

DataBases

DataBase Design Normalization Process

Objectives

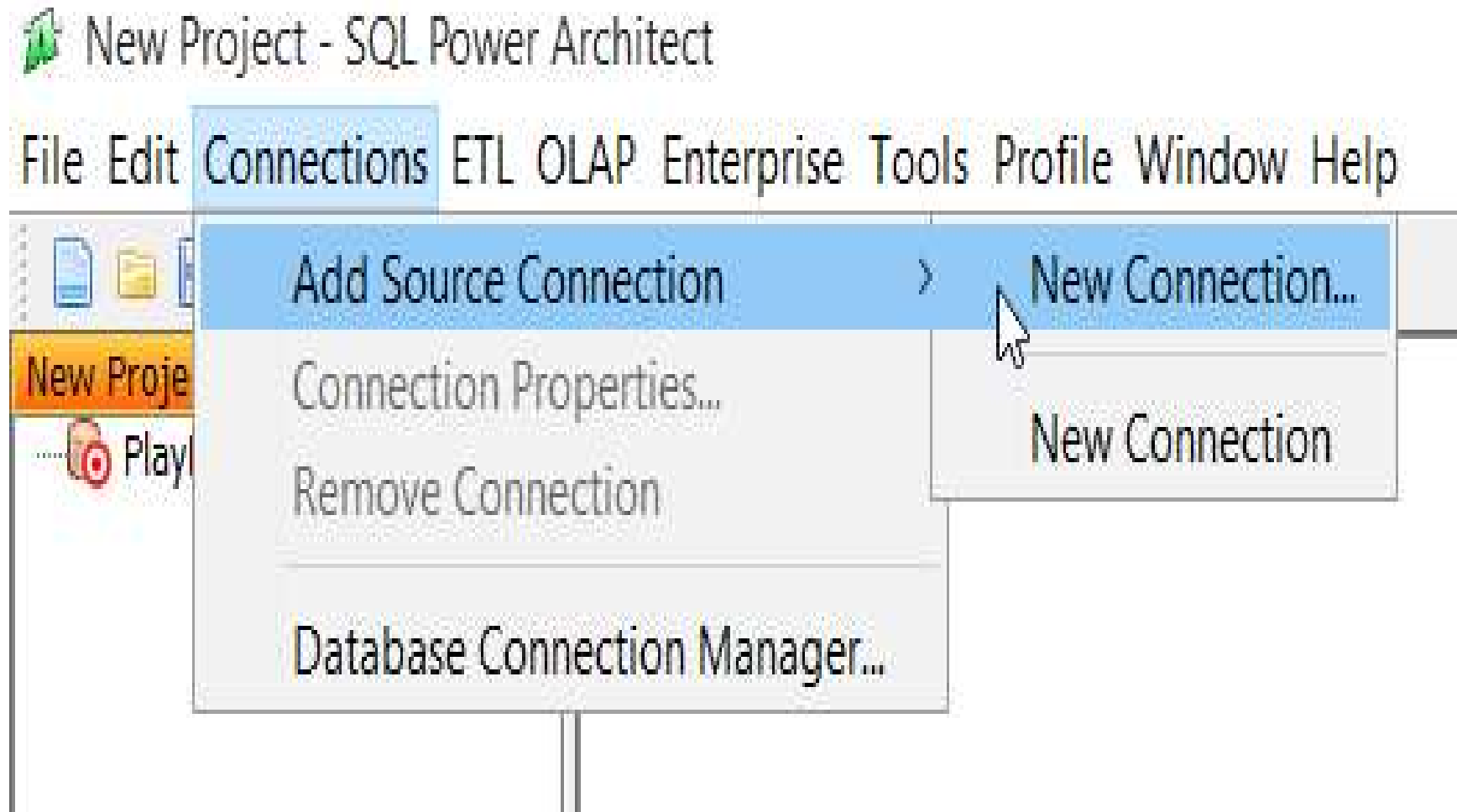
- Which database structures are considered Normal?
- Difference between snapshot and temporal databases
- Store Binary Large Objects in databases

SQL POWER ARCHITECT DATA MODELING

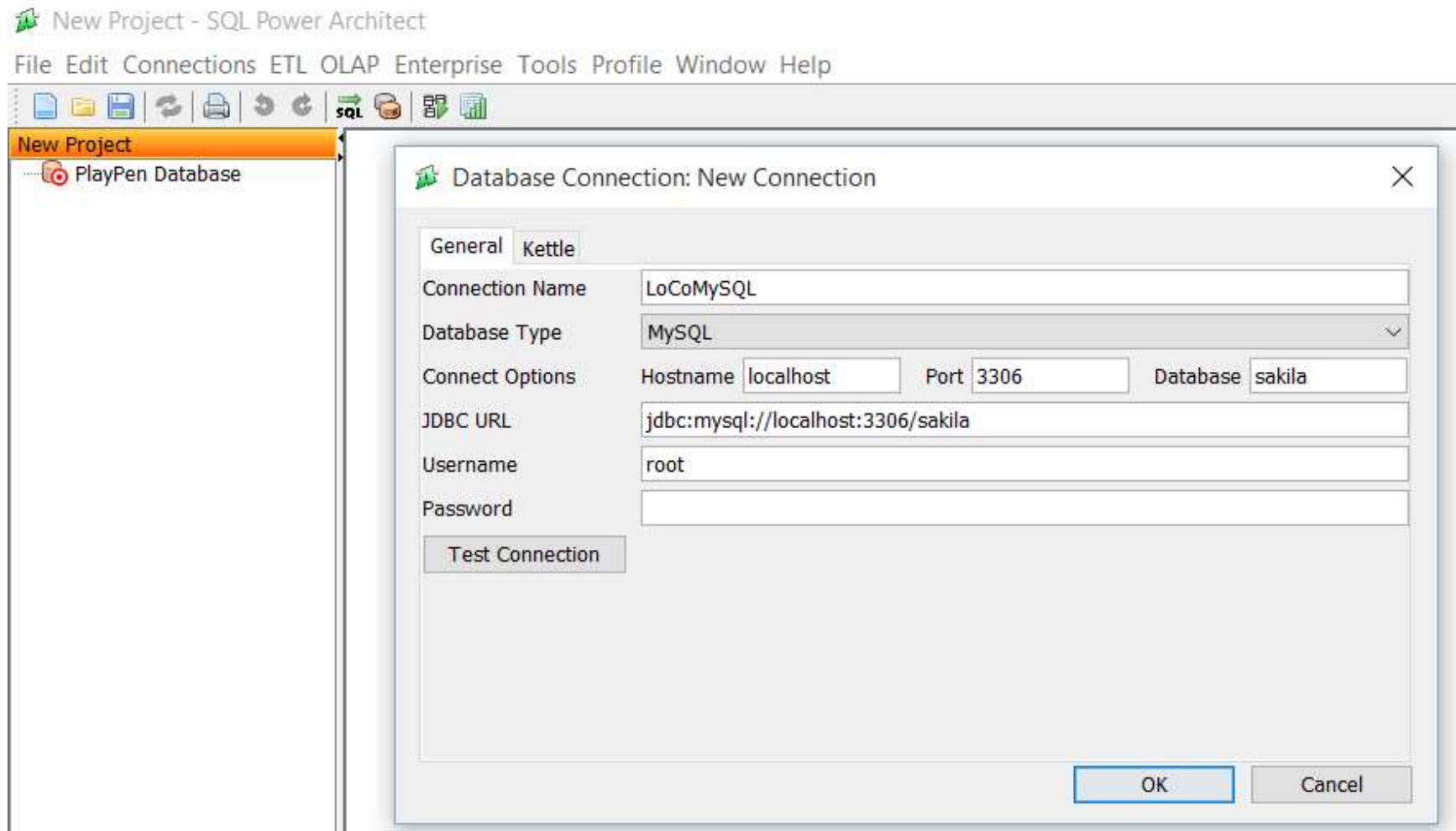
SQL Power Architect – Data Modeling

- <http://www.sqlpower.ca/page/architect>
- free database tool to design data models; forward/reverse engineer to/from any database platform; works with any database using generic framework; runs on every platform: PC, Mac, Unix

SQL Power Architect – Data Modeling



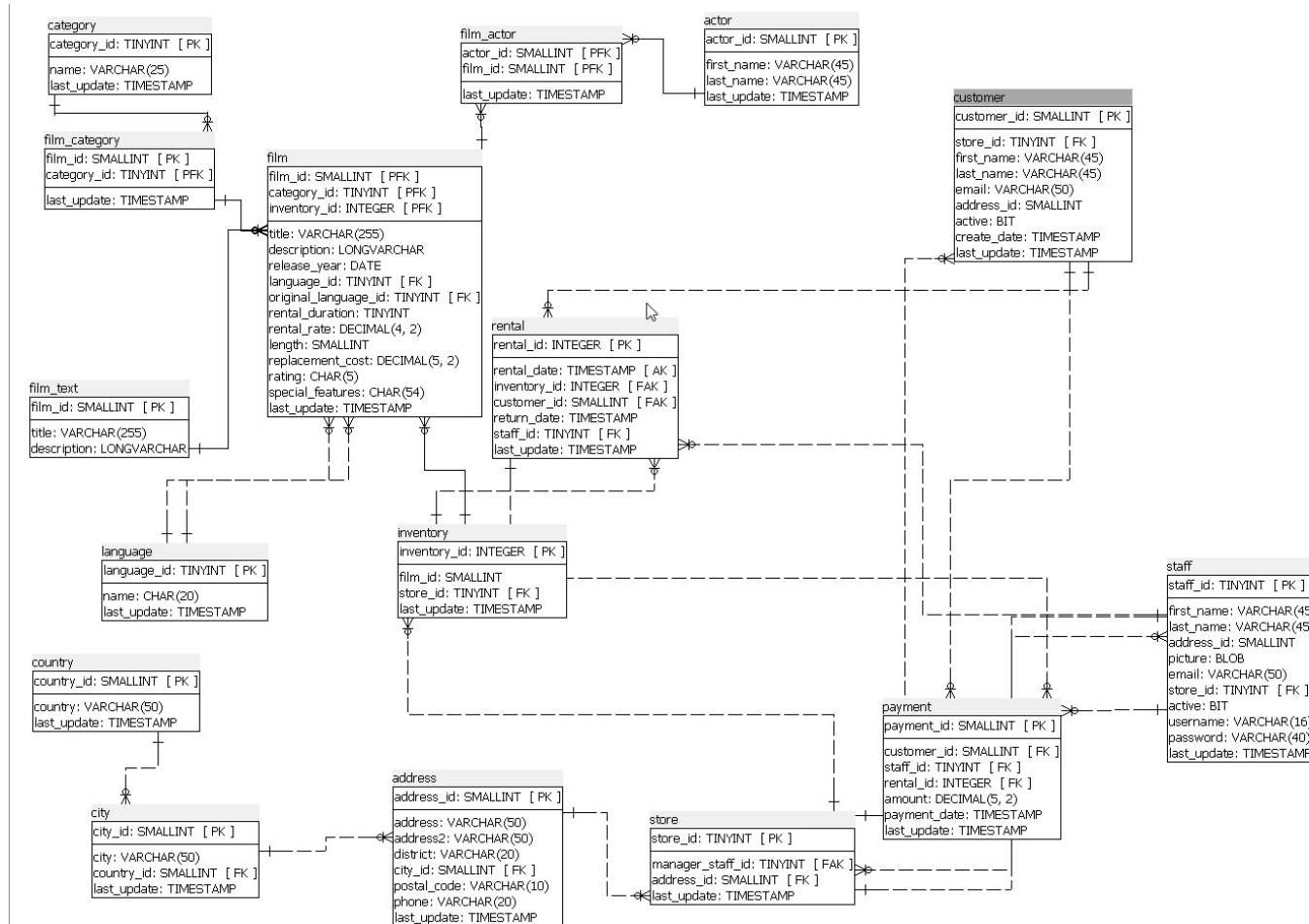
SQL Power Architect – Data Modeling



Reverse Engineering

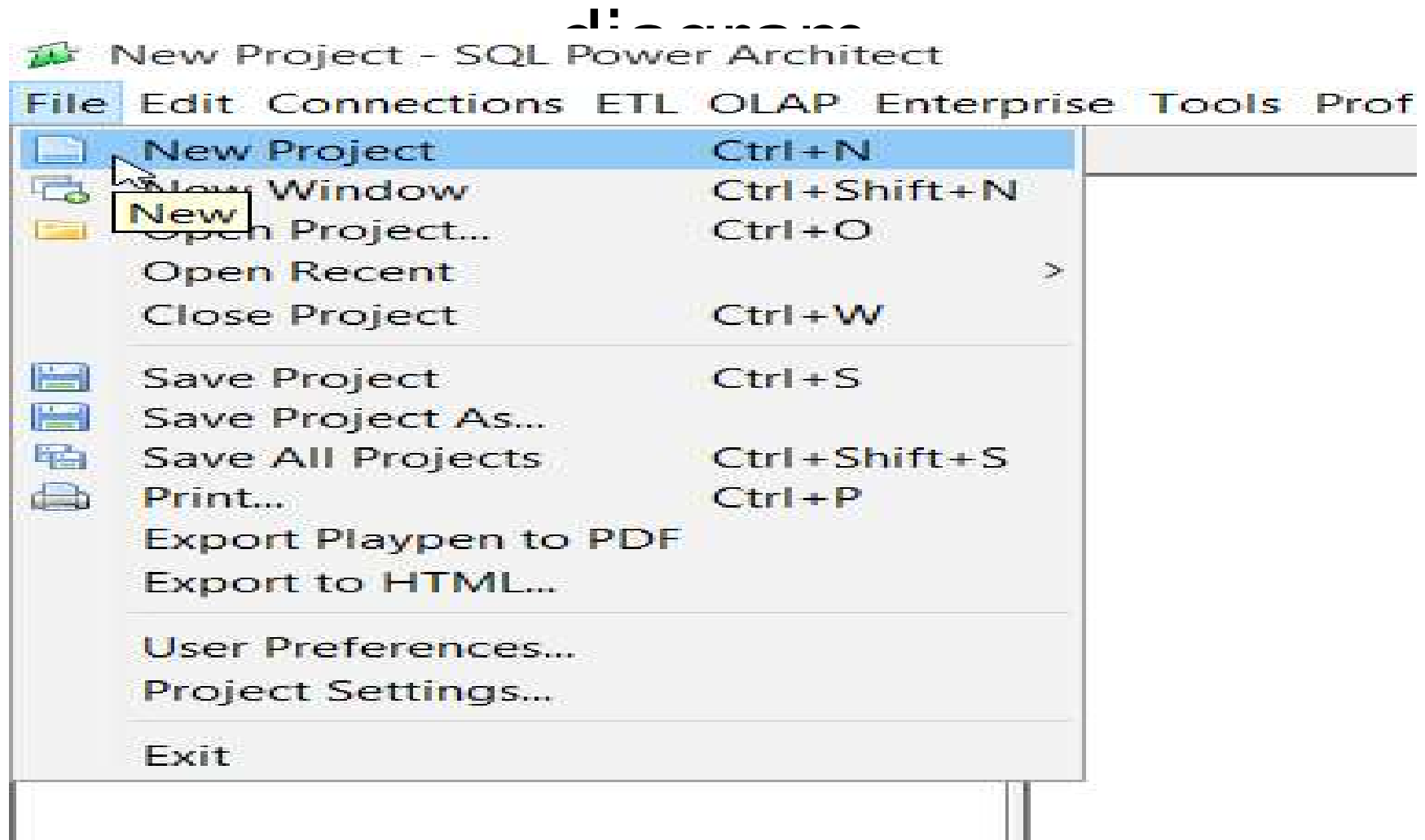


Sample database Entity Relationship diagram



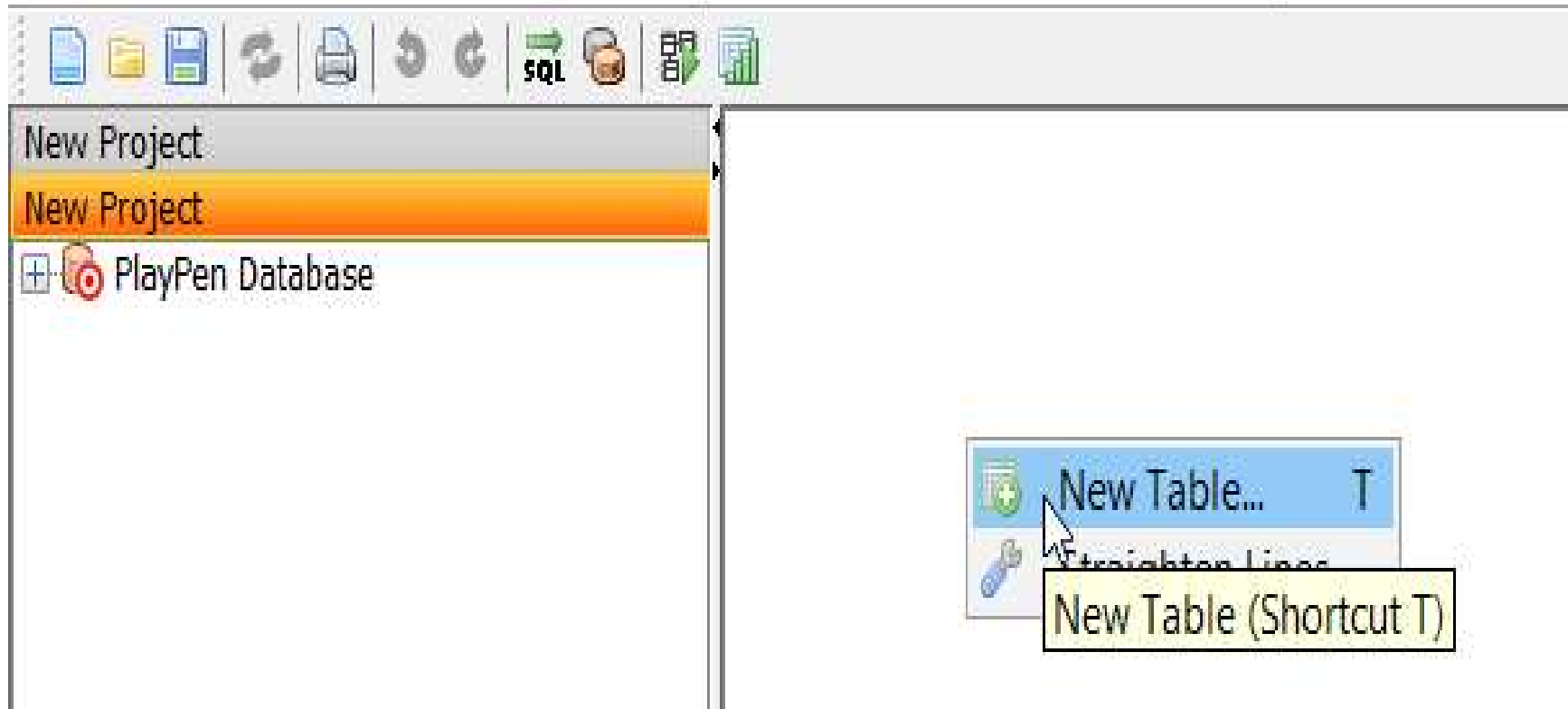
Forward Engineering

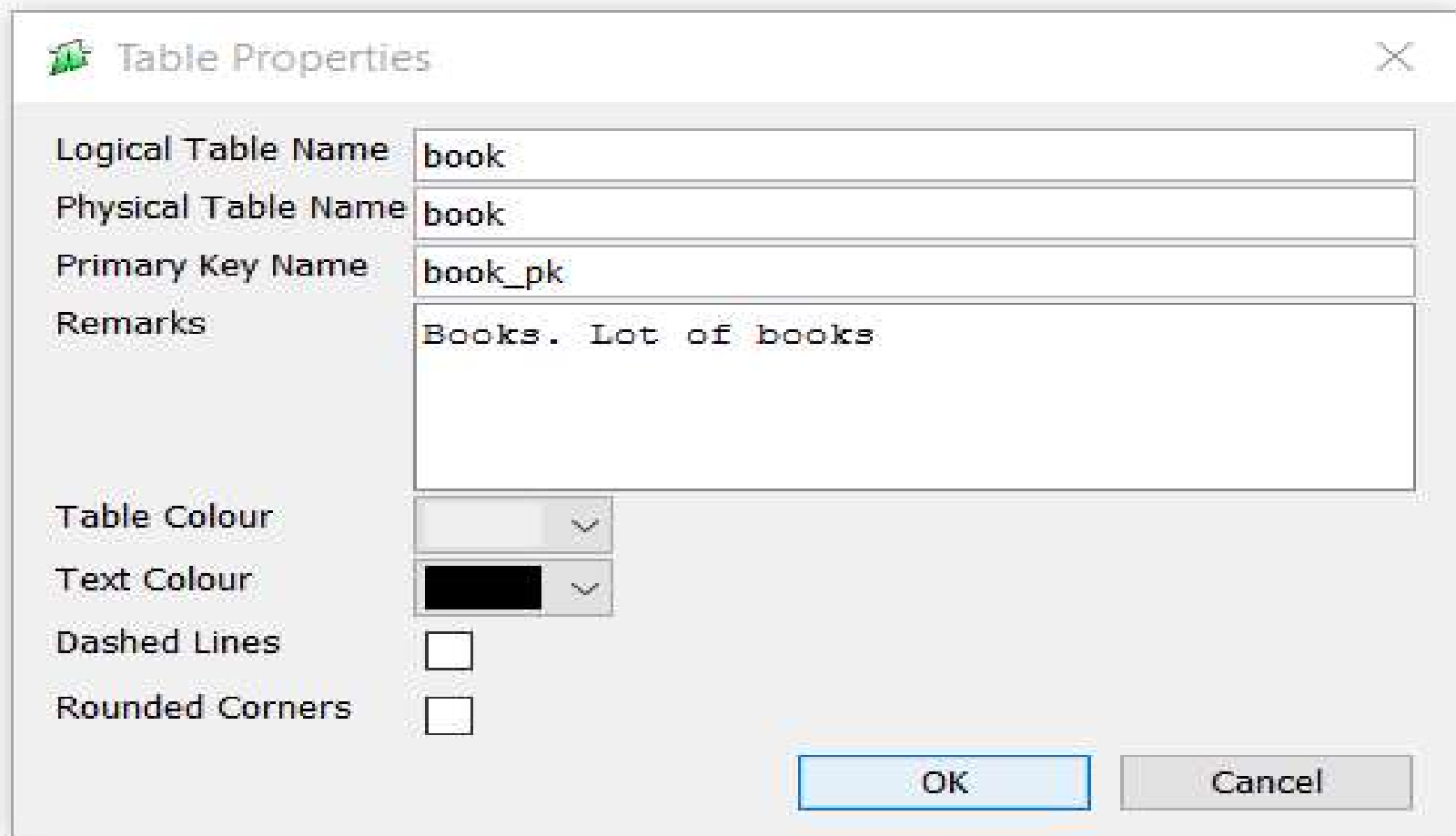
Create Entity – Relationship

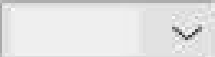



New Project - SQL Power Architect

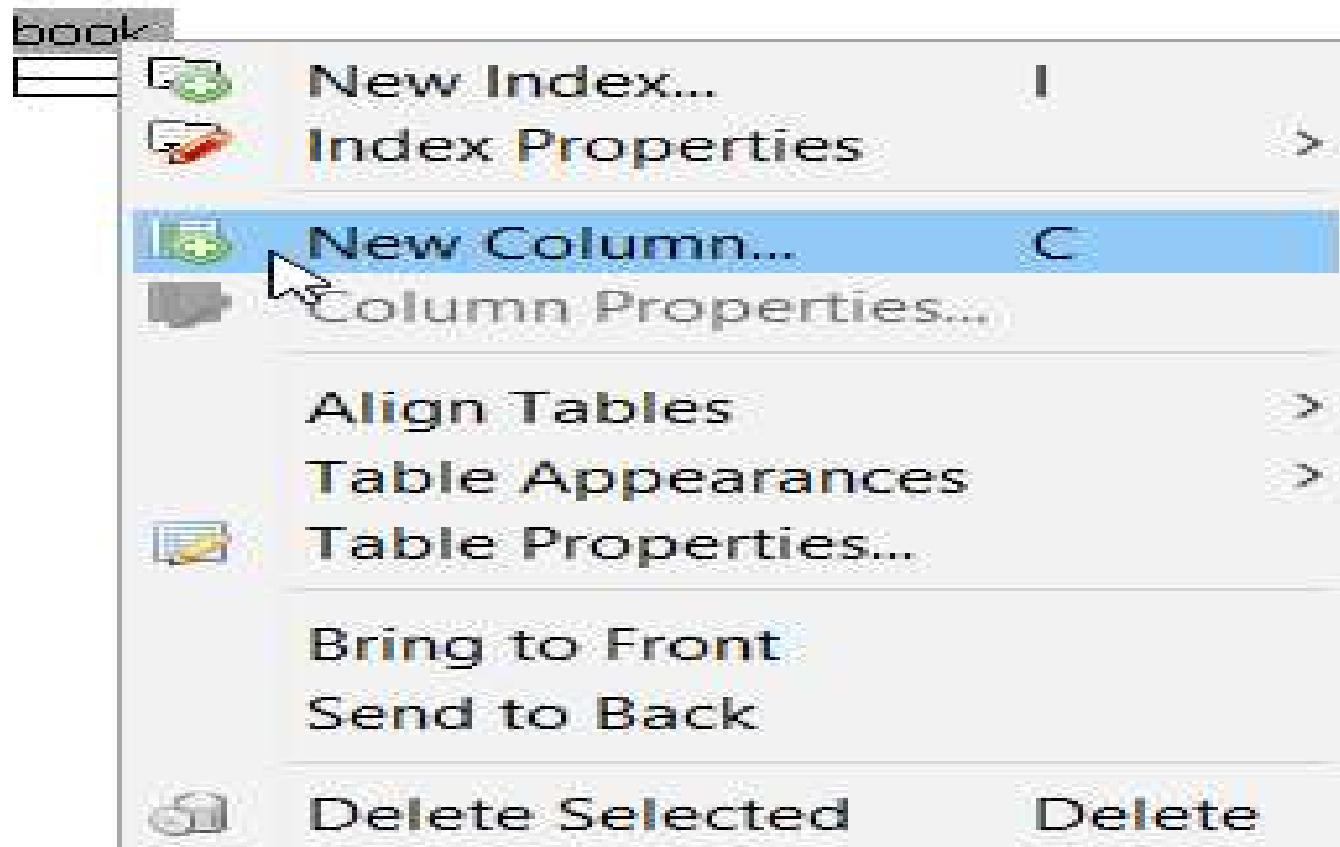
File Edit Connections ETL OLAP Enterprise Tools Profile Window Help




