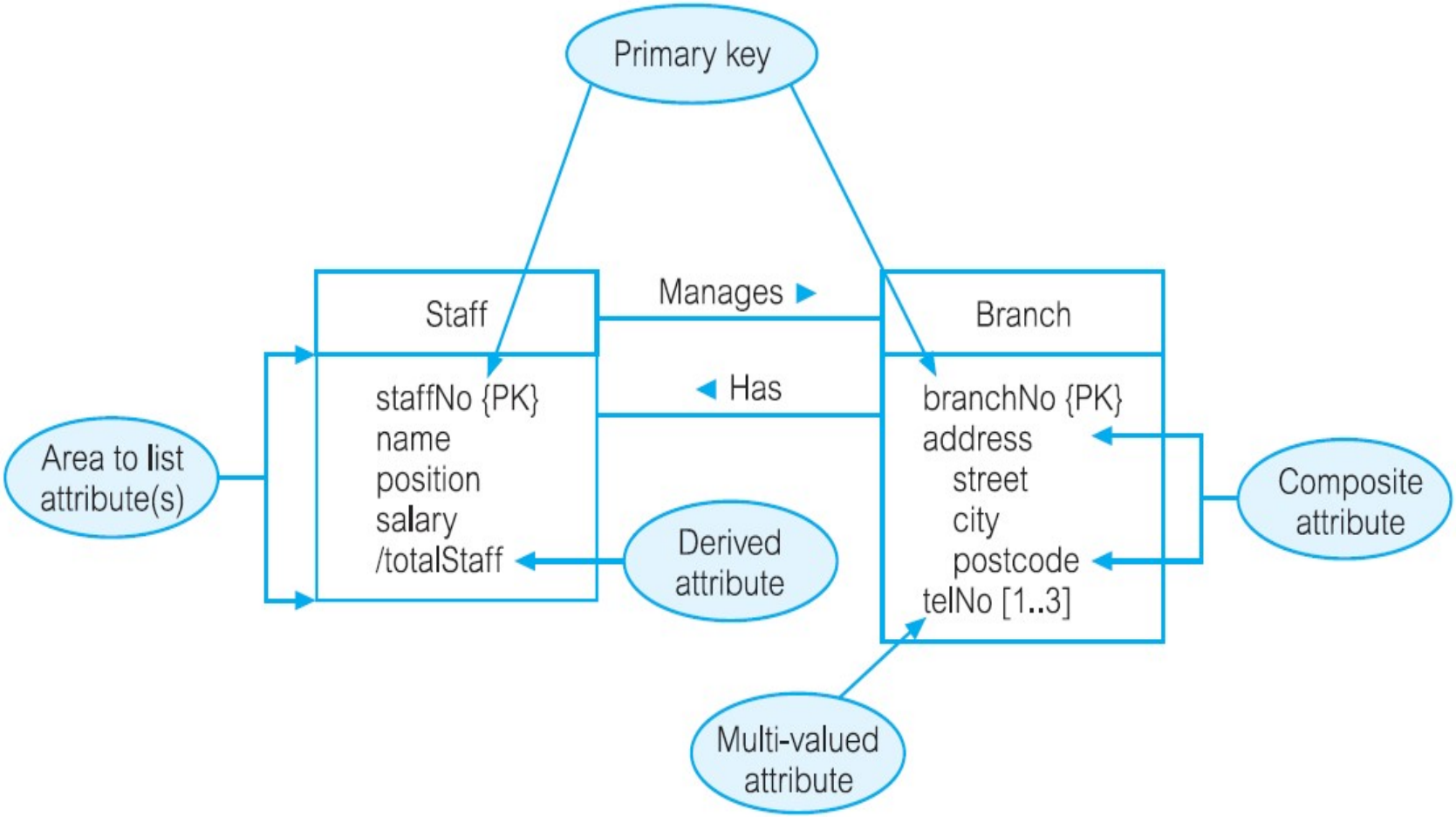
- candidate key that is selected to uniquely identify each occurrence of an entity set

Alternate Key

- consider that member of staff has unique company-defined staff number (staffNo) and also unique National Insurance Number (NIN) used by government
- choice of primary key for entity is based on considerations of attribute length, minimal number of attributes required, and future certainty of uniqueness
- other referred to as **alternate key**

- divide rectangle representing entity
- upper part displays name of entity
- lower part lists names of attributes
- first attribute(s) to be listed is primary key labeled with tag {PK}
- additional tags that can be used include alternate key {AK}

example shows ER diagram for Staff Branch entities and attributes



- can classify entity sets as being strong or weak

Strong entity set

- entity set that is *not* existence-dependent on some other entity set

Strong entity set

- if its existence does not depend upon existence of another entity type
- Staff, Branch, PropertyForRent, and Client
- each entity occurrence is uniquely identifiable using primary key attribute(s) of that entity set
- for example, we can uniquely identify each member of staff using *staffNo* attribute

