Database Modeling and Design

3rd Edition

Toby J. Teorey
University of Michigan

Lecture Notes
(last revision 10/24/98)

Contents

I. Database Systems and the Life Cycle

Introductory Concepts

data—a fact, something upon which an inference is based (information or knowledge has value, data has cost)

data item—smallest named unit of data that has meaning in the real world (examples: last name, address, ssn, political party)

data aggregate (or group) -- a collection of related data items that form a whole concept; a simple group is a fixed collection, e.g. date (month, day, year); a repeating group is a variable length collection, e.g. a set of aliases.

record—group of related data items treated as a unit by an application program (examples: presidents, elections, congresses)

file—collection of records of a single type (examples: president, election)

database—computerized collection of interrelated stored data that serves the needs of multiple users within one or more organizations, i.e. interrelated collections of records of potentially many types. Motivation for databases over files: integration for easy access and update, non-redundancy, multi-access.

database management system (DBMS) -- a generalized software system for manipulating databases. Includes logical view (schema, sub-schema), physical view (access methods, clustering), data manipulation language, data definition language, utilities - security, recovery, integrity, etc.

database administrator (DBA) -- person or group responsible for the effective use of database technology in an organization or enterprise. Motivation: control over all phases of the lifecycle.

Objectives of Database Management

1. Data availability—make an integrated collection of data available to a wide variety of users
   * at reasonable cost—performance in query update, eliminate or control data redundancy
   * in meaningful format—data definition language, data dictionary
   * easy access—query language (4GL, SQL, forms, windows, menus);
   embedded SQL, etc.; utilities for editing, report generation, sorting

2. Data integrity—insure correctness and validity
   * checkpoint/restart/recovery
   * concurrency control and multi-user updates
   * accounting, audit trail (financial, legal)

3. Privacy (the goal) and security (the means)
   * schema/sub-schema, passwords
4. Management control—DBA: lifecycle control, training, maintenance

5. Data independence (a relative term) -- avoids reprogramming of applications, allows easier conversion and reorganization

* physical data independence—program unaffected by changes in the storage structure or access methods

* logical data independence—program unaffected by changes in the schema

* Social Security Administration example (1980ís)
  - changed benefit checks from $999.99 to $9999.99 format
  - had to change 600 application programs
  - 20,000 work hours needed to make the changes (10 work years)

* Student registration system—cannot go to a 4-digit or hexadecimal course numbering system because of difficulty changing programs

*Y2K (year 2000) problem—many systems store 2-digit years (e.g. '02-OCT-98') in their programs and databases, that give incorrect results when used in date arithmetic (especially subtraction), so that '00' is still interpreted as 1900 rather than 2000. Fixing this problem requires many hours of reprogramming and database alterations for many companies and government agencies.

Relational Database Lifecycle

1. Requirements formulation and analysis
  * natural data relationships (process-independent)
  * usage requirements (process-dependent)
  * hardware/software platform (OS, DBMS)
  * performance and integrity constraints
  * result: requirements specification document, data dictionary entries

2. Logical database design
  2.1 ER modeling (conceptual design)
  2.2 View integration of multiple ER models
  2.3 Transformation of the ER model to SQL tables
  2.4 Normalization of SQL tables (up to 3NF or BCNF)
  *result: global database schema, transformed to table definitions

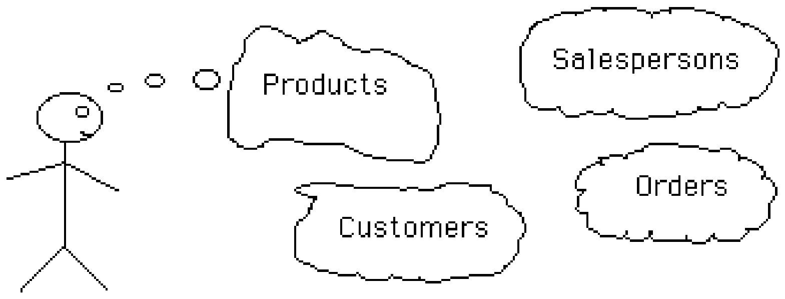
3. Physical database design
  * index selection (access methods)
  * clustering

4. Database distribution (if needed for data distributed over a network)
  * data fragmentation, allocation, replication

5. Database implementation, monitoring, and modification

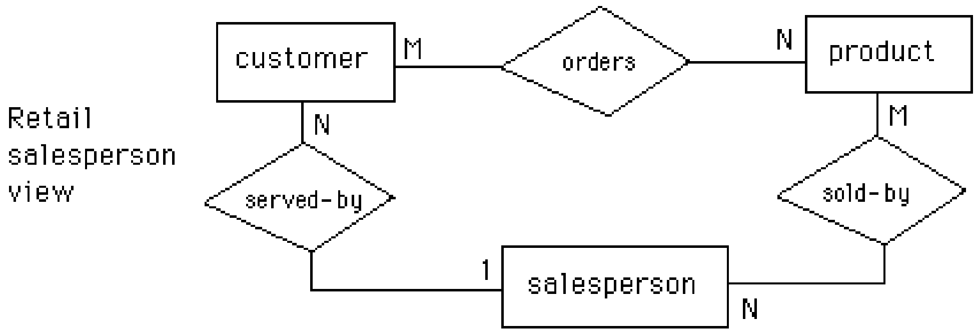
Database Life Cycle

Step I   Information Requirements (reality)

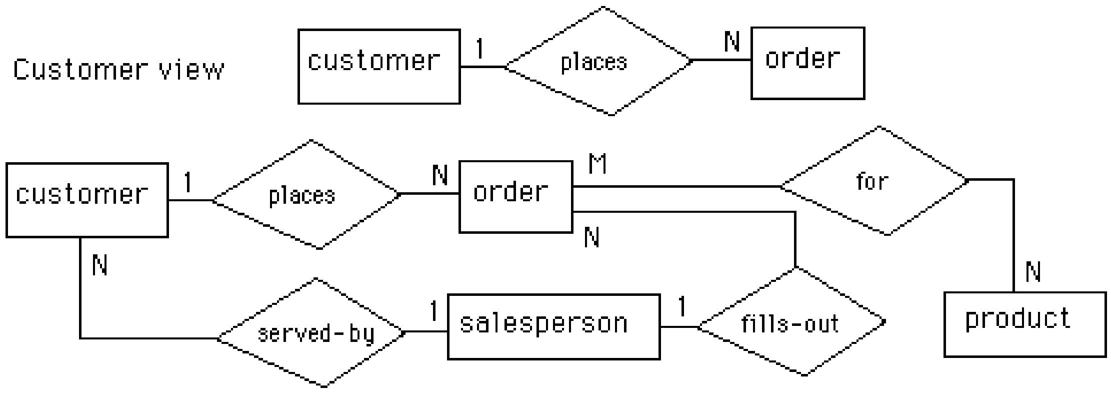


Step II   Logical design

Step II.a   ER modeling (conceptual)



Step II.b   View integration



Integration of retail salesperson's and customer's views

Step II.c Transformation of the ER diagram to SQL tables

Customer

cust- no	cust- name	....

Product

prod-no	prod-name	qty-in-stock

```
create table customer
   (cust_no integer,
   cust_name char(15),
   cust_addr char(30),
   sales_name char(15),
   prod_no integer,
   primary key (cust_no),
   foreign key (sales_name)
      references salesperson,
   foreign key (prod_no)
      references product);
```

Salesperson

sales-name	addr	dept	job-level	vacation-days

Order

order-no	sales-name	cust-no

Order-product

order-no	prod-no

Step II.d Normalization of SQL tables (3NF, BCNF, 4NF, 5NF)

Decomposition of tables and removal of update anomalies.

Salesperson

sales-name	addr	dept	job-level

Sales-vacations

job-level	vacation-days

Step III Physical Design (including denormalization)

Customer

cust-no	cust-name

$\Rightarrow$

Customer/refined

cust-no	cust-name	sales-name

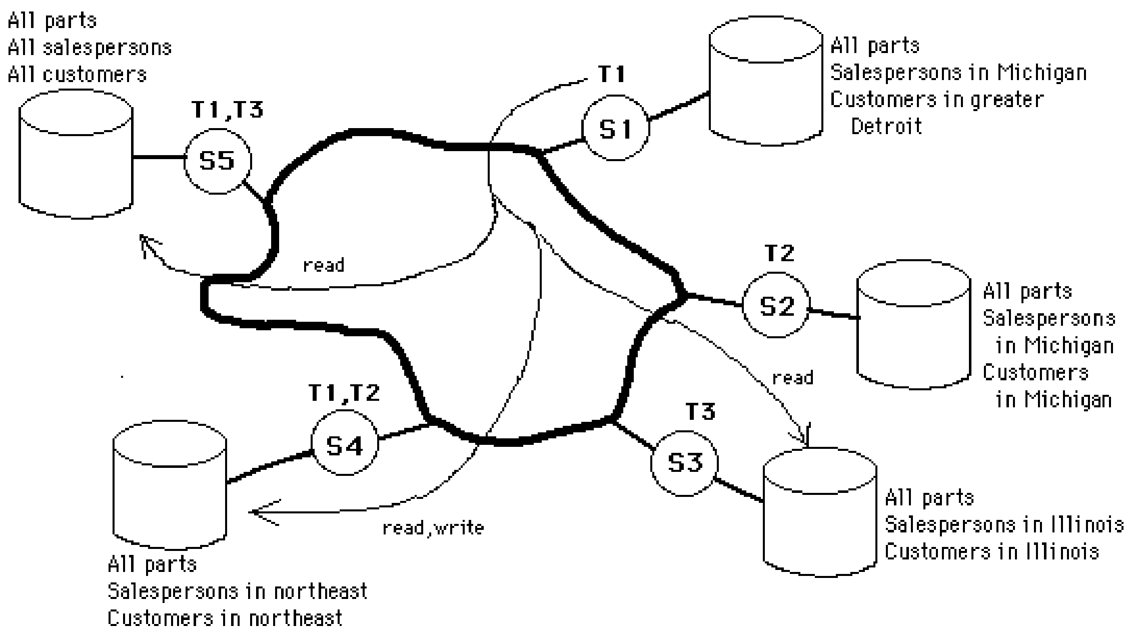
$\Leftarrow$

Order

order-no	sales-name	cust-no

Physical design parameters:
Indexing, access methods, clustering

Step IV Data distribution



S1 = Ann Arbor, S2 = Detroit, S3 = Chicago, S4 = Boston, S5 = New York

T1, T2, T3 are transactions (the figure shows all sites where they are initiated

Decisions: fragmentation, replication, allocation
Objectives: min. response time, min. communication cost, max availability

Characteristics of a Good Database Design Process

* iterative requirements analysis
- interview top-down
- use simple models for data flow and data relationships
- verify model

* stepwise refinement and iterative re-design

* well-defined design review process to reduce development costs review team
-database designers
-DBMS software group
-end users in the application areas when to review
- after requirements analysis & conceptual design
- after physical design
- after implementation (tuning) meeting format
- short documentation in advance
- formal presentation
- criticize product, not person
- goal is to locate problems, do solutions off line
- time limit is 1-2 hours

II. Requirements Analysis

Purpose - identify the real-world situation in enough detail
to be able to define database components. Collect two types of data: natural data (input to the database) and processing data (output from the database).

Natural data requirements (what goes into the database)

1. Organizational objectives
   - sell more cars this year
   - move into to recreational vehicle market

2. Information system objectives
   - keep track of competitors' products and prices
   - improve quality and timing of data to management regarding production schedule delays, etc.
   - keep track of vital resources needed to produce and market a product

3. Organizational structure/chart

4. Administrative and operational policies
   - annual review of employees
   - weekly progress reports
   - monthly inventory check
   - trip expense submission

5. Data elements, relationships, constraints, computing environment

Processing requirements (what comes out of the database)

1. Existing applications - manual, computerized

2. Perceived new applications

* quantifies how data is used by applications

* should be a subset of data identified in the natural relationships (but may not be due to unforeseen applications)

* problem - many future applications may be unknown

Data and Process Dictionary Entries for Requirements Analysis in the Database Design Lifecycle

Entity Description (possibly in a data dictionary)

Name customer	
Reference-no	4201
Cardinality	10,000
Growth rate	100 per month
Synonyms	user, buyer
Role (or description)	someone who purchases or rents a product made by the company.
Security level	0 (customer list is public)
Subtypes	adults, minors
Key attribute(s)	cust-no
Non-key attribute(s)	cust-name, addr, phone, payment-status
Relationship to other entities	salesperson, order, product
Used in which applications	billing, advertising

Attribute description (data elements in a data dictionary)

Name cust-no	
Reference-no	4202
Range of legal values	1 to 999,999
Synonyms	cno, customer-number
Data type	integer
Description	customer id number set by the company.
Key or nonkey	key
Source of data	table of allowable id numbers
Used in applications	billing
Attribute trigger	/*describes actions that occur when a data element is queried or updated*/

Relationship description

Name purchase	
Reference-no	511037
Degree	binary
Entities and connectivity	customer(0,n), product(1,n)
Synonyms	buy
Attributes (of the relationship)	quantity, order-no
Assertions	a customer must have purchased at least one product, but some products may not have been purchased as yet by any customers.

Process (application) description

Name payroll	
Reference-no	163
Frequency	bi-weekly
Priority	10
Deadline	noon Fridays
Data elements used	emp-name, emp-salary
Entities used	employee
Data volume (how many entities)	implicit from entity cardinality

Interviews at different levels

Top management - business definition, plan/objectives, future plans

Middle management - functions in operational areas, technical areas, job-titles, job functions

Employees - individual tasks, data needed, data out

Specific end-users of a DBMS - applications and data of interest

Basic rules in interviewing

1. Investigate the business first

2. Agree with the interviewee on format for documentation (ERD, DFD, etc.)

3. Define human tasks and known computer applications

4. Develop and verify the flow diagram(s) and ER diagram(s)

5. Relate applications to data (this helps your programmers)

Example: order entry clerk

Function: Take customer orders and either fill them or make adjustments.
Frequency: daily

Task Def	Volume	Data Elements
1. Create order	2000	A, B, E, H
2. Validate order	2000	A, B, G, H, J
3. Fill out error form	25	A, C
4. Reserve item/price	6000	A, D, H
5. Request alternate items	75	A, E, I, K,M
6. Enter unit price	5925	A, F, J, N

III. Entity-Relationship (ER) Modeling

Basic ER Modeling Concepts

Entity - a class of real world objects having common characteristics and properties about which we wish to record information.

Relationship - an association among two or more entities

* occurrence - instance of a relationship is the collective instances of the related entities

* degree - number of entities associated in the relationship (binary, ternary, other
n-ary)

* connectivity - one-to-one, one-to-many, many-to-many

* existence dependency (constraint) - optional/mandatory

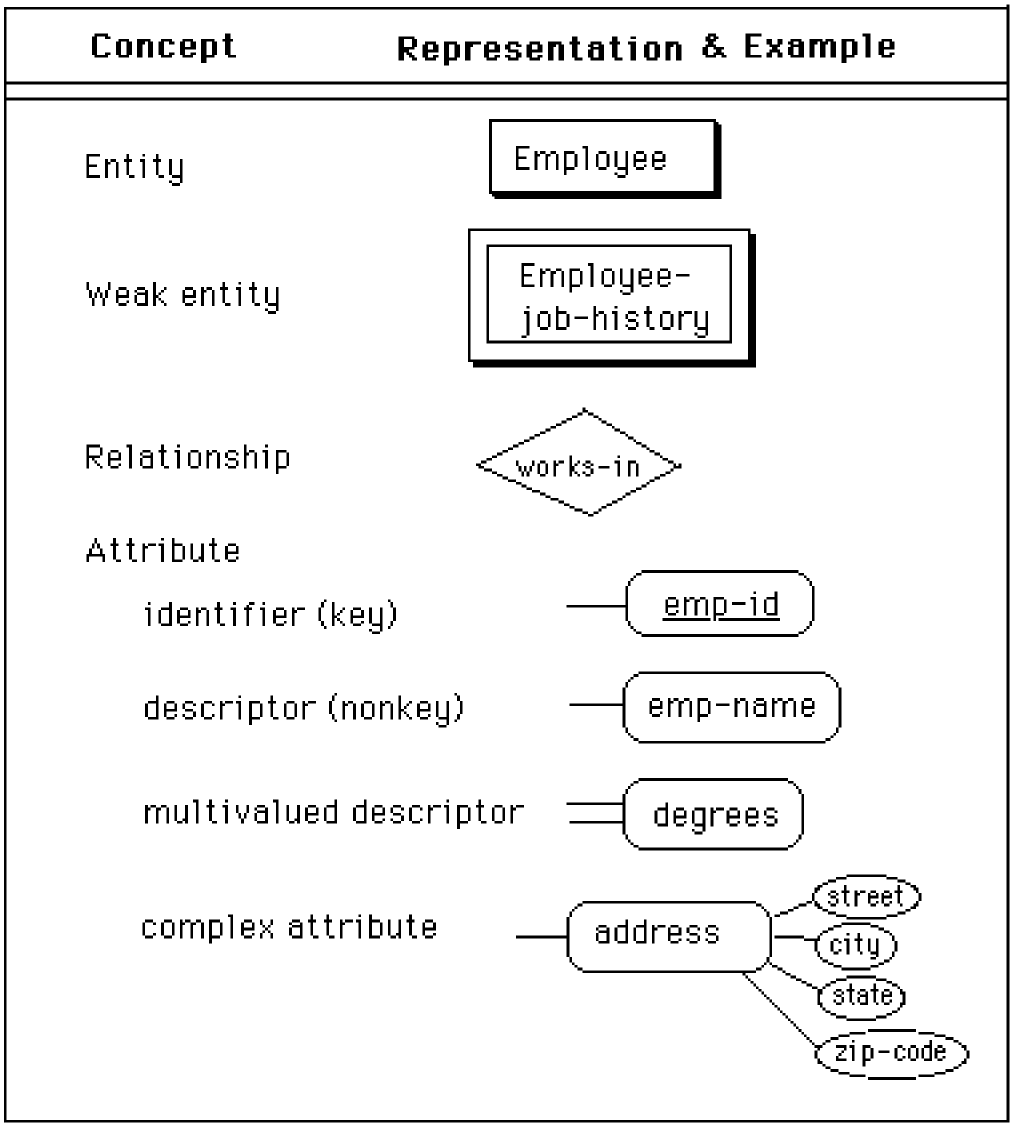
Attribute - a characteristic of an entity or relationship

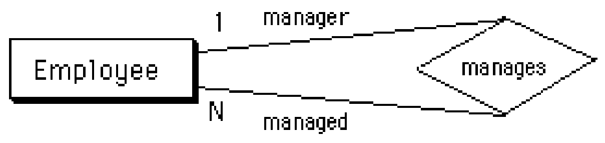
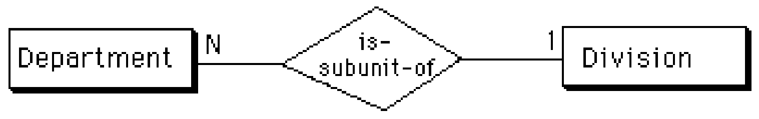
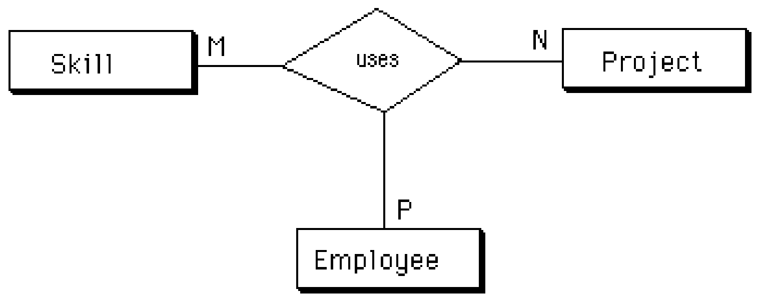
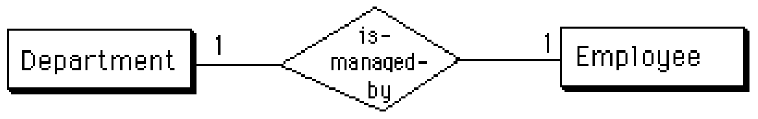
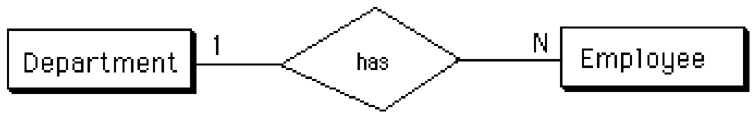
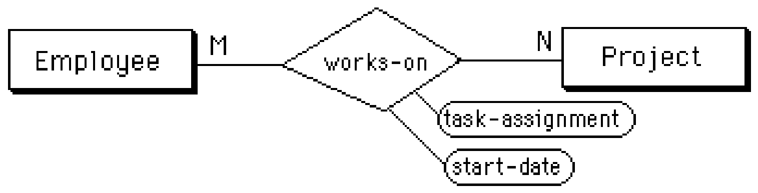
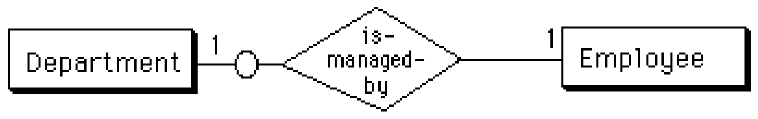
* Identifier - uniquely determines an instance of an entity