A screenshot of a 'Table Properties' dialog box. The dialog has a title bar with a green icon and a close button. It contains several input fields and checkboxes. The 'Logical Table Name' field is set to 'book'. The 'Physical Table Name' field is also set to 'book'. The 'Primary Key Name' field is set to 'book_pk'. The 'Remarks' field contains the text 'Books. Lot of books'. There are two color selection fields: 'Table Colour' (light gray) and 'Text Colour' (black). At the bottom, there are two checkboxes: 'Dashed Lines' and 'Rounded Corners', both of which are unchecked. The dialog ends with 'OK' and 'Cancel' buttons.

Logical Table Name	book
Physical Table Name	book
Primary Key Name	book_pk
Remarks	Books. Lot of books
Table Colour	
Text Colour	
Dashed Lines	<input type="checkbox"/>
Rounded Corners	<input type="checkbox"/>

OK Cancel



 Column Properties of New Column ✕

Source for ETL Mapping
None Specified

Logical Name
ISBN

Physical Name
ISBN

☒ In Primary Key

Type
CHAR

☒ **Precision** 130 ☐ **Scale** 0

☒ **Allows Nulls** No

☐ **Auto Increment** No

☐ **Default Value**

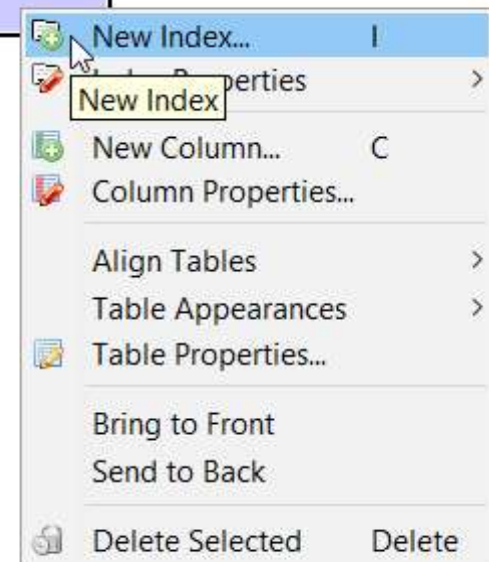
Sequence Name (Only applies to target platforms that use sequences)
book_ISBN_seq

Remarks
No remarks

OK Cancel

book
ISBN: CHAR(130) [PK]
author_id: INTEGER
publisher_id: INTEGER
title: VARCHAR(80)
price: FLOAT

author
id_author: INTEGER [PK]
name: VARCHAR(50)





Index Properties



Index Name

☒ Unique

☐ Primary Key

☐ Clustered

Index Type

In Index	Column	Asc/Des
<input type="checkbox"/>	id_author: INTEGER	UNSPECIFIED
<input checked="" type="checkbox"/>	name: VARCHAR(50)	UNSPECIFIED

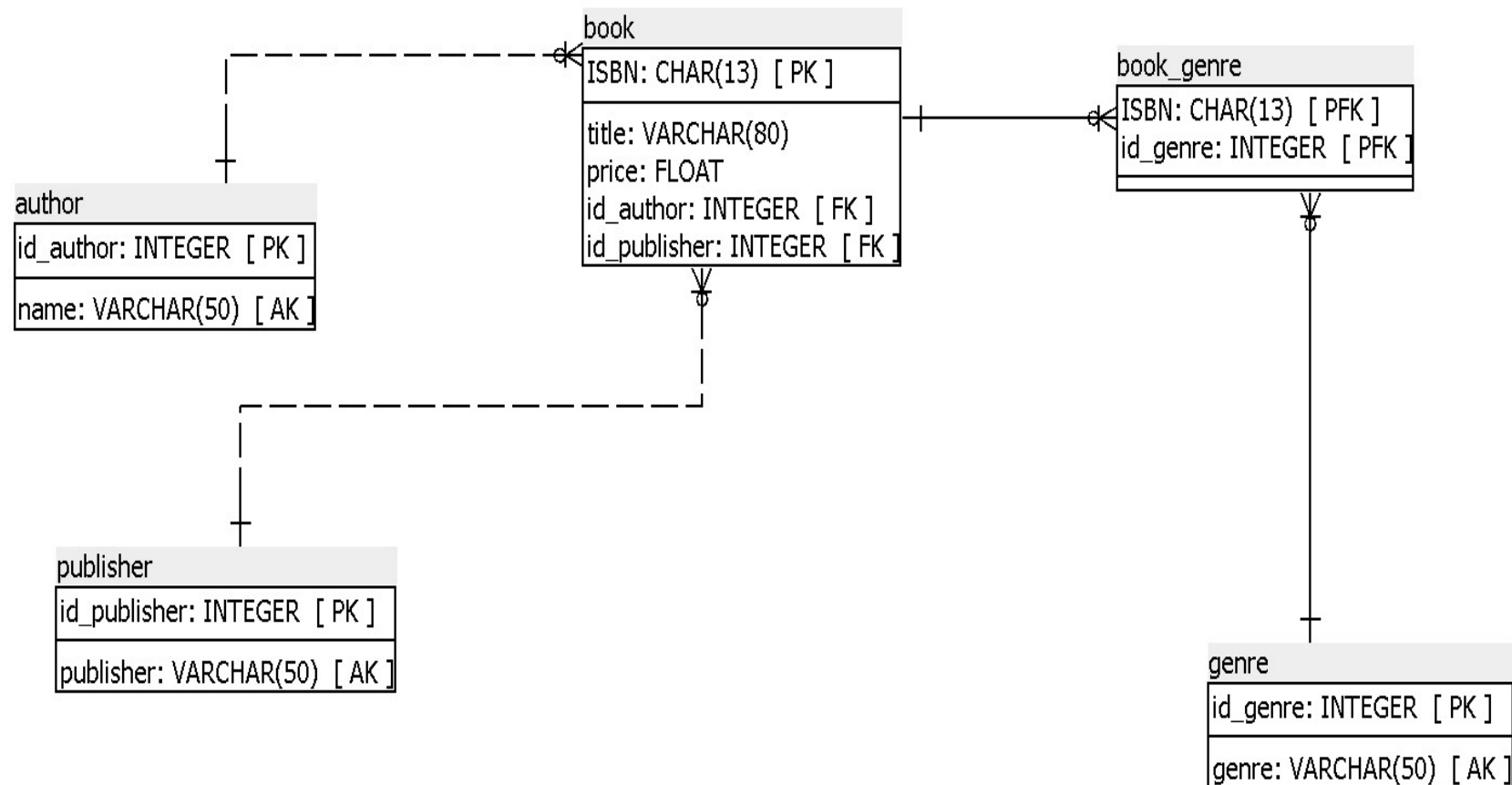


New Identifying Relationship

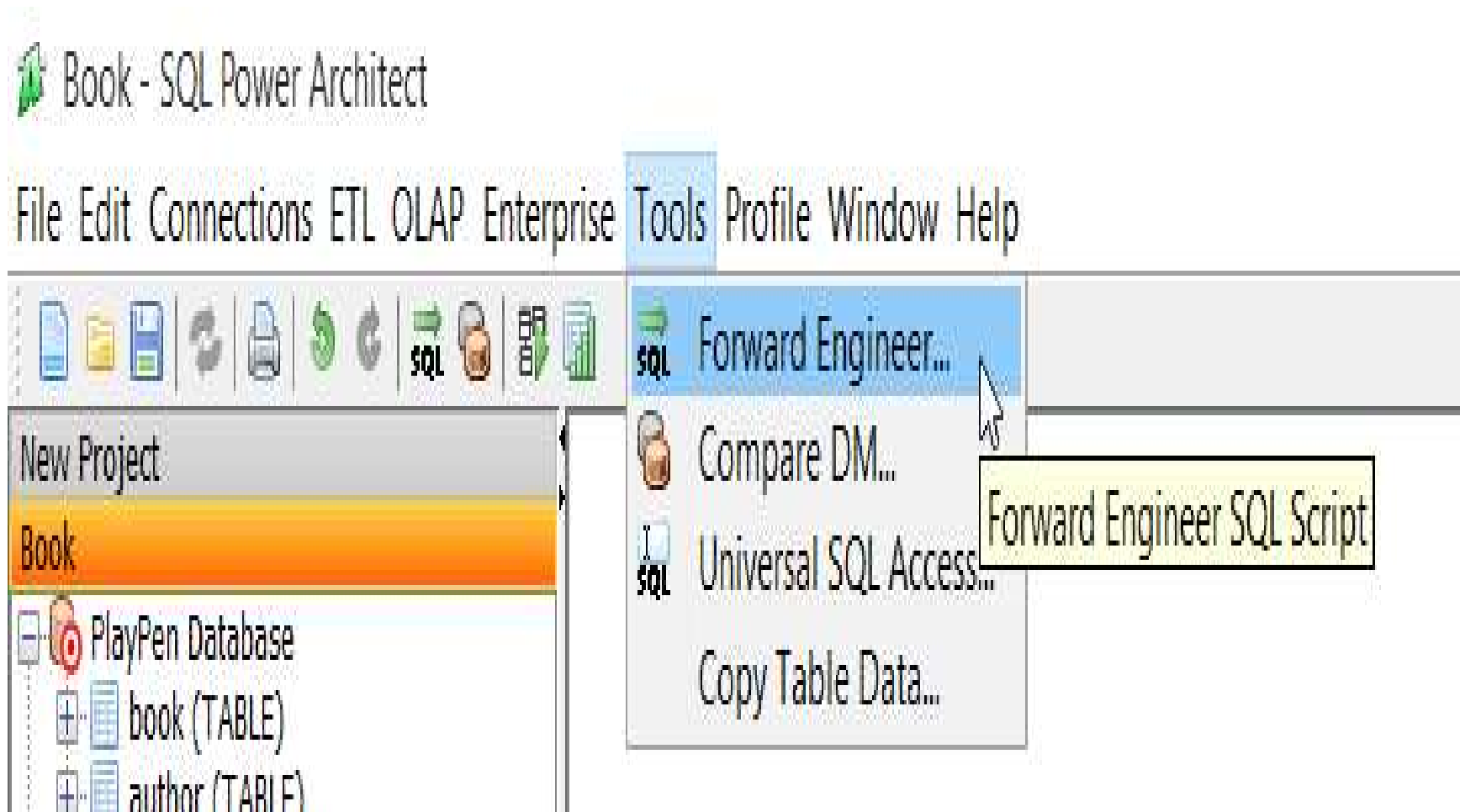
book_genre
ISBN: CHAR(13) [PK]
genre_id: INTEGER [PK]
id_genre: VARCHAR [PFK]



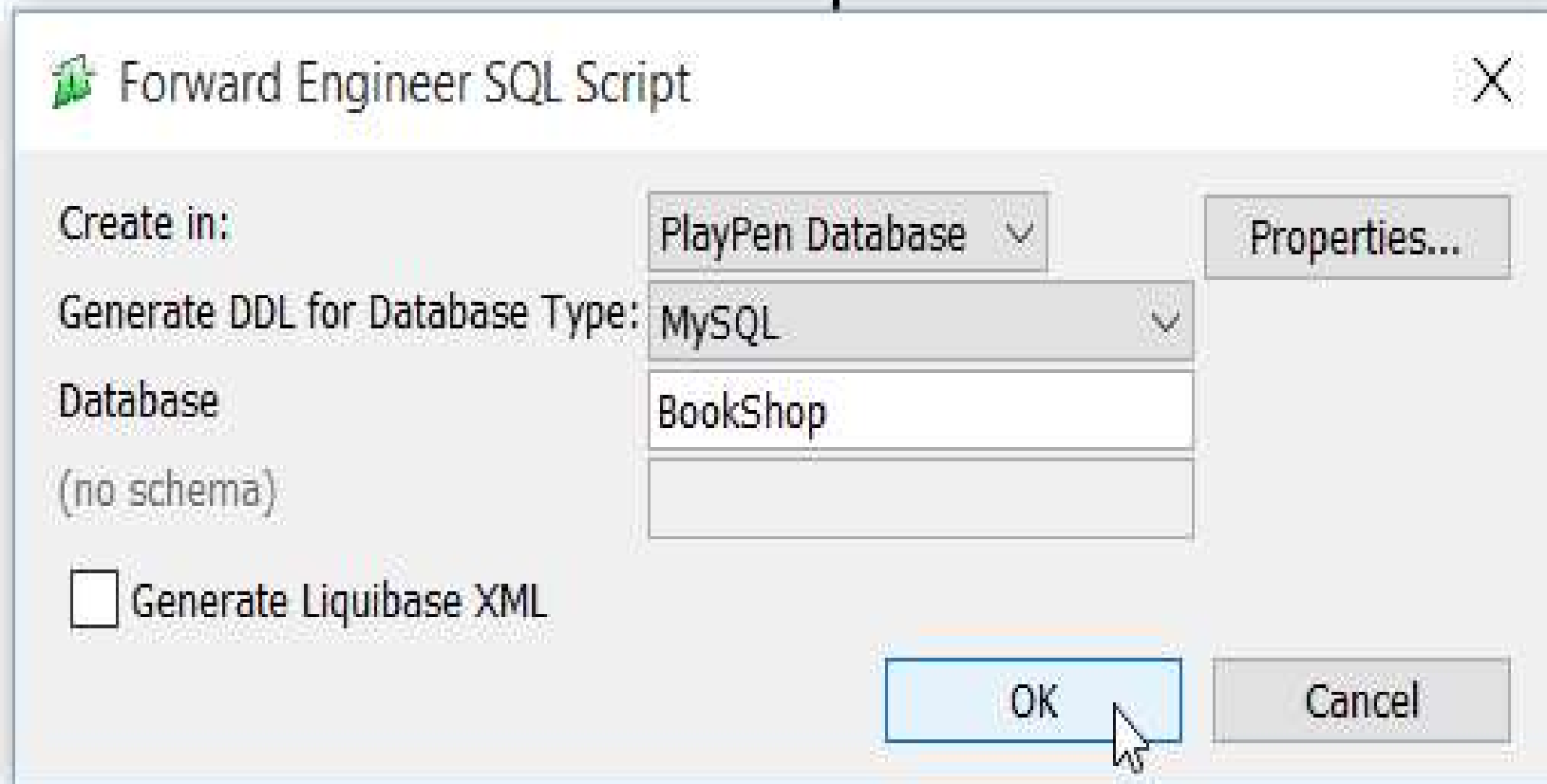
genre
id_genre: VARCHAR [PK]
genre: VARCHAR(50) [AK]



Forward Engineering



Forward Engineering



Forward Engineering

- Save SQL file
- Start DataBase Engine
 - MySQL / Microsoft SQL Server
- Start Front-End tool
 - MySQL WorkBench / Management Studio
- **CREATE SCHEMA / DATABASE**
- Run SQL file

- Generic SQL 92 standard
- Oracle
- PostgreSQL
- Microsoft SQL Server
- MySQL
- IBM DB2
- HSQLDB

DataBases

DataBase Design Normalization Process

“Data modeling is not optional”

no database or system was ever
built without at least an implicit
model

- software programs are designed to implement a process model (or functional specification)
 - specifying business processes that the system is to perform
- same way, database is specified by **data model**, describing what sort of data will be held and how it will be organized

Design – Choice & Creativity

- in design, we do not expect to find a single correct answer, although we will certainly be able to identify many that are incorrect
- 2 data modelers given same set of requirements may produce quite different solutions

- First choice of what symbols or codes we use to represent real-world facts in database
- person's age could be represented by Birth Date, Age at Date of ..., or even by code corresponding to range

- Second choice there is usually more than one way to organize (classify) data
 - into tables and columns in relational model

- Third choice requirements from which we work in practice are usually incomplete, or at least loose enough to accommodate a variety of different solutions

- Fourth choice we have some options as to which part of system will handle each business requirement

- Finally, and perhaps most importantly
 - new information systems seldom deliver value simply by automating current way of doing things
- to exploit information technology fully, we generally need to change our business processes and data required to support them
- data modeler becomes a player in helping to design new way of doing business, rather than merely reflecting the old

- We want you to learn not only to produce sound, workable models (that will not fall down) but to be able to develop and compare different options
- *not throw away rule book, not suggest that anything goes*

Data Model Important

- Leverage
- Conciseness
- Data Quality

Leverage

- small change to data model may have major impact on system as whole
- programs are far more complex and take longer to specify and construct than database
 - their content and structure are heavily influenced by database design

Leverage

- well-designed data model can make programming simpler and cheaper
- poor data organization can be expensive to fix

Conciseness

- data model is powerful tool for expressing information systems requirements and capabilities
- implicitly defines whole set of screens, reports, and processes needed to capture, update, retrieve, and delete specified data

Data Quality

- database is usually valuable business asset built up over long period
- establishing common understanding of what is to be held in each table and column, and how it is to be interpreted
- problems with data quality can be traced to lack of consistency
 - in defining and interpreting data
 - implementing mechanisms to enforce definitions

What Makes a Good Data Model?

- Completeness
- Non-redundancy
- Enforcement of Business Rules
- Data Reusability
- Stability and Flexibility
- Elegance
- Communication
- Integration
- Conflicting Objectives
- Performance

Completeness

- Does the model support all the necessary data?

Non-redundancy

- Does the model specify a database in which the same fact could be recorded more than once?

Enforcement of Business Rules

- How accurately does the model reflect and enforce the rules that apply to the business' data?

Data Reusability

- Will data stored in database be re-useable for purposes beyond those anticipated in process model?
- once an organization has captured data to serve particular requirement, other potential uses and users almost invariably emerge

Stability and Flexibility

- How well will the model cope with possible changes to business requirements?
- can any new data required to support such changes be accommodated in existing tables?
 - or will we be forced to make major structural changes, with corresponding impact on the rest of the system?

Stability and Flexibility

- data model is **stable** in the face of a change to requirements if we do not need to modify it at all
 - we can talk of models being more or less stable, depending on level of change required
- data model is **flexible** if it can be readily extended to accommodate likely new requirements with only minimal impact on existing structure

Elegance

- Does the data model provide a reasonably neat and simple classification of the data?
- simple
- consistent
- easily described and summarized

Communication

- How effective is the model in supporting communication among various stakeholders in the design of a system?
- Do tables and columns represent business concepts that users and business specialists are familiar with and can easily verify?
- Will programmers interpret model correctly?

Integration

- How will the proposed database fit with organization's existing and future databases?
 - common for the same data to appear in more than one database and for problems to arise
- How easy is it to keep different versions in step, or to assemble a complete picture?

Conflicting Objectives

- above aims will conflict with one another
- elegant but radical solution may be difficult to communicate to conservative users
- can be attracted to elegant model that we exclude requirements that do not fit
- model that accurately enforces large number of business rules will be unstable if some rules change
- model that is easy to understand because (reflects perspectives of immediate system users) may not support reusability or integrate well with other

Performance

- system user will not be satisfied if our complete, non-redundant, flexible, and elegant database cannot meet throughput and response-time requirements
- differs from our other criteria because it depends heavily on software and hardware platforms on which database will run
 - exploiting their capabilities is technical task
 - quite different from more business-focused modeling

Performance

- performance requirements are usually “added to the mix” at a later stage than other criteria, and then only when necessary
- usual (and recommended) procedure is to develop data model without considering performance
- attempt to implement it with available hardware and software
- if it is not possible to achieve adequate performance in this way do we consider modifying the model itself

Objectives

- Describe data normalization process
- Perform data normalization process
- Test tables for irregularities using data normalization process

Database Design

- Conceptual Data Modeling
- Normalization Process
- Implementing Base Table Structures

NORMALIZATION PROCESS

Normalization

- process of taking entities and attributes that have been discovered and making them suitable for the relational database
- process does this by removing redundancies and shaping data in manner that the relational engine desires

Normalization

- based on a set of levels, each of which achieves level of correctness or adherence to a particular set of rules
- rules formally known as forms, normal forms
- First Normal Form(1NF)
 - which eliminates data redundancy
- and continue through to
- Fifth Normal Form (5NF)
 - which deals with decomposition of ternary relationships

Normalization

- each level of normalization indicates an increasing degree of adherence to the recognized standards of database design
- as you increase degree of normalization of your data, you'll naturally tend to create an increasing number of tables of decreasing width (fewer columns)

Why Normalize?