Weak entity set

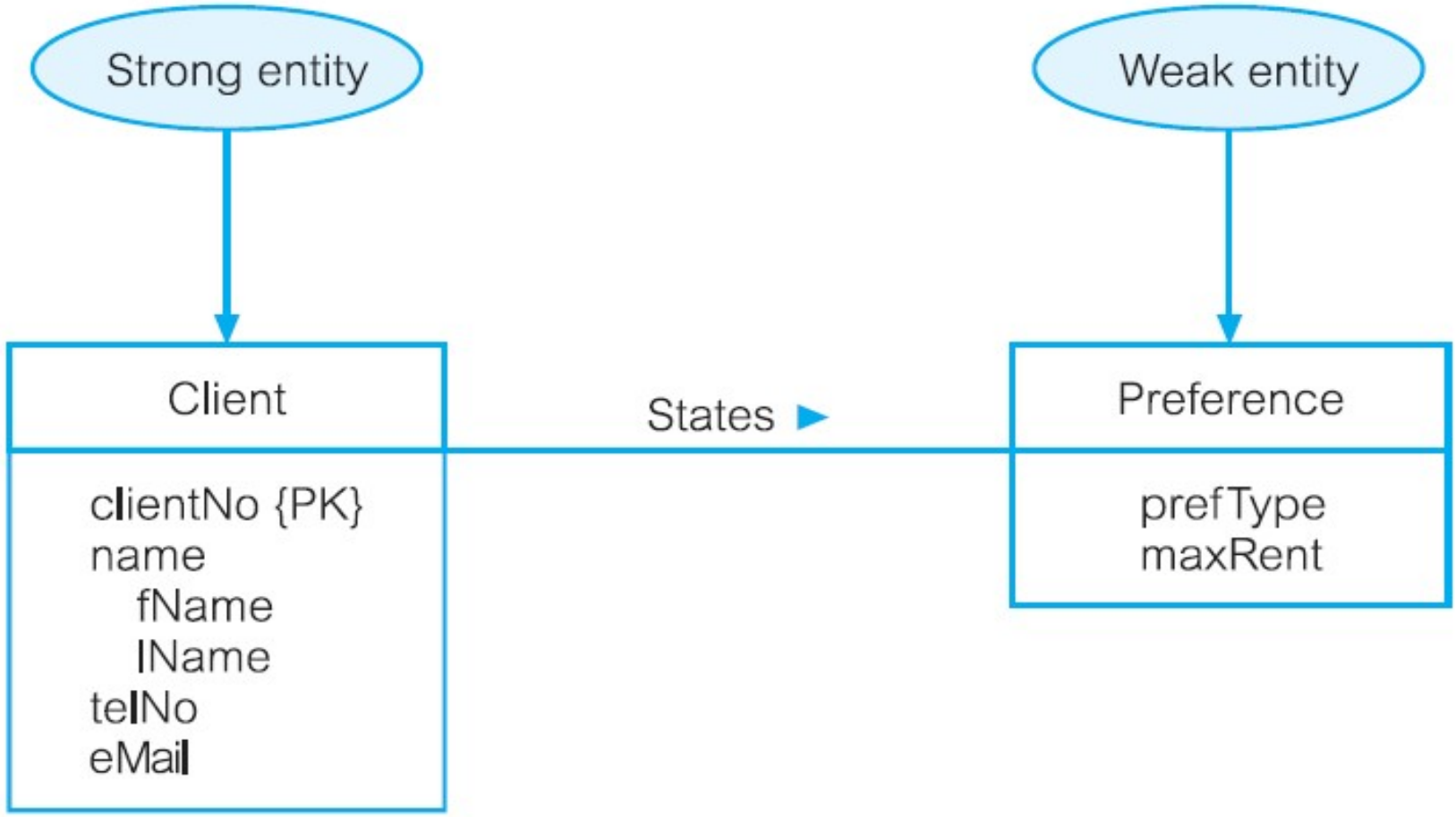
- entity set that is existence-dependent on some other entity set

Weak entity set

- dependent on existence of another entity set
- each entity occurrence cannot be uniquely identified using only attributes associated with that entity set
- there is no primary key for *Preference* entity
- can uniquely identify each *Preference* only through relationship that preference has with client uniquely identifiable using primary key for *Client* entity set, namely *clientNo*

- *Preference* entity is described as having existence dependency for *Client* entity, which is referred to as being owner entity
- weak entity sets referred to as *child, dependent,* or *subordinate* entities
- strong entity types as *parent, owner,* or *dominant* entities

strong entity set *Client*
weak entity set *Preference*



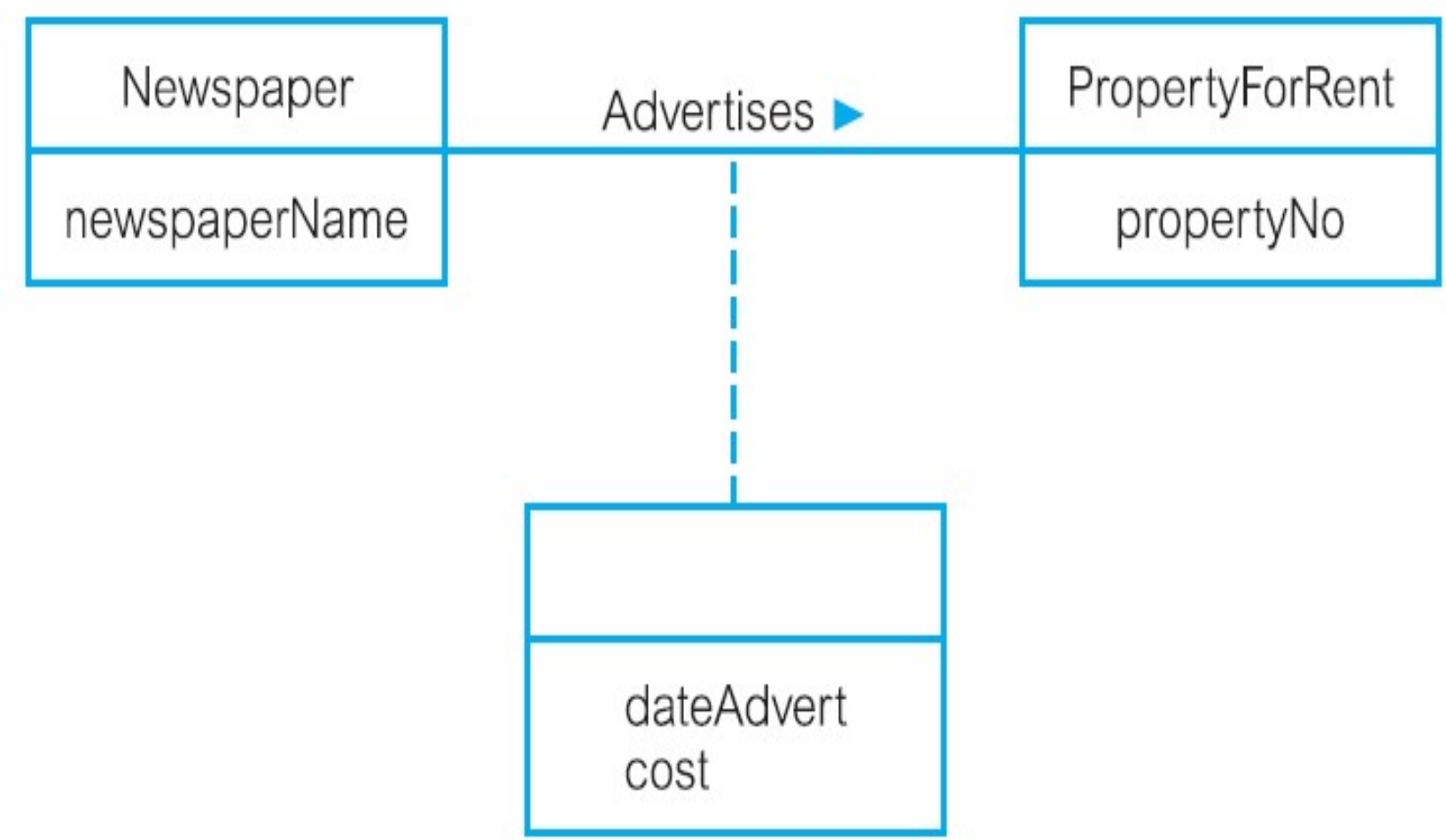
Attributes on Relationships

- attributes can also be assigned to relationships
- for example, consider relationship *Advertises*, which associates *Newspaper* and *PropertyForRent* entity sets
- to record date property was advertised and cost, we associate this information with *Advertises* relationship as attributes