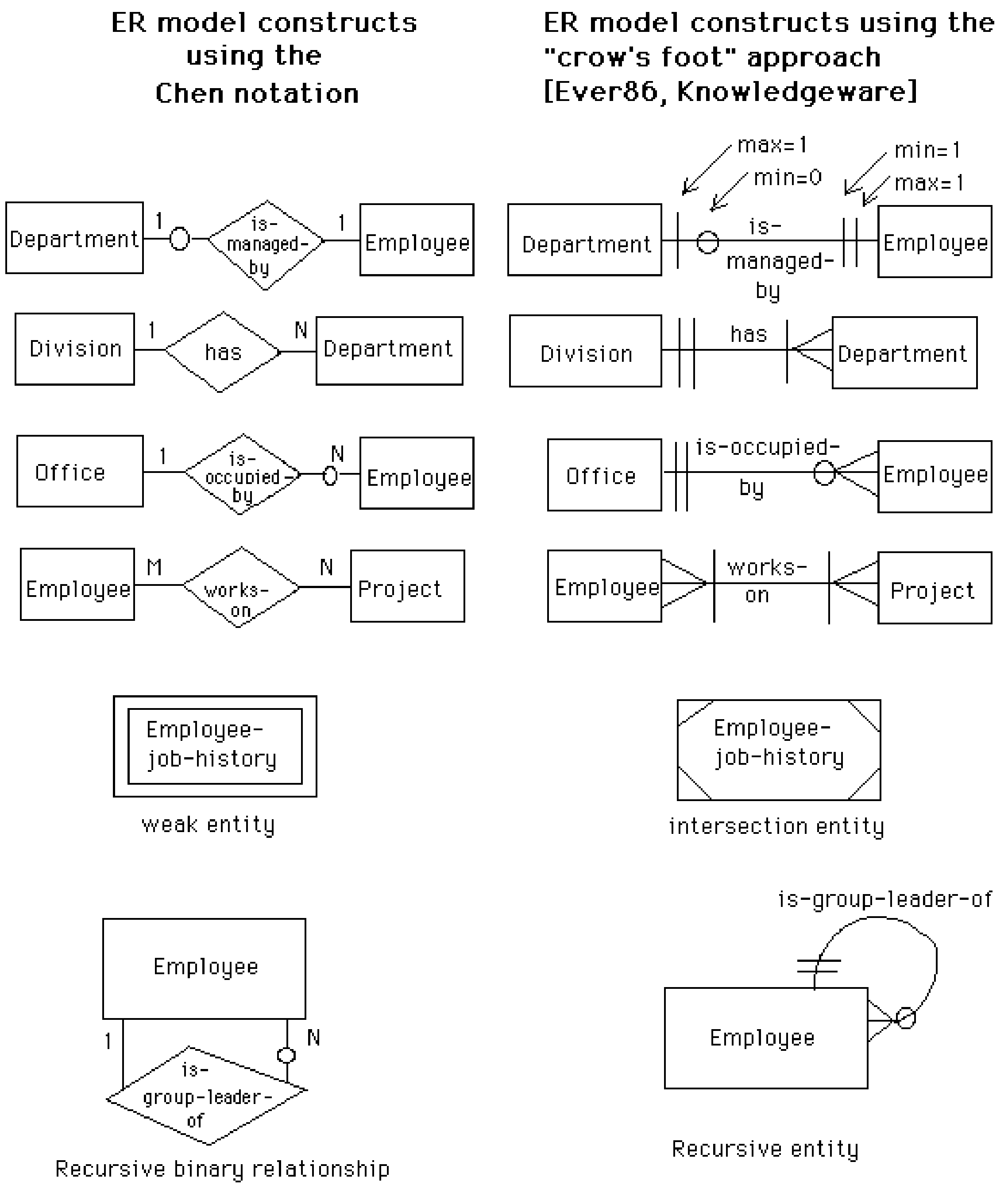
* Identity dependence - when a portion of an identifier is inherited from another entity

* Multi-valued - same attribute having many values for one entity

* Surrogate - system created and controlled unique key (e.g. Oracle's "create sequence")


Concept
Representation & Example
Entity
Employee
Weak entity
Employee-job-history
Relationship
works-in
Attribute
identifier (key)
emp-id
descriptor (nonkey)
emp-name
multivalued descriptor
degrees
complex attribute
address
street
city
state
zip-code

Concept	**Representation & Example**
Degree recursive binary	
binary	
ternary	
Connectivity one-to-one	
one-to-many	
many-to-many	 Employee M works-on N Project task-assignment start-date
Existence optional	



(a) ER construct comparisons

Super-class (super-type)/subclass (subtype) relationship

Generalization

* similarities are generalized to a super-class entity, differences are specialized to a subclass entity, called an “ISA” relationship (“specialization” is the inverse relationship)

* disjointness constraint - there is no overlap among subclasses

* completeness constraint - constrains subclasses to be all-inclusive of the super-class or not (i.e. total or partial coverage of the superclass)

* special property: hierarchical in nature

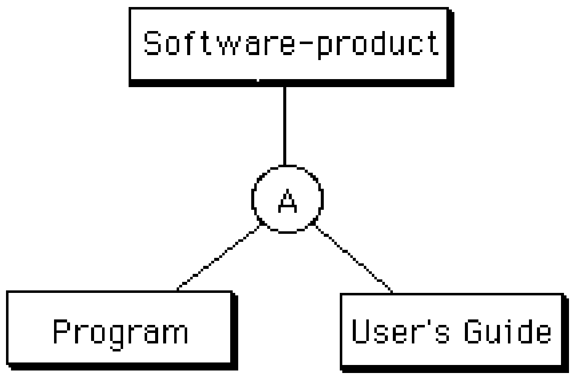
* special property: inheritance - subclass inherits the primary key of the super-class, super-class has common nonkey attributes, each subclass has specialized non-key attributes

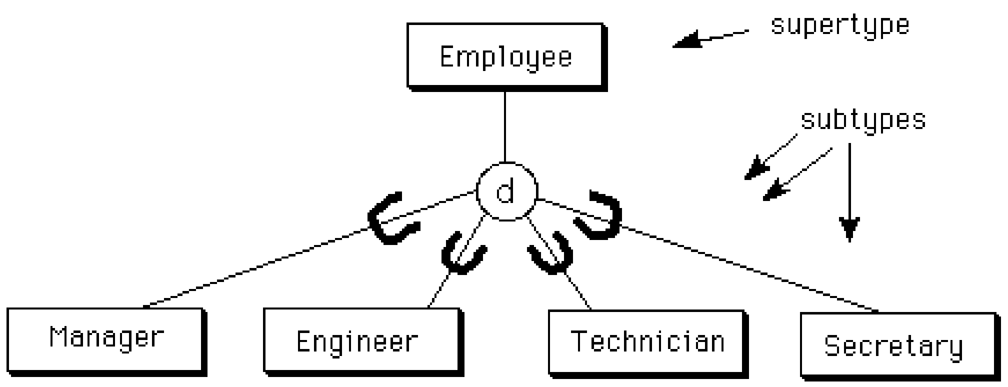
Aggregation

* “part-of” relationship among entities to a higher type aggregate entity (“contains” is the inverse relationship)

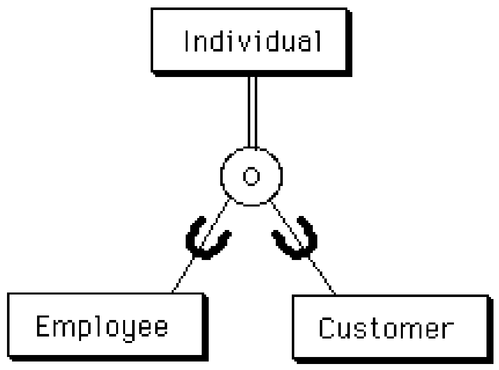
* attributes within an entity, data aggregate (mo-day-year)

* entity clustering variation: membership or “is-member-of” relationship





(a) Generalization with disjoint subtypes



(b) Generalization with overlapping subtypes and completeness constraint

Constraints in ER modeling

* role - the function an entity plays in a relationship

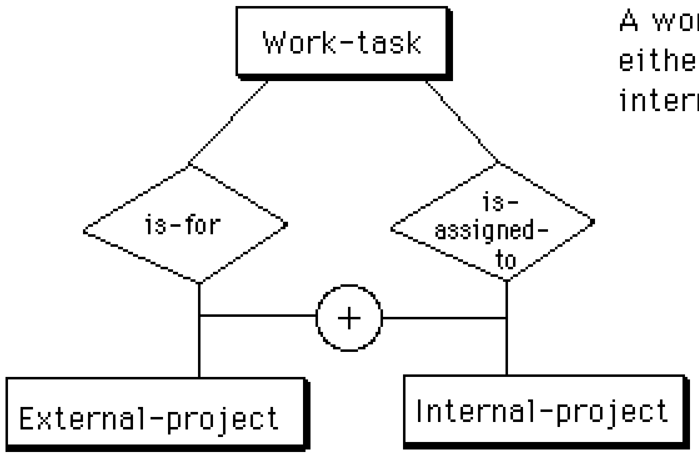
* existence constraint (existence dependency) - weak entity

* exclusion constraint - restricts an entity to be related to only of several other

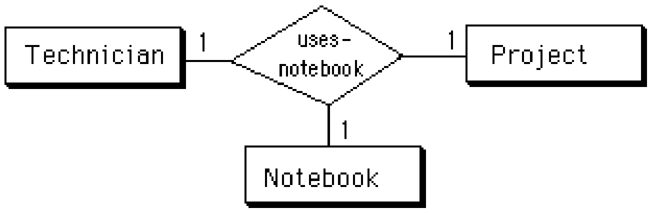
* entities at a given point in time

  - mandatory/optional
  - specifies lower bound of connectivity of entity instances
  - participating in a relationship as 1 or 0

* uniqueness constraint – one-to-one functional dependency among key attributes in a relationship: binary, ternary, or higher n-ary



A work task can be assigned to either an external project or an internal project, but not both.

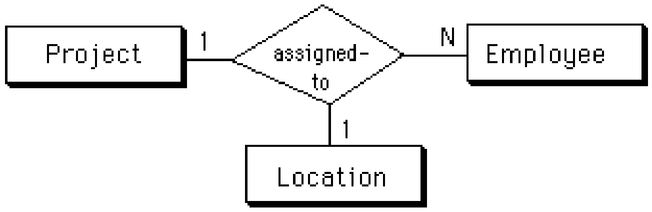


A technician uses exactly one notebook for each project. Each notebook belongs to one technician for each project. Note that a technician may still work on many projects and maintain different notebooks for different projects.

Functional dependencies

emp-id, project-name -> notebook-no
emp-id, notebook-no -> project-name
project-name, notebook-no -> emp-id

(a) one-to-one-to-one ternary relationship

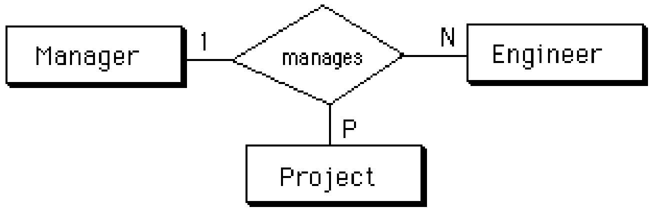


Each employee assigned to a project works at only one location for that project, but can be at different locations for different projects. At a particular location, an employee works on only one project. At a particular location, there can be many employees assigned to a given project.

Functional dependencies

emp-id, loc-name -> project-name
emp-id, project-name -> loc-name

(b) one-to-one-to-many ternary relationship

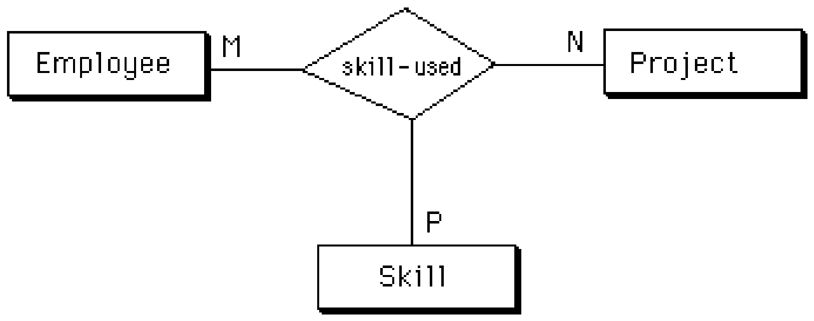


Each engineer working on a particular project has exactly one manager, but each manager of a project may manage many engineers, and each manager of an engineer may manage that engineer on many projects.

Functional dependency

project-name, emp-id -> mgr-id

(c) one-to-many-to-many ternary relationship



Employees can use many skills on any one of many projects, and each project has many employees with various skills.

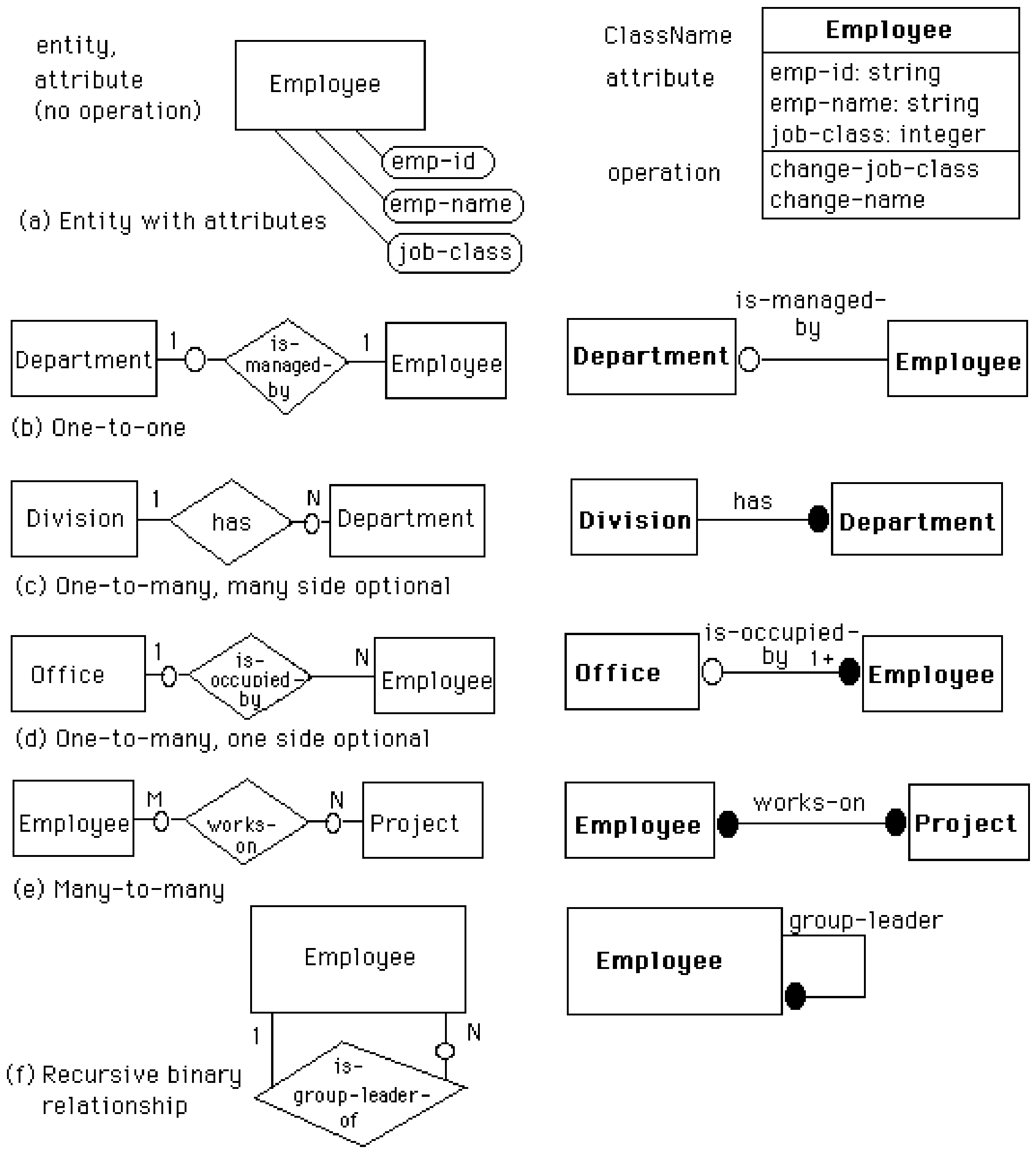
<u>Functional dependencies</u>

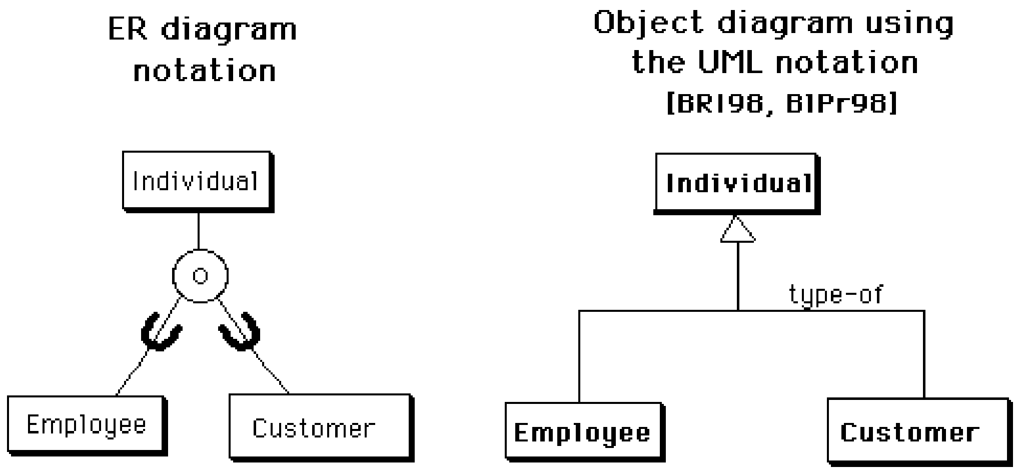
None

(d) many-to-many-to-many ternary relationship

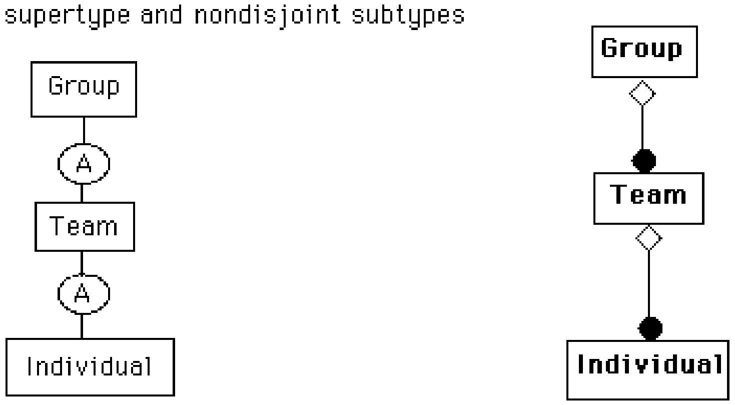
ER diagram notation

Object diagram using the Blaha and Premerlani variations of UML notation [BRJ98, BlPr98]

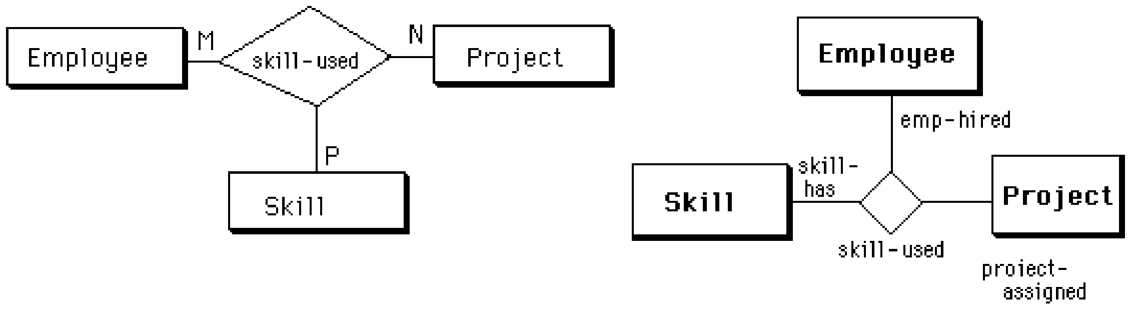




(a) Generalization ("is-a") relationship
with supertype and nondisjoint subtypes



(b) Aggregation ("part-of") relationship



(c) ternary relationship

Schema Integration Methods

Goal in schema integration

- to create a non-redundant unified (global) conceptual schema

(1) completeness - all components must appear in the global schema
(2) minimality - remove redundant concepts in the global schema
(3) understandability - does global schema make sense?