- eliminate data that's duplicated, chance it won't match when you need it
- avoid unnecessary coding needed to keep duplicated data in sync
- keep tables thin, increase number of values that will fit on a page (8K) – decrease number of reads that will be needed
- maximizing use of clustered indexes – allow for more optimum data access and joins
- lowering number of indexes per table - indexes are costly to maintain

Eliminating duplicated data

- any piece of data that occurs more than once in database is an error waiting to happen

Process of Normalization

- take entities that are complex and extract simpler entities from them
- continues until every table in database represents one thing and every column describes that thing

3 categories of normalization steps

- entity and attribute shape
- relationships between attributes
- multi-valued and join dependencies in entities

Entity and attribute shape

First Normal Form

- all attributes must be atomic, that is, only a single value represented in a single attribute in a single instance of an entity
- all instances of an entity must contain the same number of values
- all instances of an entity must be different

First Normal Form

- violations often manifest themselves in implemented model with data handling being far less optimal, usually because of having to decode multiple values stored where a single one should be or because of having duplicated rows that cannot be distinguished from one another

- for example, consider group of data like 1, 2, 3, 5, 7
- likely represents five separate values
- atomicity is to consider whether you would ever need to deal with part of column without other parts of data in that same column
- 1, 2, 3, 5, 7 list always treated as single value, it might be acceptable to store value in single column
- if you might need to deal with value 3 individually, then the list is definitely not in First Normal Form
- if there is not plan to use list elements individually, you should consider whether it is still better to store each value individually to allow for future possible usage

E-Mail Addresses

- [name1@domain1.com](#)
- AccountName: name1
- Domain: domain1.com

E-Mail Addresses

- intend to access individual parts
- all you'll ever do is send e-mail, then single column is perfectly acceptable
- need to consider what domains you have e-mail addresses stored for, then it's a completely different matter

Telephone Numbers

- AAA-EEE-NNNN (XXXX):
- AAA area code – indicates calling area located within a state
- EEE exchange - indicates a set of numbers within an area code
- NNNN number - used to make individual phone numbers unique
- XXXX extension - number that must be dialed after connecting

Mailing Addresses

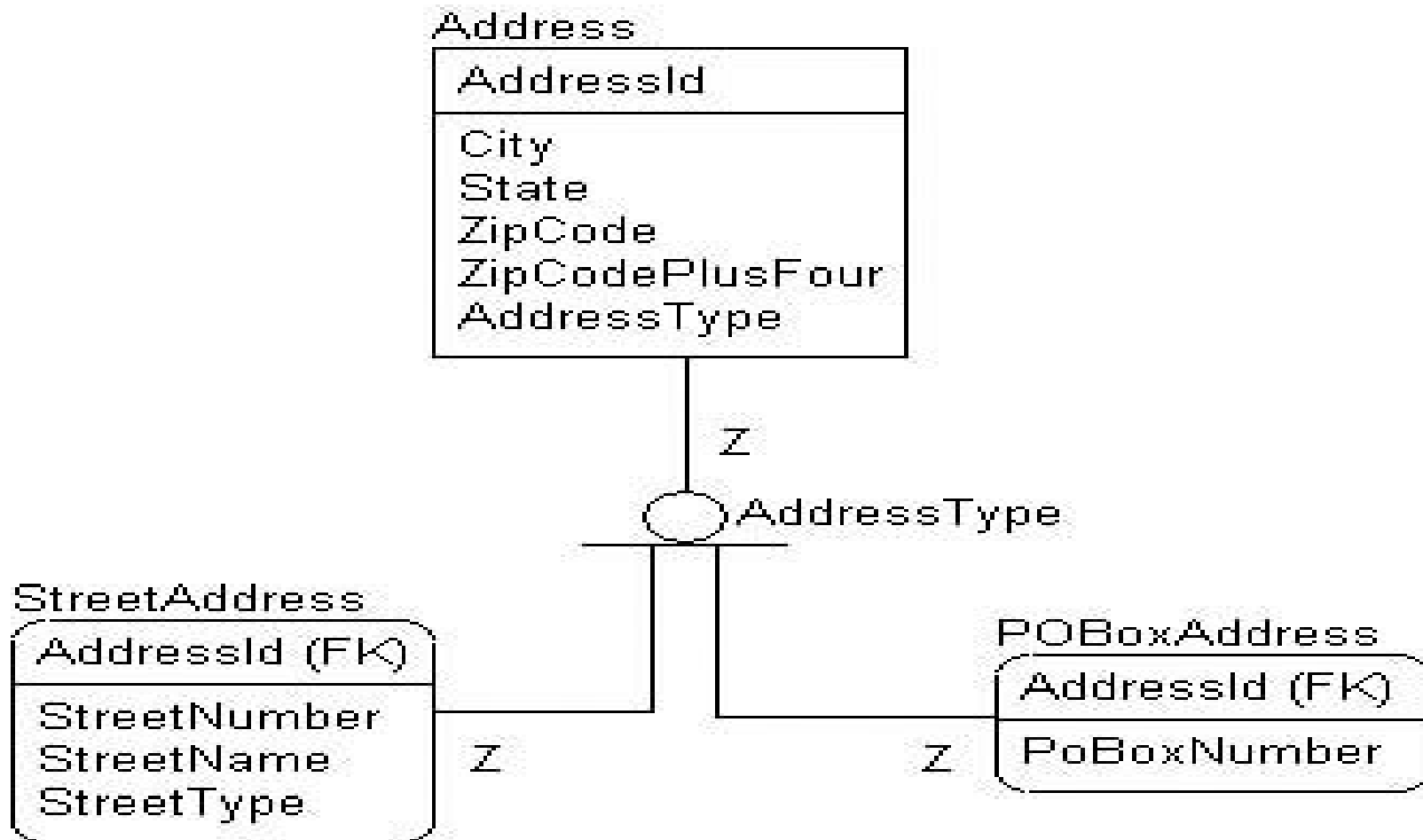
Address

AddressId
AddressLine1
AddressLine2
City
State
ZipCode
ZipCodePlusFour

Address

AddressId
StreetName
StreetNumber
StreetType
City
State
ZipCode
ZipCodePlusFour

Mailing Addresses



All instances in entity contain same number of values

- entities have a fixed number of attributes
 - and tables have a fixed number of columns
- entities should be designed such that every attribute has a fixed number of values associated with it
- example of a violation of this rule in entities that have several attributes with same base name suffixed (or prefixed) with a number, such as Payment1, Payment2, and so on,

Programming Anomalies avoided by First Normal Form

- modifying lists in single column
- modifying multipart values
- dealing with a variable number of facts in an instance

Clues that design is not in First Normal Form

- string data that contains separator-type characters
- attribute names with numbers at the end
- tables with no or poorly defined keys

Relationships Between Attributes

- Second Normal Form
 - relationships between non-key attributes and part of the primary key
- Third Normal Form
 - relationships between non-key attributes
- BCNF (Boyce Codd Normal Form)
 - relationships between non-key attributes and any key
- *Non-key attributes must provide a detail about the key, the whole key, and nothing but the key.*

Second Normal Form

- entity must be in First Normal Form.
- each attribute must be a fact describing the entire key
- technically relevant only when a composite key (a key composed of two or more columns) exists in the entity

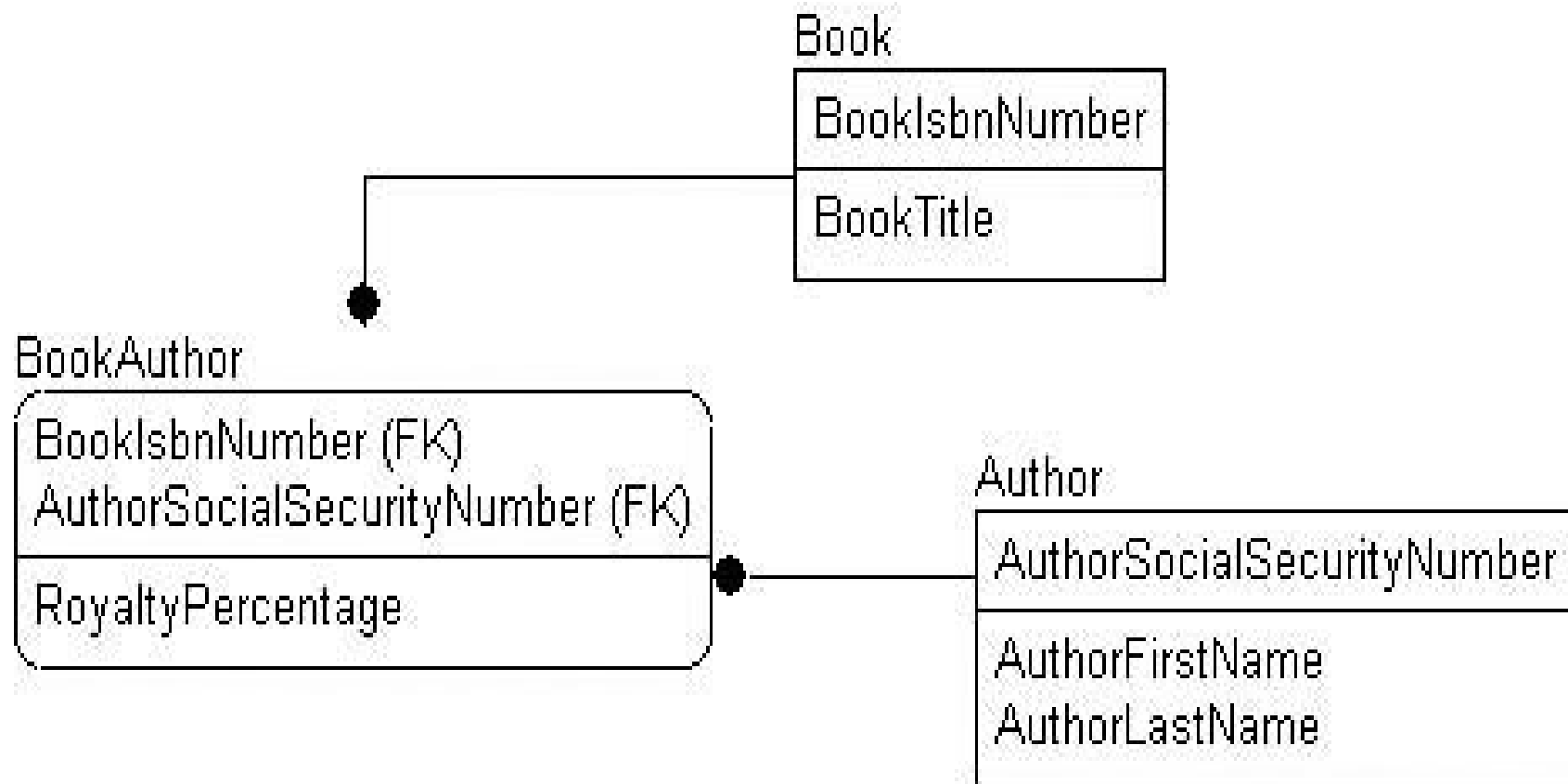
Each non-key attribute must describe entire key

BookAuthor

AuthorSocialSecurityNumber BooksbnNumber
RoyaltyPercentage BookTitle AuthorFirstName AuthorLastName

- BookIsbnNumber attribute uniquely identifies book
- AuthorSocialSecurityNumber uniquely identifies author
- two columns create key that uniquely identifies an author for book
- BookTitle describes book
 - but doesn't describe author at all
- AuthorFirstName and AuthorLastName, describe author
 - but not book

- BookIsbnNumber → BookTitle
- AuthorSocialSecurityNumber → AuthorFirstName
- AuthorSocialSecurityNumber → AuthorLastName
- BookIsbnNumber, AuthorSocialSecurityNumber → RoyaltyPercentage



Programming problems avoided

- all programming issues that arise with Second Normal Form (as well as Third and Boyce-Codd Normal Forms) deal with functional dependencies that can end up corrupting data

Book Author Edit

Author Information

SSN	111-11-1111
First Name	Fred
Last Name	Smith

Book Information

ISBN	1234567890
Title	Database Design

Royalty Percentage 15

OK

Cancel

- same author's information would have to be duplicated amongst all books
- cannot delete only book and keep author around
- cannot insert only author without book

Annomalies

- UPDATE
 - duplicate data, have to update multiple rows
- INSERT
 - cannot insert data for an entity without relationship to any other entity
- DELETE
 - cannot delete data for an entity without risk of losing info. about related entity

Clues that entity is not in Second Normal Form

- repeating key attribute name prefixes, indicating that values are probably describing some additional entity
- data in repeating groups, showing signs of functional dependencies between attributes
- composite keys without foreign key, which might be sign you have key values that identify multiple things in key

Third Normal Form

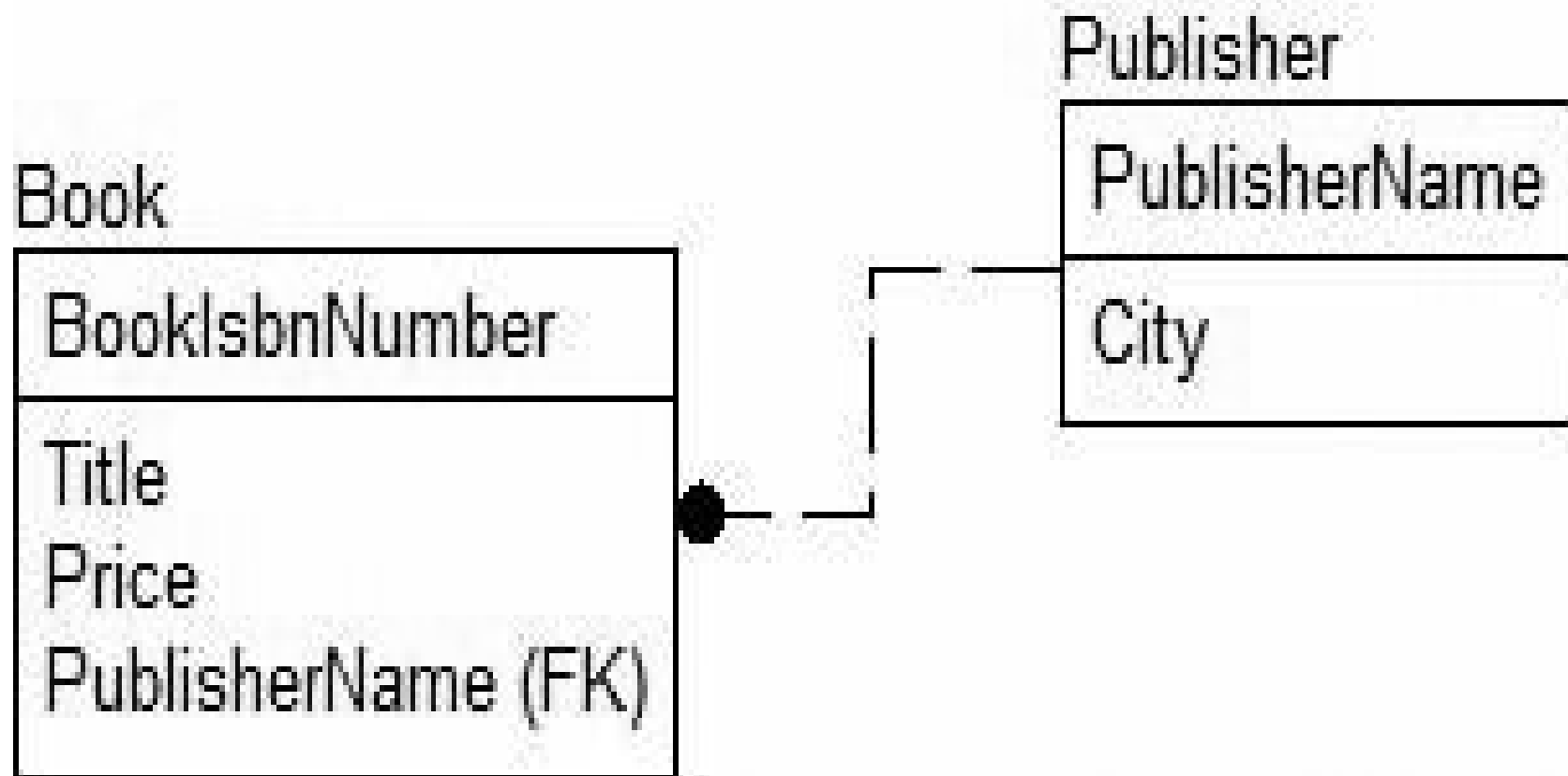
- entity must be in Second Normal Form.
- non-key attributes cannot describe other non-key attributes

non-key attributes cannot describe
other non-key attributes

Book

BooksbnNumber
Title
Price
PublisherName
PublisherCity

- Title defines title for the book defined by BookIsbnNumber
- Price indicates price of the book
- PublisherName describes the book's publisher
- PublisherCity also sort of describes something about the book, in that it tells where the publisher was located
- doesn't make sense in this context, because location of publisher is directly dependent on what publisher is represented by PublisherName



- Publisher entity has data concerning only publisher
- Book entity has book information
 - now if we want to add information to our schema concerning the publisher, contact information or address, it's obvious where we add that information
- City attribute clearly identifying publisher
 - not the book

Clues that entities are not in Third Normal Form

- multiple attributes with same prefix
 - much like Second Normal Form, only this time not in the key
- repeating groups of data
- summary data that refers to data in a different entity altogether
 - Price in Invoice as $\text{SUM}(\text{Quantity} * \text{ProductCost})$ from LineItems

Boyce-Codd Normal Form

- Ray Boyce, Edgar F. Codd
- entity is in First Normal Form.
- all attributes are fully dependent on a key
- every determinant is a key

Entity in BCNF if every Determinant is key

- *Any attribute or combination of attributes on which any other attribute or combination of attributes is functionally dependent.*
- BCNF extends previous normal forms by saying that entity might have many keys, and all attributes must be dependent on one of these keys

- Third Normal Form table which does not have multiple overlapping candidate keys is guaranteed to be in BCNF
- Third Normal Form table with two or more overlapping candidate keys may or may not be in BCNF

Court Bookings

Court	Start Time	End Time	Rate Type
1	09:30	10:30	SAVER
1	11:00	12:00	SAVER
1	14:00	15:30	STANDARD
2	10:00	11:30	PREMIUM-B
2	11:30	13:30	PREMIUM-B
2	15:00	16:30	PREMIUM-A

Court Bookings

- hard court (Court1) and grass court (Court2)
- booking defined by Court and period for which the Court is reserved
- booking has Rate Type associated
 - SAVER for hard made by members
 - STANDARD for hard made by non-members
 - PREMIUM-A for grass made by members
 - PREMIUM-B for grass made by non-members

Court Bookings - candidate keys

- {Court, Start Time}
- {Court, End Time}
- {Rate Type, Start Time}
- {Rate Type, End Time}

- table adheres to both 2NF and 3NF
- table does not adhere to BCNF
- because of dependency Rate Type → Court, in which the determining attribute (Rate Type) is neither a candidate key nor a superset of a candidate key

Rate Types

Rate Type	Court	Member Flag
SAVER	1	Yes
STANDARD	1	No
PREMIUM-A	2	Yes
PREMIUM-B	2	No

Court Bookings

Court	Start Time	End Time	Member Flag
1	09:30	10:30	Yes
1	11:00	12:00	Yes
1	14:00	15:30	No
2	10:00	11:30	No
2	11:30	13:30	No
2	15:00	16:30	Yes

- candidate keys for Rate Types table are {Rate Type} and {Court, Member Flag}
- candidate keys for Court Bookings table are {Court, Start Time} and {Court, End Time}
- both tables are in BCNF
- having one Rate Type associated with two different Courts is now impossible
- anomaly affecting original table has been eliminated

Multivalued Dependencies

- Third Normal Form is generally considered pinnacle of proper database design
- serious problems might still remain in logical design

Fourth Normal Form

- entity must be in BCNF
- there must not be more than one multivalued dependency between an attribute and the key of the entity

Fourth Normal Form

- table is in 4NF if and only if, for every one of its non-trivial multivalued dependencies $X \twoheadrightarrow Y$, X is a super key, X is either candidate key or superset thereof

Fourth Normal Form violations

- ternary relationships
- lurking multivalued attributes
- temporal data/value history

Restaurant	Pizza Variety	Delivery Area
A1 Pizza	Thick Crust	Springfield
A1 Pizza	Thick Crust	Shelbyville
A1 Pizza	Thick Crust	Capital City
A1 Pizza	Stuffed Crust	Springfield
A1 Pizza	Stuffed Crust	Shelbyville
A1 Pizza	Stuffed Crust	Capital City
Elite Pizza	Thin Crust	Capital City
Elite Pizza	Stuffed Crust	Capital City
Vincenzo's Pizza	Thick Crust	Springfield
Vincenzo's Pizza	Thick Crust	Shelbyville
Vincenzo's Pizza	Thin Crust	Springfield
Vincenzo's Pizza	Thin Crust	Shelbyville

- table has no non-key attributes
- meets all normal forms up to BCNF
- not in 4NF, non-trivial multivalued dependencies
- $\{\text{Restaurant}\} \twoheadrightarrow \{\text{Pizza Variety}\}$
- $\{\text{Restaurant}\} \twoheadrightarrow \{\text{Delivery Area}\}$
- eliminate possibility of anomalies

Restaurant	Pizza Variety
A1 Pizza	Thick Crust
A1 Pizza	Stuffed Crust
Elite Pizza	Thin Crust
Elite Pizza	Stuffed Crust
Vincenzo's Pizza	Thick Crust
Vincenzo's Pizza	Thin Crust

Restaurant	Delivery Area
A1 Pizza	Springfield
A1 Pizza	Shelbyville
A1 Pizza	Capital City
Elite Pizza	Capital City
Vincenzo's Pizza	Springfield
Vincenzo's Pizza	Shelb

- in contrast, if pizza varieties offered by restaurant sometimes did legitimately vary from one delivery area to another, the original three-column table would satisfy 4NF

Fifth Normal Form

- not every ternary relationship can be broken down into two entities related to a third
- aim of 5NF is to ensure that any ternary relationships that still exist in 4NF that can be decomposed into entities are decomposed
- eliminates problems with update anomalies due to multivalued dependencies
 - much like in 4NF only these are trickier to find.

Denormalization

- used primarily to improve performance in cases where overnormalized structures are causing overhead to query processor
- whether slightly slower (but 100 percent accurate) application is preferable to a faster application of lower accuracy
- during logical modeling, we should never step back from our normalized structures to performance-tune our applications proactively

Key Terms

- Data normalization
- Database design / Data structures
- Logical database design
- Entity-relationship diagram conversion
- Functional dependencies
- Redundant data
- 1st Normal Form, 2nd NF, 3rd NF
- Boyce – Codd Normal Form

Normalization of Database Tables

Another Example

Report generated

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.8
		101	John G. News *	Database Designer	105.00	56.3
		114	Annelise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

- an employee can be assigned to more than one project
- each project includes only a single occurrence of any one employee

- project number (PROJ_NUM) is apparently intended to be a primary key or at least a part of a PK, but it contains nulls
 - PROJ_NUM + EMP_NUM will define each row
- table entries invite data inconsistencies
 - for example, JOB_CLASS value “Elect. Engineer” might be entered as “Elect.Eng.” in some cases, “El. Eng.” in others, and “EE” in still others

data redundancies yield the following anomalies:

- Update anomalies
- Insertion anomalies
- Deletion anomalies

Update anomalies

- Modifying the JOB_CLASS for employee number 105 requires (potentially) many alterations, one for each EMP_NUM = 105.