Attributes on Relationships

- represent attributes associated with relationship using same symbol as an entity set; rectangle representing the attribute(s) is associated with relationship using dashed line

example of relationship *Advertises* with attributes *dateAdvert* and *cost*



Cardinality

- number (or range) of possible occurrences of an entity set that may relate to single occurrence of associated entity set through particular relationship
- describes maximum number of possible relationship occurrences for entity participating in given relationship
- main type of structural constraint on relationships is called ***cardinality***

Cardinality

- constrains way that entities are related
- representation of business rules established by enterprise
- binary relationships are generally referred to as being one-to-one (1:1), one-to-many (1:m), or many-to-many (m:n)

Cardinality

- (1:1) - a member of staff manages a (single) branch
- (1:m) - a member of staff oversees (many) properties for rent
- (m:n) – (many) newspapers advertise (many) properties for rent

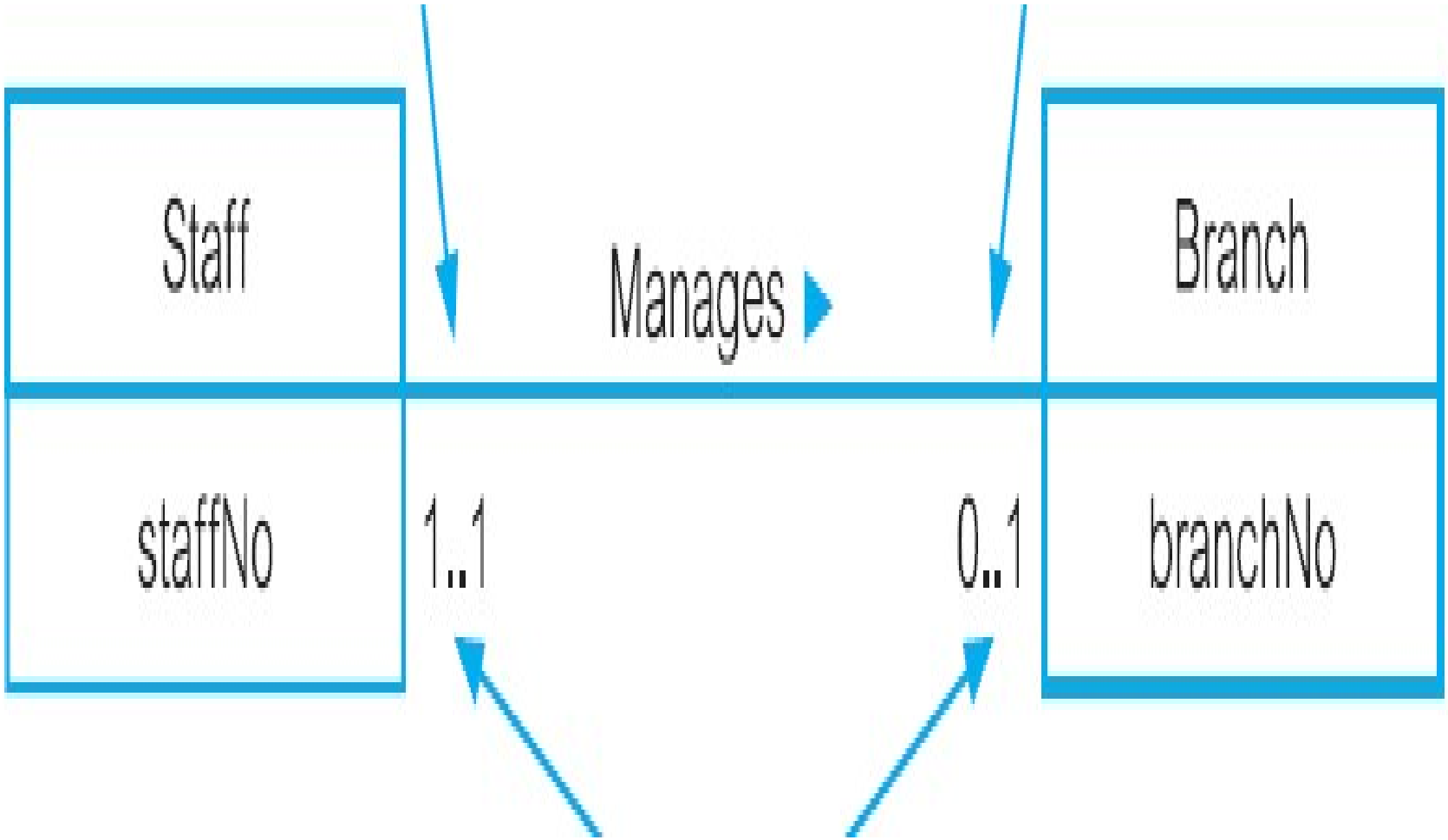
One-to-One (1:1) Relationships

- relationship *Manages*, which relates *Staff* and *Branch*
- represent each entity occurrence using values for primary key attributes of *Staff* and *Branch* entities, namely *staffNo* and *branchNo*

One-to-One (1:1) Relationships

- member of staff can manage *zero or one* branch and each branch is managed by *one* member of staff
- there is maximum of one branch for each member of staff involved in relationship and maximum of one member of staff for each branch

Cardinality of Staff *Manages* Branch - one-to-one (1:1)