1. Comparing of schemas

* look for correspondence (identity) among entities

* detect possible conflicts

- naming conflicts
  homonyms - same name for different concepts
  synonyms - different names for the same concept

- structural conflicts
  type conflicts - different modeling construct for the same concept (e. g. “order” as an entity, attribute, relationship)

- dependency conflicts - connectivity is different for different views (e.g. job-title vs. job-title-history)

- key conflicts - same concept but different keys are assigned (e.g. ID-no vs. SSN)

- behavioral conflicts - different integrity constraints (e.g. null rules for optional/mandatory: insert/delete rules)

* determine inter-schema properties

- possible new relationships to combine schemas

- possible abstractions on existing entities or create new super-classes (super-types)

2. Conforming of schemas

* resolve conflicts (often user interaction is required)

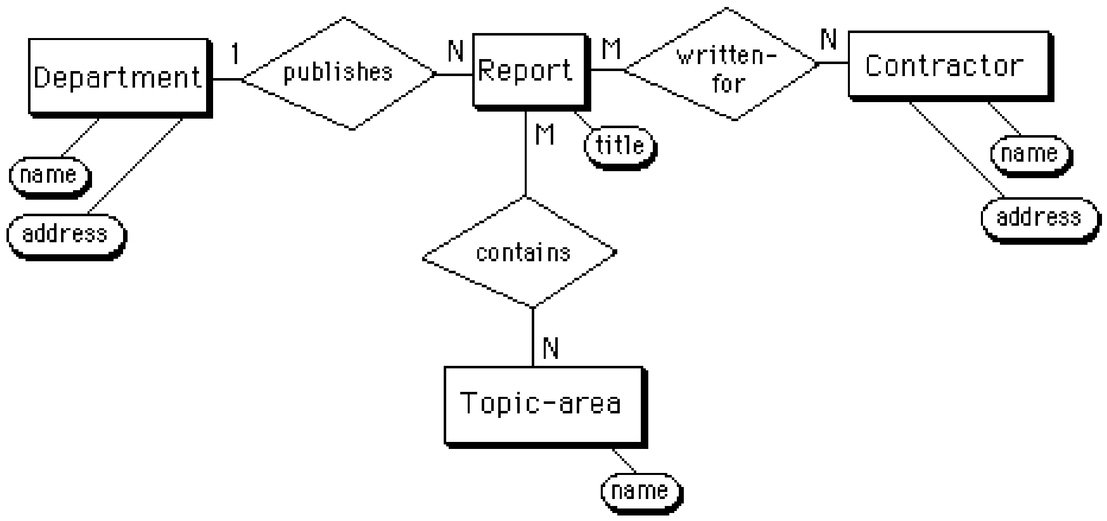
* conform or align schemas to make compatible for integration

* transform the schema via
  - renaming (homonyms, synonyms, key conflicts)
  - type transformations (type or dependency conflicts)
  - modify assertions (behavioral conflicts)

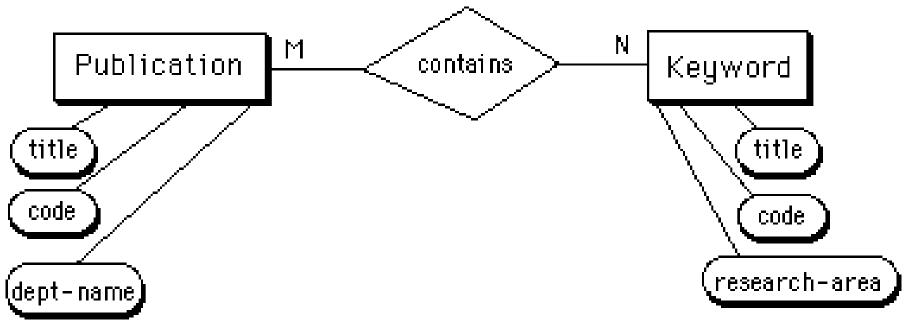
3. Merging and restructuring

* superimpose entities

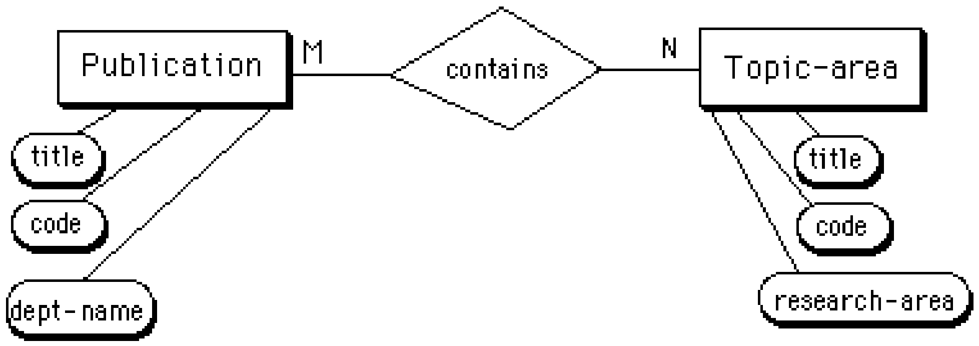
* restructure result of superimposition



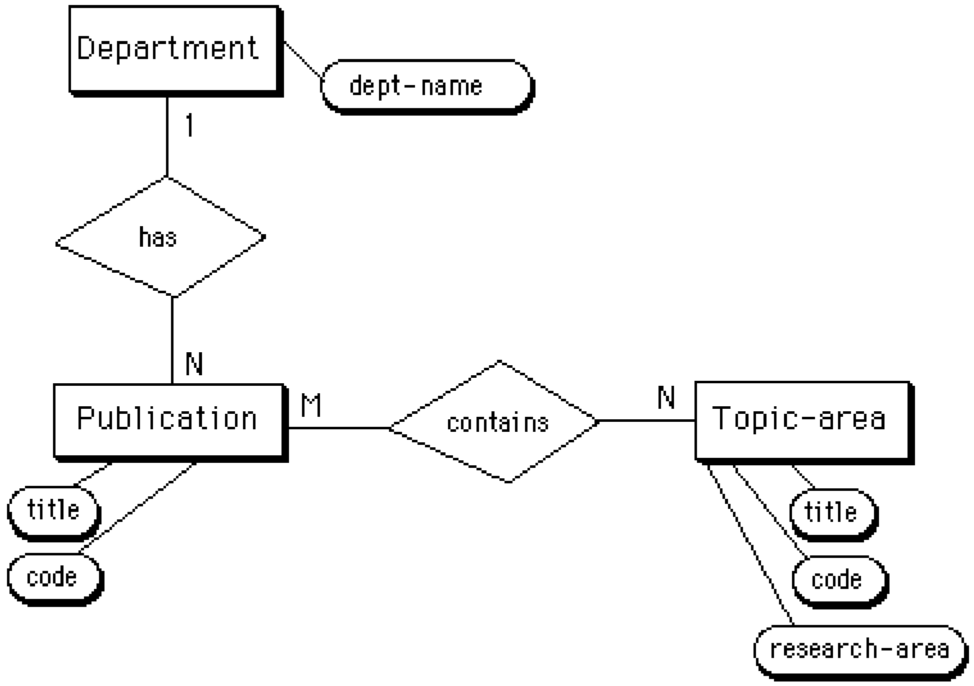
(a) Original schema 1, focused on reports



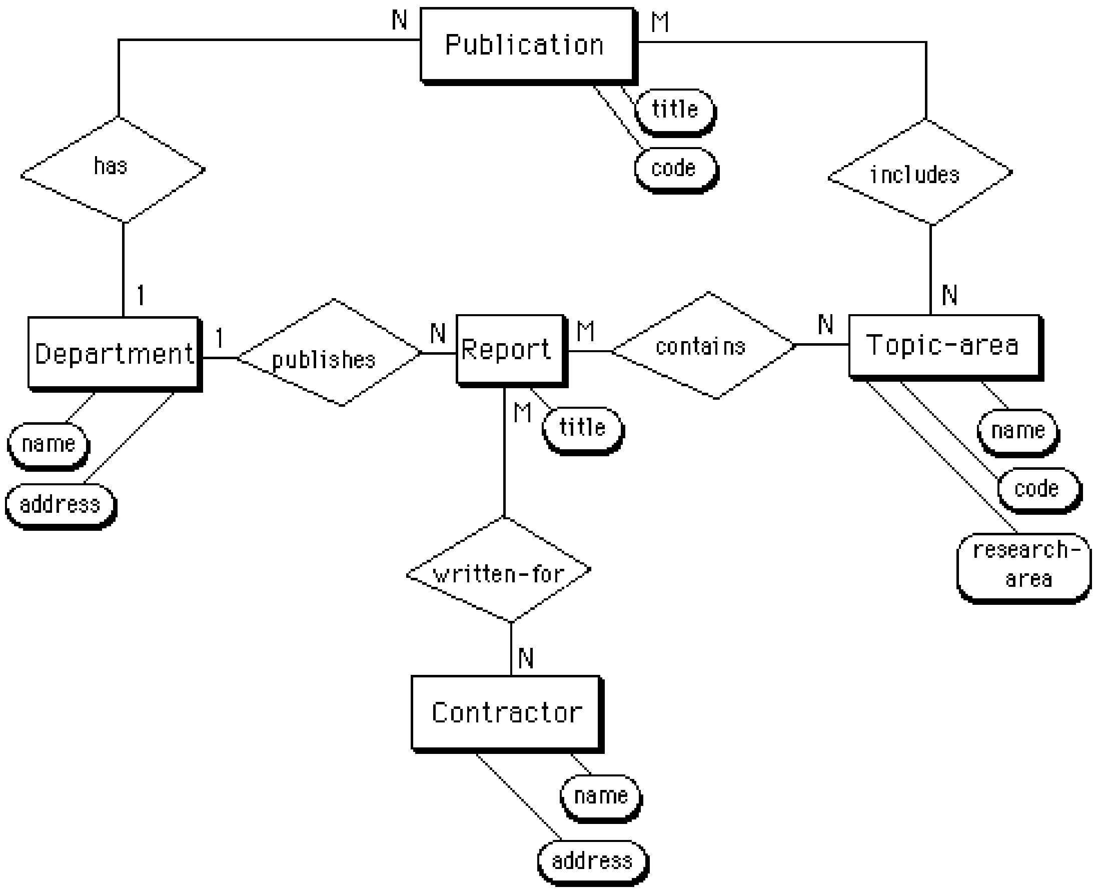
(b) Original schema 2, focused on publications



(a) Schema 2.1, in which Keyword has changed to Topic-area



(b) Schema 2.2, in which the attribute dept-name has changed to an attribute and an entity


Publication
N
M
title
code
has
includes
1
N
Department
1
publishes
N
Report
M
contains
N
Topic-area
name
address
M
title
name
code
research-area
written-for
N
Contractor
name
address

Entity-Relationship Clustering

Motivation

* conceptual (ER) models are difficult to read and understand for large and complex databases, e.g. 10,000 or more data elements

* there is a need for a tool to abstract the conceptual database schema (e. g. clustering of the ER diagram)

* potential applications

  - end user communication

  - application design team communication

  - documentation of the database conceptual schema (in coordination with the data dictionary)

Clustering Methodology

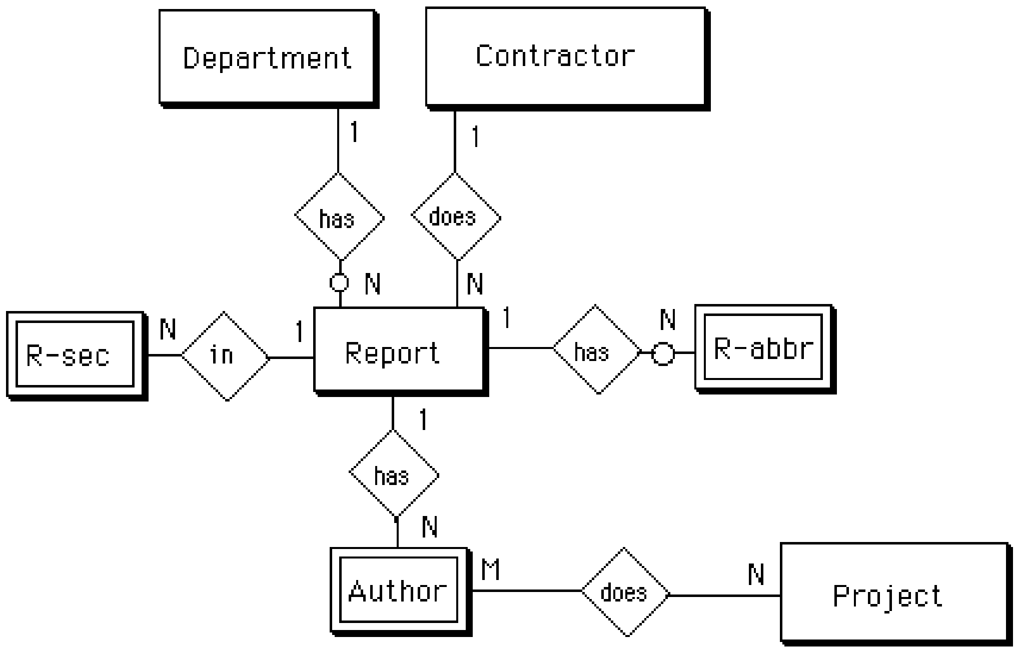
Given an extended ER diagram for a database.....

Step 1. Define points of grouping within functional areas.

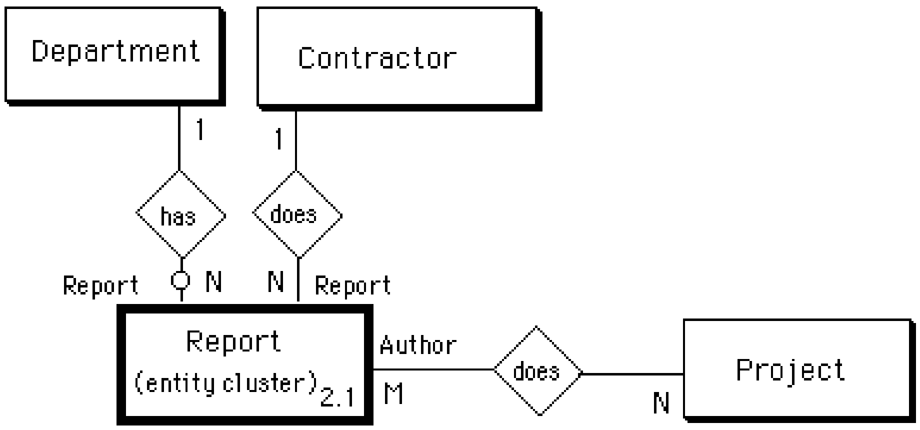
Step 2. Form entity clusters
  * group entities within the same functional area
  * resolve conflicts by combining at a higher functional grouping

Step 3. Form higher entity clusters.

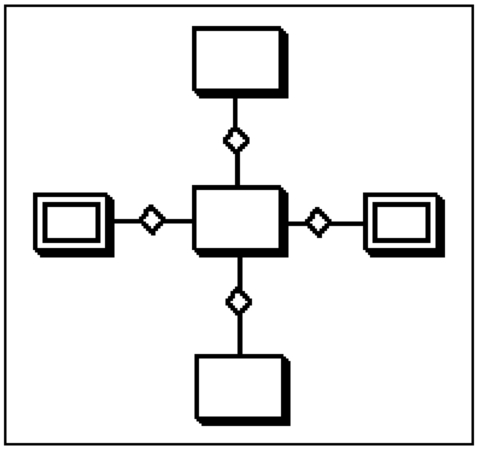
Step 4. Validate the cluster diagram.
  * check for consistency of interfaces.
  * end-users must concur with each level.