Insertion anomalies

- Just to complete a row definition, a new employee must be assigned to a project
- If the employee is not yet assigned, a phantom project must be created to complete the employee data entry

Deletion anomalies

- Suppose that only one employee is associated with a given project
- If that employee leaves the company and the employee data are deleted, the project information will also be deleted
- To prevent the loss of the project information, a fictitious employee must be created just to save the project information

well-formed relations

- objective of normalization is to ensure that each table conforms to the concept of *well-formed relations*, that is, tables that have the following characteristics:

- Each table represents a single subject
 - for example, course table will contain only data that directly pertain to courses
 - similarly, student table will contain only student data
- No data item will be unnecessarily stored in more than one table (tables have minimum controlled redundancy)
 - the reason for this requirement is to ensure that the data are updated in only one place

- All nonprime attributes in a table are dependent on the primary key, the entire primary key and nothing but the primary key
 - reason for this requirement is to ensure that the data are uniquely identifiable by primary key value
- each table is void of insertion, update, or deletion anomalies
 - to ensure integrity and consistency of data

Functional dependence

- attribute B is fully functionally dependent on attribute A if each value of A determines one and only one value of B
- Example: $\text{PROJ_NUM} \rightarrow \text{PROJ_NAME}$
 - read as “PROJ_NUM functionally determines PROJ_NAME”
 - attribute PROJ_NUM is known as “determinant” attribute, and attribute PROJ_NAME is known as “dependent” attribute

Functional dependence

- Attribute A determines attribute B (that is, B is functionally dependent on A) if all of the rows in table that agree in value for attribute A also agree in value for attribute B

Step 1:

Eliminate the Repeating Groups

- Start by presenting the data in a tabular format, where each cell has a single value and there are no repeating groups.
- To eliminate the repeating groups, eliminate the nulls by making sure that each repeating group attribute contains an appropriate data value.

Table in first normal form 1NF

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
15	Evergreen	101	John G. News	Database Designer	105.00	19.4
15	Evergreen	105	Alice K. Johnson *	Database Designer	105.00	35.7
15	Evergreen	106	William Smithfield	Programmer	35.75	12.6
15	Evergreen	102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
18	Amber Wave	118	James J. Frommer	General Support	18.36	45.3
18	Amber Wave	104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	96.75	48.4
22	Rolling Tide	113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	26.87	22.0
22	Rolling Tide	106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
25	Starflight	115	Travis B. Bawangi	Systems Analyst	96.75	45.8
25	Starflight	101	John G. News *	Database Designer	105.00	56.3
25	Starflight	114	Annelise Jones	Applications Designer	48.10	33.1
25	Starflight	108	Ralph B. Washington	Systems Analyst	96.75	23.6
25	Starflight	118	James J. Frommer	General Support	18.36	30.5
25	Starflight	112	Darlene M. Smithson	DSS Analyst	45.95	41.4

first normal form (1NF)

- describes tabular format in which:
- all of key attributes are defined
- there are no repeating groups in the table
- each row/column intersection contains only atomic values, one and only one value, not set of values
- all attributes are dependent on primary key
- all relational tables satisfy 1NF requirements

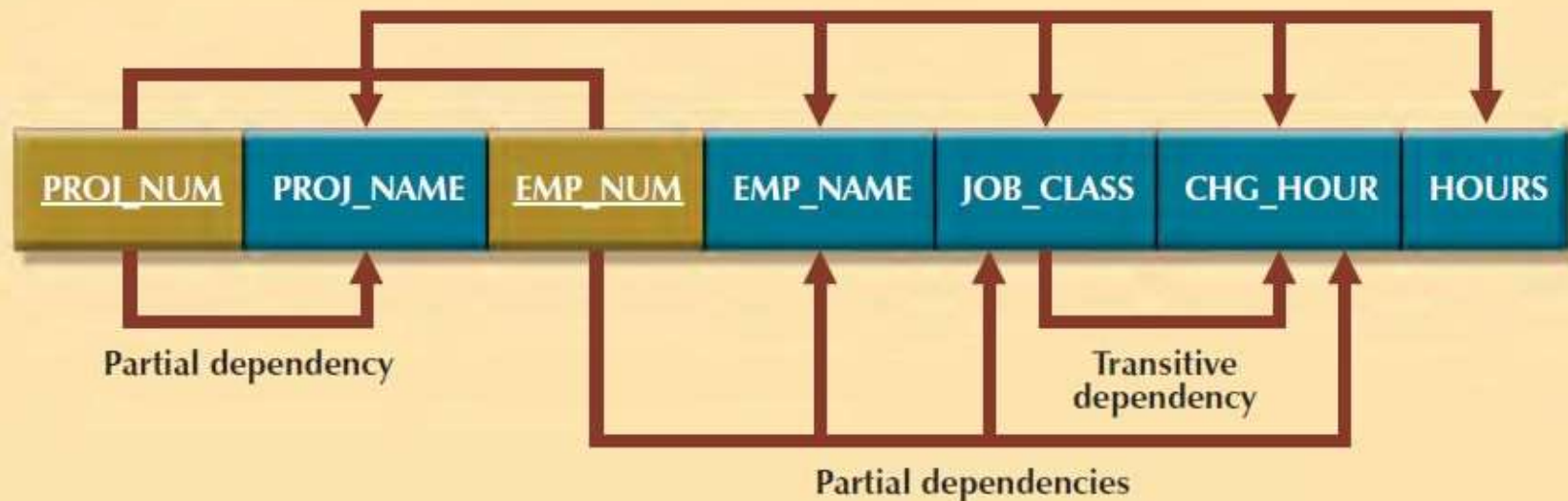
Step 2: Identify the Primary Key

- that will uniquely identify any attribute value must be composed of a combination of PROJ_NUM and EMP_NUM
- identification of PK in Step 2 means that you have already identified following dependency:
- PROJ_NUM, EMP_NUM → PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, HOURS

Step 3: Identify All Dependencies

- PROJ_NUM \rightarrow PROJ_NAME
- EMP_NUM \rightarrow EMP_NAME, JOB_CLASS, CHG_HOUR
- JOB_CLASS \rightarrow CHG_HOUR

Dependency Diagram



1NF (PROJ_NUM, EMP_NUM, PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOURS, HOURS)

PARTIAL DEPENDENCIES:

(PROJ_NUM \Rightarrow PROJ_NAME)

(EMP_NUM \Rightarrow EMP_NAME, JOB_CLASS, CHG_HOUR)

TRANSITIVE DEPENDENCY:

(JOB CLASS \Rightarrow CHG_HOUR)

Dependency Diagram

1. primary key attributes are bold, underlined, and shaded in different color.
2. arrows above attributes indicate all desirable dependencies, that is, dependencies that are based on PK; entity's attributes are dependent on *combination of PROJ_NUM and EMP_NUM*
3. arrows below indicate less desirable dependencies; Two types of such dependencies exist:

1. *Partial dependencies* - you need to know only PROJ_NUM to determine the PROJ_NAME; that is, PROJ_NAME is dependent on only part of PK; and you need to know only EMP_NUM to find EMP_NAME, JOB_CLASS, and CHG_HOUR. Dependency based on only a part of a composite primary key is a partial dependency
2. *Transitive dependencies* - CHG_HOUR is dependent on JOB_CLASS; neither CHG_HOUR nor JOB_CLASS is prime attribute (at least part of key) condition is transitive dependency. Transitive dependency is a dependency of one nonprime attribute on another. Transitive dependencies still yield data anomalies

Data Redundancies ... Anomalies

- occur because row entry requires duplication of data
- if Alice K. Johnson submits her work log, user would have to make multiple entries during course of a day
 - for each entry, EMP_NAME, JOB_CLASS, and CHG_HOUR must be entered each time, even though attribute values are identical for each row entered
- duplicate effort inefficient, helps create anomalies: nothing prevents from typing slightly different versions of employee name, position, hourly pay
- data anomalies violate relational database's integrity and consistency rules

Conversion to Second Normal Form

- Converting to 2NF is done only when the 1NF has a composite primary key.
- If the 1NF has a single-attribute primary key, then the table is automatically in 2NF.

Step 1: Make New Tables to Eliminate Partial Dependencies

- for each component of primary key that acts as determinant in a partial dependency, create a new table with copy of that component as primary key
- while these components are placed in new tables, it is important that they also remain in original table as well - they will be foreign keys for the relationships that are needed to relate these new tables to original table

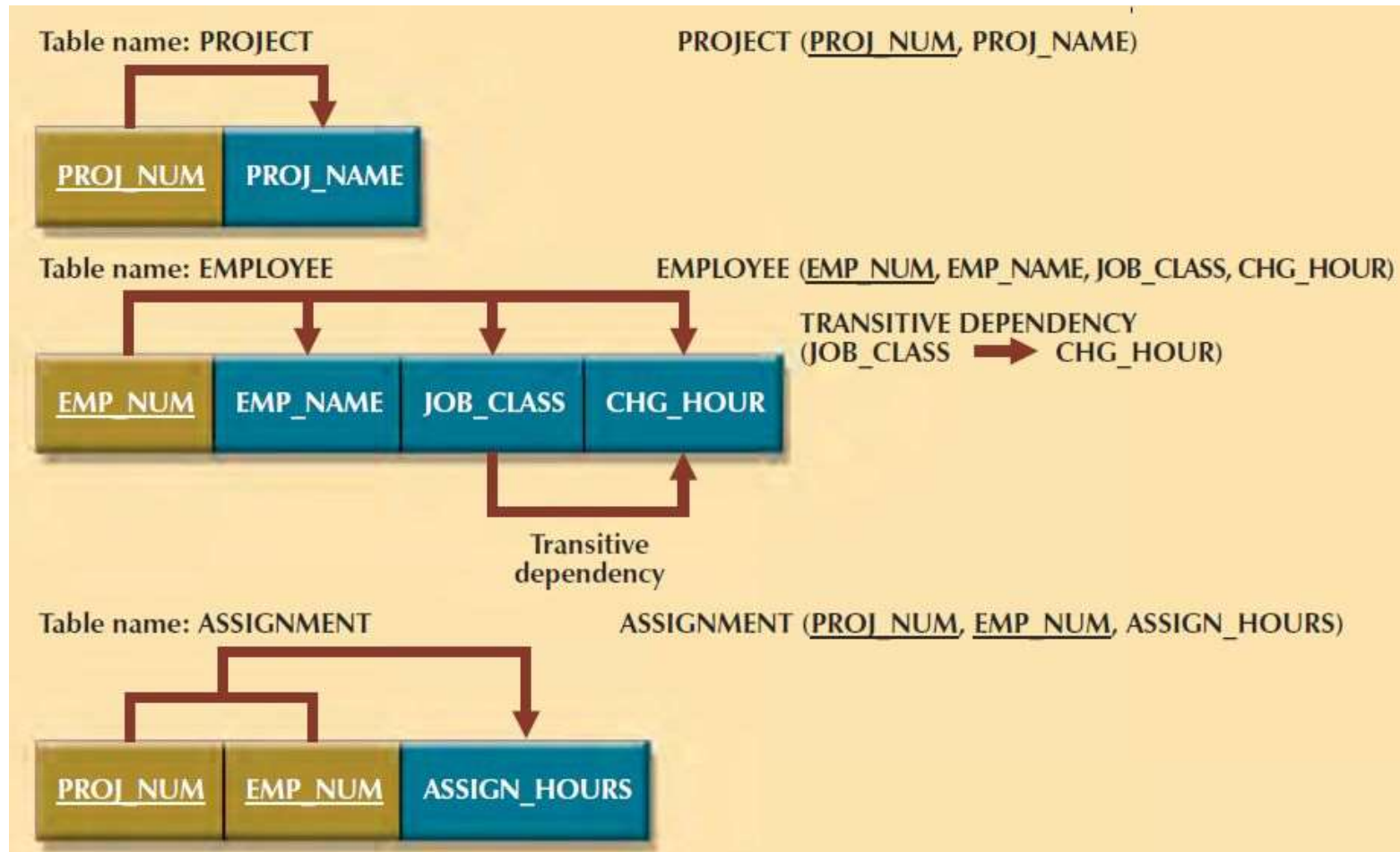
- PROJ_NUM
- EMP_NUM
- PROJ_NUM EMP_NUM
- each component will become key in new table
- original table is now divided into three tables (PROJECT, EMPLOYEE, and ASSIGNMENT)

Step 2: Reassign Corresponding Dependent Attributes

- dependencies for original key components are found by examining arrows below dependency diagram
- attributes that are dependent in a partial dependency are removed from original table and placed in new table with its determinant
- any attributes that are not dependent in partial dependency will remain in original table

- PROJECT (PROJ_NUM, PROJ_NAME)
- EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS, CHG_HOUR)
- ASSIGNMENT (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)

Second normal form (2NF)



second normal form (2NF)

- table is in second normal form (2NF) when:
 - it is in 1NF
 - and
 - it includes no partial dependencies; that is, no attribute is dependent on only portion of primary key

- Because number of hours spent on each project by each employee is dependent on both PROJ_NUM and EMP_NUM in ASSIGNMENT table, you leave those hours in ASSIGNMENT table
- ASSIGNMENT table contains composite primary key composed of attributes PROJ_NUM and EMP_NUM
- primary key/foreign key relationships have been created

- most of the anomalies discussed earlier have been eliminated
- for example, if you now want to add, change, or delete a PROJECT record, you need to go only to PROJECT table and make the change to only one row
- transitive dependency can generate anomalies
- for example, if charge per hour changes for job classification held by many employees, that change must be made for each of those employees
- if you forget to update some of employee records that are affected by charge per hour change, different employees with same job description will generate different hourly charges

Conversion to Third Normal Form

Step 1: Make New Tables to Eliminate Transitive Dependencies

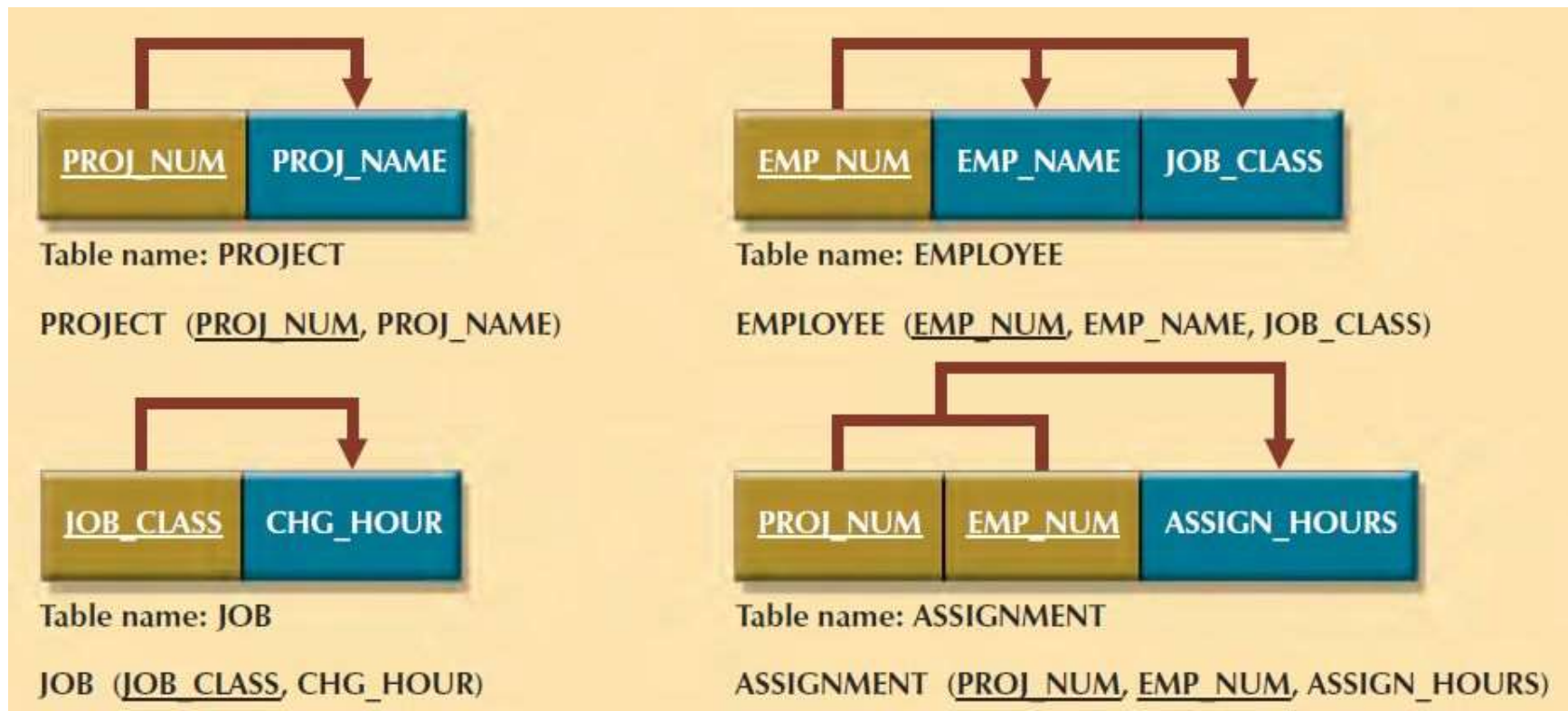
- for every transitive dependency, write a copy of its determinant as primary key for new table
- determinant is any attribute whose value determines other values within a row
- if you have three different transitive dependencies, you will have three different determinants
- determinant remain in original table to serve as foreign key

- determinant for this transitive dependency as:
- JOB_CLASS

Step 2: Reassign Corresponding Dependent Attributes

- identify the attributes that are dependent on each determinant identified
- place dependent attributes in new tables with their determinants and remove them from their original tables

Third normal form (3NF)



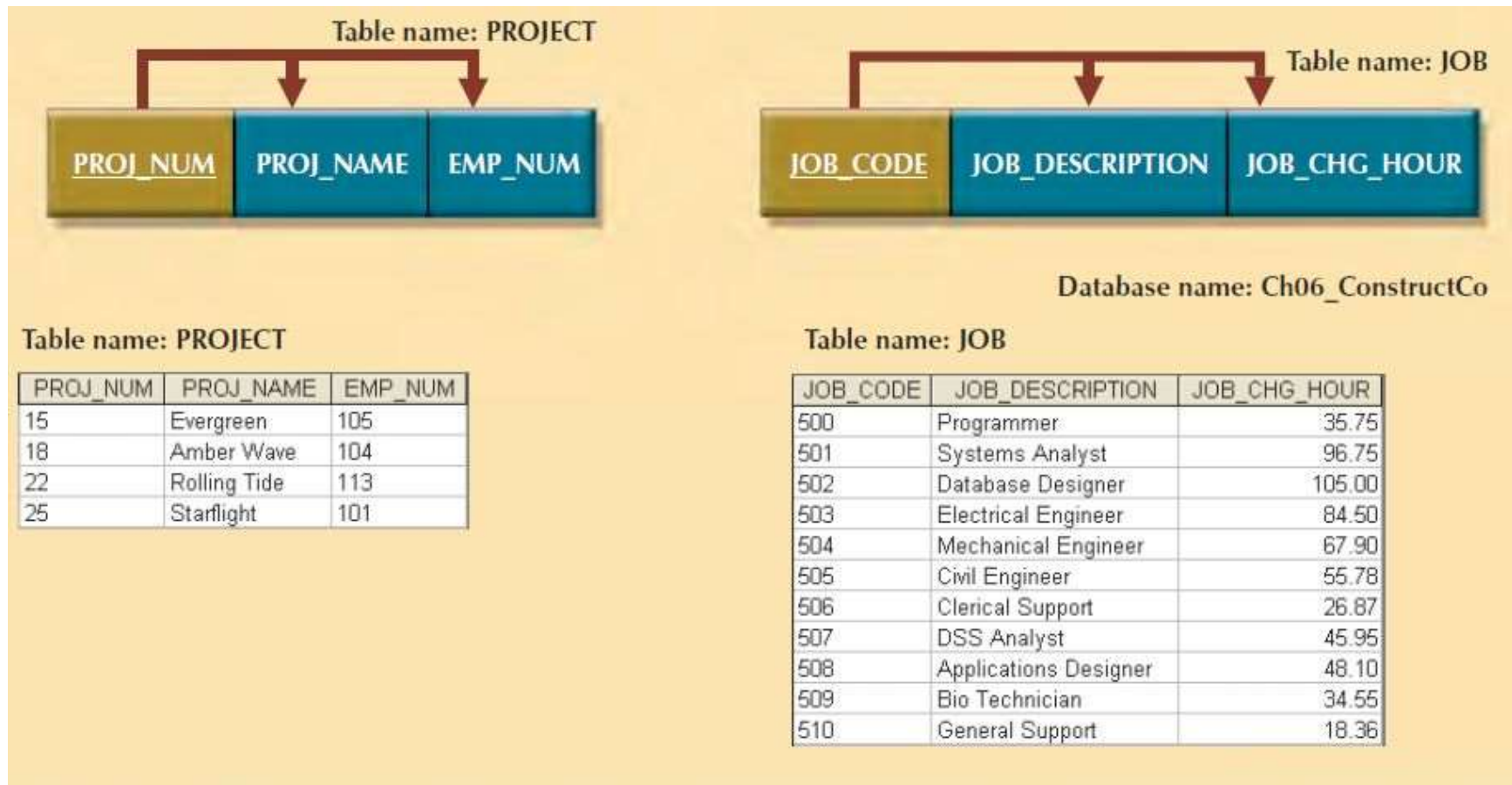
third normal form (3NF)

- table is in third normal form (3NF) when:
- it is in 2NF
- and
- it contains no transitive dependencies

- PROJECT (PROJ_NUM, PROJ_NAME)
- EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS)
- JOB (JOB_CLASS, CHG_HOUR)
- ASSIGNMENT (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)

- technique is the same, ...
- it is imperative that 2NF be achieved before moving on to 3NF;
- be certain to resolve partial dependencies before resolving transitive dependencies

Improving Design



Completed Database

Table name: ASSIGNMENT

Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-10	15	103	2.6	84.50	219.70
1002	04-Mar-10	18	118	1.4	18.36	25.70
1003	05-Mar-10	15	101	3.6	105.00	378.00
1004	05-Mar-10	22	113	2.5	48.10	120.25
1005	05-Mar-10	15	103	1.9	84.50	160.55
1006	05-Mar-10	25	115	4.2	96.75	406.35
1007	05-Mar-10	22	105	5.2	105.00	546.00
1008	05-Mar-10	25	101	1.7	105.00	178.50
1009	05-Mar-10	15	105	2.0	105.00	210.00
1010	06-Mar-10	15	102	3.8	96.75	367.65
1011	06-Mar-10	22	104	2.6	96.75	251.55
1012	06-Mar-10	15	101	2.3	105.00	241.50
1013	06-Mar-10	25	114	1.8	48.10	86.58
1014	06-Mar-10	22	111	4.0	26.87	107.48
1015	06-Mar-10	25	114	3.4	48.10	163.54
1016	06-Mar-10	18	112	1.2	45.95	55.14
1017	06-Mar-10	18	118	2.0	18.36	36.72
1018	06-Mar-10	18	104	2.6	96.75	251.55
1019	06-Mar-10	15	103	3.0	84.50	253.50
1020	07-Mar-10	22	105	2.7	105.00	283.50
1021	08-Mar-10	25	108	4.2	96.75	406.35
1022	07-Mar-10	25	114	5.8	48.10	278.98
1023	07-Mar-10	22	106	2.4	35.75	85.80

Table name: EMPLOYEE



Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joebrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Higher Level Normal Forms

- Tables in 3NF will perform suitably in business transactional databases.
- However, there are occasions when higher normal forms are useful
- special case of 3NF, known as Boyce-Codd normal form(BCNF)
- fourth normal form (4NF)