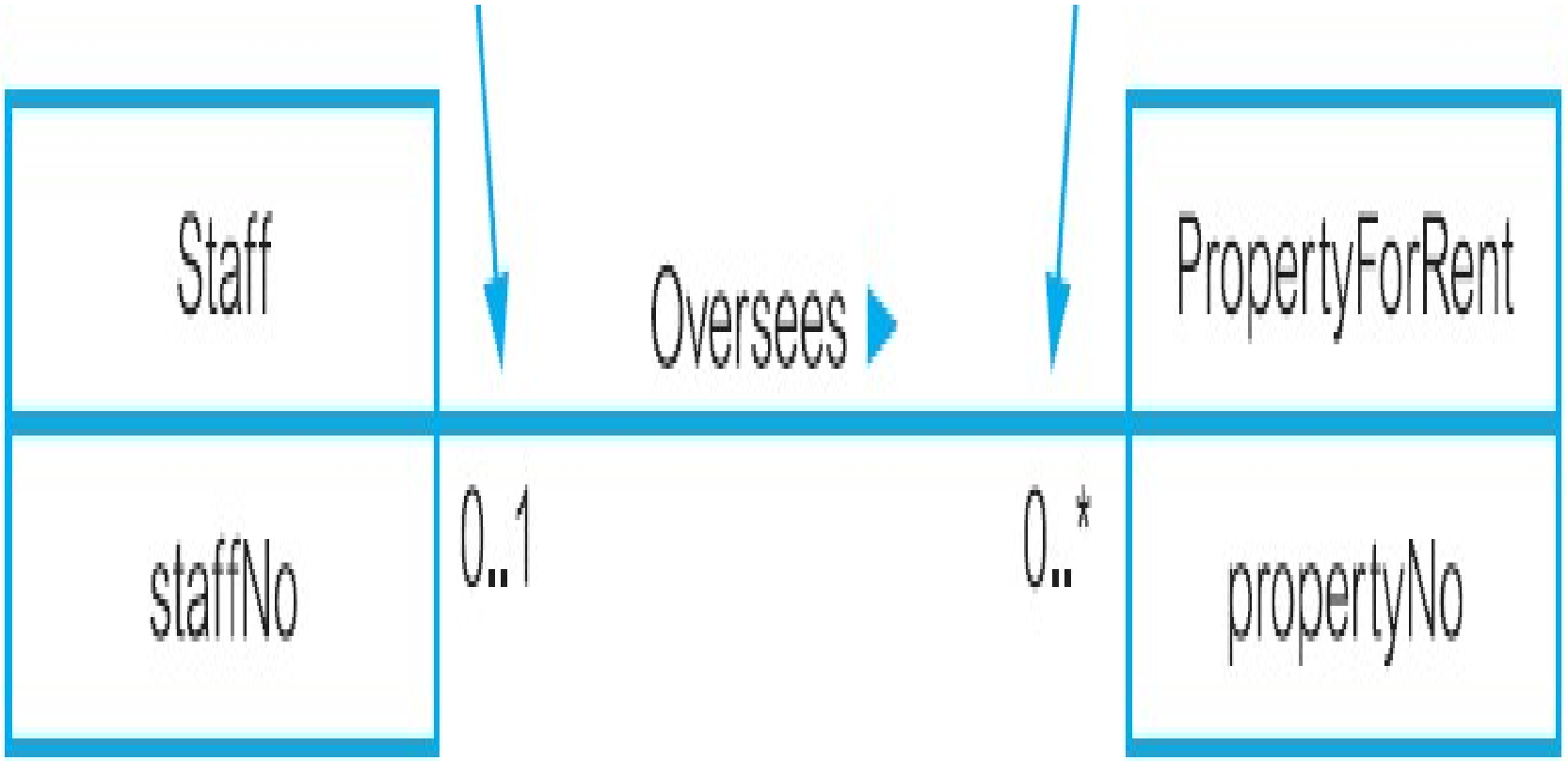
One-to-Many (1:m) Relationships

- relationship *Oversees*, which relates *Staff* and *PropertyForRent* entity sets
- represent each entity occurrence using values for primary key attributes of *Staff* and *PropertyForRent* entities, namely *staffNo* and *propertyNo*

One-to-Many (1:m) Relationships

- a member of staff can oversee *zero or more* properties for rent and property for rent is overseen by *zero or one* member of staff
- for members of staff participating in this relationship there are *many* properties for rent, and for properties participating in this relationship there is maximum of *one* member of staff