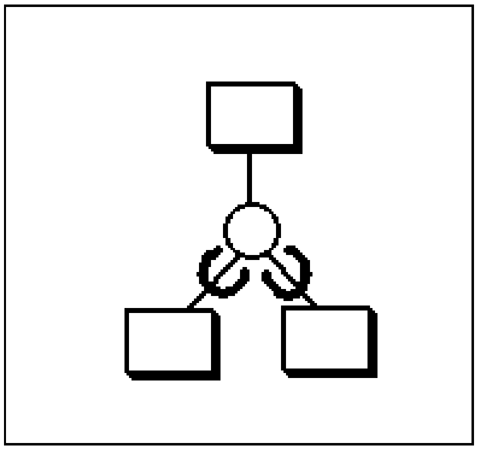


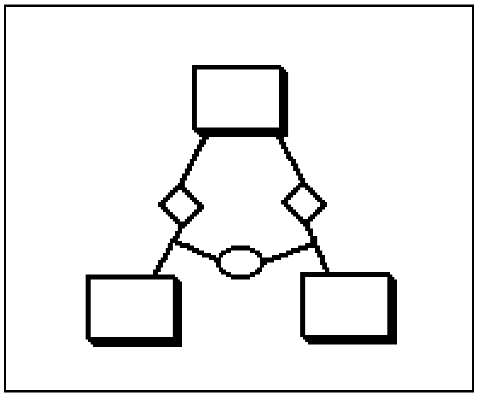
(a) ER model before clustering

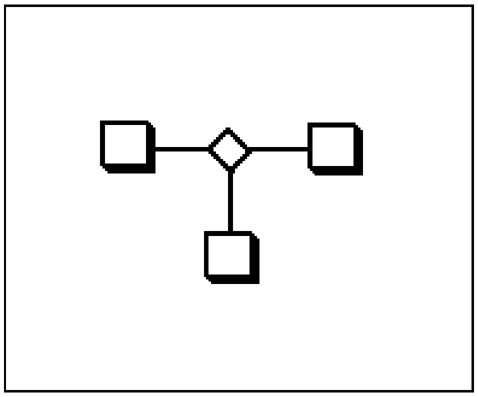


(b) ER model after clustering


(a) Dominance grouping


(b) Abstraction grouping


(c) Constraint grouping


(d) Relationship grouping

Transformations from ER diagrams to SQL Tables

*** Entity** – directly to a SQL table

*** Many-to-many binary relationship** – directly to a SQL table, taking the 2 primary keys in the 2 entities associated with this relationship as foreign keys in the new table

*** One-to-many binary relationship** – primary key on “one” side entity copied as a foreign key in the “many” side entity’s table

*** Recursive binary relationship** – same rules as other binary relationships

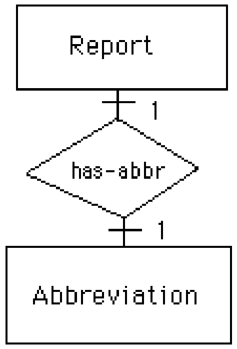
*** Ternary relationship** – directly to a SQL table, taking the 3 primary keys of the 3 entities associated with this relationship as foreign keys in the new table

*** Attribute of an entity** – directly to be an attribute of the table transformed from this entity

*** Generalization super-class (super-type) entity** – directly to a SQL table

*** Generalization subclass (subtype) entity** – directly to a SQL table, but with the primary key of its super-class (super-type) propagated down as a foreign key into its table

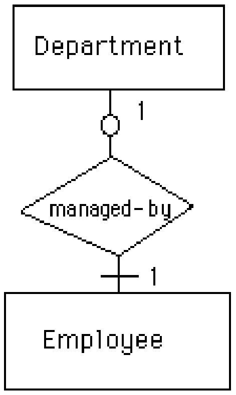
*** Mandatory constraint (1 lower bound) on the “one” side of a one-to-many relationship** – the foreign key in the “many” side table associated with the primary key in the “one” side table should be set as “not null” (when the lower bound is 0, nulls are allowed as the default in SQL)



(a) one-to-one, both
entities mandatory

Every report has one abbreviation, and every
abbreviation represents exactly one report.

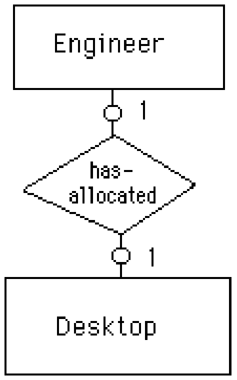
```
create table report
        (report_no  integer,
         report_name  varchar(256),
         primary key(report_no);
create table abbreviation
        (abbr_no  char(6),
         report_no  integer not null unique,
         primary key (abbr_no),
         foreign key (report_no) references report
            on delete cascade on update cascade);
```



(b) one-to-one, one entity
optional, one mandatory

Every department must have a manager, but an
employee can be a manager of at most one department.

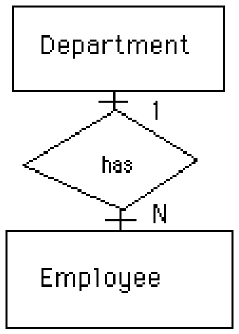
```
create table department
        (dept_no  integer,
         dept_name  char(20),
         mgr_id  char(10) not null unique,
         primary key (dept_no),
         foreign key (mgr_id) references employee
            on delete set default on update cascade);
create table employee
        (emp_id  char(10),
         emp_name  char(20),
         primary key (emp_id));
```



(c) one-to-one,
both entities optional

Some desktop computers are allocated to engineers,
but not necessarily to all engineers.

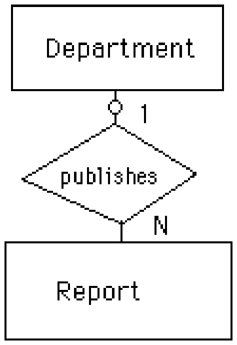
```
create table engineer
        (emp_id  char(10),
         desktop_no  integer,
         primary key (emp_id));

create table desktop
        (desktop_no  integer,
         emp_id  char(10),
         primary key (desktop_no),
         foreign key (emp_id) references engineer
            on delete set null on update cascade);
```



(d) one-to-many, both entities mandatory

Every employee works in exactly one department, and each department has at least one employee.

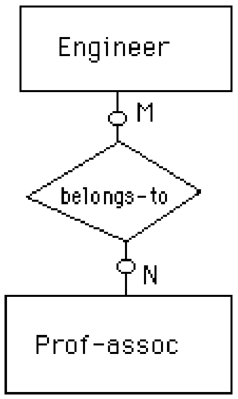
```
create table department
        (dept_no  integer,
         dept_name  char(20),
         primary key (dept_no));
create table employee
        (emp_id  char(10),
         emp_name  char(20),
         dept_no  integer not null,
         primary key (emp_id),
         foreign key (dept_no) references department
           on delete set default on update cascade);
```



(e) one-to-many, one entity optional, one unknown

Each department publishes one or more reports. A given report may not necessarily be published by a department.

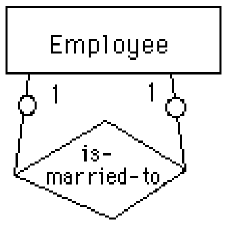
```
create table department
        (dept_no  integer,
         dept_name  char(20),
         primary key (dept_no));
create table report
        (report_no  integer,
         dept_no  integer,
         primary key (report_no),
         foreign key (dept_no) references department
           on delete set null on update cascade);
```



(f) many-to-many, both entities optional

Every professional association could have none, one, or many engineer members. Each engineer could be a member of none, one, or many professional associations.

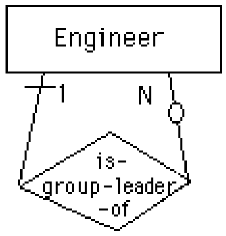
```
create table engineer
        (emp_id  char(10),
         primary key (emp_id));
create table prof_assoc
        (assoc_name  varchar(256),
         primary key (assoc_name));
create table belongs_to
        (emp_id  char(10),
         assoc_name  varchar(256),
         primary key (emp_id, assoc-name),
         foreign key (emp_id) references engineer
           on delete cascade on update cascade,
         foreign key (assoc_name ) references prof-assoc
           on delete cascade on update cascade);
```



(a) one-to-one, both
sides optional

Any employee is allowed to be married
to another employee in this company.

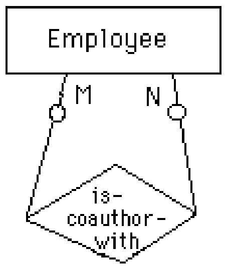
```
create table employee
        (emp_id  char(10),
         emp_name  char(20),
         spouse_id  char(10),
         primary key (emp_id),
         foreign key (spouse_id) references employee
           on delete set null on update cascade);
```



(b) one-to-many, one side
mandatory, many side optional

Engineers are divided into groups for certain
projects. Each group has a leader.

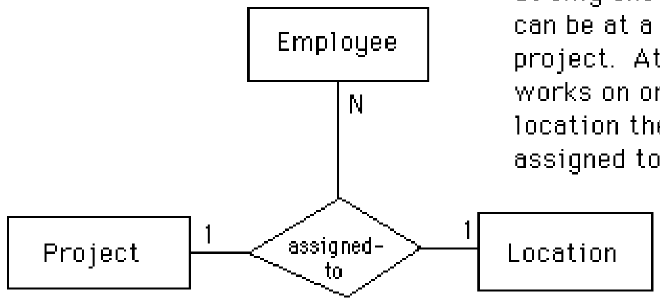
```
create table engineer
        (emp_id  char(10),
         leader_id  char(10) not null,
         primary key (emp_id),
         foreign key (leader_id) references engineer
           on delete set default on update cascade);
```



(c) many-to-many,
both sides optional

Each employee has the opportunity to coauthor a
report with one or more other employees, or to
write the report alone.

```
create table employee
        (emp_id  char(10),
         emp_name  char(20),
         primary key (emp_id));

create table coauthor
        (author_id  char(10),
         coauthor_id  char(10),
         primary key (author_id, coauthor-id),
         foreign key (author_id) references employee
           on delete cascade on update cascade,
         foreign key (coauthor_id) references employee
           on delete cascade on update cascade);
```



Each employee assigned to a project works at only one location for that project, but can be at a different location for a different project. At a given location, an employee works on only one project. At a particular location there can be many employees assigned to a given project.

```
create table employee (emp_id  char(10),
                       emp_name  char(20),
                       primary key (emp_id));
create table project (project_name  char(20),
                      primary key (project_name));
create table location  (loc_name  char(15),
                        primary key (loc_name));
create table assigned_to (emp_id  char(10),
                          project_name  char(20),
                          loc_name  char(15) not null,
                          primary key (emp_id, project_name),
                          foreign key (emp_id) references employee
                            on delete cascade on update cascade,
                          foreign key (project_name) references project
                            on delete cascade on update cascade,
                          foreign key (loc_name) references location
                            on delete cascade on update cascade),
                          unique (emp_id, loc_name));
```

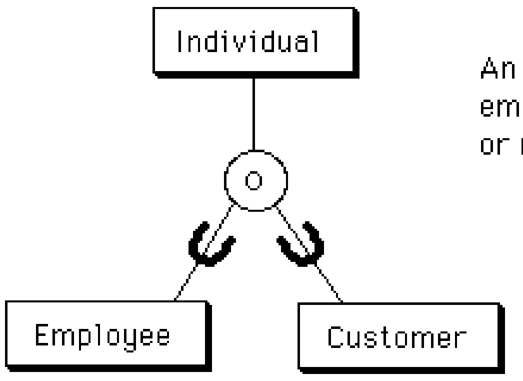
assigned_to

emp_id	project_name	loc_name
48101	forest	B66
48101	ocean	E71
20702	ocean	A12
20702	river	D54
51266	river	G14
51266	ocean	A12
76323	hills	B66

Functional dependencies

emp_id, loc_name -> project_name
emp_id, project_name -> loc_name

(b) one-to-one-to-many ternary relationships



An individual may be either an employee or a customer, or both, or neither.

```
create table individual (indiv_id  char(10),
                          indiv_name  char(20),
                          indiv_addr  char(20),
                          primary key (indiv_id));
create table employee (emp_id  char(10),
                        job_title  char(15),
                        primary key (emp_id),
                        foreign key (emp_id) references individual
                          on delete cascade on update cascade);
create table  customer (cust_no  char(10),
                         cust_credit  char( 12),
                         primary key (cust_no),
                         foreign key (cust_no) references individual
                           on delete cascade on update cascade);
```

IV. Normalization and Normal Forms

First normal form (1NF) to third normal form (3NF) and BCNF

Goals of normalization

1. Integrity
2. Maintainability

Side effects of normalization

* Reduced storage space required (usually, but it could increase)
* Simpler queries (sometimes, but some could be more complex)
* Simpler updates (sometimes, but some could be more complex)

First normal form (1NF) -- a table R is in 1NF iff all underlying domains contain only atomic values, i.e. there are no repeating groups in a row.

functional dependency—given a table R, a set of attributes B is functionally dependent on another set of attributes A if at each instant of time each A value is associated with only one B value. This is denoted by A -> B. A trivial FD is of the form XY --> X (subset).

super-key -- a set of one or more attributes, which, when taken collectively, allows us to identify uniquely an entity or table.

candidate key—any subset of the attributes of a super-key that is also a super-key, but not reducible.

primary key -- arbitrarily selected from the set of candidate keys, as needed for indexing.

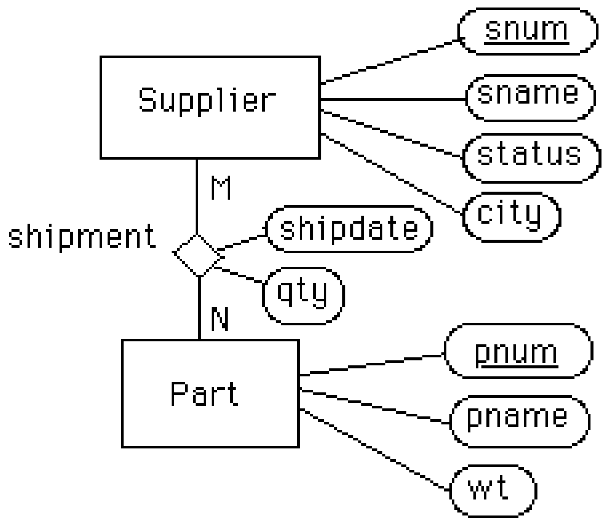
Third normal form (3NF)
-- a table is in 3NF if, for every nontrivial FD X --> A, either:

(1) attribute X is a super-key, or
(2) attribute A is a member of a candidate key (prime attribute)

Boyce-Codd normal form (BCNF)
-- a table is in BCNF if, for every nontrivial FD X --> A,

(1) attribute X is a super-key.

Tables, Functional Dependencies, and Normal Forms



First Normal Form

TABLE SUPPLIER_PART (100k rows, 73 bytes/row => 7.3 MB)

SNUM	SNAME	STATUS	CITY	PNUM	PNAME	WT	QTY	SHIPDATE
S1	SMITH	20	LONDON	P1	NUT	12	3	1-4-90
S1	SMITH	20	LONDON	P2	BOLT	22	2	2-17-90
S1	SMITH	20	LONDON	P3	WRENCH	27	6	11-5-89
S1	SMITH	20	LONDON	P4	WRENCH	24	2	6-30-91
S1	SMITH	20	LONDON	P5	CLAMP	22	1	8-12-91
S1	SMITH	20	LONDON	P6	LEVEL	19	5	4-21-91
S2	JONES	10	PARIS	P1	NUT	12	3	5-3-90
S2	JONES	10	PARIS	P2	BOLT	22	4	12-31-90
S3	BLAKE	10	PARIS	P3	WRENCH	27	4	3-25-91
S3	BLAKE	10	PARIS	P5	CLAMP	22	2	3-27-91
S4	CLARK	20	LONDON	P2	BOLT	22	2	10-31-89
S4	CLARK	20	LONDON	P4	WRENCH	24	3	7-14-90
S4	CLARK	20	LONDON	P5	CLAMP	22	7	8-20-90
S5	ADAMS	30	ATHENS	P5	CLAMP	22	5	8-11-91

Functional dependencies

SNUM --> SNAME, STATUS,CITY
CITY --> STATUS
PNUM --> PNAME, WT
SNUM,PNUM,SHIPDATE --> QTY

Attribute sizes (bytes)

SNUM	5	PNAME	10
SNAME	20	WT	5
STATUS	2	QTY	5
CITY	10	SHIPDATE	8
PNUM	8	Total size	73

Third Normal Form

TABLE PART (100 rows, 23 bytes/row => 2.3 KB)

PNUM	PNAME	WT	Functional dependencies
P1	NUT	12	PNUM --> PNAME, WT
P2	BOLT	17	
P3	WRENCH	17	
P4	WRENCH	24	
P5	CLAMP	12	
P6	LEVEL	19	

TABLE SHIPMENT (100k rows, 26 bytes/row => 2.6 MB)

SNUM	PNUM	QTY	SHIPDATE	Functional dependency
S1	P1	31-4-90	SNUM, PNUM, SHIPDATE--> QTY	
S1	P2	2	2-17-90	
S1	P3	6	11-5-89	
S1	P4	2	6-30-90	
S1	P5	1	8-12-91	
S1	P6	5	4-21-91	
S2	P1	3	5-3-90	
S2	P2	4	12-31-90	
S3	P3	4	3-25-91	
S3	P5	2	3-27-91	
S4	P2	2	10-31-89	
S4	P4	3	7-14-90	
S4	P5	7	8-20-90	
S5	P5	58-11-91		

NOT Third Normal Form

TABLE SUPPLIER (200 rows, 37 bytes/row => 7.4 KB)

SNUM	SNAME	STATUS	CITY	Functional dependencies
S1	SMITH	20	LONDON	SNUM --> SNAME, STATUS, CITY
S2	JONES	10	PARIS	CITY --> STATUS
S3	BLAKE	10	PARIS	
S4	CLARK	20	LONDON	
S5	ADAMS	30	ATHENS	

Decomposition of Table Supplier into two Third Normal Form (3NF) Tables

Third Normal Form

TABLE SUPPLIER_W/O_STATUS (200 rows, 35 bytes/row => 7 KB)

SNUM	SNAME	CITY	Functional dependency
S1	SMITH	LONDON	SNUM --> SNAME, CITY
S2	JONES	PARIS	
S3	BLAKE	PARIS	
S4	CLARK	LONDON	
S5	ADAMS	ATHENS	

TABLE CITY_AND_STATUS (100 rows, 12 bytes/row => 1.2 KB)

CITY	STATUS	Functional dependency
LONDON	20	CITY --> STATUS
PARIS	10	
ATHENS	30	

Relational tables predicted by the ER model, with no functional dependencies given, just those implied by the diagram.
Table 1: emphistory (jobtitle, <u>startdate</u>, enddate, <u>empid</u>)
Table 2: employee (<u>empid</u>

Example of Table Design and Normalization (3NF) from a collection of FDs and an ER diagram

Functional dependencies (FDs) given

empid, startdate --> jobtitle, enddate
empid --> empname, phoneno, officeno, projno, deptno
phoneno --> officeno
projno --> projname, startdate, enddate
deptno --> deptname, mgrid
mgrid --> deptno

Functional Dependency Inference rules
(Armstrong's Axioms)

1. Reflexivity
If Y is a subset of the attributes of X, then X->Y.
X = ABCD, Y = ABC => X->Y
X->X trivial case

2. Augmentation
If X->Y and Z is a subset of table R (i.e. Z is any set of attributes in R), then XZ -> YZ .

3. Transitivity
If X->Y and Y->Z then X->Z.

4. Pseudo-transitivity
If X->Y and YW->Z then XW->Z.
(transitivity is a special case of pseudo-transitivity when W is null)

5. Union
If X->Y and X->Z then X->YZ.

6. Decomposition
If X->YZ then X->Y and X->Z.

<u>**Superkey Rule 1.**</u> **Any FD involving all attributes of a table defines a super-key on the LHS of the FD.**

Given: any FD containing all attributes in the table R(W,X,Y,Z), i.e. XY -> WZ.
Proof:

(1) XY -> WZ	given
(2) XY -> XY	by the reflexivity axiom
(3) XY -> XYWZ	by the union axiom

(4) XY uniquely determines every attribute in table R, as shown in (3)
(5) XY uniquely defines table R, by the definition of a table as having no duplicate rows
(6) XY is therefore a super-key, by the definition of a super-key.

<u>**Super-key Rule 2.**</u> **Any attribute that functionally determines a Super-key of a table, is also a super-key for that table.**

Given: Attribute A is a super-key for table R(A,B,C,D,E), and E -> A.
Proof:
(1) Attribute A uniquely defines each row in table R, by the def. of a super-key

(2) A -> ABCDE	by the definition of a super-key and a relational table
(3) E -> A given	
(4) E -> ABCDE	by the transitivity axiom

(5) E is a super-key for table R, by the definition of a super-key.

3NF Synthesis Algorithm (Bernstein)

Basic definitions

g e H set of FDs

H^+	closure of H - set of all FDs derivable from H using all the FD inference rules
H'	cover of H - any set of FDs from which every FD in H^+ can be derived

H'(non-redundant) – non-redundant cover of H, i.e. a cover which contains no proper subset which is also a cover. Can be determined with quadratic complexity $O(n^2)$.

Example

Given a set of FDs H, determine a minimal set of tables in 3NF,
while preserving all FDs and maintaining only lossless decomposition/joins.

```
H:      AB->C              DM->NP            D->KL
        A->DEFG            D->M
        E->G               L->D
        F->DJ              PR->S
        G->DI              PQR->ST
```

Step 1: Eliminate any extraneous attributes in the left hand sides of the FDs. We want to reduce the left hand sides of as many FDs as possible. In general: XY->Z and X->Z => Y is extraneous **(Reduction Rule 1)**

XYZ->W and X->Y => Y is extraneous **(Reduction Rule 2)**

For this example we mix left side reduction with the union and decomposition axioms:

```
DM->NP =>          D->NP => D -> MNP
D->M               D->M

PQR->ST       =>  PQR->S, PQR->T =>   PQR->.T
PR->S               PR->S             PR->S
```

Step 2: Find a non-redundant cover H' of H, i.e. eliminate any FD derivable from others in H using the inference rules (most frequently the transitivity axiom).