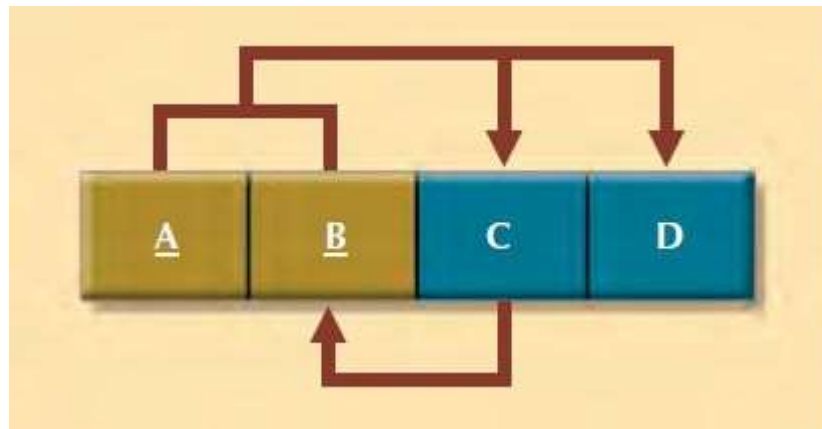
Boyce-Codd normal form (BCNF)

- when every determinant in table is candidate key
- when table contains only one candidate key, the 3NF and the BCNF are equivalent

Boyce-Codd normal form (BCNF)

- table is in 3NF when it is in 2NF and there are no transitive dependencies
- nonkey attribute is determinant of key attribute
- condition does not violate 3NF, yet it fails to meet the BCNF requirements because BCNF requires that every determinant in the table be candidate key

table that is in 3NF but not in BCNF



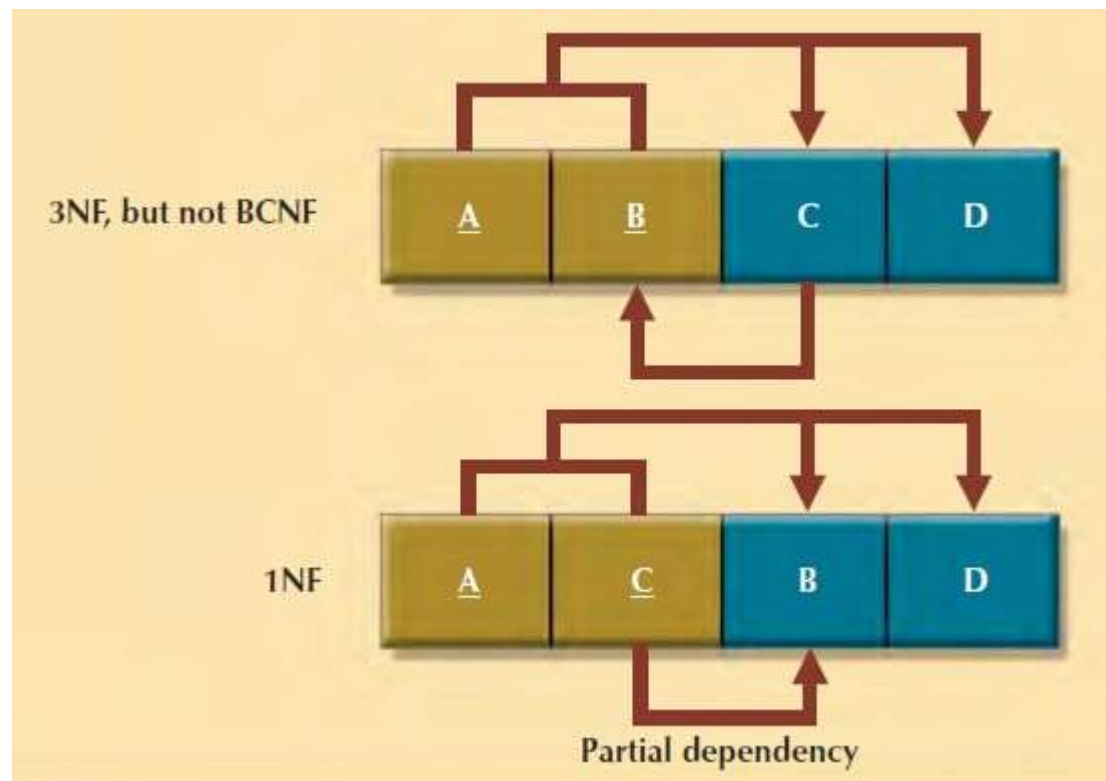
functional dependencies

- $A + B \rightarrow C, D$
- $A + C \rightarrow B, D$
- $C \rightarrow B$
- two candidate keys: $(A + B)$ and $(A + C)$
- has no partial dependencies, nor does it contain transitive dependencies
- $C \rightarrow B$ indicates that nonkey attribute determines part of the PK and that dependency is not transitive or partial because dependent is prime attribute

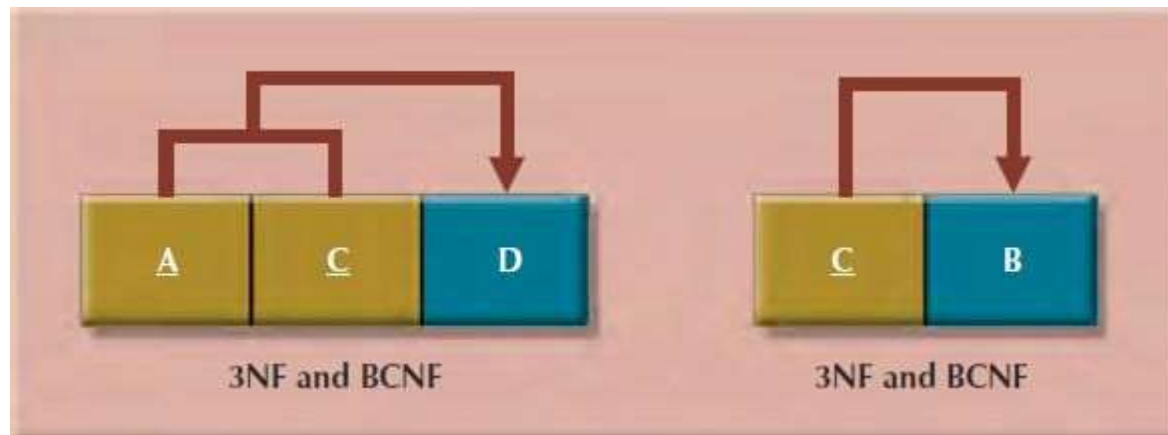
convert into BCNF

- change the primary key to $A + C$
- appropriate action because dependency $C \rightarrow B$ means that C is, in effect, superset of B
- at this point, table is in 1NF because it contains partial dependency, $C \rightarrow B$
- next, follow standard decomposition procedures to produce results

Decomposition to BCNF



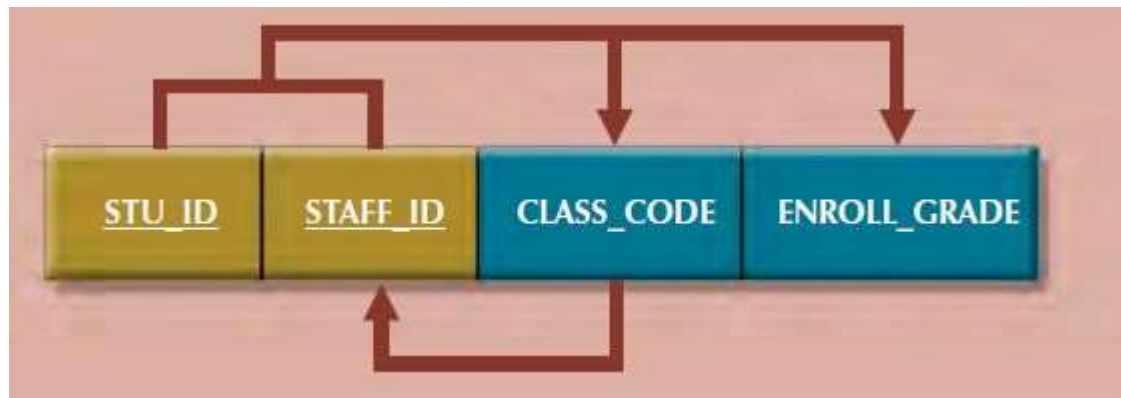
Decomposition to BCNF



Sample Data for a BCNF Conversion

STU_ID	STAFF_ID	CLASS_CODE	ENROLL_GRADE
125	25	21334	A
125	20	32456	C
135	20	28458	B
144	25	27563	C
144	20	32456	B

- each CLASS_CODE identifies class uniquely, course might generate many classes
 - for example, course labeled INFS 420 might be taught in two classes (sections), each identified by unique code to facilitate registration
 - CLASS_CODE 32456 might identify INFS 420, class section 1
 - CLASS_CODE 32457 might identify INFS 420, class section 2
 - CLASS_CODE 28458 might identify QM 362, class section 5
- student can take many classes
- staff member can teach many classes, but each class is taught by only one staff member



- $\text{STU_ID} + \text{STAFF_ID} \rightarrow \text{CLASS_CODE}, \text{ENROLL_GRADE}$
- $\text{CLASS_CODE} \rightarrow \text{STAFF_ID}$

- table represented by this structure has major problem, because it is trying to describe two things: staff assignments to classes and student enrollment information
- dual-purpose table structure will cause anomalies
- if different staff member is assigned to teach class 32456, two rows will require updates, thus producing an update anomaly
- if student 135 drops class 28458, information about who taught that class is lost, thus producing a deletion anomaly



forth normal form (4NF)

- multivalued attributes exist
- Consider possibility that an employee can have multiple assignments and can also be involved in multiple service organizations
- Suppose employee 10123 does volunteer work for Red Cross and United Way; same employee might be assigned to work on three projects: 1, 3, and 4

Table name: VOLUNTEER_V1

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	UW	3
10123		4

Table name: VOLUNTEER_V3

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	RC	3
10123	UW	4

Table name: VOLUNTEER_V2

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	
10123	UW	
10123		1
10123		3
10123		4

- attributes ORG_CODE and ASSIGN_NUM each may have many different values
- situation is referred to as multivalued dependency
- one key determines multiple values of two other attributes and those attributes are independent of each other
- one employee can have many service entries and many assignment entries
- one EMP_NUM can determine multiple values of ORG_CODE and multiple values of ASSIGN_NUM
- ORG_CODE and ASSIGN_NUM are independent

- if versions 1 and 2 are implemented, tables are likely to contain quite a few null values; do not even have a viable candidate keyv
- version 3 at least has a PK, but it is composed of all of attributes in the table; meets 3NF requirements, yet it contains many redundancies that are clearly undesirable

set of tables in 4NF

Table name: PROJECT

PROJ_CODE	PROJ_NAME	PROJ_BUDGET
1	BeThere	1023245.00
2	BlueMoon	20198608.00
3	GreenThumb	3234456.00
4	GoFast	5674000.00
5	GoSlow	1002500.00

Table name: ASSIGNMENT

ASSIGN_NUM	EMP_NUM	PROJ_CODE
1	10123	1
2	10121	2
3	10123	3
4	10123	4
5	10121	1
6	10124	2
7	10124	3
8	10124	5

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME
10121	Rogers
10122	O'Leery
10123	Panera
10124	Johnson

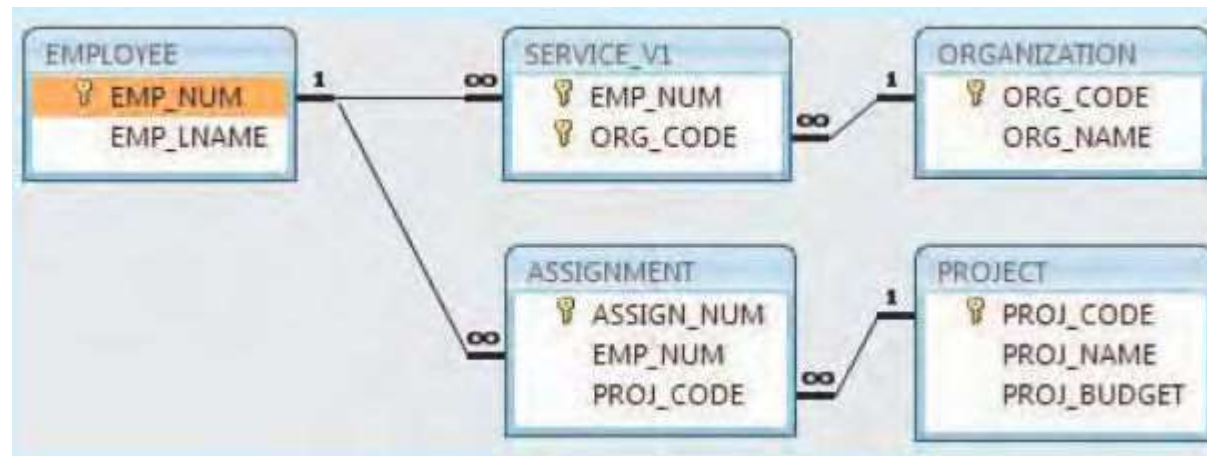
Table name: ORGANIZATION

ORG_CODE	ORG_NAME
RC	Red Cross
UW	United Way
WF	Wildlife Fund

Table name: SERVICE_V1

EMP_NUM	ORG_CODE
10123	RC
10123	UW
10123	WF

relational diagram



fourth normal form (4NF)

- table is in fourth normal form (4NF) when it is in 3NF and has no multivalued dependencies
- *Normalization should be part of the design process*

Data-Modeling Checklist

Business rules

- Properly document and verify all business rules with the end users.
- Ensure that all business rules are written precisely, clearly, and simply. The business rules must help identify entities, attributes, relationships, and constraints.
- Identify the source of all business rules, and ensure that each business rule is justified, dated, and signed off by an approving authority.

Naming Conventions

- All names should be limited in length (database-dependent size).
- Entity Names:
 - Should be nouns that are familiar to business and should be short and meaningful
 - Should document abbreviations, synonyms, and aliases for each entity
 - Should be unique within the model
 - For composite entities, may include a combination of abbreviated names of the entities linked through the composite entity

Naming Conventions

- Attribute Names:
 - Should be unique within the entity
 - Should use the entity abbreviation as a prefix
 - Should be descriptive of the characteristic
 - Should use suffixes such as `_ID`, `_NUM`, or `_CODE` for the PK attribute
 - Should not be a reserved word
 - Should not contain spaces or special characters such as `@`, `!`, or `&`

Naming Conventions

- Relationship Names:
 - Should be active or passive verbs that clearly indicate the nature of the relationship

Entities

- Each entity should represent a single subject.
- Each entity should represent a set of distinguishable entity instances.
- All entities should be in 3NF or higher. Any entities below 3NF should be justified.
- The granularity of the entity instance should be clearly defined.
- The PK should be clearly defined and support the selected data granularity.

Attributes

- Should be simple and single-valued (atomic data)
- Should document default values, constraints, synonyms, and aliases
- Derived attributes should be clearly identified and include source(s)
- Should not be redundant unless this is required for transaction accuracy, performance, or maintaining a history
- Nonkey attributes must be fully dependent on the PK attribute

Relationships

- Should clearly identify relationship participants
- Should clearly define participation, connectivity, and document cardinality

ER Model

- Should be validated against expected processes: inserts, updates, and deletes
- Should evaluate where, when, and how to maintain a history
- Should not contain redundant relationships except as required (see attributes)
- Should minimize data redundancy to ensure single-place updates
- Should conform to the minimal data rule: *“All that is needed is there, and all that is there is needed.”*

DataBase

Database Design

Information System

- Databases are a part of a larger picture called an information system
- database designers must recognize that database is a critical means to an end rather than an end in itself
 - managers want the database to serve their management needs
 - many databases seem to require that managers alter their routines to fit database requirements

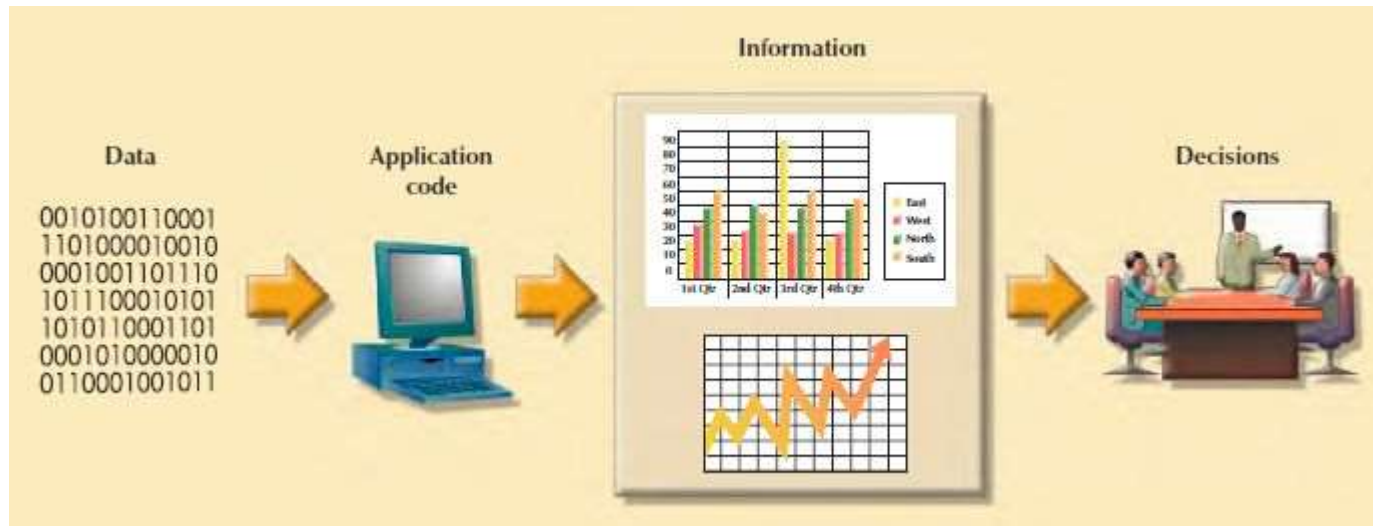
Systems Development Life Cycle (SDLC)

- creation and evolution of information systems follows an iterative pattern called the Systems Development Life Cycle (SDLC)
- continuous process of creation, maintenance, enhancement, and replacement of the information system
- similar cycle applies to databases

Information System

- facilitates the transformation of data into information
- it allows for management of both data and information
- composed of people, hardware, software, database(s), application programs, and procedures
- Systems analysis
 - process that establishes need for and extent of an information system
- Systems development
 - process of creating an information system
- Purpose

Generating information for decision making



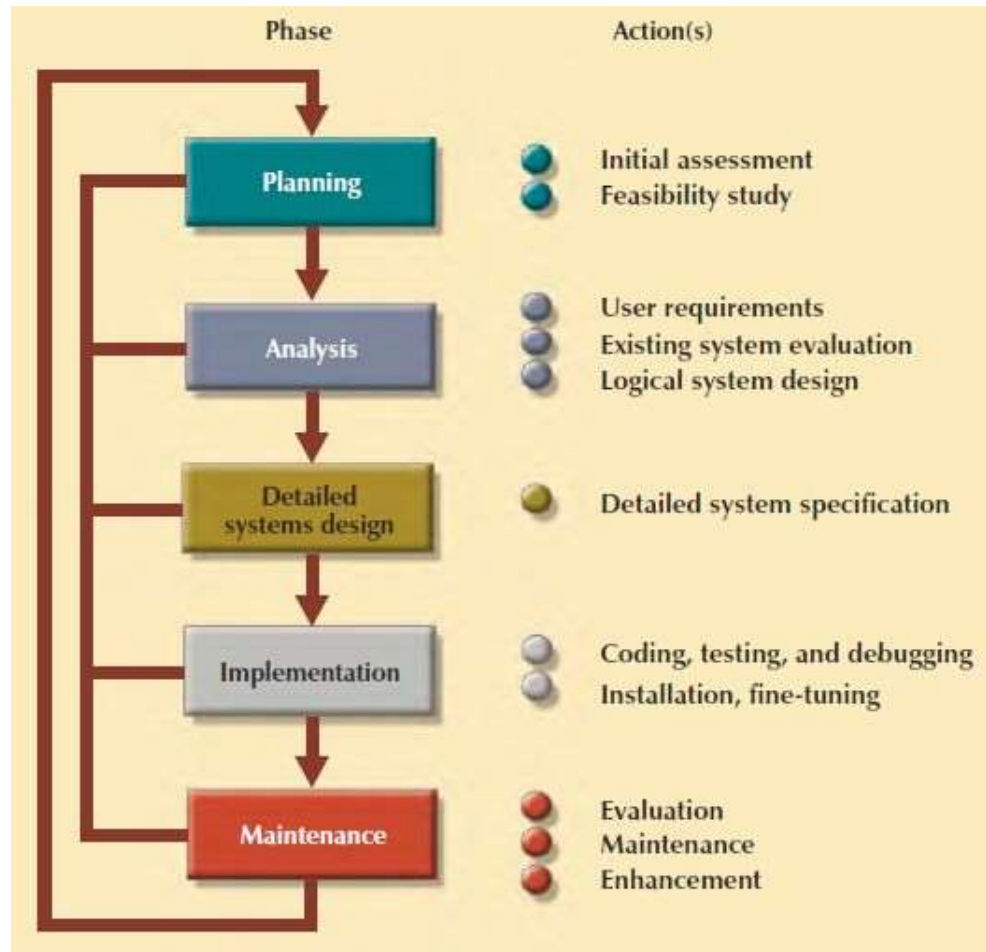
Performance of information system

- Database design and implementation
- Application design and implementation
- Administrative procedures

Systems Development Life Cycle

- general framework through which you can track and understand activities required to develop and maintain information systems
- there are alternative methodologies, such as:
 - Unified Modeling Language (UML)
 - Rapid Application Development (RAD)
 - Agile Software Development

Systems Development Life Cycle



Systems Development Life Cycle

- traces the history (life cycle) of an information system
- traditional is divided into five phases
 - planning, analysis, detailed systems design, implementation, and maintenance
- iterative process, rather than sequential

Planning

- yields a general overview of objectives
- Should the existing system be continued?
- Should the existing system be modified?
- Should the existing system be replaced?