One-to-Many (1:m) Relationships



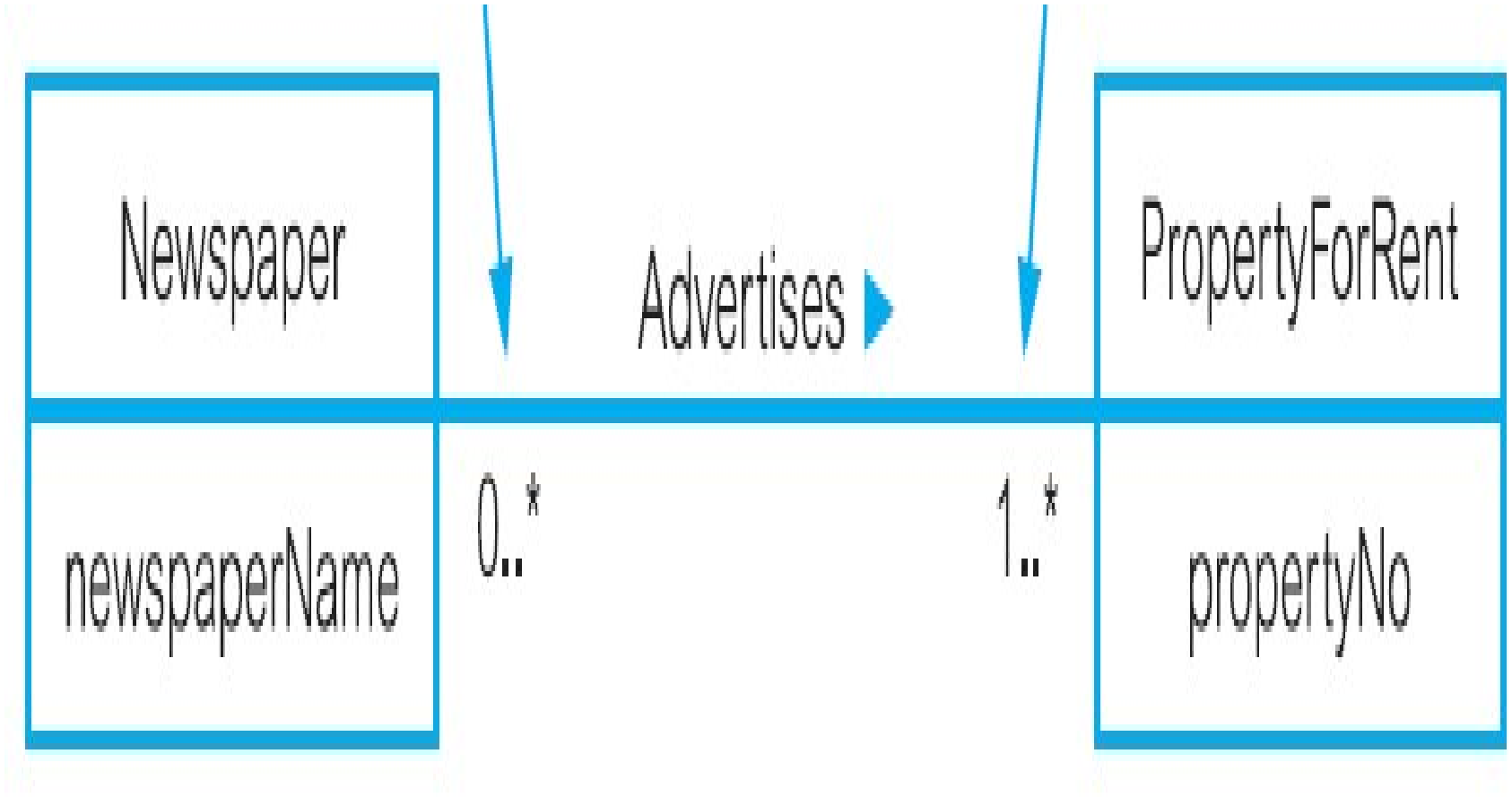
Many-to-Many (m:n) Relationships

- relationship *Advertises* relates *Newspaper* and *PropertyForRent* entity sets
- represent each entity occurrence using values for primary key attributes of *Newspaper* and *PropertyForRent* entity sets, namely *newspaperName* and *propertyNo*

Many-to-Many (m:n) Relationships

- *one* newspaper advertises *one or more* properties for rent and *one* property for rent is advertised in *zero or more* newspapers
- there are *many* properties for rent, and for each property for rent participating in this relationship there are *many* newspapers

Many-to-Many (m:n) Relationships



Participation

- determines whether all or only some entity occurrences participate in relationship

- participation constraint represents whether all entity occurrences are involved in particular relationship (referred to as ***mandatory*** participation) or only some (referred to as ***optional*** participation)

Participation

- optional participation is represented as minimum value of 0
- mandatory participation is shown as minimum value of 1
- participation for given entity in relationship is represented by minimum value on *opposite* side of relationship

Participation

- optional participation for *Staff* entity in *Manages* relationship is shown as minimum value of 0 for cardinality beside *Branch* entity
- mandatory participation for *Branch* entity in *Manages* relationship is shown as minimum value of 1 for cardinality beside *Staff* entity

Problems: connection traps

- occur due to misinterpretation of meaning of certain relationships: *fan traps* and *chasm traps*
- to identify connection traps we must ensure that meaning of relationship is fully understood and clearly defined
- if we do not understand relationships we may create model that is not true representation of “real world”

Fan Trap

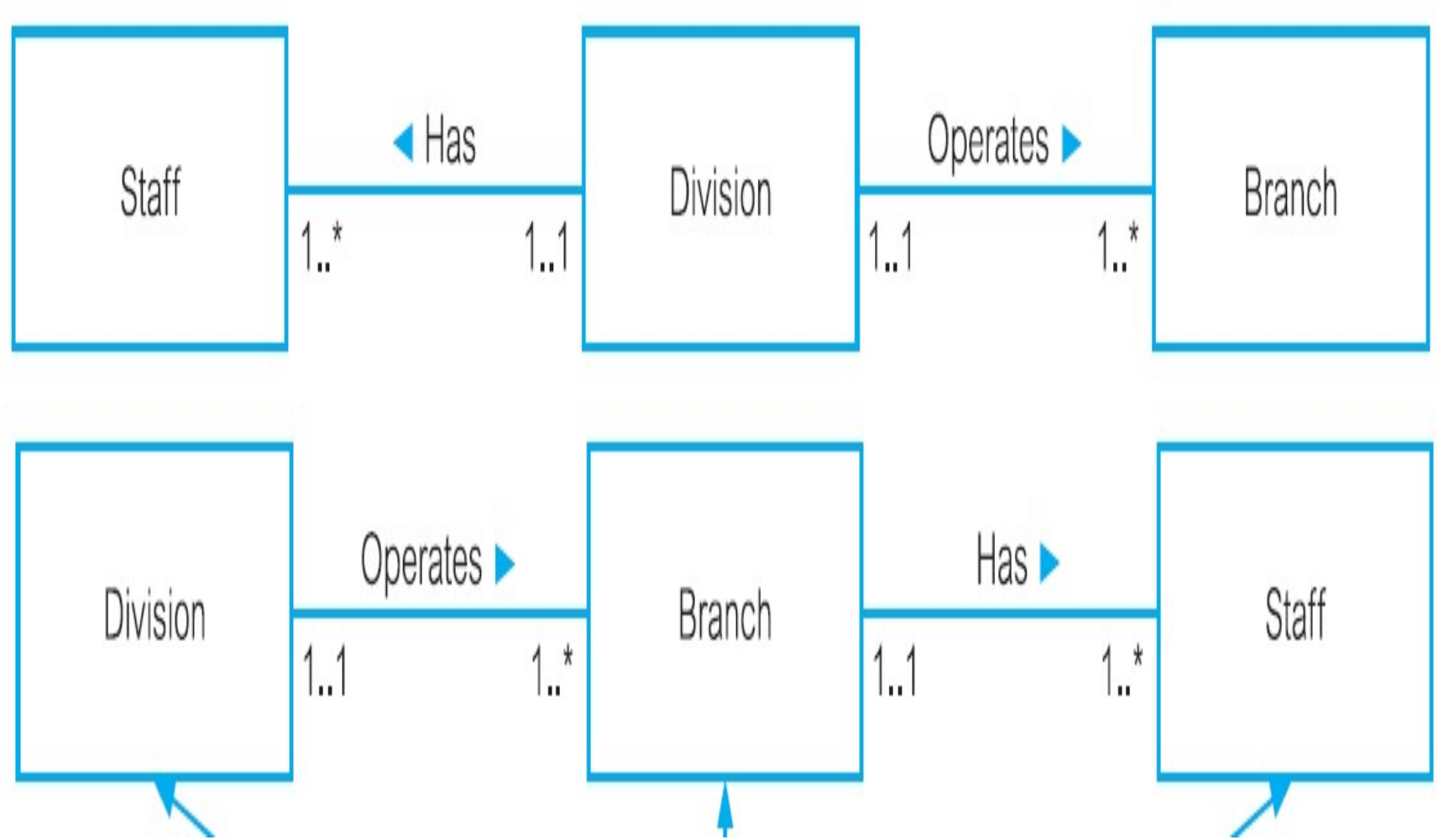
- model represents relationship between entity sets, but pathway between certain entity occurrences is ambiguous