A->E->G => eliminate A->G from the cover
A->F->D => eliminate A->D from the cover

Step 3: Partition H' into tables such that all FDs with the same left side are in one table, thus eliminating any non-fully functional FDs. (Note: creating tables at this point would be a feasible solution for 3NF, but not necessarily minimal.)

R1: AB->C	R4: G->DI	R7: L->D
R2: A->EF	R5: F->DJ	R8: PQR->T
R3: E->G	R6: D->KLMNP	R9: PR->S

Step 4: Merge equivalent keys, i.e. merge tables where all FD's satisfy 3NF.

4.1 Write out the closure of all LHS attributes resulting from Step 3, based on transitivities.

4.2 Using the closures, find tables that are subsets of other groups and try to merge them. Use Rule 1 and Rule 2 to establish if the merge will result in FDs with super-keys on the LHS. If not, try using the axioms to modify the FDs to fit the definition of super-keys.

4.3 After the subsets are exhausted, look for any overlaps among tables and apply Rules 1 and 2 (and the axioms) again.

In this example, note that R7 (L->D) has a subset of the attributes of R6 (D->KLMNP). Therefore we merge to a single table with FDs D->KLMNP, L->D because it satisfies 3NF: D is a super-key by Rule 1 and L is a super-key by Rule 2.

Final 3NF (and BCNF) table attributes, FDs, and candidate keys:

R1: ABC (AB->C with key AB)	R5: DFJ (F->DJ with key F)
R2: AEF (A->EF with key A)	R6: DKLMNP (D->KLMNP, L->D, w/keys D, L)
R3: EG (E->G with key E)	R7: PQRT (PQR->T with key PQR)
R4: DGI (G->DI with key G)	R8: PRS (PR->S with key PR)

Step 4a. Check to see whether all tables are also BCNF. For any table that is not BCNF, add the appropriate partially redundant table to eliminate the delete anomaly.

Maier’s Example using 3NF Synthesis

[Maier,D. *The Theory of Relational Databases*, Computer Science Press, 1983]

R = {A,B,C,D,E,F,G,H,I,J,K }

Functional dependencies (FDs):

(1) E --> A B C D F G H I J K	(7) H I --> J
(2) A B C --> E D F G H I J K	(8) I J --> H
(3) A B D --> E C F G H I J K	(9) H J --> I
(4) G --> H I J	
(5) C F --> K	
(6) D F --> K	

Step 1 - No reduction of determinants necessary.

Step 2 - Find nonredundant cover.

(4) G->HIJ => eliminate HIJ from (1), (2), and (3)
(7) HI->J => reduce (4) to G->HI, eliminating J from (4)
(5) CF -> K => eliminate K from (1) and (3)
(6) DF->K => eliminate K from (2)
(1) E->DFG => eliminate DFG from (2)
(1) E->CFG => eliminate CFG from (3)

Step 3 - Partition into groups with the same left side.

G1: E->ABCDFG	G6: DF->K
G2: ABC->E	G7: HI->J
G3: ABD->E	G8: IJ->H
G4: G->HI	G9: HJ->I
G5: CF->K	

Step 4 - Merge equivalent keys, forming new groups. Construct final set of tables, attributes, FDs, and candidate keys.

R1: ABCDEFG (E->ABCDFG, ABC->E, ABD->E with keys E, ABC, ABD)
R2: GHI (G->HI with key G)
R3: CFK (CF->K with key CF)
R4: DFK (DF->K with key DF
R5: HIJ (HI->J, IJ->H, HJ->I with keys HI, IJ, HJ)

Example of a 3NF table that is not BCNF, i.e. it has further anomalies:

S = student, C = course, I = instructor

SC -> I   For each course, each student is taught by only one instructor. A course may be taught by more than one instructor.

I -> C   Each instructor teaches only one course.

This table is 3NF with a candidate key SC:

SCI	student	course	instructor
	Sutton	Math	Von Neumann
	Sutton	Journalism	Murrow
	Niven	Math	Von Neumann
	Niven	Physics	Fermi
	Wilson	Physics	Einstein

Delete anomaly: If Sutton drops Journalism, then we have no record of Murrow teaching Journalism.
How can we decompose this table into BCNF?

Decomposition 1 (bad)........eliminates the delete anomaly

SC (no FDs) and I -> C (two tables)

Problems - 1. lossy join

2. dependency SC -> I is not preserved

SC	student	course	IC	instructor course	
	Sutton	Math		Von Neumann	Math
	Sutton	Journalism		Murrow Journalism	
	Niven	Math		Fermi	Physics
	Niven	Physics	Einstein	Physics	
	Wilson	Physics			

----------------join SC and IC ------------------

SCI'	student	course	instructor	
	Sutton	Math	Von Neumann	
	Sutton	Journalism	Murrow	
	Niven	Math	Von Neumann	
	Niven	Physics	Fermi	
	Niven	Physics	Einstein	**(spurious row)**
	Wilson	Physics	Fermi	**(spurious row)**
	Wilson	Physics	Einstein	

Decomposition 2 (better).....eliminates the delete anomaly
SI (no FD) and I -> C
Advantages – eliminates the delete anomaly, lossless
Disadvantage - dependency SC -> I is not preserved

SI

student	instructor
Sutton	Von Neumann
Sutton	Murrow
Niven	Von Neumann
Niven	Fermi
Wilson	Einstein
Sutton	Dantzig **(new)**

IC

instructor	course
Von Neumann	Math
Murrow	Journalism
Fermi	Physics
Einstein	Physics
Dantzig	Math **(new)**

The new row is allowed in SI using unique(student,instructor) in the create table command, and the join of SI and IC is lossless. However, a join of SI and IC now produces the following two rows:

student	course	instructor	
Sutton	Math	Von Neumann	
Sutton	Math	Dantzig	which violates the FD SC -> I.

Oracle, for instance, has no way to automatically check SC->I, although you could write a procedure to do this at the expense of a lot of overhead.

Decomposition 3 (tradeoff between integrity and performance)
SC -> I and I -> C (two tables with redundant data)
Problems -extra updates and storage cost

Fourth Normal Form (4NF)

Fourth normal form (4NF) -- a table R is in 4NF iff it is in BCNF and whenever there exists a nontrivial multi-value dependency (MVD) in R, say X-->>Y, X is a super-key for R.

Multi-valued dependency (MVD)
X -->> Y holds whenever a valid instance of R(X,Y,Z) contains a pair of rows that contain duplicate values of X, then the instance also contains the pair of rows obtained by interchanging the Y values in the original pair.

Multi-valued Dependency Inference rules
(Berri, Fagin, Howard...1977 ACM SIGMOD Proc.)
1. Reflexivity X -->> X
2. Augmentation If X -->> Y, then XZ -->> Y.
3. Transitivity If X -->>Y and Y -->> Z then X -->> (Z-Y).
4. Pseudo-transitivity If X -->> Y and YW -->> Z then XW -->> (Z-YW).
(transitivity is a special case of pseudo-transitivity when W is null)
5. Union If X -->> Y and X -->> Z then X -->> YZ.
6. Decomposition If X -->> Y and X -->> Z, then X -->> Y n Z and X -->> (Z-Y)
7. Complement If X -->> Y and Z=R-X-Y, then X -->> Z.
8. FD => MVD If X -> Y, then X -->> Y.
9. FD, MVD mix If X -->> Y and Z -->> W (where W is contained in Y and Y n Z is not empty), then X->W.
Note: n = intersect

Why is 4NF useful?
Avoids certain update anomalies/inefficiencies.

1. delete anomaly - two independent facts get tied together unnaturally so there may be bad side effects of certain deletes, e.g. in “skills_required” the last record of a skill may be lost if employee is temporarily not working on any projects).

2. update inefficiency - adding a new project in “skills_required” requires insertions for many records (rows) that to include all required skills for that new project. Likewise, loss of a project requires many deletes.

3. 4NF maintains smaller pieces of information with less redundancy.

Example of a ternary relationship (many-to-many-to-many) that can be BCNF or 4NF depending on the semantics associated with it.

Table name	**NF**	**2-way decomp.**	**3-way decomp.**	**Nontrivial MVDs**
skill_available	BCNF	yes	yes	6
skill_required	BCNF	yes	yes	2
skill_in_common	4NF	no	yes	0

Semantics and analysis of each relationship

skill_required—an employee must have all the required skills for a project to work on that project.

skill_required

empno	project	skill	Nontrivial MVDs
101	3	A	project->>skill
101	3	B	project->>empno
101	4	A	
101	4	C	
102	3	A	
102	3	B	
103	5	D	

empno	project
101	3
101	4
102	3
103	5

empno	skill
101	A
101	B
101	C
102	A
102	B
103	D

project	skill
3	A
3	B
4	A
4	C
5	D

2-way lossless join occurs when skill_required is projected over {empno, project} and {project, skill}. Projection over {empno, project} and {empno, skill}, and over {empno, skill} and {project, skill}, however, are not lossless. 3-way lossless join occurs when skill_required is projected over {empno, project}, {empno, skill}, {project, skill}.

skill_in_common—an employee must apply the intersection of available skills to the skills needed for different project In other words if an employee has a certain skill and he or she works on a given project that requires that skill, then he or she must provide that skill for that project (this is less restrictive than skill_required because the employee need not supply all the required skills, but only those in common).

skill_in_common

empno	project	skill
101	3	A
101	3	B
101	4	A
101	4	B
102	3	A
102	3	B
103	3	A
103	4	A
103	5	A
103	5	C

empno	project	empno	skill	project	skill	
	101	3	101	A	3	A
	101	4	101	B	3	B
	102	3	102	A	4	A
	103	3	102	B	4	B
	103	4	103	A	5	A
	103	5	103	C	5	C

This has a 3-way lossless decomposition. There are no 2-way lossless decompositions and no MVDs, thus the table is in 4NF.

V. Access Methods

Types of Queries

Query type 1: access all records of a given type

“Increase everyone’s salary by 10%”

access method: sequential processing

Query type 2: access at most one record

“Find the address of John Smith,
whose id number is 333-44-5555”

access methods: hashing, B^+ tree index

Query type 3: access a subset of records of a given type

“Find all employees who have C programming experience and over three years with the company”

access method: secondary indexing (Oracle uses B+trees for this)

Sequential Access Methods

lra = n	logical record accesses
sba = ceil(n/bf)	sequential block accesses
rba = 0	random block accesses

iotime = sba*Tsba + rba*Trba seconds

where Tsba is the average disk i/o service time for a sequential block and Trba is the average disk i/o service time for a random block access

Disk service time in a dedicated environment

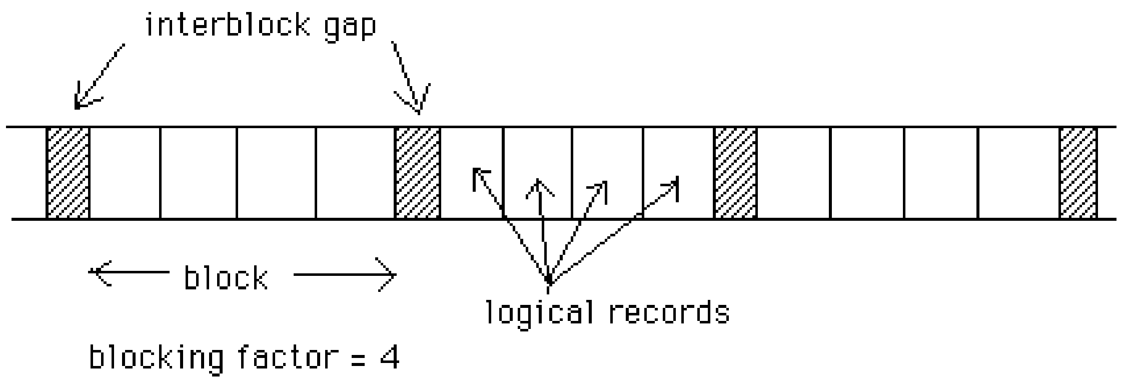
sequential block access:

Tsba = rot/2 + bks/tr

where rot is the disk rotation time (for a full rotation),
bks is the block size in bytes (bf*record size), and
tr is the disk transfer rate in bytes per second.

Trba = seek(avg) + rot/2 + bks/tr

where seek(avg) is the average seek time over the extent of the file on disk



Disk service time in a shared environment

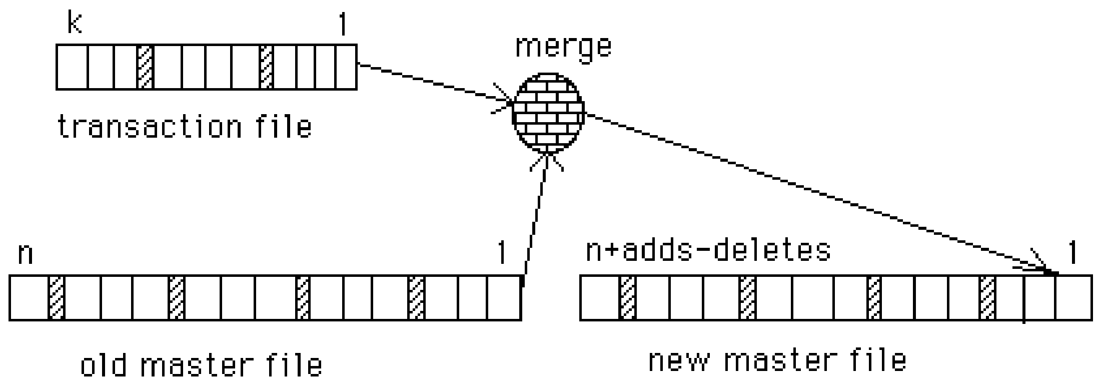
Tsba = **Trba** = seek(avg) + rot/2 + bks/tr
where seek(avg) is the average disk seek time over the extent of the entire disk.

Batch processing of k sequentially stored records

read the transaction file:
lra = k where k = number of transaction records
sba = ceil(k/tfbf) where tfbf is the transaction file blocking factor

read the master file:
lra = n
sba = ceil(n/bf) where bf is the master file blocking factor

write a new master file:
lra = n + adds - deletes
sba = ceil((n+adds-deletes)/bf)
where adds is the number of records added or inserted, and deletes is the number of records deleted.



Random Access Methods

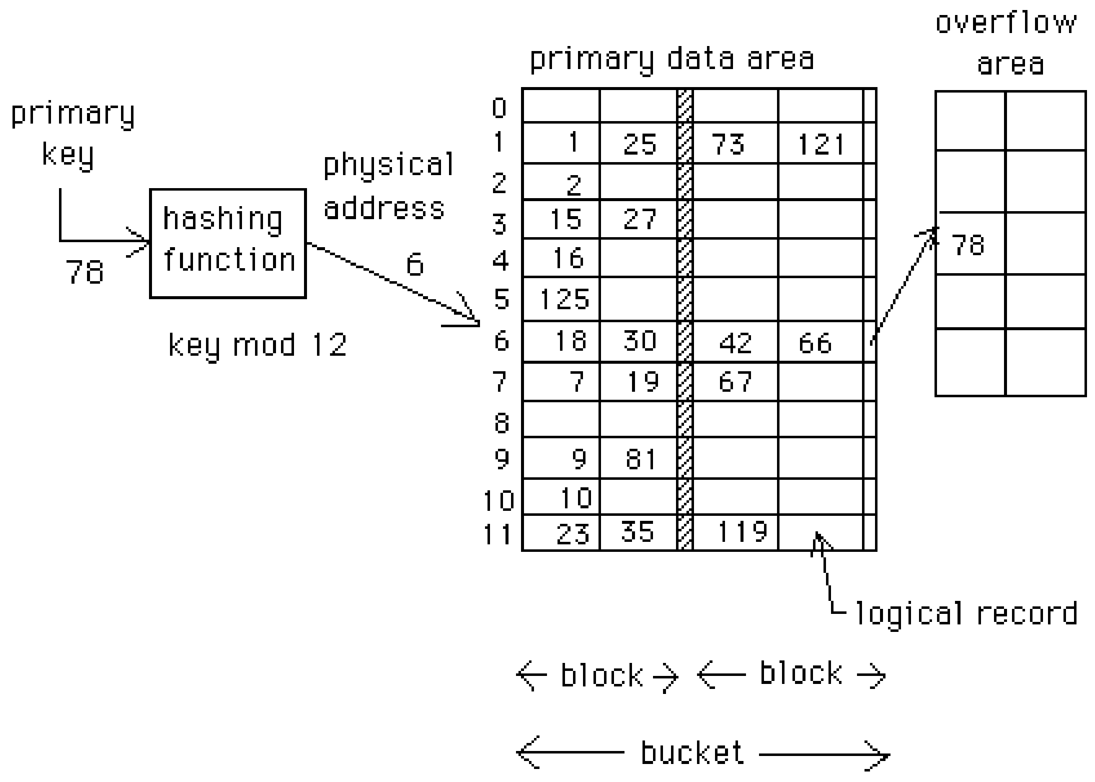
Hashing

Basic mechanism – transformation of a primary key directly to a physical address, called a bucket (or indirectly via a logical address)

Collisions – handled by variations of chained overflow techniques

<u>random access to a hashed file</u>
lra = 1 + overflow(avg)
rba = 1 + overflow(avg)

<u>insertion into a hashed file</u>
lra = 1 + overflow(avg) + rewrite
rba = 1 + overflow(avg)
rba=1 for the rewrite



Extendible Hashing

* number of buckets grow or contracts
* bucket splits when it becomes full (based on first i bits of hash value)
* collisions are resolved immediately, no long overflow chains
* primary key transformed to an entry in the Bucket Address Table (BAT), typically in RAM
* BAT has pointers to disk buckets that hold the actual data
* Retrieve a single record = 1 rba (access the bucket in one step)