Feasibility study

- technical aspects of hardware and software requirements
- system cost
- operational cost
 - human, technical, and financial resources to keep system operational

Analysis

- problems defined during planning phase are examined in greater detail of both individual needs and organizational needs
- What are the requirements of the current system's end users?
- Do those requirements fit into the overall information requirements?
- Is a thorough audit of user requirements

Detailed Systems Design

- completes the design of the system's processes
- includes all necessary technical specifications for screens, menus, reports, and other
- steps are laid out for conversion from the old to the new system
- training principles and methodologies are also planned and must be submitted for management's approval

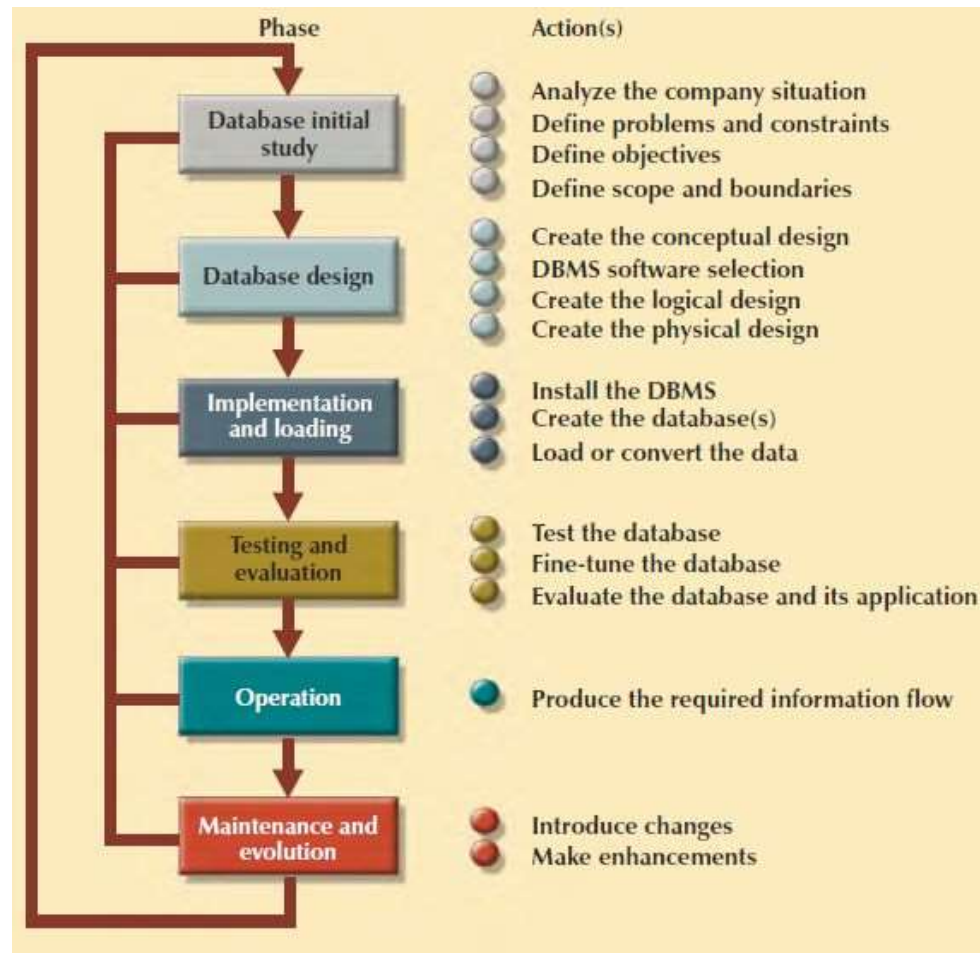
Implementation

- hardware, software, DBMS and application programs are installed
- database design is implemented
- system enters into cycle of coding, testing, and debugging until it is ready to be delivered
- database contents loaded
 - customized user programs
 - database interface programs.
 - conversion programs
- system is subjected to exhaustive testing until it is ready for use
- final documentation is reviewed and printed
- end users are trained
- system is in full operation
- will be continuously evaluated and fine-tuned

Maintenance

- almost as soon as system is operational, end users begin to request changes
- corrective maintenance in response to systems errors
- adaptive maintenance due to changes in the business environment
- perfective maintenance to enhance the system

Database Life Cycle



Database Life Cycle

- within larger information system, database is subject to a life cycle
- contains six phases:
 - database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution

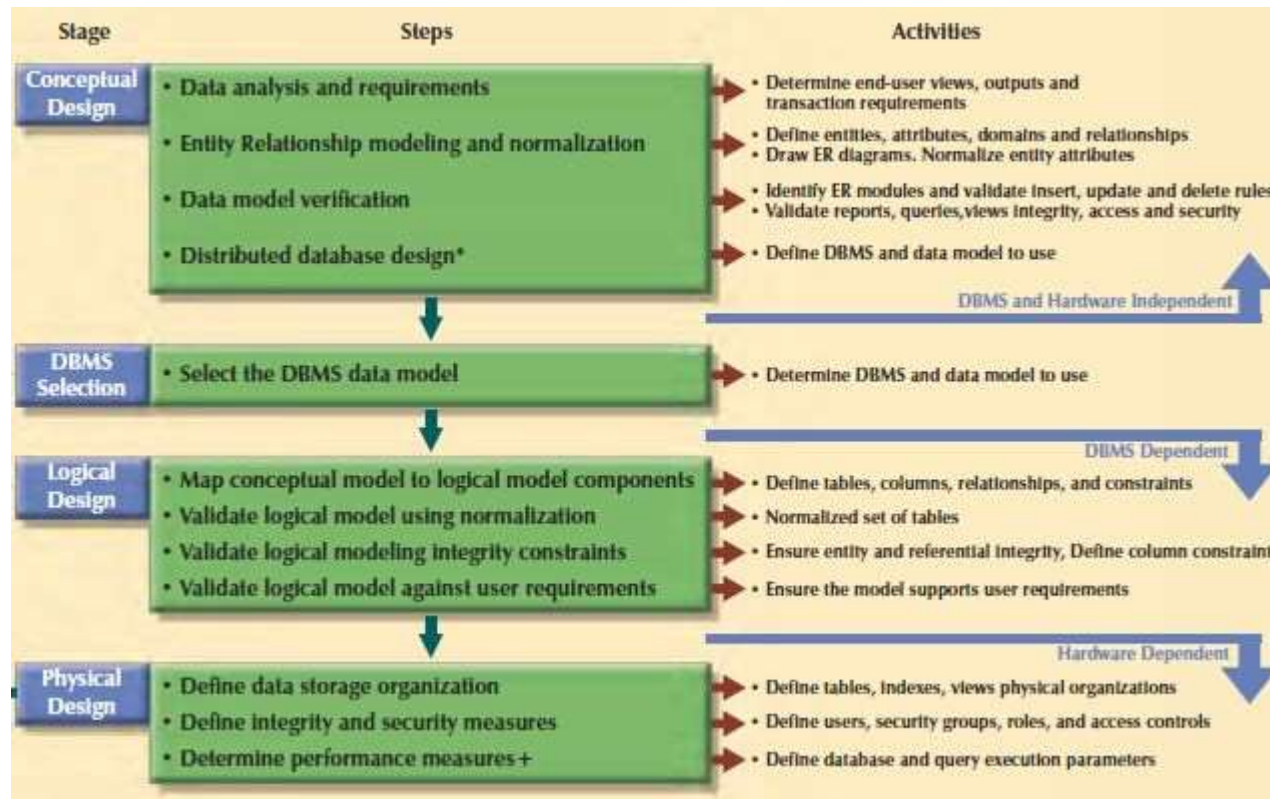
Database Initial Study

- examine current system's operation
 - determine how and why current system fails
- database design is a technical business,
- it is also people-oriented
- overall purpose of database initial study is to:
 - analyze company situation
 - define problems and constraints
 - define objectives
 - define scope and boundaries

Database Design

- arguably the most critical DBLC phase
 - making sure that final product meets user and system requirements
- business view of data as a source of information
- designer's view of data structure, its access, and activities required to transform data into information

Database design process



Implementation and Loading

- output of database design phase is a series of instructions detailing creation of tables, attributes, domains, views, indexes, security constraints, and storage and performance guidelines
- actually implement all these design specifications
- install DBMS
- create database(s)
- load or convert data
 - typically, data migrated from prior version of system

Testing and Evaluation

- occurs in conjunction with applications programming
- ensure integrity, security, performance, and recoverability of the database
- evaluate the database and its application programs

Testing and Evaluation

- physical security allows only authorized personnel physical access to specific areas
- password security allows assignment of access rights to specific authorized users
- access rights can be established through the use of database software
 - restrict operations (CREATE, UPDATE, DELETE, and so on) on predetermined objects such as databases, tables, views, queries, and reports
- audit trails are usually provided by DBMS to check for access violations
- data encryption can be used to render data useless to unauthorized users who might have violated some of database security layers

Fine-Tune database

- different systems will place different performance requirements on database
- database performance tuning and query optimization

Common sources of database failure

- Software
 - traceable to operating system, DBMS software, application programs, or viruses
- Hardware
 - may include memory chip errors, disk crashes, bad disk sectors, and “*disk full*” errors
- Programming exemptions
 - application programs or end users may roll back transactions when certain conditions are defined
 - caused by malicious or improperly tested code

Common sources of database failure

- Transactions
 - system detects deadlocks and aborts one of transactions
- External factors
 - backups are especially important when a system suffers complete destruction from natural disaster

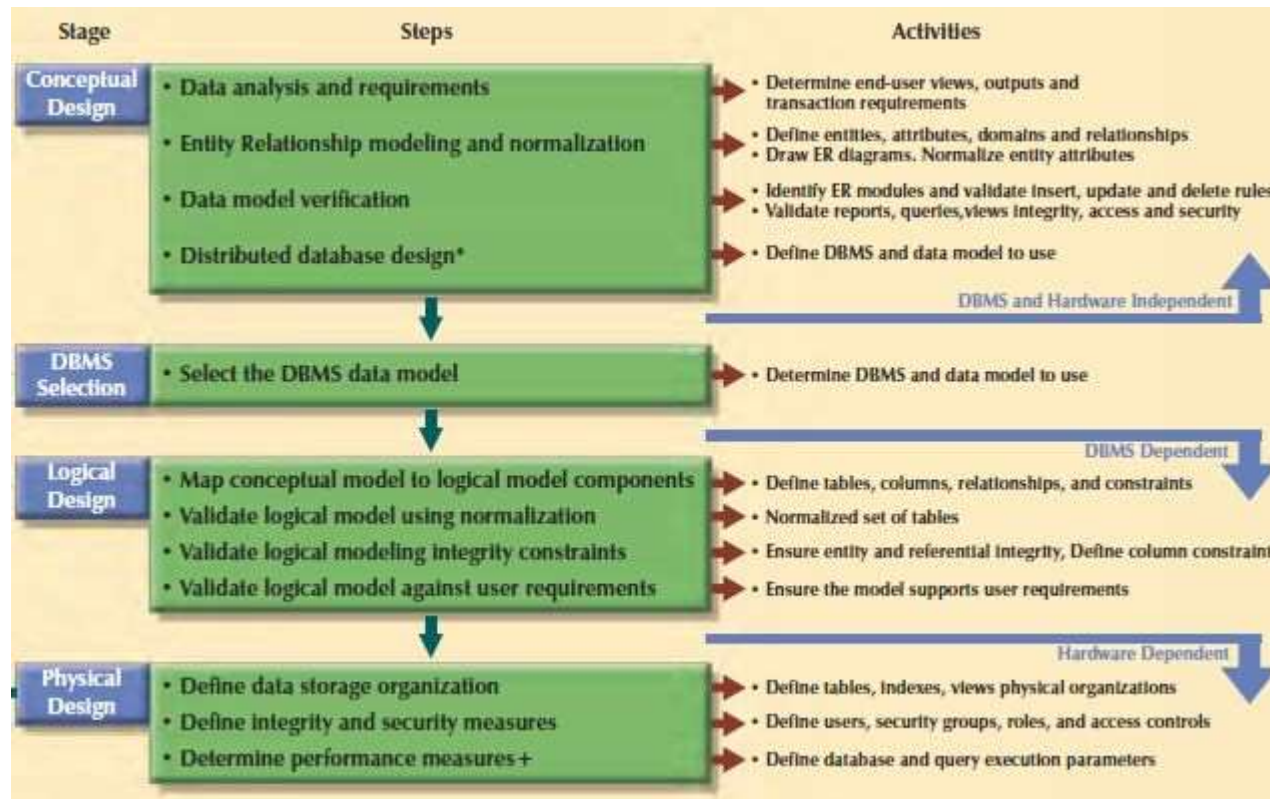
Operation

- database passed evaluation stage, it is considered to be operational
- database, management, users, application programs constitute a complete information system

Maintenance and Evolution

- DBA perform routine maintenance activities
- preventive maintenance (backup)
- corrective maintenance (recovery)
- adaptive maintenance
 - enhancing performance, add entities and attributes ...
- assignment of access permissions for new and old users
- generation of database access statistics to improve efficiency of system audits , to monitor system performance
- periodic security audits, system-usage summaries for internal billing or budgeting purposes

Database design process



Conceptual design

- design database that is independent of database software and physical details
- output of this process is a conceptual data model that describes the main data entities, attributes, relationships, and constraints of a given problem domain - descriptive and narrative in form
- generally composed of a graphical representation
- textual descriptions of main data elements, relationships, and constraints

Conceptual design

- Data analysis and requirements
- Entity relationship modeling and normalization
- Data model verification
- Distributed database design

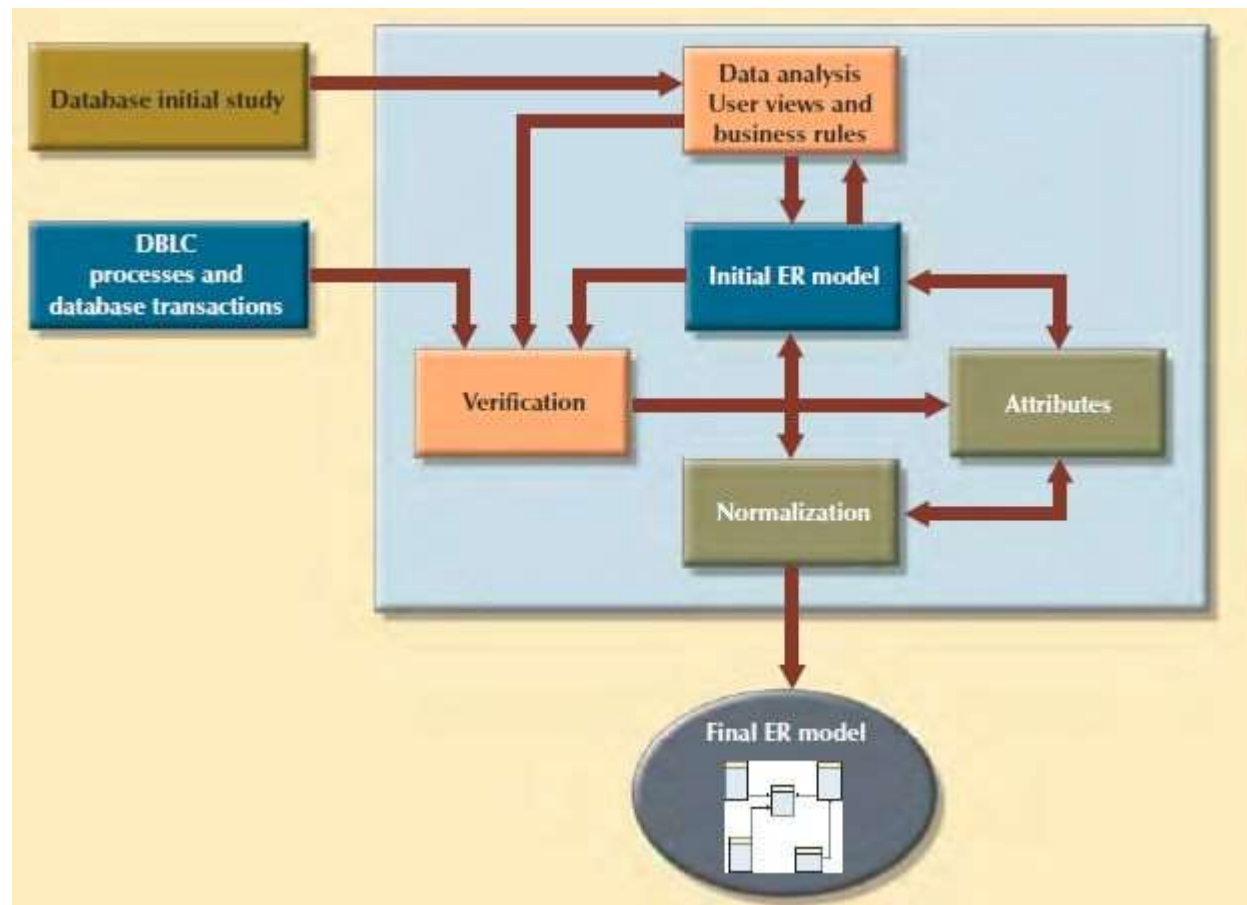
Data analysis and requirements

- information needs
- information users
- information sources
- information constitution
 - data elements needed to produce information

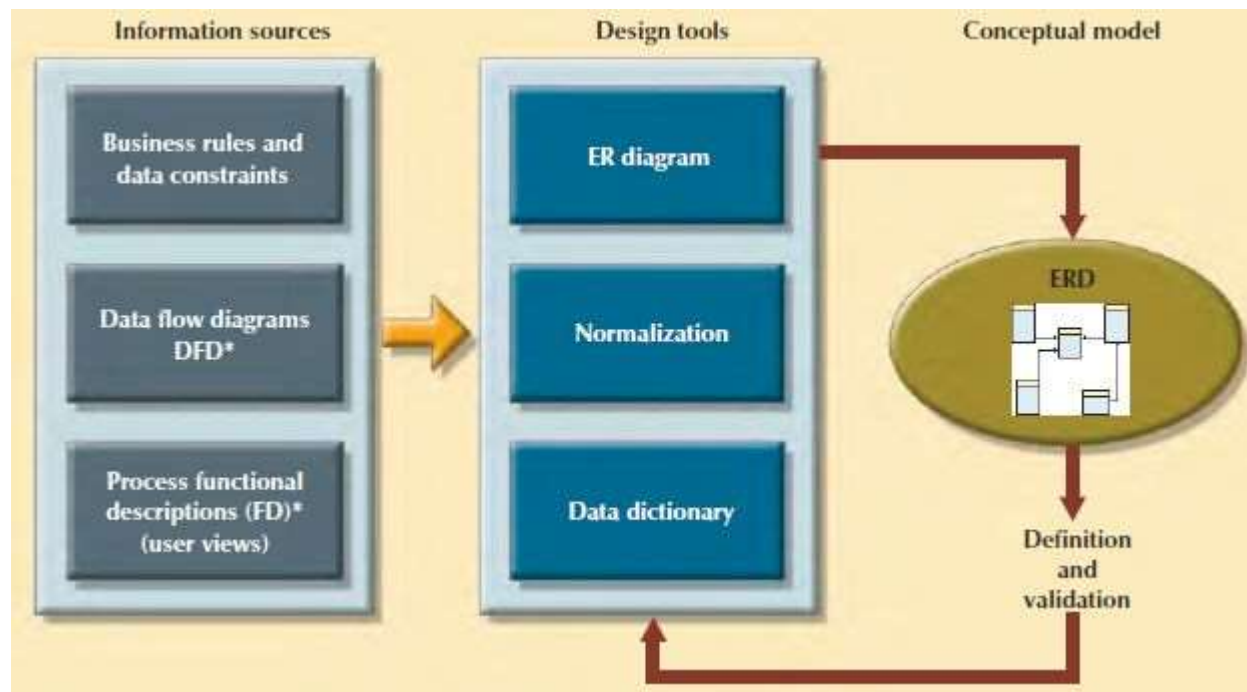
Entity relationship modeling and normalization

1. identify, analyze, and refine business rules
2. identify main entities
3. define relationships among entities
4. define attributes, primary keys, and foreign keys for each of the entities
5. normalize the entities
6. complete initial ER diagram
7. validate ER model against end users' information and processing requirements
8. modify ER model

iterative process based on many activities



information sources



Data model verification

1. identify the ER model's central entity.
2. identify each module and its components.
3. identify each module's transaction requirements:
 1. Internal: Updates/Inserts/Deletes/Queries/Reports
 2. External: Module interfaces
4. verify all processes against system requirements
5. make all necessary changes suggested
6. repeat Steps 2–5 for all modules

Distributed database design

- although not a requirement for most databases
- sometimes database may need to be distributed among multiple geographically disperse locations
 - processes that access database may also vary from one location to another
- if database data and processes are to be distributed across system, portions of a database, known as database fragments, may reside in several physical locations
- database fragment
 - subset of database that is stored at given location
 - composed of subset of rows or columns from one or multiple tables

DBMS Software Selection

- Common factors affecting purchasing decision:
 - cost
 - DBMS features and tools
 - underlying model
 - Portability
 - DBMS hardware requirements

Logical design

- requires that all objects in conceptual model be mapped to specific constructs used by selected database model
- includes specifications for relations (tables), relationships, and constraints (i.e., domain definitions, data validations, and security views)

Logical design

1. map conceptual model to logical model components
2. validate logical model using normalization
3. validate logical model integrity constraints
4. validate logical model against user requirements

Mapping conceptual model to relational model

1. map strong entities
2. map supertype/subtype relationships
3. map weak entities
4. map binary relationships
5. map higher degree relationships

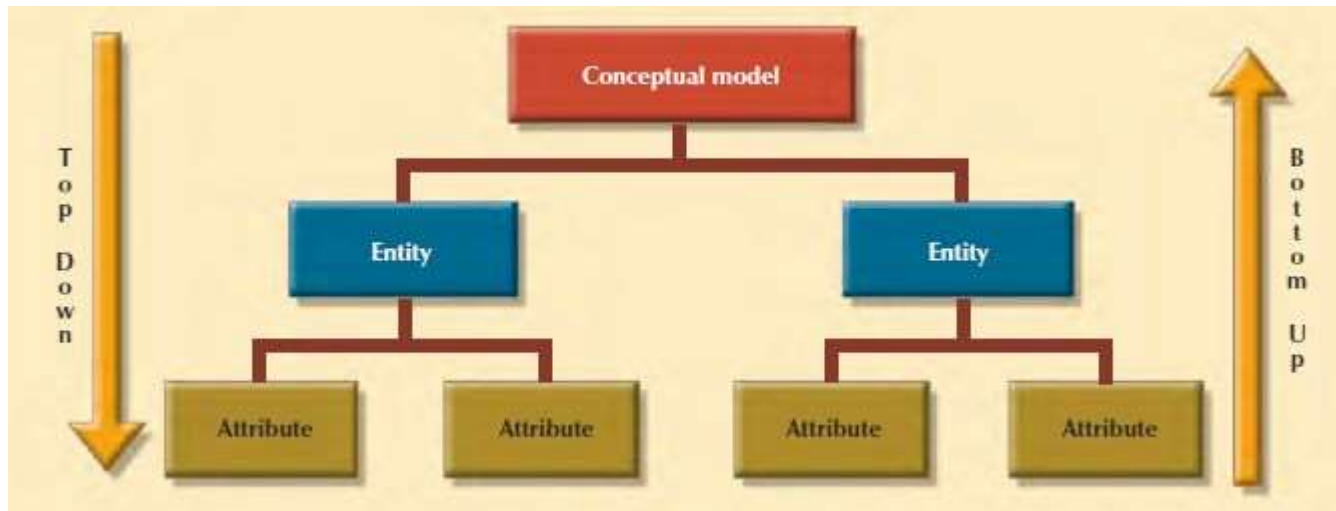
Physical design

- process of determining data storage organization and data access characteristics
- storage characteristics are function of types of devices supported by hardware, type of data access methods supported by system, and the DBMS
- relational databases are more insulated from physical details than older hierarchical and network models

Database Design Strategies

- top-down design
 - starts by identifying data sets and then defines data elements for each of those sets
 - involves identification of different entity types and definition of each entity's attributes
- bottom-up design
 - first identifies data elements (items) and then groups them together in data sets
 - first defines attributes, and then groups them to form entities

Top-down vs. bottom-up design sequencing



Centralized design

- productive when data component is composed of a relatively small number of objects and procedures
- can be successfully done by single person or by small, informal design team
- operations and scope of problem are sufficiently limited

Decentralized design

- might be used when data component of the system has a considerable number of entities and complex relations on which very complex operations are performed
- likely to be employed when problem itself is spread across several operational sites and each element is a subset of entire data set

Aggregation problems

- synonyms and homonyms
 - various departments might know same object by different names (synonyms), or they might use same name to address different objects (homonyms) - object can be entity, attribute, or relationship
- entity and entity subtypes
 - entity subtype might be viewed as separate entity by one or more departments
 - must integrate such subtypes into higher-level entity
- conflicting object definitions
 - attributes can be recorded as different types (character, numeric), or different domains can be defined for same attribute
 - constraint definitions can vary
 - must remove such conflicts from model

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”