- exist where two or more 1:m relationships fan out from same entity

Fan Trap

- relationships (1:m) *Has* and *Operates* emanating from same entity *Division*
- a single division operates *one or more* branches and has *one or more* staff
- problem arises when we want to know which members of staff work at particular branch

Example of Fan Trap



Chasm Trap

- model suggests existence of relationship between entity sets, but pathway does not exist between certain entity occurrences

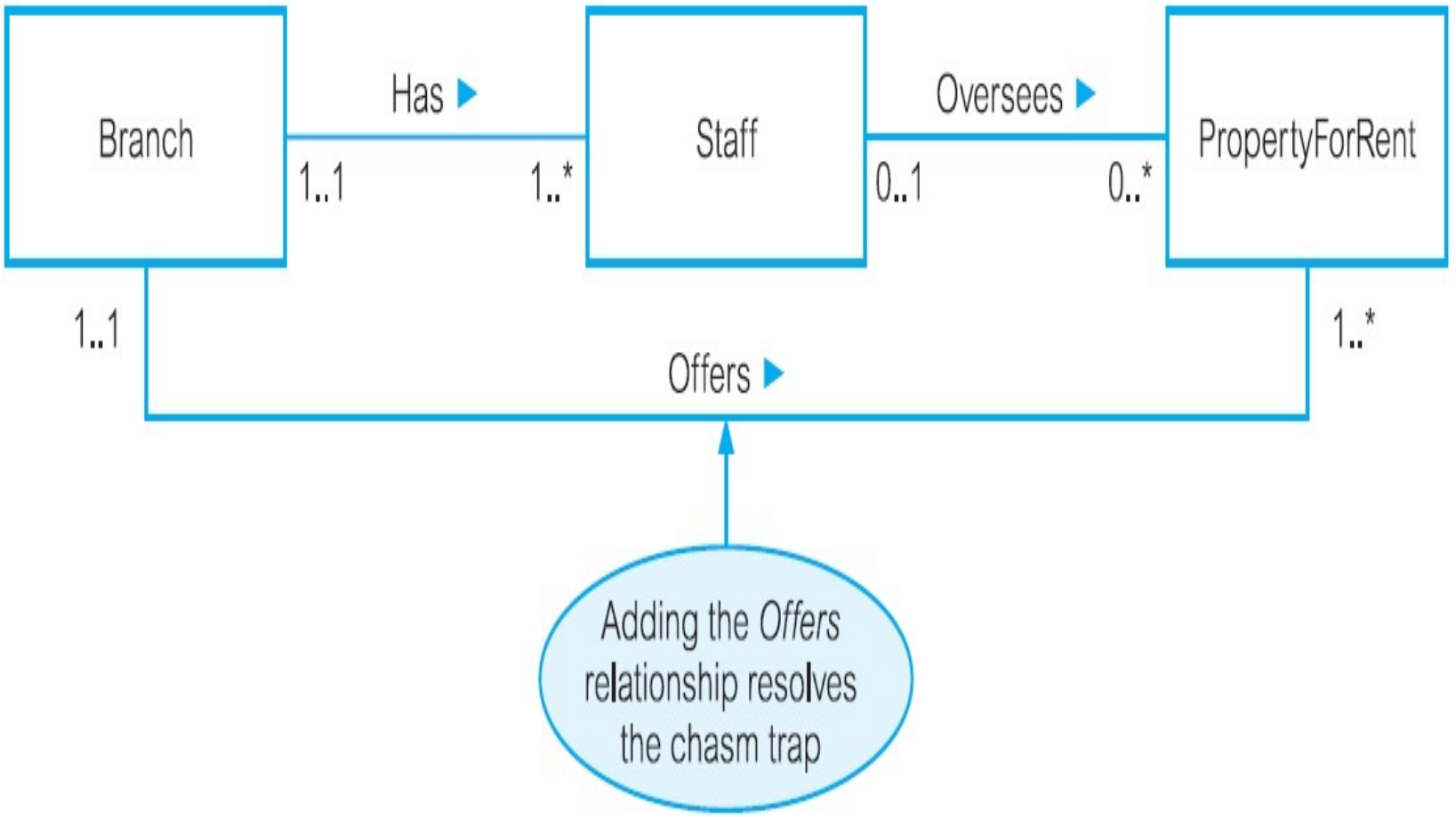
Chasm Trap

- occur where there are one or more relationships with minimum multiplicity of zero (that is, optional participation) forming part of pathway between related entities

Chasm Trap

- potential chasm trap illustrated in relationships between *Branch*, *Staff*, and *PropertyForRent* entities; model represents facts that single branch has *one or more* staff who oversee *zero or more* properties for rent; not all staff oversee property and not all properties are overseen
- problem arises when we want to know which properties are available at each branch

Example of Chasm Trap



SPECIALIZATION / GENERALIZATION

Specialization/Generalization

- associated with special types of entities known as superclasses and subclasses, and process of attribute inheritance
- hierarchy of entities containing superclasses and subclasses

Superclass

- entity set that includes one or more distinct subgroupings of its occurrences, which must be represented in data model

Subclass

- distinct subgrouping of occurrences of entity set, which must be represented in data model

for example

- entities that are members of *Staff* entity set may be classified as *Manager*, *SalesPersonnel*, and *Secretary*
- *Staff* entity is referred to as superclass of
- *Manager*, *SalesPersonnel*, and *Secretary* subclasses
- superclass/subclass relationship

Superclass/Subclass Relationships

- each member of subclass is also member of superclass; but has distinct role
- relationship between superclass and subclass is one-to-one (1:1)

Superclass/Subclass Relationships

- some superclasses may contain overlapping subclasses (illustrated by member of *Staff* who is both *Manager* and *Sales Personnel*)
- not every member of superclass is necessarily member of subclass (members of *Staff* without distinct job role such as *Manager* or *Sales Personnel*)

Possible (but not indicated)

- All Up – use superclass without any subclasses
- All Down – use many subclasses without superclass