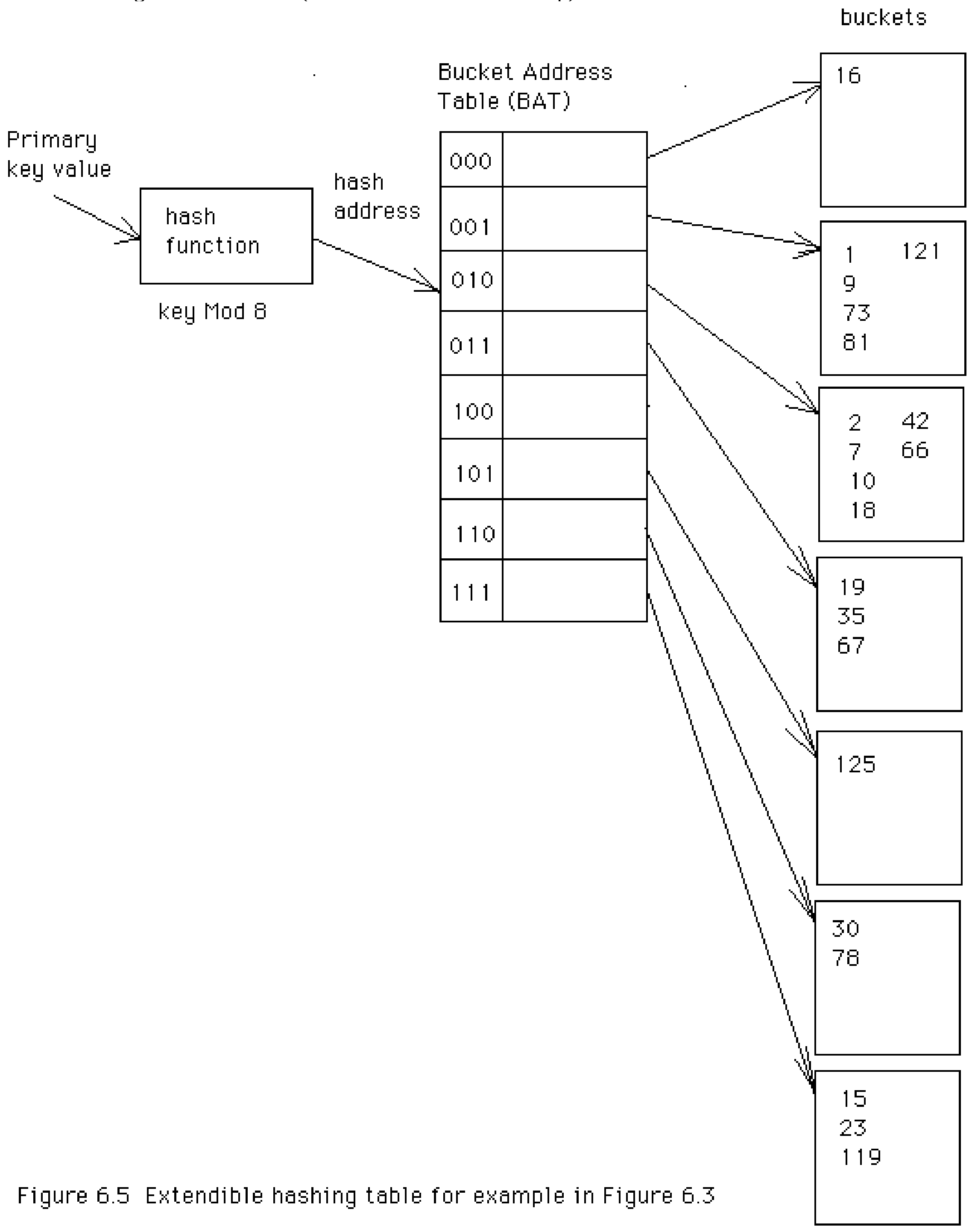


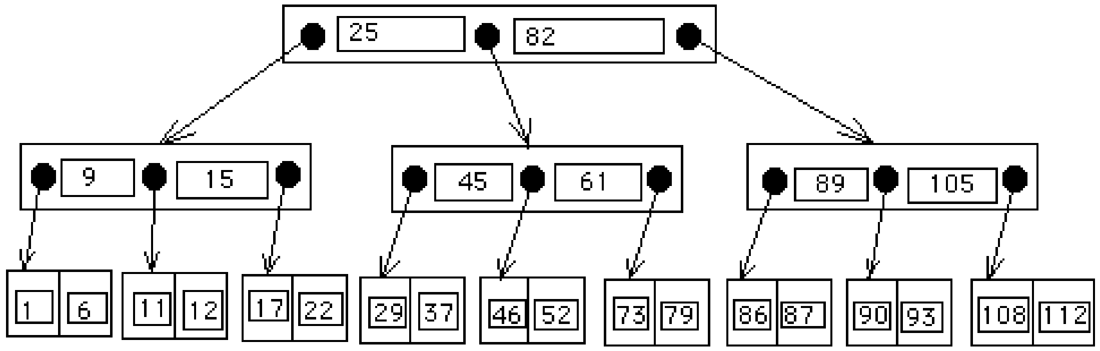
Figure 6.5 Extendible hashing table for example in Figure 6.3

* Cost (service time) of I/O for updates, inserts, and deletes is the same as for B+-trees

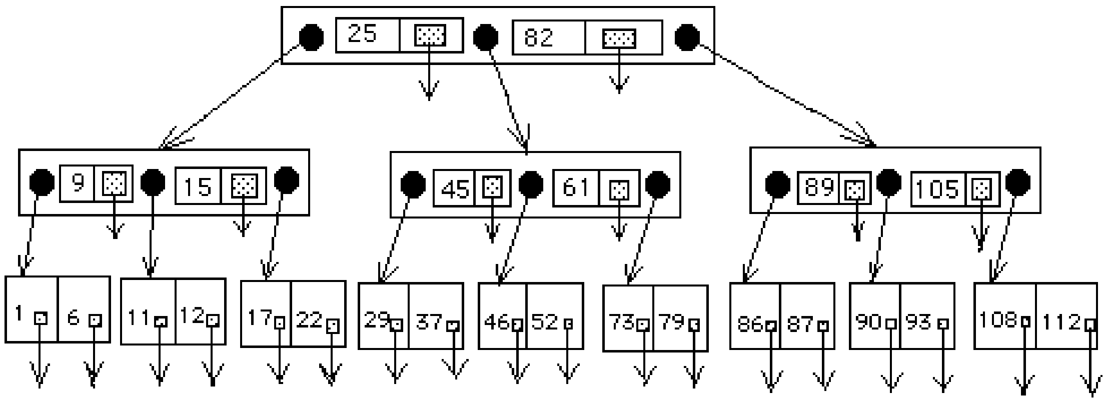
B-trees and B$^+$-trees

B-tree index basic characteristics

* each node contains p pointers and p-1 records
* each pointer at level i is for a data and pointer block at level i+1
* i=1 denotes the root level (single node or block)
· ▯   can be inefficient for searching because of the overhead in each search level



(a) B-tree with embedded records at each node



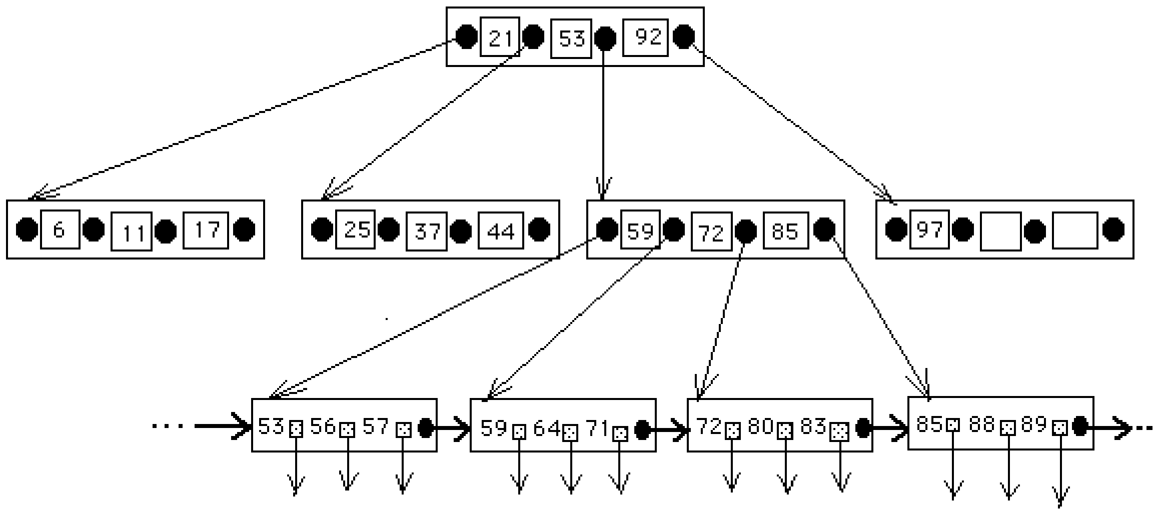
(b) B-tree with key-data pointer pairs in each node

● tree pointer

⊡ data pointer

B+-tree index basic characteristics

* eliminates data pointers from all nodes except the leaf nodes
* each non-leaf index node has p pointers and p-1 key values
* each pointer at level i is for an index block (of key/pointer pairs) at level i+1
* each leaf index has a key value/pointer pair to point to the actual data block (and record) containing that primary key value
* leaf index nodes can be logically connected via pointers for ordered sequence search
* hybrid method for efficient random access and sequential search



Example: B^+-tree

To determine the order of a B^+-tree, let us assume that the database has 500,000 records of 200 bytes each, the search key is 15 bytes, the tree and data pointers are 5 bytes, and the index node (and data block size) is 1024 bytes. For this configuration we have
non-leaf index node size = 1024 bytes = p*5 + (p-1)*15 bytes
p = floor((1024+15)/20) = floor(51.95) = 51
number of search key values in the leaf nodes = floor ((1024-5)/(15+5))=50
h = height of the B+-tree (number of index levels, including the leaf index nodes
n = number of records in the database (or file); all must be pointed at from the next to last level, h-1

$p^{h-1}(p-1) > n$
$(h-1)\log p + \log(p-1) > \log n$
$(h-1)\log p > \log n - \log(p-1)$
$h > 1 + (\log n - \log(p-1)) / \log p$
$h > 1 + (\log 500{,}000 - \log 49)/\log 50 = 3.34$, h=4 (nearest higher integer)

A good approximation can be made by assuming that the leaf index nodes are implemented with p pointers and p key values:

$p^h > n$
$h \log p > \log n$
$h > \log n/\log p$

In this case, the result above becomes h > 3.35 or h = 4.

B+-tree performance

read a single record (B^+-tree) = h+1 rba

update a single record (B^+-tree) = search cost + rewrite data block
= (h+1) rba + 1 rba

general update cost for insertion (B^+-tree)
=search cost (i.e., h+1 reads)
+simple rewrite of data block and leaf index node pointing to the data block (i.e., 2 rewrites)
+nos*(write of new split index node
+ rewrite of the index node pointer to the new index node)
+ nosb*(write of new split data block)

= (h+1) rba + 2 rba + nos*(2 rba) + nosb*(1 rba)

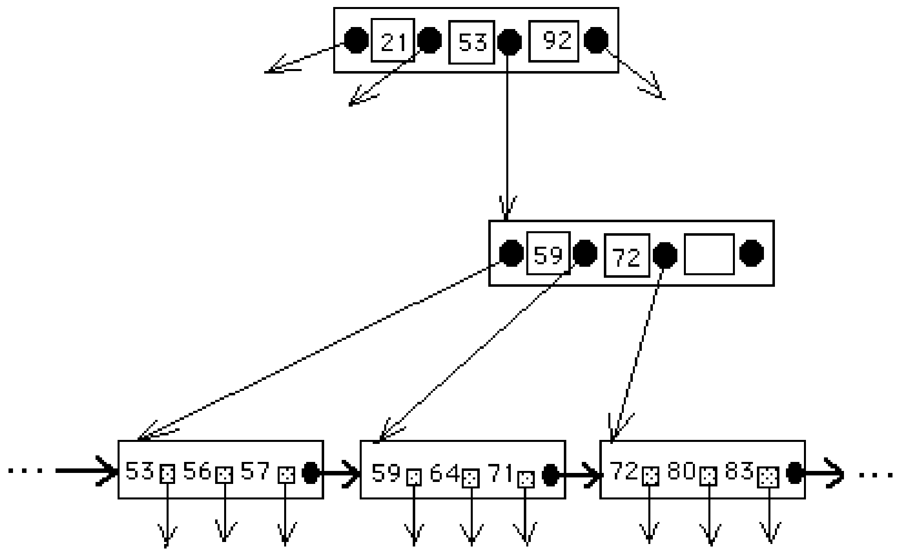
where nos is the number of index split node operations required and nosb is the number of data split block operations required

general update cost for deletion (B^+-tree)
= search cost (i.e., h+1 reads)
+ simple rewrite of data block and leaf index node pointing to the data block (i.e., 2 rewrites)
+ noc*(rewrite of the node pointer to the remaining node)

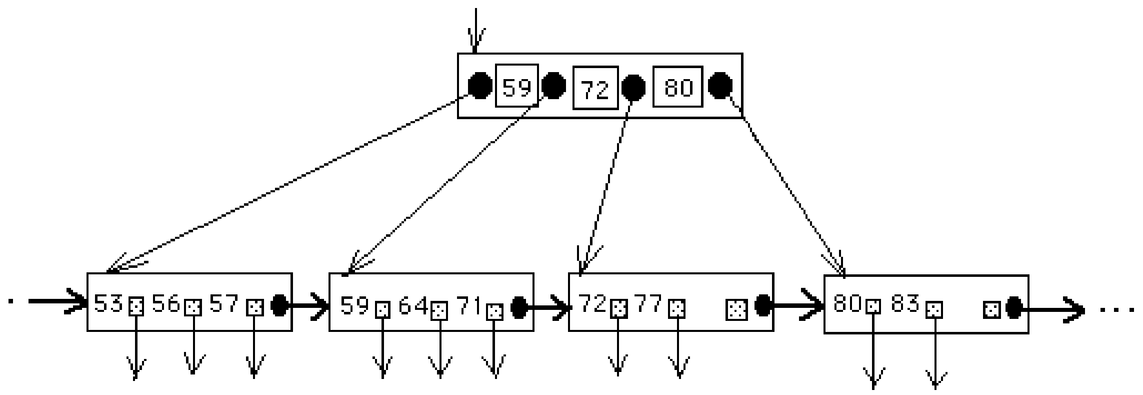
= (h+1) rba + 2 rba + noc*(1 rba)
where noc is the number of consolidations of index nodes required.

As an example, consider the insertion of a node (with key value 77) to the B^+-tree shown in Fig. 6.6. This insertion requires a search (query) phase and an insertion phase with one split node. The total insertion cost for height 3 is

insertion cost = (3 + 1) rba search cost + (2 rba) rewrite cost
+ 1 split *(2 rba rewrite cost)
= 8 rba



(a) B$^+$-tree before the insertion of record with key value 77



(b) B$^+$-tree after the insertion and split block operation

Secondary Indexes

Basic characteristics of secondary indexes
* based on Boolean search criteria (AND, OR, NOT) of attributes that are not the primary key

* attribute type index is level 1 (usually in RAM)

* attribute value index is level 2 (usually in RAM)

* accession list is level 3 (ordered list of pointers to blocks containing records with the given attribute value)

* one accession list per attribute value; pointers have block address and record offset typically

* accession lists can be merged to satisfy the intersection (AND) of records that satisfy more than one condition

Boolean query cost (secondary index)

= search attribute type index + search attribute value index
+ search and merge m accession lists + access t target records

= (0 + 0 + sum of m accession list accesses) sba + t rba

= (sum of m accession list cost) sba + t rba
where m is the number of accession lists to be merged and t is the number of target records to be accessed after the merge operation.

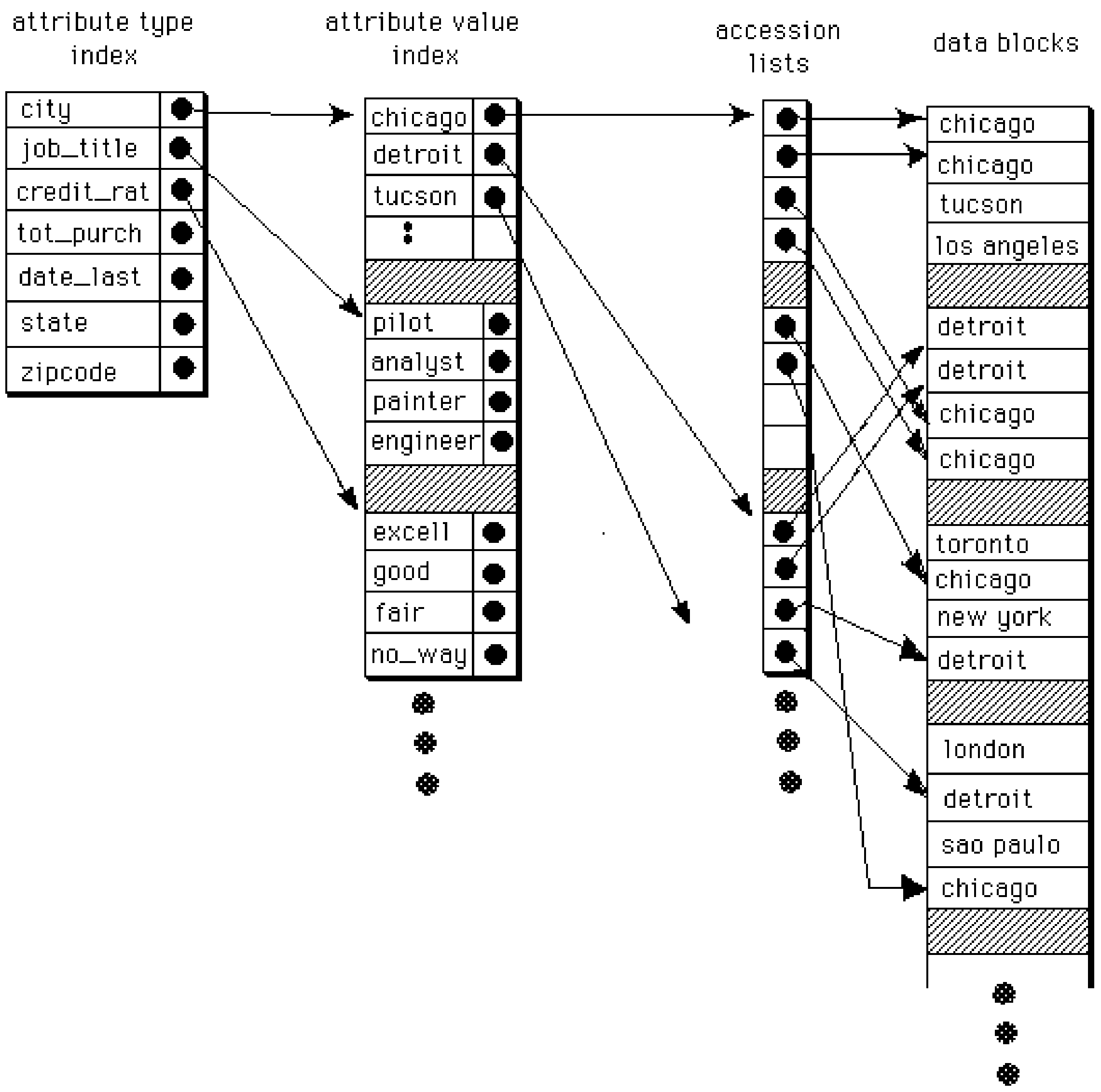
accession list cost (for accession list j) = ceil(pj/bfac) sba
where pj is the number of pointer entries in the jth accession list and bfac is the blocking factor for all accession lists

bfac = block_size/pointer_size

* assume all accesses to the accession list are sequential

* ignore the error incurred by assuming the first record access is sequential

· **use the 1% rule**
(any variable affecting the result by less than 1% is ignored)



Example: Mail Order Business
Assume we have a file of 10,000,000 records of mail order customers for a large commercial business. Customer records have attributes for customer name, customer number, street address, city, state, zip code, phone number, employer, job title, credit rating, date of last purchase, and total amount of purchases. Assume that the record size is 250 bytes; block size is 5000 bytes (bf=20); and pointer size, including record offset, is 5 bytes (bfac=1000). The query to be analyzed is "Find all customers whose job title is 'engineer', city is 'chicago', and total amount of purchases is greater than $1,000." For each AND condition we have the following hit rates, that is, records that satisfy each condition:
job title is 'engineer': 84,000 records
city is 'chicago': 210,000 records
total amount of purchases > $1000: 350,000 records
total number of target records that satisfy all three conditions = 750

query cost (inverted file)
= merge of 3 accession lists + access 750 target records

= [ceil(n1/bfac) + ceil(n2/bfac) + ceil(n3/bfac)] sba + 750 rba

= [ceil(84,000/1000) + ceil(210,000/1000) + ceil(350,000/1000] sba
+ 750 rba
= (84+210+350) sba + 750 rba
= 644 sba + 750 rba

If we assume Tsba is 10 milliseconds and Trba is 25 milliseconds, we obtain
query iotime (secondary index)
= 644 sba*10 ms + 750 rba*25 ms
= 25190 ms
= 25.19 sec (much more efficient than sequential scan, see below)

query iotime (sequential scan)
= ceil(n/bf) sba *Tsba
= ceil(10,000,000/20)*10 ms
= 5,000,000 ms
= 5000 sec

Secondary Indexes using B$^+$-trees

* used by Oracle and many others for non-unique indexes

* index nodes contain key/pointer pairs in the same way as a primary key index using a B+-tree