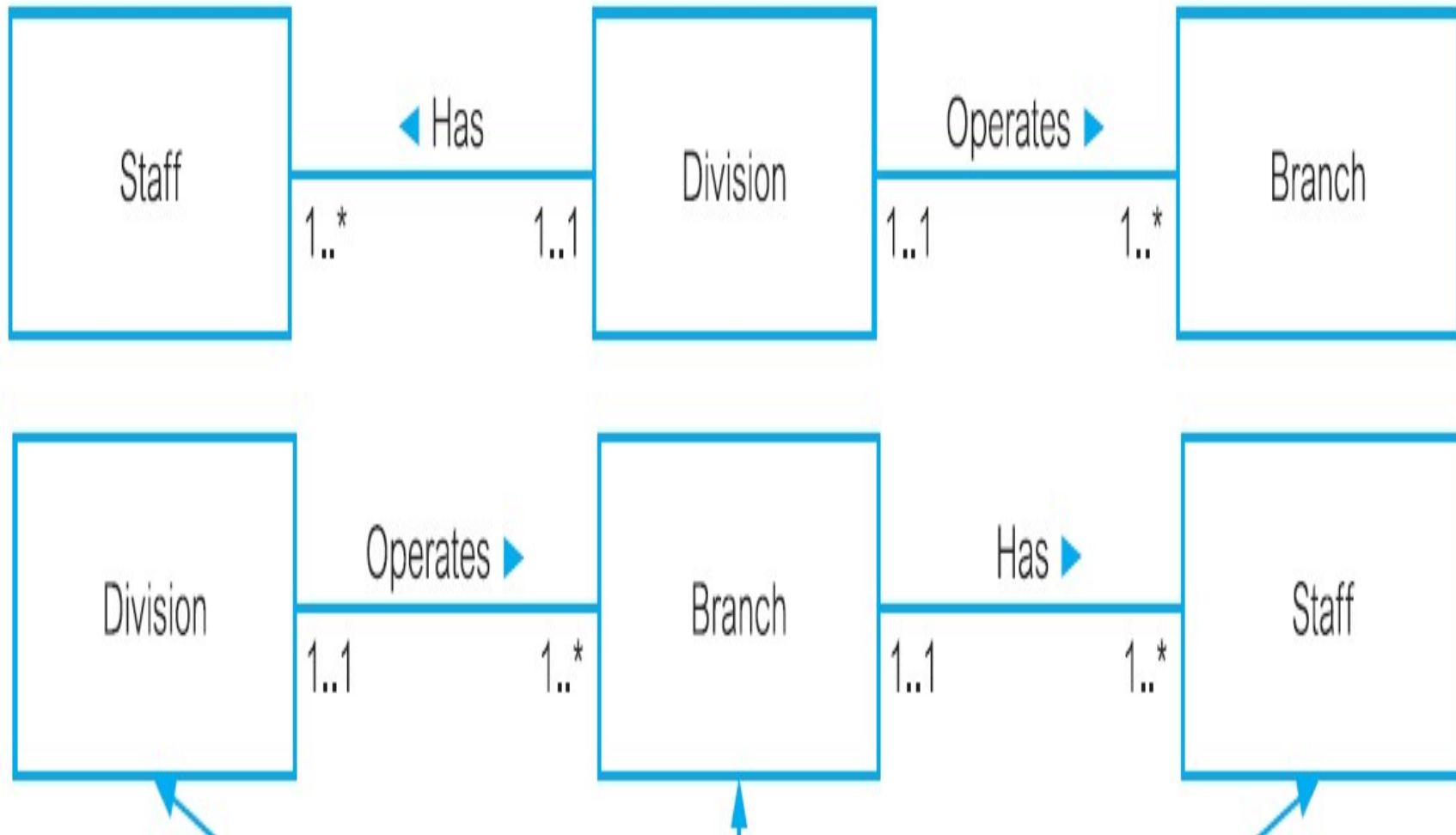
Fan Trap

- 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

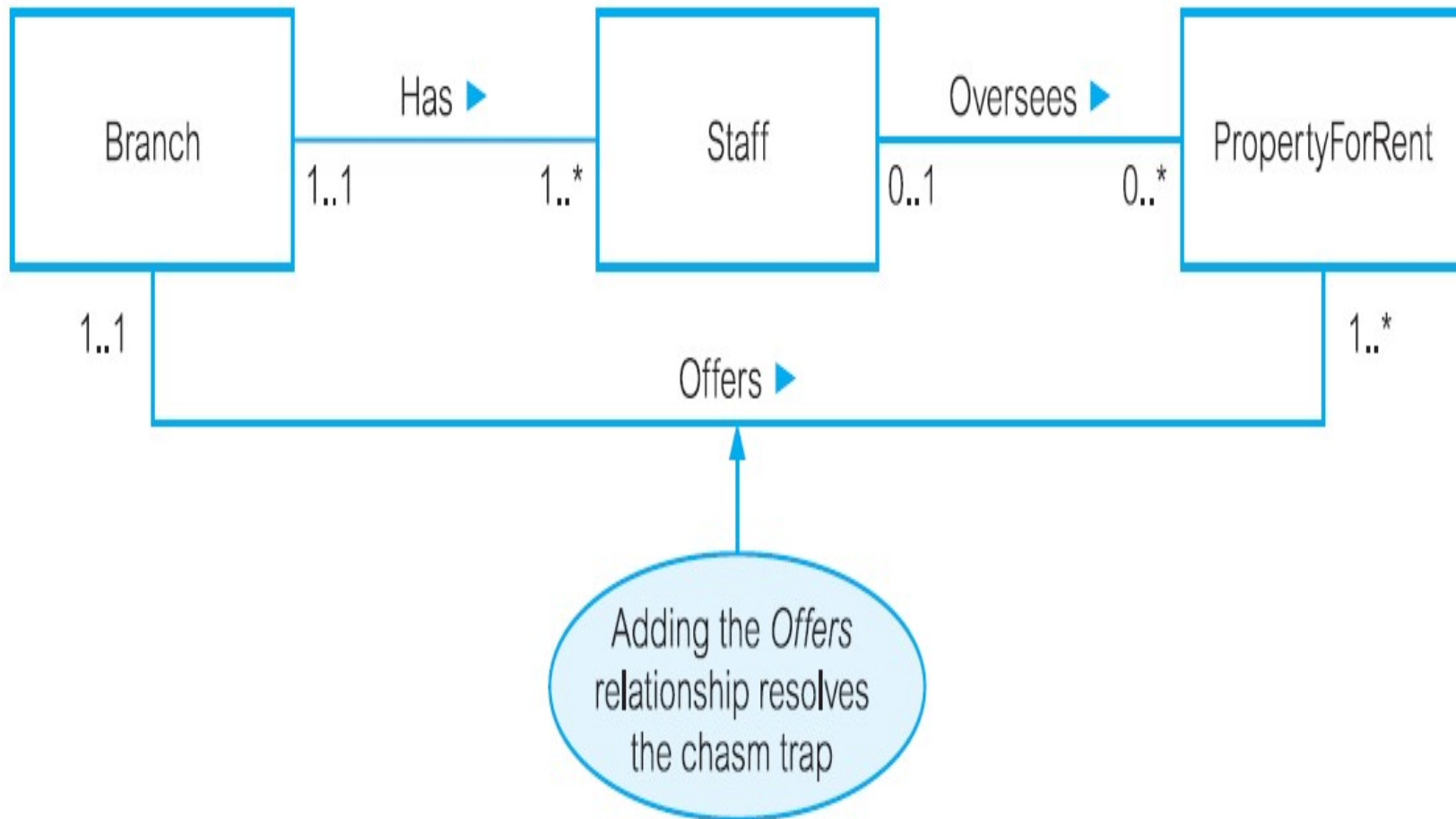
Chasm Trap

- 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*)

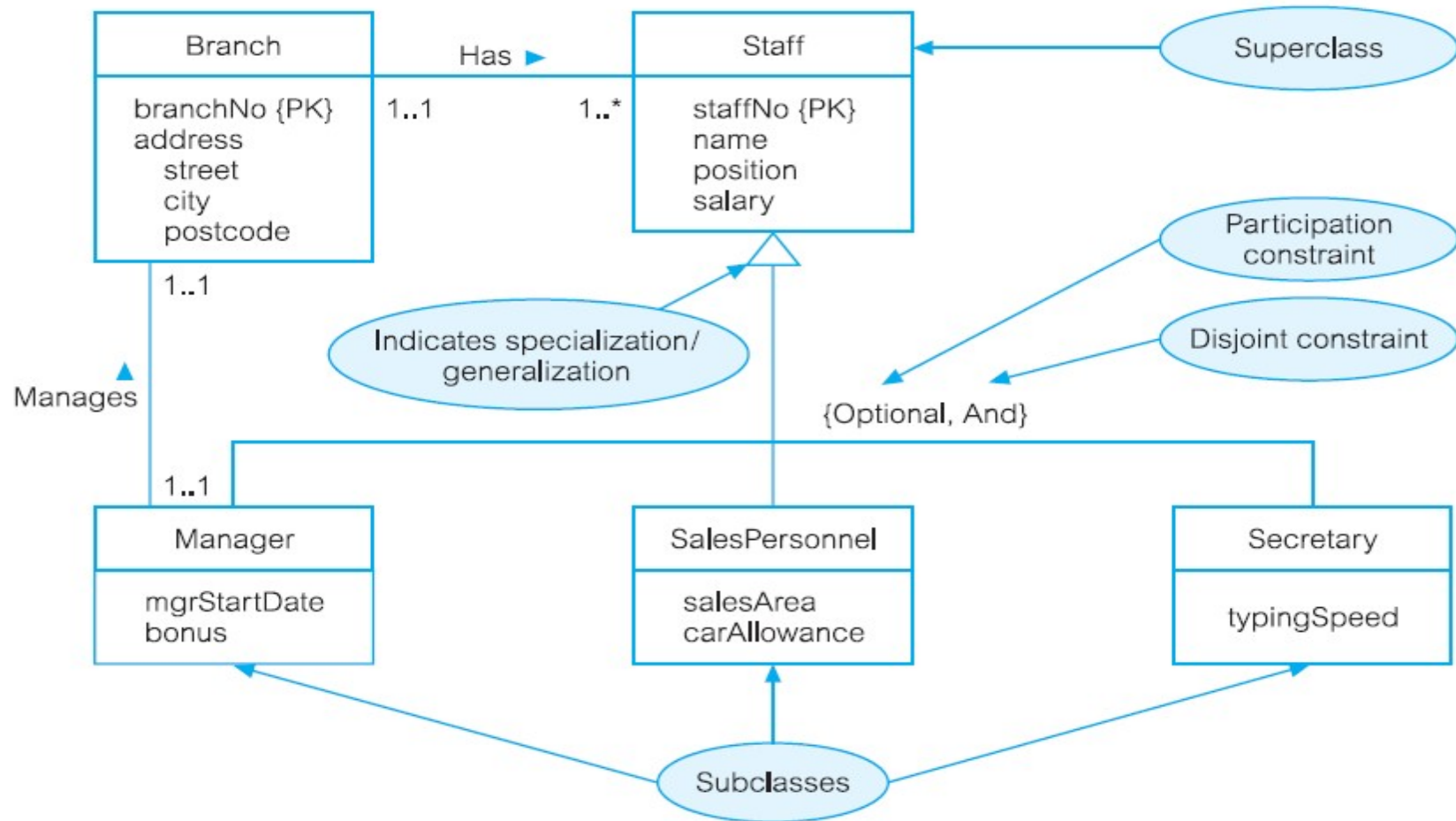
Specialization process

- 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

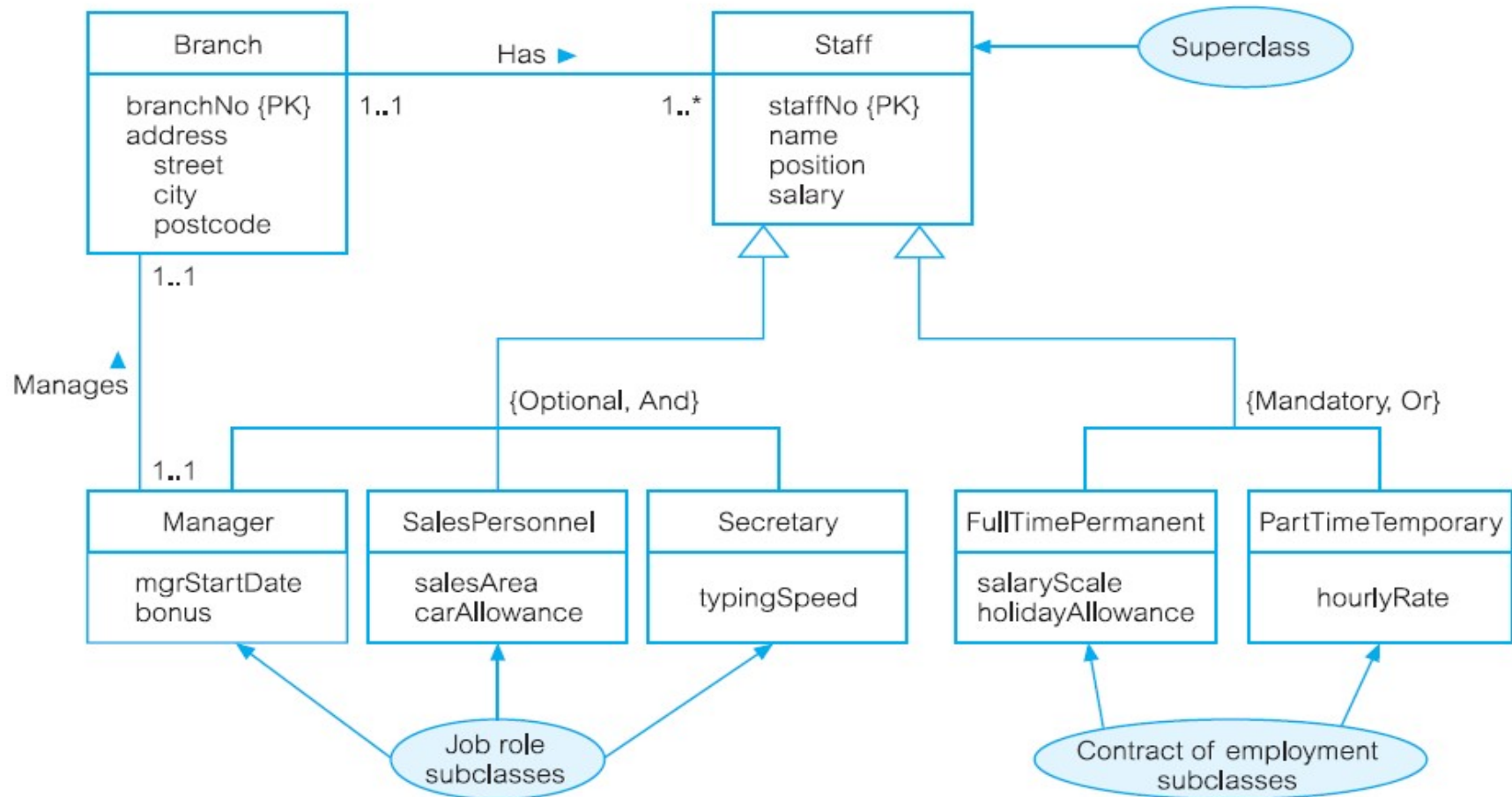


- may have several specializations of same entity based on different distinguishing characteristics

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**

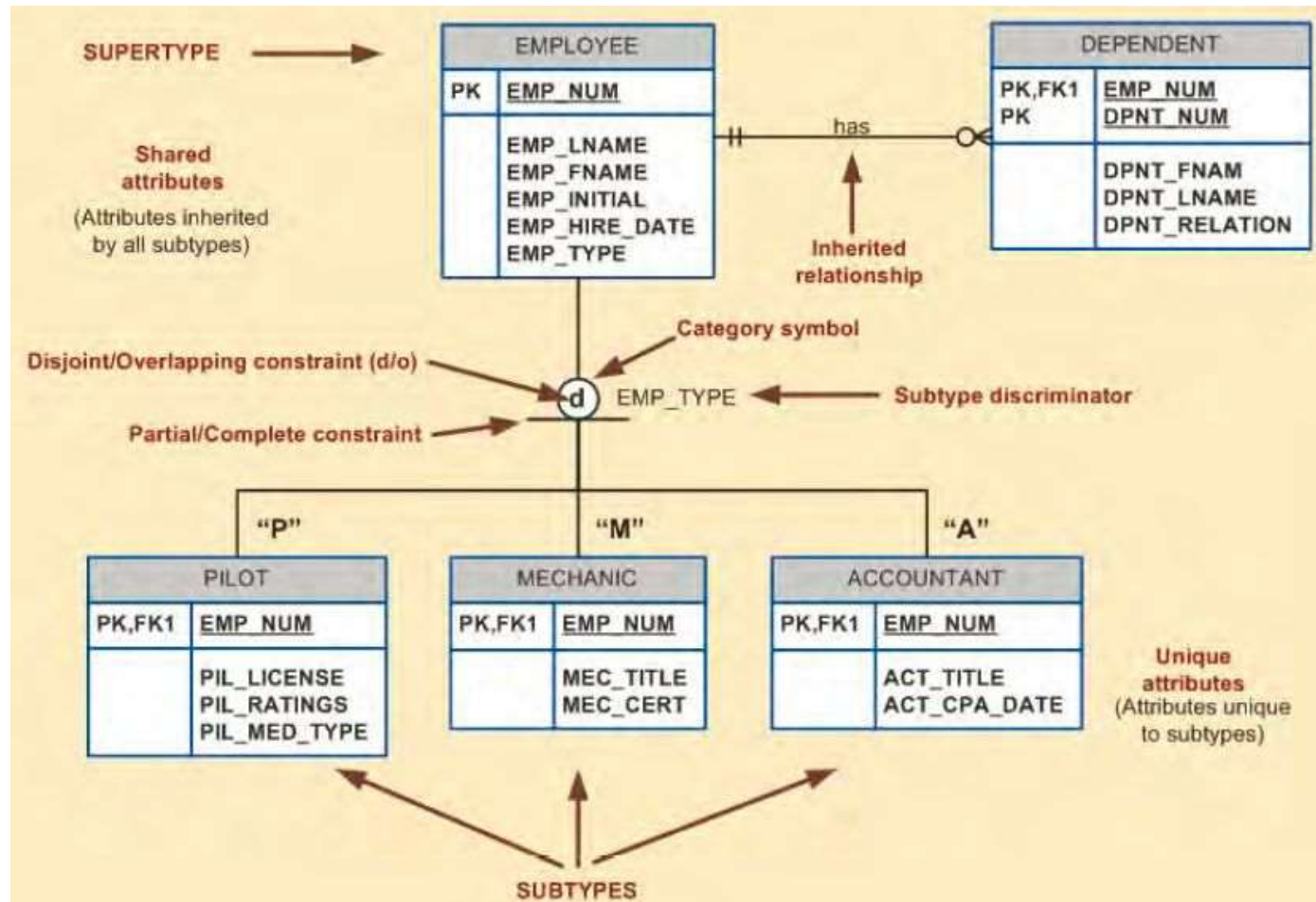
Participation constraints

- 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

specialization hierarchy



case of an aviation business

- employs pilots, mechanics, secretaries, accountants, database managers, and many others
- pilots share certain characteristics with other employees
 - such as last name (EMP_LNAME) and hire date (EMP_HIRE_DATE)
- unlike other employees, pilots must meet special requirements such as flight hour restrictions, flight checks, and periodic training
- if all employee characteristics and special qualifications were stored in a single EMPLOYEE entity, you would have a lot of nulls

- based on hierarchy conclude that PILOT is subtype of EMPLOYEE, and that EMPLOYEE is supertype of PILOT
- entity supertype is generic (more general) entity type that is related to one or more specific (specialized) entity subtypes
- entity supertype contains common characteristics
- entity subtypes contain unique characteristics of each entity subtype
- There must be different, identifiable kinds or types of the entity in the user's environment.
- The different kinds or types of instances should have one or more attributes that are unique to that kind or type

- pilots meet both criteria of being an identifiable kind of employee and having unique attributes that other employees do not possess, it is appropriate to create PILOT as a subtype of EMPLOYEE
- CLERK would not be an acceptable subtype of EMPLOYEE because it only satisfies one of the criteria—it is an identifiable kind of employee—but there are not any attributes that are unique to just clerks

Specialization Hierarchy

- Entity supertypes and subtypes are organized in a *specialization hierarchy*
- specialization hierarchy formed by an EMPLOYEE supertype and three entity subtypes—PILOT, MECHANIC, and ACCOUNTANT
- 1:1 relationship between EMPLOYEE and its subtypes
 - for example PILOT subtype occurrence is related to one instance of the EMPLOYEE

- relationships depicted within specialization hierarchy are sometimes described in terms of “is-a” relationships
- for example pilot is an employee
- subtype can exist only within the context of a supertype
- subtype can have only one supertype to which it is directly related
- specialization hierarchy can have many levels of supertype/subtype relationships

Inheritance

- enables entity subtype to inherit attributes and relationships of supertype
- pilots, mechanics, and accountants all inherit employee number, last name, first name, middle initial, hire date, and so on from EMPLOYEE entity
- pilots have attributes that are unique; same is true for mechanics and accountants
- all entity subtypes inherit their primary key attribute from their supertype
 - EMP_NUM attribute is the primary key for each of the subtypes

EMPLOYEE-PILOT

supertype-subtype relationship

Table Name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIRE_DATE	EMP_TYPE
100	Koistycz	Xavier	T	15-Mar-88	
101	Lewis	Marion		25-Apr-89	P
102	Vandem	Jean		20-Dec-93	A
103	Jones	Victoria	R	28-Aug-03	
104	Lange	Edith		20-Oct-97	P
105	Williams	Gabriel	U	08-Nov-97	P
106	Duzak	Mario		05-Jan-04	P
107	Drante	Venite	L	02-Jul-97	M
108	Wesenberg	Joni		18-Nov-95	M
109	Travis	Brett	T	14-Apr-01	P
110	Genlazi	Stan		01-Dec-03	A

Table Name: PILOT

EMP_NUM	PILOT_LICENSE	PILOT_RATINGS	PILOT_MED_TYPE
101	ATP	SEL/MEL/Instr/CFI	1
104	ATP	SEL/MEL/Instr	1
105	COM	SEL/MEL/Instr/CFI	2
106	COM	SEL/MEL/Instr	2
109	COM	SEL/MEL/SES/Instr/CFI	1

Subtype Discriminator

- attribute in supertype entity that determines to which subtype the supertype occurrence is related
- common practice to show subtype discriminator and its value for each subtype in the ER diagram

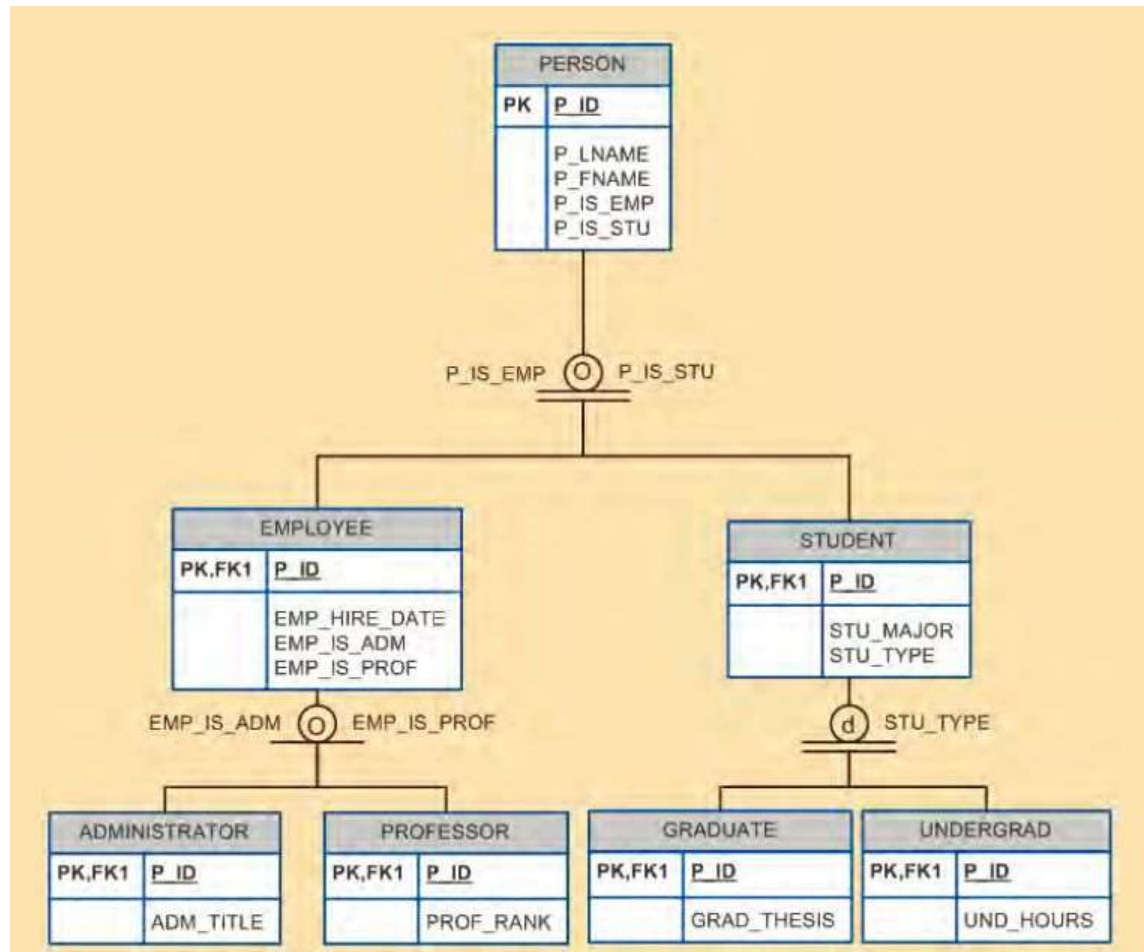
Disjoint or overlapping entity subtypes

- Employee can be a pilot or a mechanic or an accountant
- business rules dictates that employee cannot belong to more than one subtype at a time
- Disjoint subtypes, also known as non-overlapping subtypes, are subtypes that contain unique subset of the supertype entity set

Overlapping subtypes

- if business rule specifies that employees can have multiple classifications, EMPLOYEE supertype may contain overlapping job classification subtypes
- contain non-unique subsets of the supertype entity set
- for example, in university environment, person may be an employee or student or both


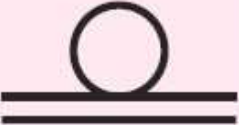
Specialization hierarchy with overlapping subtypes



Completeness constraint

- specifies whether each entity supertype occurrence must also be a member of at least one subtype
- partial completeness means that not every supertype occurrence is a member of a subtype
 - there may be some supertype occurrences that are not members of any subtype
- total completeness means that every supertype occurrence must be a member of at least one subtype

Specialization Hierarchy Constraint Scenarios

TYPE	DISJOINT CONSTRAINT	OVERLAPPING CONSTRAINT
Partial 	Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.	Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique.
Total 	Every supertype occurrence is a member of a (at least one) subtype. Subtype discriminator cannot be null. Subtype sets are unique.	Every supertype occurrence is a member of a (at least one) subtype. Subtype discriminators cannot be null. Subtype sets are not unique.