- use superclasses and subclasses to avoid describing different types of *Staff* with possibly different attributes within single entity

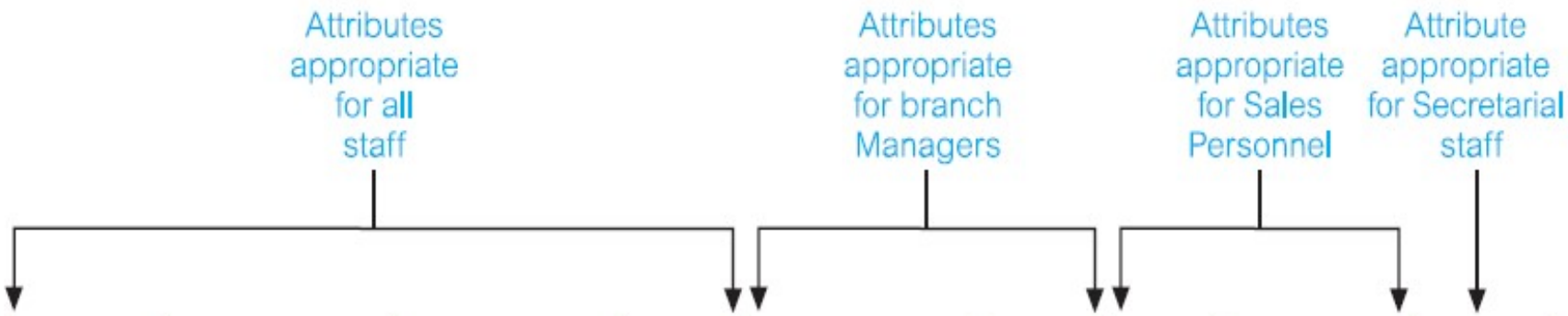
- avoids describing similar concepts more than once - saving time for designer
- adds more semantic information to design in form that is familiar to many
- "*Manager* IS-A member of *Staff*" - communicates significant semantic content in concise form

- for example, *Sales Personnel* may have special attributes such as *salesArea* and *carAllowance*
- if all are described by single *Staff* entity, this may result in a lot of nulls for job-specific attributes

- can also show relationships that are associated with only particular types of staff (subclasses) and not with staff, in general;
- for example, *Sales Personnel* may have distinct relationships that are not appropriate for all *Staff*, such as *SalesPersonnel Uses Car*

- *Sales Personnel* have common attributes with other *Staff*, such as *staffNo*, *name*, *position*, and *salary*

relation *AllStaff*



staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary	8500					100
SL31	Robert Chin	Snr Sales Asst	17000			SA2B	3700	
SG5	Susan Brand	Manager	24000	01/06/91	2350			

Attribute Inheritance

- subclass represents same “real world” object as in superclass, may possess subclass-specific attributes, as well as those associated with superclass
- *SalesPersonnel* subclass inherits all attributes of *Staff* superclass - *staffNo*, *name*, *position*, and *salary* together with those specifically associated with *SalesPersonnel* subclass - *salesArea* and *carAllowance*

Hierarchy

- subclass is an entity in its own right and so it may also have one or more subclasses
- Specialization hierarchy (for example, *Manager* is a specialization of *Staff*)
- Generalization hierarchy (for example, *Staff* is a generalization of *Manager*)
- IS-A hierarchy (for example, *Manager* IS-A (member of) *Staff*)

Specialization process

- process of maximizing differences between members of an entity by identifying their distinguishing characteristics

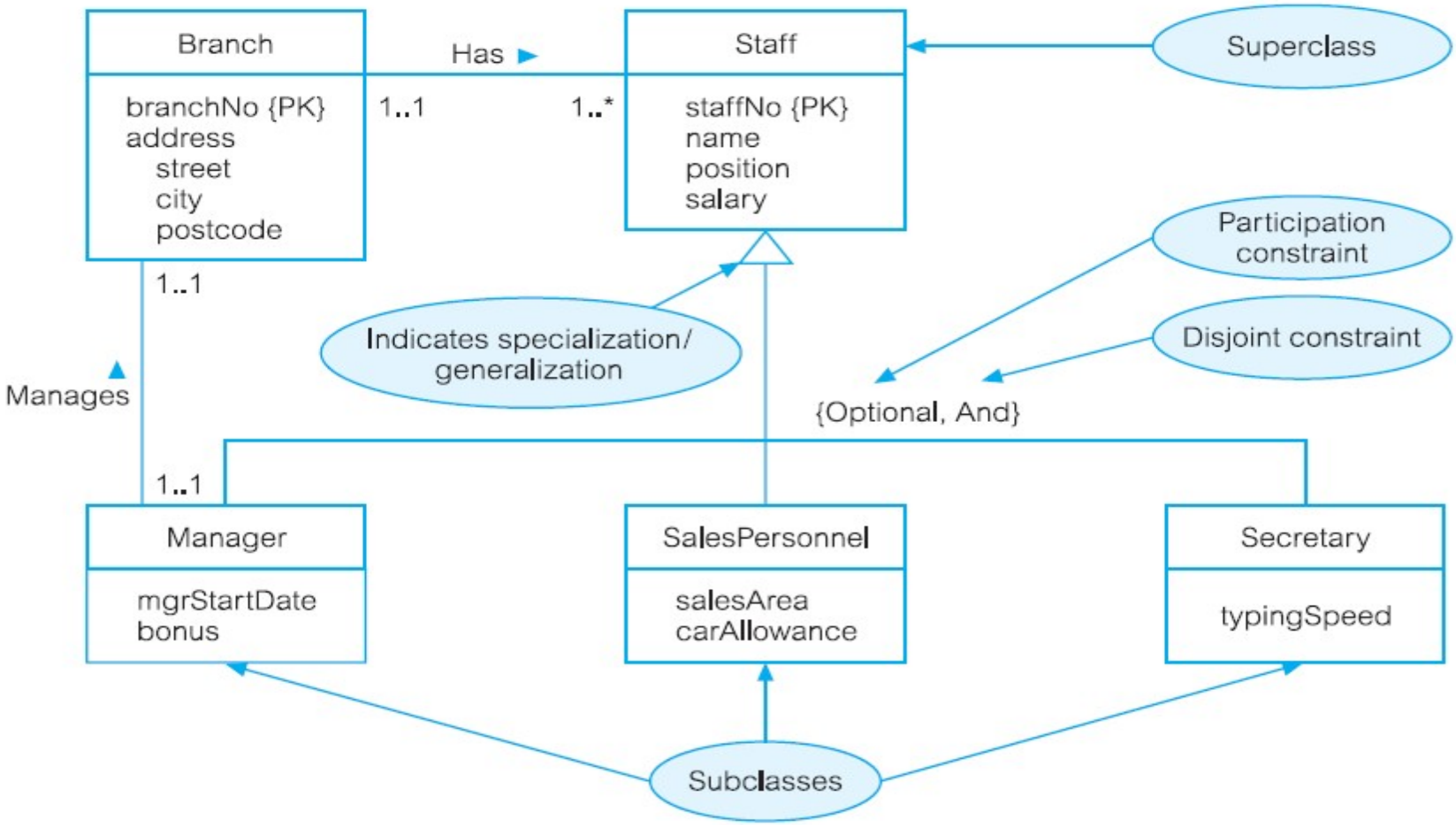
- top-down approach to defining set of superclasses and their related subclasses