* key at each level, leaf and non-leaf, is the concatenation of attributes used in the query , e.g. jobtitle, city, total_purchases (as attributes of consumer)

* leaf node pointers are to the blocks containing records with the given combination of attribute values indicated in the concatenated keys

* analysis of queries and updates for this type of index proceeds in the same way as a primary key (unique) index, keeping in mind that the key formats are different in the two cases

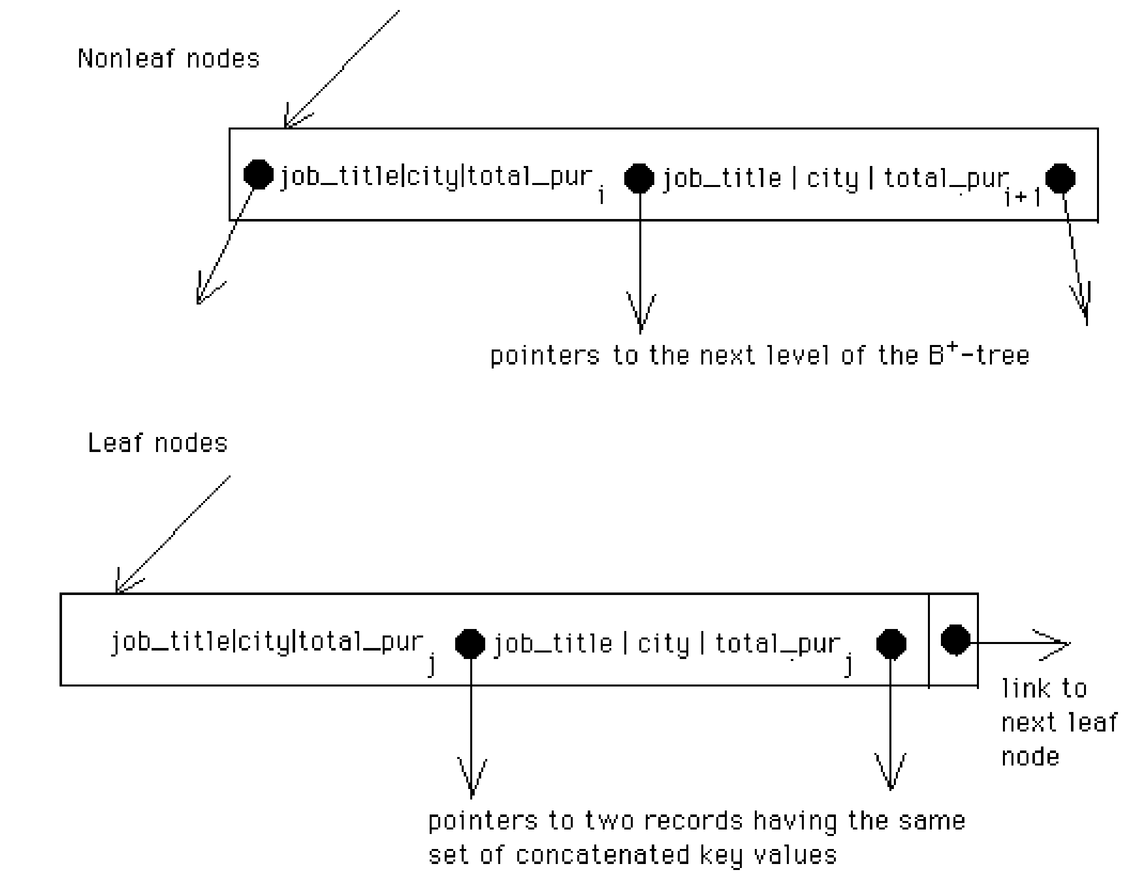

Nonleaf nodes
job_title|city|total_pur i
job_title | city | total_pur i+1
pointers to the next level of the B+-tree
Leaf nodes
job_title|city|total_pur j
job_title | city | total_pur j
link to next leaf node
pointers to two records having the same set of concatenated key values

Figure 6.10 Using a B$^+$-tree for a secondary index

Denormalization

* motivation – poor performance by normalized databases

* search for potential denormalizations that avoid or minimize delete anomalies

To illustrate the effect of denormalization, let us assume that the table **review** is associated with the tables **employee** and **manager** as the table that follows shows. The extension of the **review** table, **review-ext**, is shown as a means of reducing the number of joins required in the query shown below. This extension results in a real denormalization, that is,

review_no -> emp_id -> emp_name, emp_address

with the side effects of add and update anomalies. However, the delete anomaly cannot occur because the original data is redundant in the extended schema.

Original Tables and Process (Query)

Table	**Primary Key**	**Nonkeys**
employee	emp_id	emp_name, emp_address, mgr_id

Table Denormalization Algorithm

1. Select the dominant processes based on such criteria as high frequency of execution, high volume of data accessed, response time constraints, or explicit high priority.

2. Define join tables, when appropriate, for the dominant processes.

3. Evaluate total cost for storage, query, and update for the database schema, with and without the extended table, and determine which configuration minimizes total cost.

4. Consider also the possibility of denormalization due to a join table and its side effects. If a join table schema appears to have lower storage and processing cost and insignificant side effects, then consider using that schema for physical design in addition to the original candidate table schema. Otherwise use only the original schema.

Join Strategies

1. nested loop: complexity O(mn)
2. merge-join: complexity $O(n \log_2 n)$
3. indexed join: complexity O(2m)
4. hash-join: complexity O(m+n)

where m and n are the rows of the two tables to be joined
Assume

* **assigned_to** table has 50,000 rows
* **project** table has 250 rows
* let the blocking factors for the **assigned_to** and **project** tables be 100 and 50, respectively, and the block size is equal for the two tables.
* the common join column is project_name.

```
select project_name, emp_id
      from project as p, assigned_to as a
      where p.project_name = a.project_name;
```

Nested Loop Case 1: **assigned_to** is the outer loop table.

join cost = m/bfm + m*n/bfn
= 50,000/100 + 50,000*250/50
= 500 + 250,000
= 250,500 sequential block accesses (sba)

If a sequential block access requires an average of 10 ms, the total time required is 2505 seconds.

Nested Loop Case 2: **project** is the outer loop table.

join cost = 250/50 + 250*50,000/100
= 5 + 125,000
= 125,005 sequential block accesses (or 1250 seconds)

Note that this strategy does not take advantage of row order for these tables
Merge-Join Case 1: Both **project** and **assigned_to** are already ordered by project_name.

join cost = merge time (to scan both tables)
= 50,000/100 + 250/50
= 505 sequential block accesses (or 5 seconds)

Merge-Join Case 2: Only **project** is ordered by project_name.

join cost = sort time for **assigned_to** + merge time (to scan both sorted tables)
= $(50{,}000 \cdot \log_2 50{,}000)/100 + 50{,}000/100 + 250/50$
= (50,000*16)/100 + 500 + 5
= 8505 sequential block accesses (or 85 seconds)

Merge-Join Case 3: Neither **project** nor **assigned_to** are ordered by project_name.

join cost = sort time for both tables + merge time for both tables
= (50,000*$\log_2$ 50,000)/100 +(250*$\log_2$ 250)/50 + 50,000/100
+ 250/50
= 8000 + 40 + 500 + 5
= 8545 sequential block accesses (or 85 seconds)

select project_name, emp_id
from **project** as p, **assigned_to** as a
where p.project_name = a.project_name
and p.project_name = 'financial analysis';

Indexed join basic algorithm:

join cost = scan entire first table (**assigned_to**)
+ access second table (**project**) qualifying rows = 50,000/100 sba + 100 rba = 500 sba + 100 rba

If Tsba=10 ms and Trba=40 ms, then the total iotime is 9 seconds.
As in the indexed join example above, let mt=100 and nt=5 qualifying rows for the first and second tables, respectively.
Hash join basic algorithm:
join cost = scan first table (**assigned_to**) + scan second table (**project**)
+ access qualifying rows in the two tables = 50,000/100 sba + 250/50 sba + 100 rba + 5 rba
= 505 sba + 105 rba
Thus we get iotime of 9.25 seconds for this case when Tsba=10 ms and Trba=40 ms.

VI. Database Distribution Strategies

Overview of Distributed Databases

Distributed database - a collection of multiple, logically interrelated databases distributed over a computer network [OzVa91].

Distributed Database Management System (DDBMS) - a software system that permits the management of a distributed database and makes the distribution transparent to the users. If heterogeneous, it may allow transparent simultaneous access to data on multiple dissimilar systems.

Advantages

1. Improves performance, e.g. it saves communication costs and reduces query delays by providing data at the sites where it is most frequently accessed.

2. Improves the reliability and availability of a system by providing alternate sites from where the information can be accessed.

3. Increases the capacity of a system by increasing the number of sites where the data can be located.

4. Allows users to exercise control over their own data while allowing others to share some of the data from other sites.

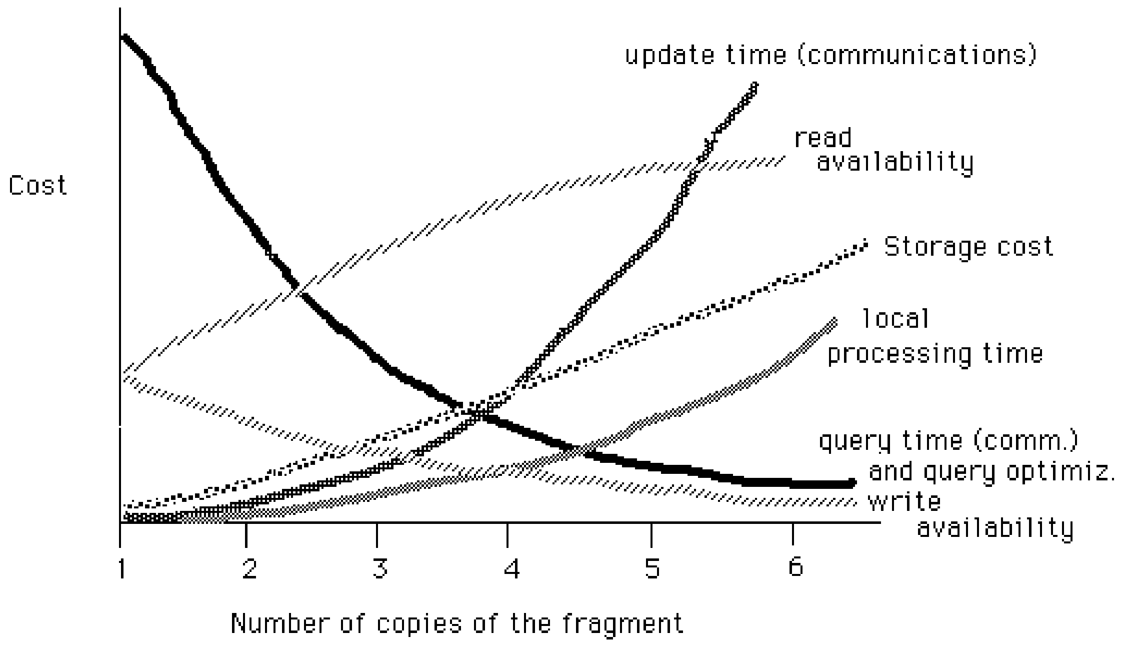
5. Helps solve more complex database problems.

Disadvantages

1. Increases the complexity of the system and introduces several technical as well as management challenges especially when geographical and organizational boundaries are crossed.

2. Makes central control more difficult and raises several security issues because a data item stored at a remote site can be always accessed by the users at the remote site.

3. Makes performance evaluation difficult because a process running at one node may impact the entire network.


Cost
update time (communications)
read availability
Storage cost
local processing time
query time (comm.) and query optimiz.
write availability
1
2
3
4
5
6
Number of copies of the fragment

Requirements of a Generalized DDBMS: Date's 12 Rules

Rule 1. Local Autonomy. Local data is locally owned and managed, even when it is accessible by a remote site. Security, integrity, and storage remain under control of the local system. Local users should not be hampered when their system is part of a distributed system.

Rule 2. No Central Site. There must be no central point of failure or bottleneck. Therefore the following must be distributed: dictionary management, query processing, concurrency control, and recovery control.

Rule 3. Continuous Operation. The system should not require a shutdown to add or remove a node from the network. User applications should not have to change when a new network is added, provided they do not need information from the added node.

Rule 4. Location Independence (or Transparency). A common global user view of the database should be supported so that users need not know where the data is located. This allows data to be moved for performance considerations or in response to storage constraints without affecting the user applications.

Rule 5. Fragmentation Independence (or Transparency). This allows tables to be split among several sites, transparent to user applications. For example, we can store New York employee records at the New York site and Boston employees at the Boston site, but allow the user to refer to the separated data as EMPLOYEES, independent of their locations.

Rule 6. Replication Independence (or Transparency). This allows several copies of a table (or portions therec to reside at different nodes. Query performance can be improved since applications can work with a local copy instead of a remote one. Update performance, however, may be degraded due to the additional copies. Availability can improve.

Rule 7. Distributed Query Processing. No central site should perform optimization; but the submitting site, which receives the query from the user, should decide the overall strategy. Other participants perform optimization at their own levels.

Rule 8. Distributed Transaction Processing. The system should process a transaction across multiple databases exactly as if all of the data were local. Each node should be capable of acting as a coordinator for distributed updates, and as a participant in other transactions. Concurrency control must occur at the local level (Rule 2), but there must also be cooperation between individual systems to ensure that a "global deadlock" does not occur.

Rule 9. Hardware Independence. The concept of a single database system must be presented regardless of the underlying hardware used to implement the individual systems.

Rule 10. Operating System Independence. The concept of a single database system must be presented regardless of the underlying operating systems used.

Rule 11. Network Independence. The distributed system must be capable of communicating over a wide variety ol networks, often different ones in the same configuration. Standard network protocols must be adhered to.

Rule 12. DBMS Independence (Heterogeneity). The distributed system should be able to be made up of individual sites running different database management systems.

What are the basic issues in the design and implementation of distributed database systems?

* Data Distribution Strategies
  - Fragmentation
  - Data allocation
  - Replication
  - Network data directory distribution

* Query Processing and Optimization

* Distribution Transparency
  - location, fragmentation, replication, update

* Integrity
  - Transaction management
  - Concurrency control
  - Recovery and availability
  - Integrity constraint checking

* Privacy and Security
  - Database administrators

* Data Manipulation Languages
  - SQL is the standard
  - Forms coming into common use

Modified Life Cycle for Data Distribution

IV.1 Fragmentation (or partitioning). Define a fragmentation schema of the database based on dominant applications' "select" predicates (set of conditions for retrieval specified in a select statement). A *fragmentation schema* describes the one-to-many mapping used to partition each global table into fragments. Fragments are logical portions of global tables which are physically located at one or several sites of the network.

IV.2 Data allocation. Create a data allocation schema that indicates where each copy of each fragment is to be stored. The *allocation schema* defines at which site(s) a fragment is located. A one-to-one mapping in the allocation schema results in non-redundancy, while a one-to-many mapping defines a redundant distributed database. The set of fragments of a global table, located at a given site, constitutes its physical image at that site.

Fragmentation
A table r is fragmented by partitioning it into a number of disjoint sub-tables (fragments) r_1, r_2, ..., r_n. These fragments contain sufficient information to reconstruct the original table r.

Horizontal fragmentation partitions the rows of a global table into subsets. A fragment r_1 is a *selection* on the global table r using a predicate P_i, its *qualification.* The reconstruction of r is obtained by taking the union of all fragments.

Vertical fragmentation subdivides the attributes of the global table into groups. The simplest form of vertical fragmentation is decomposition. A unique *row-id* may be included in each fragment to guarantee that the reconstruction through a join operation is possible.

Mixed fragmentation is the result of the successive application of both fragmentation techniques.

Rules for Fragmentation

1. Fragments are formed by the select predicates associated with dominant database transactions. The predicates specify attribute values used in the conjunctive (AND) and disjunctive (OR) form of select commands, and rows (records) containing the same values form fragments.

2. Fragments must be disjoint and their union must become the whole table. Overlapping fragments are too difficult to analyze and implement.

3. The largest fragment is the whole table. The smallest fragment is a single record. Fragments should be designed to maintain a balance between these extremes.

Data Distribution

Data distribution defines the constraints under which data allocation strategies may operate. They are determined by the system architecture and the available network database management software. The four basic data distribution approaches are :

* **Centralized**

In the centralized database approach, all the data are located at a single site. The implementation of this approach is simple. However, the size of the database is limited by the availability of the secondary storage at the central site. Furthermore, the database may become unavailable from any of the remote sites when communication failures occur, and th database system fails totally when the central site fails.

* **Partitioned**

In this approach, the database is partitioned into disjoint fragments, and each fragment is assigned to a particular sit This strategy is particularly appropriate where either local secondary storage is limited compared to the database size, the reliability of the centralized database is not sufficient, or operating efficiencies can be gained through the exploitation of the locality of references in database accesses.

* **Replicated**

The replicated data distribution strategy allocates a complete copy of the database to each site in the network. This completely redundant distributed data strategy is particularly appropriate when reliability is critical, the database is small, ar update inefficiency can be tolerated.

* **Hybrid**

The hybrid data distribution strategy partitions the database into critical and non-critical fragments. Non-critical fragments need only be stored once, while critical fragments are duplicated as desired to meet the required level of reliability

Distributed Database Requirements

Database Description

1. Conceptual schema (ER diagram)

2. Transactions: functions and data accessed

Configuration Information

1. Sources of data—where data can be located.

2. Sinks of data—where user transactions can be initiated and data transferred.

3. Transaction rate (frequency) and volume (data flow).

4. Processing capability at each site—CPU and I/O capability (speed).

5. Security—data ownership (who can update) and access authorization (who can query) for each transaction.

6. Recovery—estimated frequency and volume of backup operations.

7. Integrity—referential integrity, concurrency control, journaling overhead, etc.

Constraints

1. Network topology: ring, star, bus, etc.

2. Processing capability needed at each site.

3. Channel (link) transmission capacity.

4. Availability—related to mean-time-between-failures (MTBF) and mean-time-to-repair (MTTR).

Objective Functions

1. Response time as a function of transaction size.

2. Total system cost—communications, local I/O, cpu time, disk space.

The General Data Allocation Problem

Given

1. the application system specifications
   - A database global schema and fragmentation schema.
   - A set of user transactions and their frequencies.
   - Security, i.e. data ownership (who can update) and access authorization (who can query) for each transaction.
   - Recovery, estimated frequency and volume of backup operations.

2. The distributed system configuration and software:
   - The network topology, network channel capacities, and network control mechanism.
   - The site locations and their processing capacity (CPU and I/O processing).
   - Sources of data (where data can be located), and sinks of data (where user transactions can be initiated and data transferred).
   - The transaction processing options and synchronization algorithms.
   - The unit costs for data storage, local site processing, and communications.

Find

the allocation of programs and database fragments to sites which minimizes C, the total cost:

$$C = C_{comm} + C_{proc} + C_{stor}$$

where:

C_{comm} = communications cost for message and data.

C_{proc} = site processing cost (CPU and I/O).

C_{stor} = storage cost for data and programs at sites.

subject to possible additional constraints on:

* Transaction response time which is the sum of communication delays, local processing, and all resource queuing delays.

* Transaction availability which is the percentage of time the transaction executes with all components available.

The Non-Redundant “Best Fit” Method

A general rule for data allocation states that data should be placed as close as possible to where it will be used, and then load balancing should be considered to find a global optimization of system performance.

The non-redundant “best fit” method determines the single most likely site to allocate a fragment (which may be a file, table, or subset of a table) based on maximum benefit, where benefit is interpreted to mean total query and update references. In particular, place fragment F_i at the site s^* where the number of local query and update references by all the user transactions are maximized.

Example

System Parameters

Fragment	Size	Avg. Service Time Local Query(Update)	Avg. Service Time Remote Query(Update)
F1	300 KBytes	100 ms (150 ms)	500 ms (600 ms)
F2	500 KBytes	150 ms (200 ms)	650 ms (700 ms)
F3	1.0 Mbytes	200 ms (250 ms)	1000 ms (1100 ms)

User transactions are described in terms of their frequency of occurrence, which fragments they access, and whether the accesses are reads or writes.