Specialization

- top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype
- based on grouping unique characteristics and relationships of subtypes

Generalization

- bottom-up process of identifying higher-level, more generic entity supertype from lower-level entity subtypes
- based on grouping common characteristics and relationships of subtypes

Entity Cluster

- is a “virtual” entity type used to represent multiple entities and relationships
- formed by combining multiple interrelated entities into a single abstract entity object with the purpose of simplifying ER Diagram and thus enhancing its readability

Primary Key

- most important characteristic of entity is its primary key
- primary key's function is to guarantee entity integrity
- primary keys and foreign keys work together to implement relationships in relational model
- primary key has direct bearing on efficiency and effectiveness of database implementation

Natural Key

- Natural Key or natural identifier is a real-world, generally accepted identifier used to distinguish — that is, uniquely identify — real-world objects
- familiar to end users and forms part of their day-to-day business vocabulary

- function of the primary key is to guarantee entity integrity, not to “describe” the entity

- primary keys and foreign keys are used to implement relationships among entities
- implementation of such relationships is done mostly behind the scenes, hidden from end users
- database applications should let end user choose among multiple descriptive narratives of different objects, while using primary key values behind the scenes

Desirable Primary Key Characteristics

- **Unique values** - must uniquely identify each entity instance; it cannot contain nulls
- **Non-intelligent** - should not have embedded semantic meaning other than to uniquely identify each entity instance
- attribute with embedded semantic meaning is probably better used as descriptive characteristic of entity than as an identifier
- for example, student ID of 650973 would be preferred over Doe, John C. as primary key identifier

Desirable Primary Key Characteristics

- **No change over time** - if attribute has semantic meaning, it might be subject to updates; names do not make good primary keys
 - if you have Jane Doe as PK what happens if she changes her name when she gets married?
- PK should be permanent and unchangeable
- **Preferably single-attribute** - should have the minimum number of attributes; desirable but not required; simplify implementation of FK; multiple-attribute PK can cause PK of related entities to grow

Desirable Primary Key Characteristics

- **Preferably numeric** - unique values can be better managed when they are numeric, because database can use internal routines to implement automatically increments values with the addition of each new row
- **Security-compliant** - selected primary key must not be composed of any attribute(s) that might be considered security risk or violation
 - using Social Security number as PK in EMPLOYEE table is not a good idea

composite primary keys

- are particularly useful in two cases:
 1. as identifiers of composite entities, where each primary key combination is allowed only once in the M:N relationship
 2. as identifiers of weak entities, where the weak entity has a strong identifying relationship with the parent entity

surrogate key

- instances when primary key doesn't exist in real world or when existing natural key might not be suitable
- standard practice to create surrogate key - primary key created by database designer to simplify identification of entity instances
- has no meaning in user's environment — it exists only to distinguish one entity instance from another
- it can be generated by the DBMS to ensure that unique values are always provided

Database Design

- Data modeling and database design require skills that are acquired through experience
- experience is acquired through practice, regular and frequent repetition, applying the concepts learned to specific and different design problems
- presents four special design cases that highlight importance of flexible designs, proper identification of primary keys, and placement of foreign keys

- DESIGN CASES
- LEARNING FLEXIBLE DATABASE DESIGN
- Design Case #1: Implementing 1:1 Relationships
- Design Case #2: Maintaining History of Time-Variant Data
- Design Case #3: Fan Traps
- Design Case #4: Redundant Relationships

Design Case #1:

Implementing 1:1 Relationships

- Foreign Keys work with Primary Keys to implement relationships:
 - put PK of “one” side (parent entity)
 - put FK on “many” side (dependent entity)
- case of a 1:1 relationship between EMPLOYEE and DEPARTMENT based on business rule “one EMPLOYEE is the manager of one DEPARTMENT, and one DEPARTMENT is managed by one EMPLOYEE.”

- Place FK in both entities - is not recommended, as it would create duplicated work, and it could conflict with other existing relationships
- Place FK in one of entities (PK of one of two entities appears as FK in other entity)
 - preferred solution
 - but there is remaining question: which entity should be used

- One side is mandatory and other side is optional
 - Place PK of entity on mandatory side in entity on optional side as FK, and make FK mandatory
- Both sides are optional
 - Select FK that causes fewest nulls, or place FK in entity in which the (relationship) role is played
- Both sides are mandatory
 - Like previous
 - or consider revising your model to ensure that the two entities do not belong together in a single entity

- As a designer, you must recognize that 1:1 relationships exist in the real world, and therefore, they should be supported in the data model.
- In fact, a 1:1 relationship is used to ensure that two entity sets are not placed in the same table.

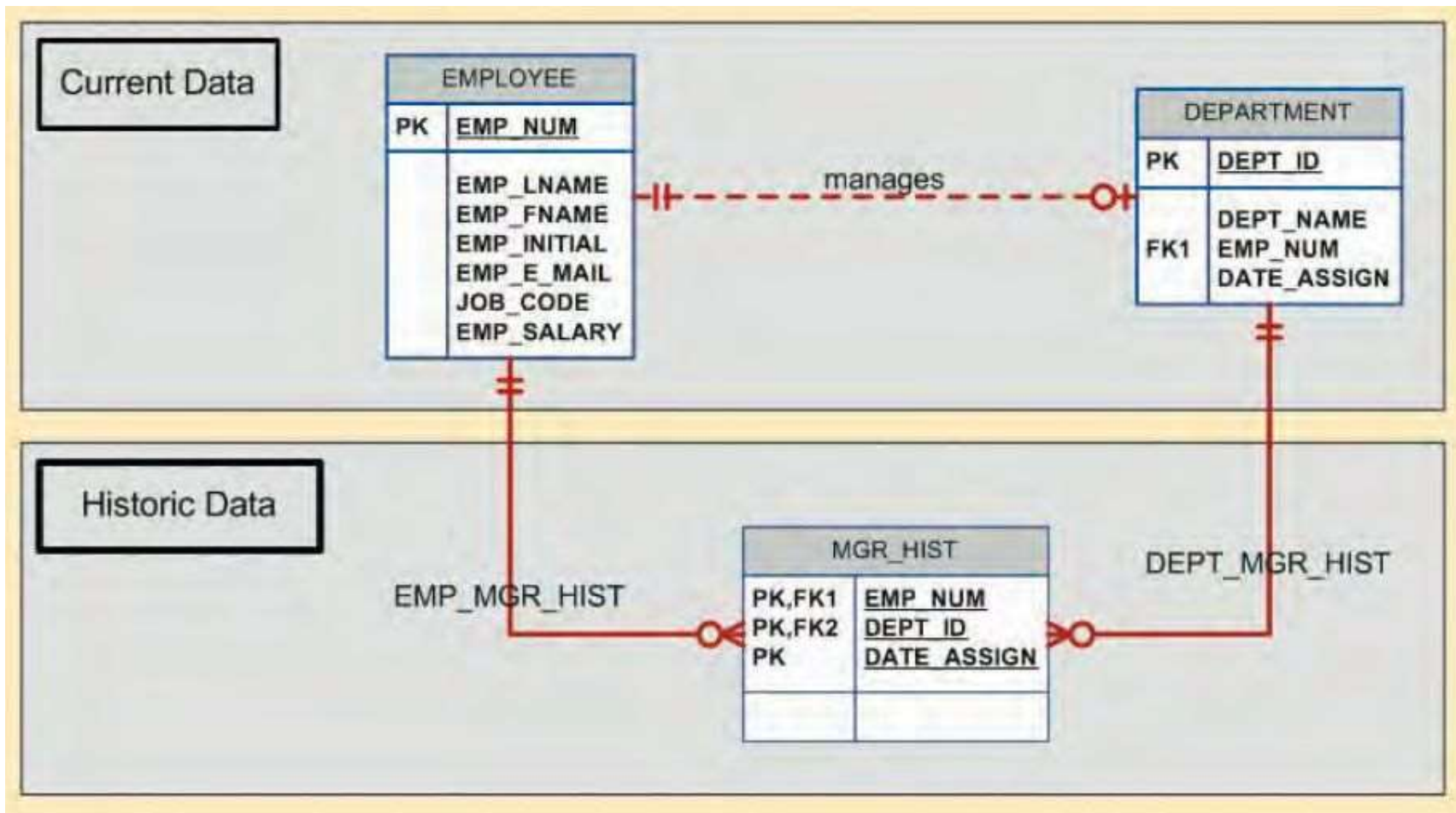
Design Case #2:

Maintaining History of Time-Variant

- Company managers generally realize that good decision making is based on information that is generated through data stored in databases. Such data reflect current as well as past events.
- Normally, data changes are managed by replacing existing attribute value with new value
- However, there are situations in which history of values for a given attribute must be preserved.
- *time-variant data* refer to data whose values change over time and for which you must keep history of the data changes

- some attribute values, such as your date of birth or your Social Security number, are not time variant
- attributes such as your student GPA or your bank account balance are subject to change over time
- sometimes data changes are externally originated and event driven, such as product price change

- keeping history of time-variant data you must create a new entity in 1:M relationship with original entity; this new entity will contain new value, date of change
- for example, if you want to keep track of current manager as well as history of all department managers, you can create model



- note that “manages” relationship is optional in theory and redundant in practice
- at any time, you could find out who manager of department is by retrieving the most recent DATE_ASSIGN date from MGR_HIST
- differentiates between current data and historic data
- current manager relationship is implemented by “manages” relationship between EMPLOYEE and DEPARTMENT
- historic data are managed through EMP_MGR_HIST and DEPT_MGR_HIST

Design Case #3:

Fan Traps

- Creating data model requires proper identification of data relationships among entities. However, due to miscommunication or incomplete understanding of business rules or processes, it is not uncommon to misidentify relationships among entities
- design trap occurs when relationship is improperly or incompletely identified and is therefore represented in a way that is not consistent with real world
- most common design trap is known as fan trap

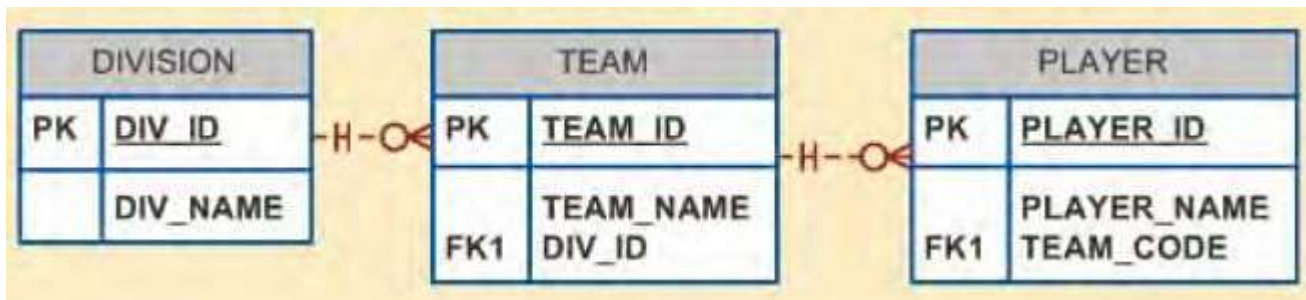
- *fan trap* occurs when you have one entity in two 1:M relationships to other entities, thus producing an association among the other entities that is not expressed in the model
- for example, assume the league has many divisions; each division has many players; each division has many teams
- might create

Incorrect ER Diagram with fan trap problem



- DIVISION is in 1:M relationship with TEAM and in 1:M relationship with PLAYER
- representation is semantically correct, relationships are not properly identified
- there is no way to identify which players belong to which team
- note that the relationship lines for DIVISION instances fan out to TEAM and PLAYER entity instances — thus “fan trap” label

Corrected ER Diagram after removal of the fan trap



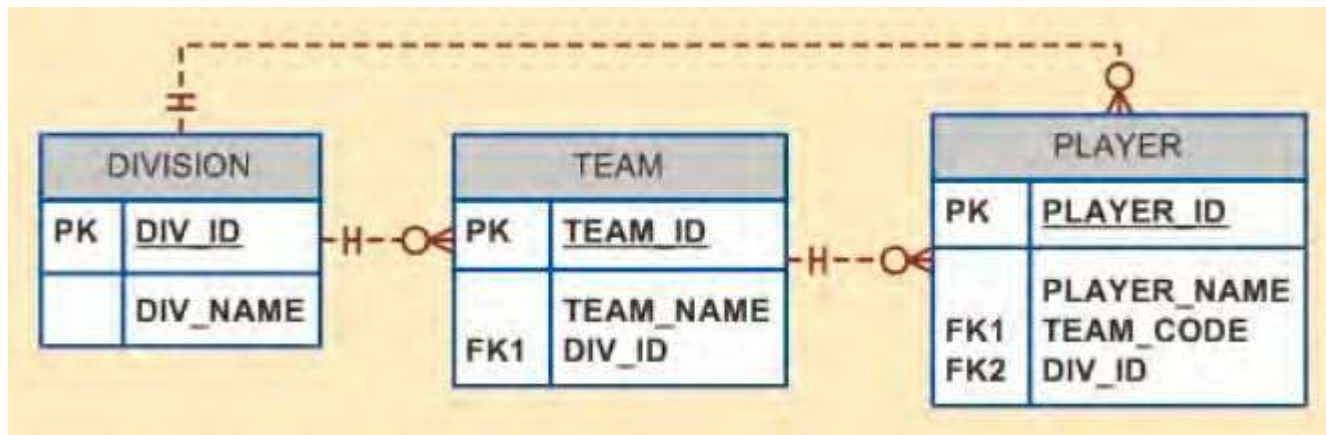
Design Case #4:

Redundant Relationships

- Although redundancy is often a good thing to have in computer environments (multiple backups in multiple places, for example), redundancy is seldom a good thing in the database environment
- Redundant relationships occur when there are multiple relationship paths between related entities
- main concern with redundant relationships is that they remain consistent across the model
- some designs use redundant relationships as a way to simplify the design

- an example of redundant relationships was introduced during discussion on maintaining a history of time-variant data
- use of redundant “manages” and “employs” relationships was justified by the fact that such relationships were dealing with current data rather than historic data

Redundant Relationship



- note transitive 1:M relationship between DIVISION and PLAYER through TEAM entity
- relationship that connects DIVISION and PLAYER is redundant
- relationship could be safely deleted without losing any information-generation capabilities in model

DataBase Design Examples

Formula 1

- <https://www.formula1.com/>
- Races
- Drivers
- Teams

Driver: Fernando ALONSO

- Team McLaren
- Country Spain
- Podiums 97
- Points 1830
- Grands Prix entered 272
- World Championships 2
- Highest race finish 1 (x32)
- Highest grid position 1
- Date of birth 29/07/1981
- Place of birth Oviedo, Spain

- Who is famous for using the catchphrase "*allons-y* Alonso" ?
- **Time** traveler

Driver: Fernando ALONSO

- *Team* *McLaren*
- **Country Spain**
- *Podiums* *97*
- *Points* *1830*
- *Grands Prix entered* *272*
- *World Championships* *2*
- *Highest race finish* *1 (x32)*
- *Highest grid position* *1*
- **Date of birth 29/07/1981**
- **Place of birth Oviedo, Spain**

Entities

- Affiliation
- Result

Affiliation

- ID of Fernando ALONSO
- ID of McLaren
- Begin Date
- End Date - could be NULL
 - NULL with meaning from Begin Date until today, tomorrow, end of time, ...

Affiliation view

- as of today()
- JOIN FROM Driver, Affiliation, Team
- WHERE Begin Date <= today() AND
- (today () <= End Date OR End Date IS NULL)

- Begin End Date are time intervals
- $\text{Begin} \leq \text{End}$ OR End IS NULL
- Time interval intersection
- $(T_{11}, T_{12}), T_{11} \leq T_{12}$
- $(T_{21}, T_{22}), T_{21} \leq T_{22}$

- Begin End Date are time intervals
- $\text{Begin} \leq \text{End}$ OR End IS NULL
- Time interval intersection
- $(T_{11}, T_{12}), T_{21} \leq T_{22}$
- $(T_{21}, T_{22}), T_{21} \leq T_{22}$
- $T_{11} \leq T_{22}$ AND $T_{12} \geq T_{21}$

Temporal Databases

- provide uniform and systematic way of dealing with historical data
- some data may be inherently historical, e.g., medical or judicial records
- regarded as enhancement of conventional (snapshot) database
- sets data into context of time, adds a time dimension

- over time, relational tables are updated, new rows are inserted, some rows are deleted and some attribute values might be modified - data in table changes over time
- if copy of table was taken each time before it is updated and if date of copy was added to all rows we could actually follow evolution of table

- in contrast, in many databases, one would only keep current copy of table - current *snapshot*
- *time cube* snapshot is time slice of such cube, conventional database always holds one slice whereas temporal database holds entire cube

- whenever an update occurs conventional database *physically* updates, throws old values away and stores new ones
- temporal database is updated *logically*, it marks old and new values with timestamps that indicate to which snapshot actually belong
- concept of *physical* vs. *logical deletion*

Set ... Picture

- UPDATE AutoDriver
 - SET Picture =
- (SELECT BulkColumn FROM OPENROWSET (Bulk 'D:\Downloads\1458061391663.jpg', SINGLE_BLOB) AS BLOB)
- WHERE id_ad = 1;

Executed Only Once in DataBase Server

- USE master
 - GO
- EXEC sp_configure 'show advanced options', 1
 - GO
- RECONFIGURE WITH OVERRIDE
 - GO
- EXEC sp_configure 'xp_cmdshell', 1
 - GO
- RECONFIGURE WITH OVERRIDE
 - GO
- EXEC sp_configure 'show advanced options', 0
 - GO

Get ... Picture

- EXEC xp_cmdshell
- 'BCP "SELECT [Picture] FROM [formula1].[dbo].[AutoDriver] WHERE [id_ad] = 1" queryout "D:\Downloads\alonso.jpg" -T -n'
- ;

Challenge

- build a “maneale” database with pictures and sound

maneie

MS SQL Server - configure

- USE master
 - GO
- EXEC sp_configure 'show advanced options', 1
 - GO
- RECONFIGURE WITH OVERRIDE
 - GO
- EXEC sp_configure 'xp_cmdshell', 1
 - GO
- RECONFIGURE WITH OVERRIDE
 - GO
- EXEC sp_configure 'show advanced options', 0
 - GO

MS SQL Server – Data Type

- for binary - VARBINARY(max)

MS SQL Server - SET

- UPDATE manea SET manea =
- (SELECT BulkColumn FROM OPENROWSET (Bulk 'D:\Downloads\Carmen Serban - Picioarele epilate sunt ca sarea in bucate.mp3', SINGLE_BLOB) AS BLOB)
- WHERE id_manea = 1;
- UPDATE manea SET manea =
- (SELECT BulkColumn FROM OPENROWSET (Bulk 'D:\Downloads\Carmen Serban-Viata bate filmul.mp3', SINGLE_BLOB) AS BLOB)
- WHERE id_manea = 2;

MS SQL Server - GET

- EXEC xp_cmdshell
- 'BCP "SELECT [manea] FROM [manele].[dbo].[manea] WHERE [id_manea] = 1" queryout "D:\Downloads\FileOut1.mp3" -T -n'
- EXEC xp_cmdshell
- 'BCP "SELECT [manea] FROM [manele].[dbo].[manea] WHERE [id_manea] = 2" queryout "D:\Downloads\FileOut2.mp3" -T -n'

MySQL - configure

- `max_allowed_packet=100M`
- in `my.ini`

MySQL Server – Data Type

- LargeBLOB (Binary Large OBject)

MySQL - SET

- UPDATE manea SET manea =
- LOAD_FILE('D:/Downloads/Carmen Serban - Picioarele epilate sunt ca sarea in bucate.mp3')
- WHERE idmanea = 1;
- UPDATE manea SET manea =
- LOAD_FILE('D:/Downloads/Carmen Serban-Viata bate filmul.mp3')
- WHERE idmanea = 2;

MySQL - GET

- SELECT manea INTO DumpFile
'D:/Downloads/FileOut1.mp3'
- FROM manea WHERE idmanea = 1;
- SELECT manea INTO DumpFile
'D:/Downloads/FileOut2.mp3'
- FROM manea WHERE idmanea = 2;

Additional optional material

WISDOM

Picioarele Epilate Sunt Ca Sarea in Bucate
Epilate Legs Are Like Salt in Food

Viață bate filmul

- Mi-am tocit hainele în coate
- Să termin o facultate
- ...
- Sunt zilier cu facultate
- La un patron fără carte
- ...
 - Să am banii de chirie

Viață bate filmul

- ...
- Eu sunt fraierul cu carte
- Toți îmi spun intelectualul
- Dar nu am cașcavalul
- Ieri eram inginerașul
- De azi să îmi spuneți nașul
- ...