Generalization process

- process of minimizing differences between entities by identifying their common characteristics

- bottom-up approach, that results in identification of generalized superclass from original entity types

Specialization/generalization of Staff subclasses representing job roles

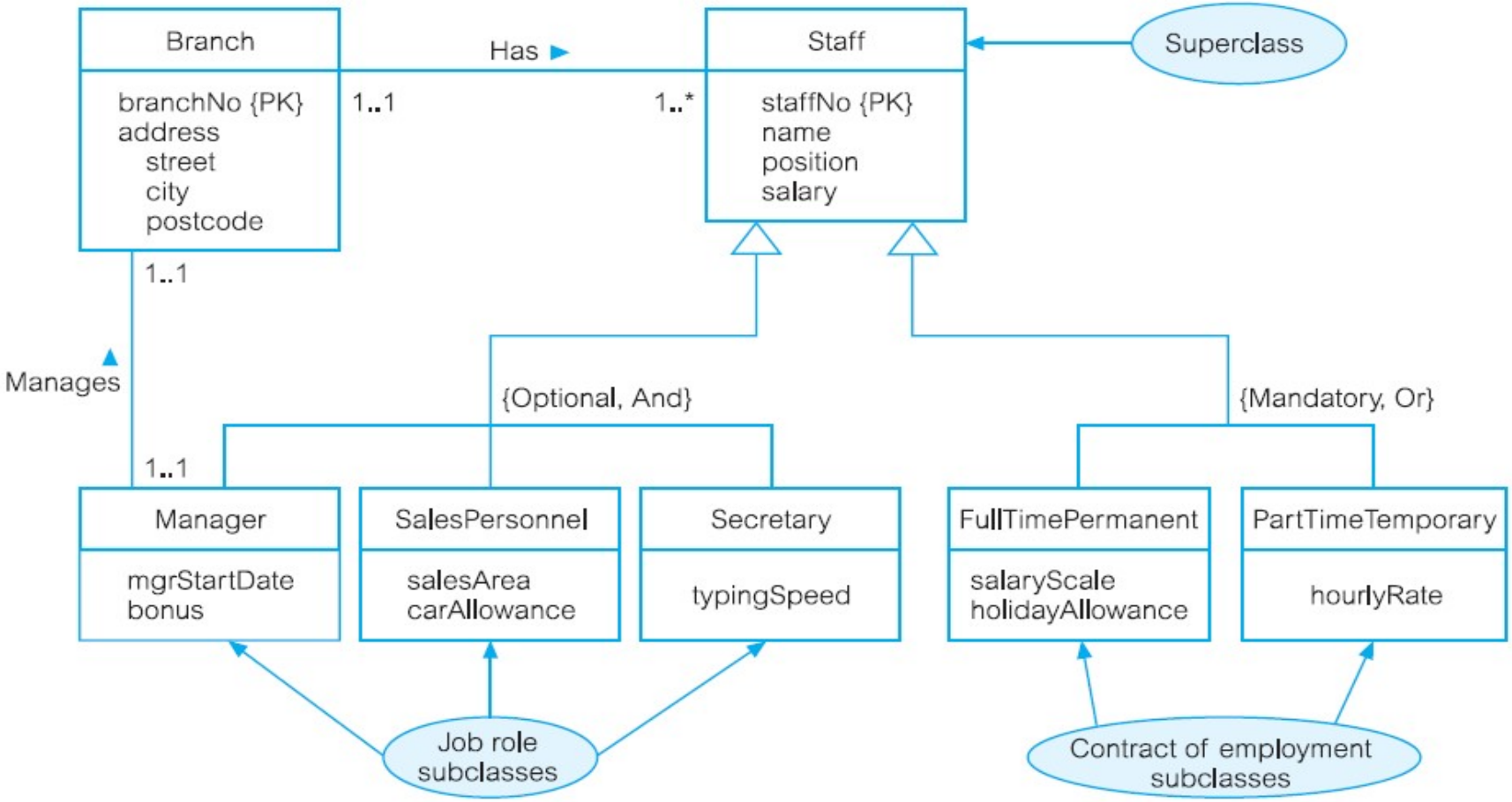


- may have several specializations of same entity based on different distinguishing characteristics

for example

- another specialization of *Staff* entity may produce subclasses *FullTimePermanent* and *PartTimeTemporary*, which distinguishes between types of employment contract for members of staff
- attributes that are specific to *FullTimePermanent* (*salaryScale* and *holidayAllowance*) and *PartTimeTemporary* (*hourlyRate*)

Staff entity: job roles & contracts of employment



Constraints on Specialization/Generalization

- two constraints that may apply to specialization/generalization called **participation constraints** and **disjoint constraints**

Participation constraints

- determines whether every member in superclass must participate as member of subclass

- may be mandatory or optional
- mandatory participation specifies that every member in superclass must also be member of a subclass

Disjoint constraints

- describes relationship between members of subclasses and indicates whether it is possible for member of superclass to be member of one, or more than one, subclass - applies when superclass has more than one subclass
- if subclasses are disjoint, then an entity occurrence can be member of only one of subclasses

Review Questions

- why is ER model considered top-down approach?
- describe four basic components of ER
- provide examples of unary, binary, ternary, relationships
- how basic ER components are represented in ER diagram
- describe what cardinality constraint represents for relationship
- describe how fan and chasm traps can occur

Review Questions

- describe situations that would call for an specialization/generalization relationship
- describe and illustrate using an example process of attribute inheritance