Transact	Site(s)	Frequency	Fragment Accesses (Reads,Writes)
T1	S1,S4,S5	1	F1 (3 reads, 1 write), F2 (2 reads)
T2	S2,S4	2	F1 (2 reads), F3 (3 reads, 1 write)
T3	S3,S5	3	F2 (3 reads, 1 write), F3 (2 reads)

Security: User transactions T1,T2,T3 can either query or update (no restrictions).
Sources of data: All sites S1 - S5.
Sinks of data (possible locations of transactions): All sites S1 - S5.

Local Reference Computations

Our goal is to compute the number of local references to each fragment residing at each site, one by one. The site that maximizes the local references to a given fragment is chosen as the site where that fragment should reside.

Fragment	Site	Trans. T1(freq)	T2(freq)	T3(freq)	Total local refs
F1	S1	3 read,1 write(1)	0	0	4
	S2	0	2 read(2)	0	4
	S3	0	0	0	0
	S4	3 read,1 write(1)	2 read(2)	0	8 (max.)
	S5	3 read,1 write(1)	0	0	4
F2	S1	2 read(1)	0	0	2
	S2	0	0	0	0
	S3	0	0	3 read,1 write(3)	12
	S4	2 read(1)	0	0	2
	S5	2 read(1)	0	3 read,1 write(3)	14 (max.)
F3	S1	0	0	0	0
	S2	0	3 read,1 write(2)	0	8 (max.)
	S3	0	0	2 read(3)	6
	S4	0	3 read,1 write(2)	0	8 (max.)
	S5	0	0	2 read(3)	6

Table. Local references for each fragment at each of five possible sites.

Allocation Decision

Allocate F1 at site S4.
Allocate F2 at site S5.
Allocate F3 at either site S2 or S4

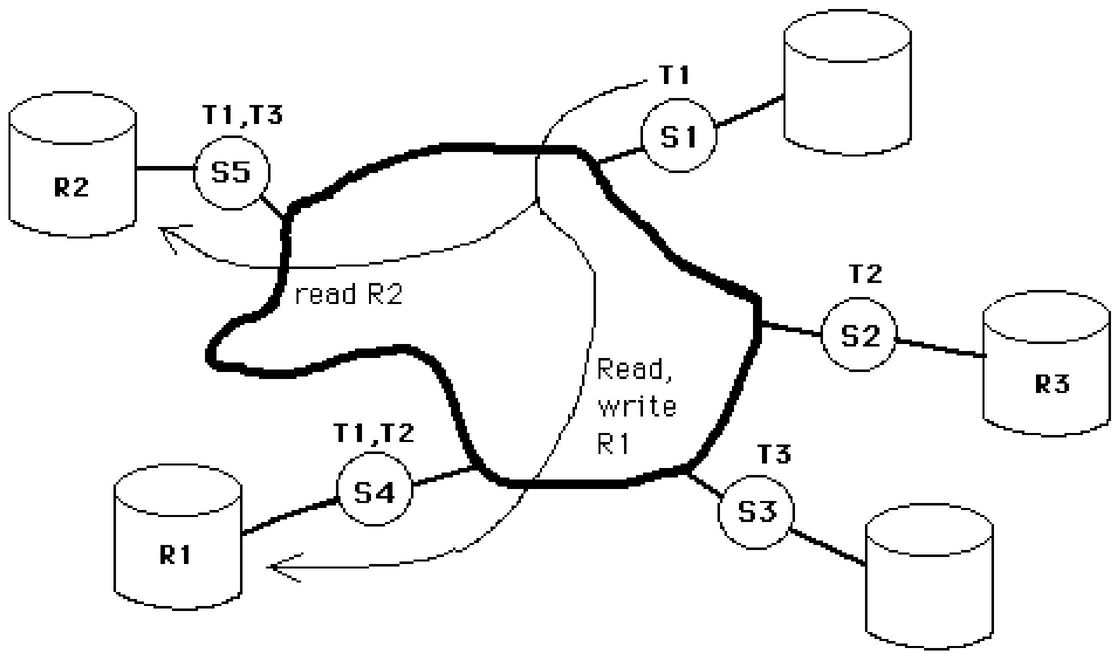
Additional information is needed to choose this allocation. For instance, if maximum availability of data is a major consideration, then choose site S2 because site S4 already has fragment F1 allocated to it and putting F3 there as well would decrease the potential availability of data should site S4 crash.

Advantages

- simple algorithm

Disadvantages

- number of local references may not accurately characterize time or cost (reads and writes given equal weights)
- no insights regarding replication



Relations (tables): R1, R2, R3
Sites: S1, S2, S3, S4, S5
Transactions: T1, T2, T3

The Redundant “All Beneficial Sites” Method

This method can be used for either the redundant or non-redundant case. It selects all sites for a fragment allocation where the benefit is greater than the cost for one additional copy of that fragment. You are assumed to start with zero copies.

The **benefit** for fragment F at site S is measured by the difference in elapsed time to do a remote query to fragment F from site S (i.e. no replicated copy available locally) and a local query to fragment F at site S (i.e. replicated copy available locally).
Total benefit for fragment F at site S is the weighted sum of benefit for each query times the frequency of queries.

The **cost** for fragment F at site S is the total elapsed time for all the local updates of fragment F, plus the total elapsed time for all the remote updates for the given fragment at that site.
Total cost for fragment F at site S is weighted sum of cost for each update transaction times the frequency of update transactions.

Example
Cost/Benefit Computations for “All Beneficial Sites”

Fragment	Site	Remote updates (local updates)	No. of writes*freq*time	**Cost**
F1	S1	T1 from S4 and S5 (T1 from S1)	2*1*600 ms +(1*1*150)	1350 ms
	S2	T1 from S1, S4, S5	3*1*600 ms	1800 ms
	S3	T1 from S1, S4, S5	3*1*600 ms	1800 ms
	S4	T1 from S1 and S5 (T1 from S4)	2*1*600 ms +(1*1*150)	1350 ms
	S5	T1 from S1 and S4 (T1 from S5)	2*1*600 ms +(1*1*150)	1350 ms
F2	S1	T3 from S3 and S5	2*3*700 ms	4200 ms
	S2	T3 from S3 and S5	2*3*700 ms	4200 ms
	S3	T3 from S5 (T3 from S3)	1*3*700 ms +(1*3*200)	2700 ms
	S4	T3 from S3 and S5	2*3*700 ms	4200 ms
	S5	T3 from S3 (T3 from S5)	1*3*700 ms +(1*3*200)	2700 ms
F3	S1	T2 from S2 and S4	2*2*1100 ms	4400 ms
	S2	T2 from S4 (T2 from S2)	1*2*1100 ms +(1*2*250)	2700 ms
	S3	T2 from S2 and S4	2*2*1100 ms	4400 ms
	S4	T2 from S2 (T2 from S4)	1*2*1100 ms +(1*2*250)	2700 ms
	S5	T2 from S2 and S4	2*2*1100 ms	4400 ms

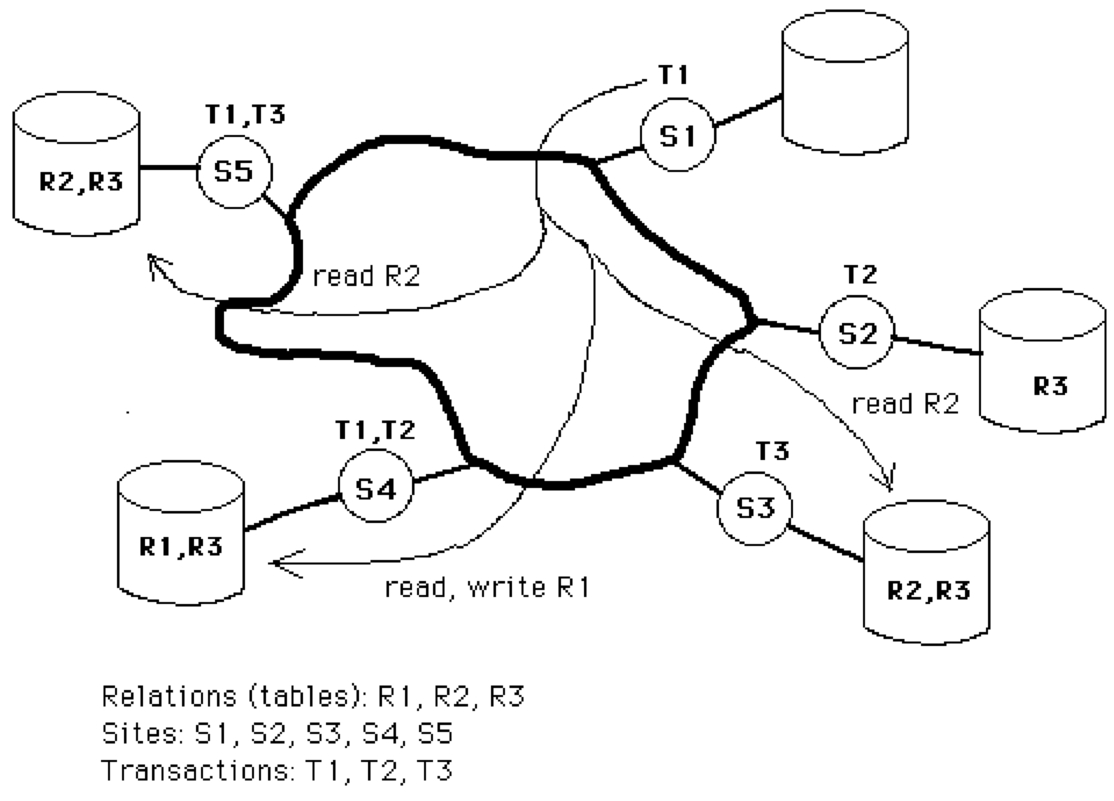
Fragment	Site	Query (read) sources	No. of reads*freq*(remote-local time)	**Benefit**
F1	S1	T1 at S1	3*1*(500 - 100)	1200 ms
	S2	T2 at S2	2*2*(500 - 100)	1600 ms
	S3	None	0	0
	S4	T1 and T2 at S4	(3*1 + 2*2)*(500 - 100)	2800 ms**
	S5	T1 at S5	3*1*(500 - 100)	1200 ms

F2	S1	T1 at S1	2*1*(650 - 150)	1000 ms
	S2	None	0	0
	S3	T3 at S3	3*3*(650 - 150)	4500 ms**
	S4	T1 at S4	2*1*(650 - 150)	1000 ms
	S5	T1 and T3 at S5	(2*1 + 3*3)*(650 - 150)	5500 ms**
F3	S1	None	0	0
	S2	T2 at S2	3*2*(1000 - 200)	4800 ms**
	S3	T3 at S3	2*3*(1000 - 200)	4800 ms**
	S4	T2 at S4	3*2*(1000 - 200)	4800 ms**
	S5	T3 at S5	2*3*(1000 - 200)	4800 ms**

**sites where benefit > cost
Table. Cost and benefit for each fragment located at five possible sites.

Advantages
- simple algorithm
- can be applied to either redundant or nonredundant case
- reads and writes given appropriate weights

Disadvantages
- global averages of query and update time may not be realistic
- network topology and protocols not taken into account



VII. Data Warehousing, OLAP, and Data Mining

Data warehouse – a large repository of historical data that can be integrated for decision support

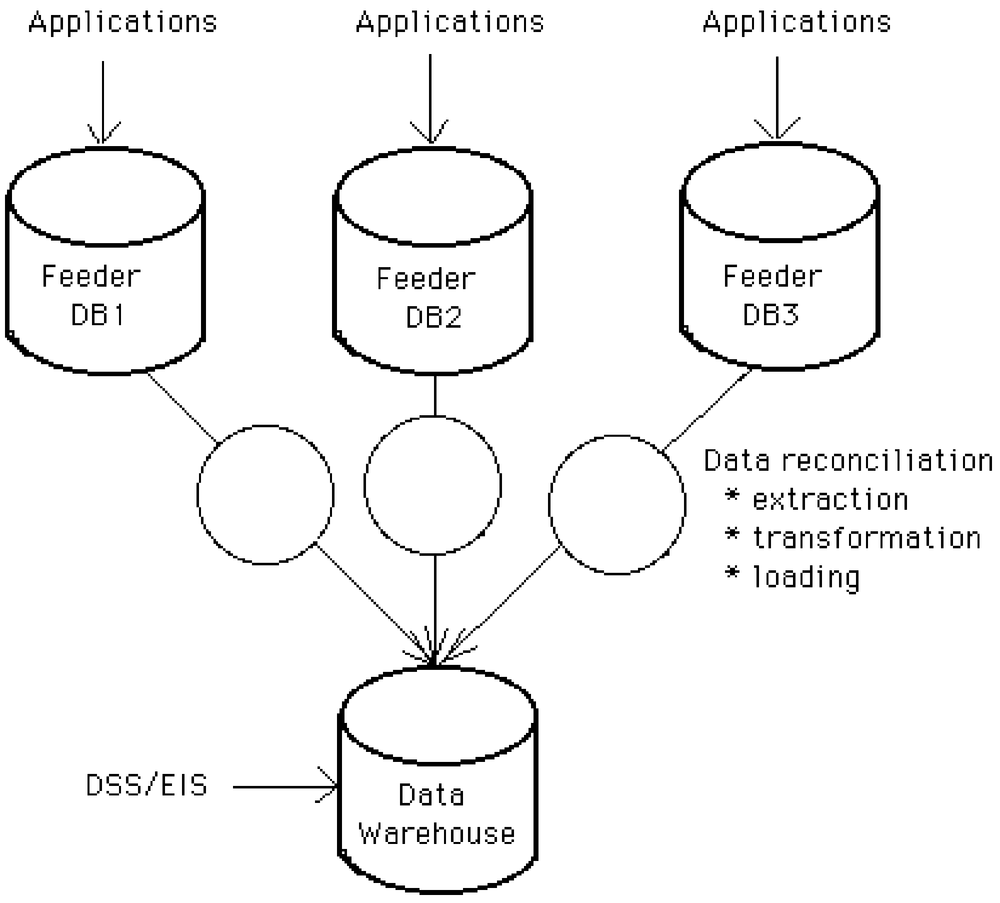


Figure 9.1 Data warehouse architecture

OLTP	Data Warehouse
Transaction oriented	Subject oriented
Thousands of users	Few users (typically under 100)
Small (MB up to several GB)	Large (100s of GB up to several TB)
Current data	Historical data
Normalized data (many tables, few columns per table)	Denormalized data (few tables, many columns per table)
Continuous updates	Batch updates
Simple to complex queries	Usually very complex queries

Table 9 1 Comparison between OLTP and Data Warehouse databases

Core Requirements for Data Warehousing

1. DWs are organized around subject areas.

2. DWs should have some integration capability.

3. The data is considered to be nonvolatile and should be mass loaded.

4. Data tends to exist at multiple levels of granularity.

5. The DW should be flexible enough to meet changing requirements rapidly. .

6. The DW should have a capability for rewriting history, that is, allowing “what-if” analysis.

7. A usable DW user interface should be selected.

8. Data should be either centralized or distributed physically.

Data Warehouse Life Cycle

I. Requirements analysis and specification

1.1 Analyze the end-user requirements and develop a requirements specification. This step follows the practice used by conventional relational databases (see Chapter 1).

1.2 Define the DW architecture and do some initial capacity planning for servers and tools. Integrate the servers, storage elements, and client tools.

1.3 Use enterprise data modeling

II. Logical database design

2.1 Design the enterprise DW schema and views.

2.2 Star schema is the most often used format –- good performance, ease of use

> Fact table (one) – very large table containing numeric and/or non numeric attributes, including the primary keys from the dimension tables; similar to intersection tables between entities with many-to-many relationships
>
> Dimension tables (several) - smaller tables containing mostly non numeric attributes; similar to relational tables based on entities

2.3 Snowflake schema – similar to star schema, except dimension tables are normalized

2.4 Fact table family (constellation) – multiple fact tables interact with dimension tables

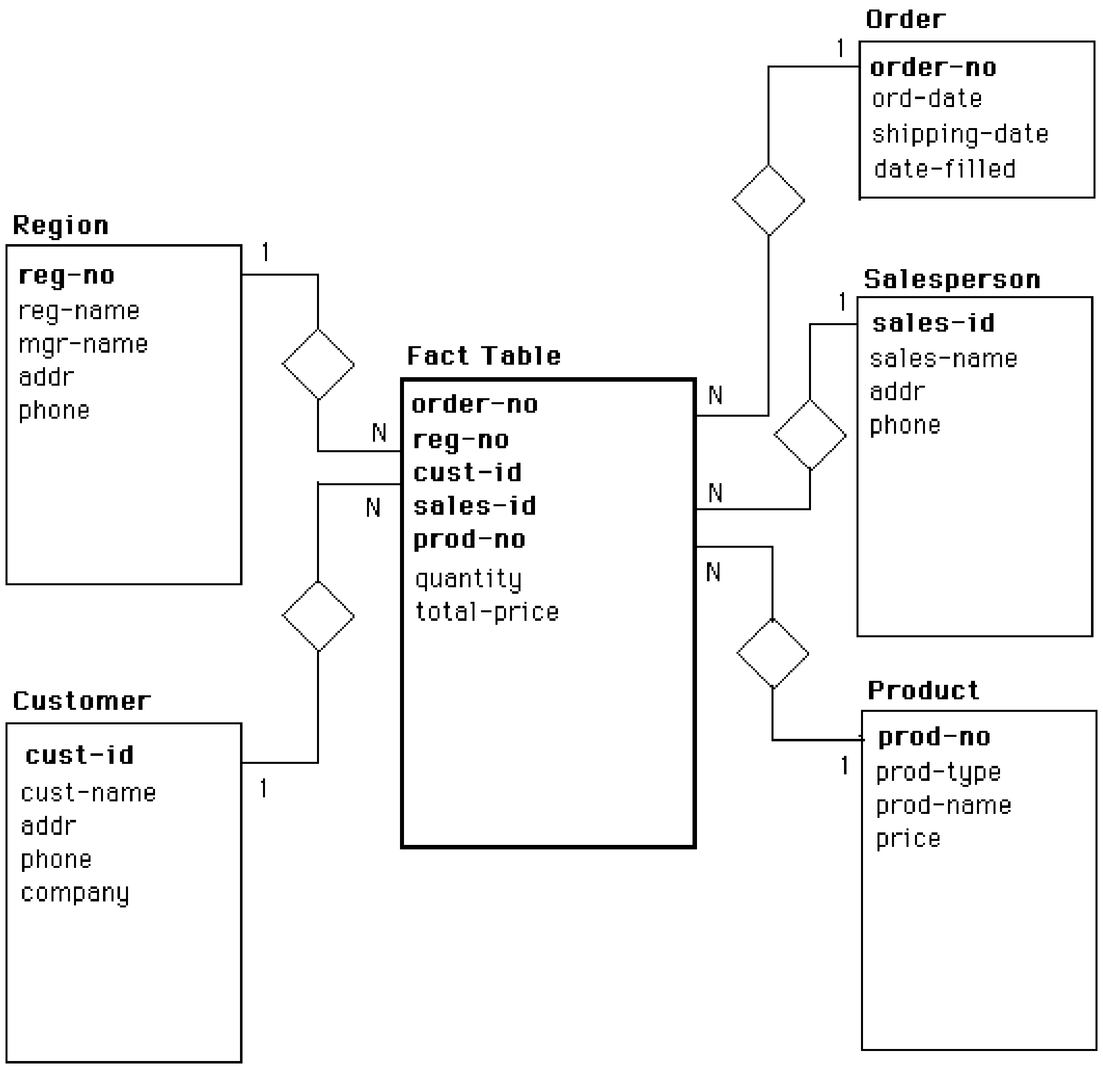


Figure 9.4 Star schema for the "order" data warehouse

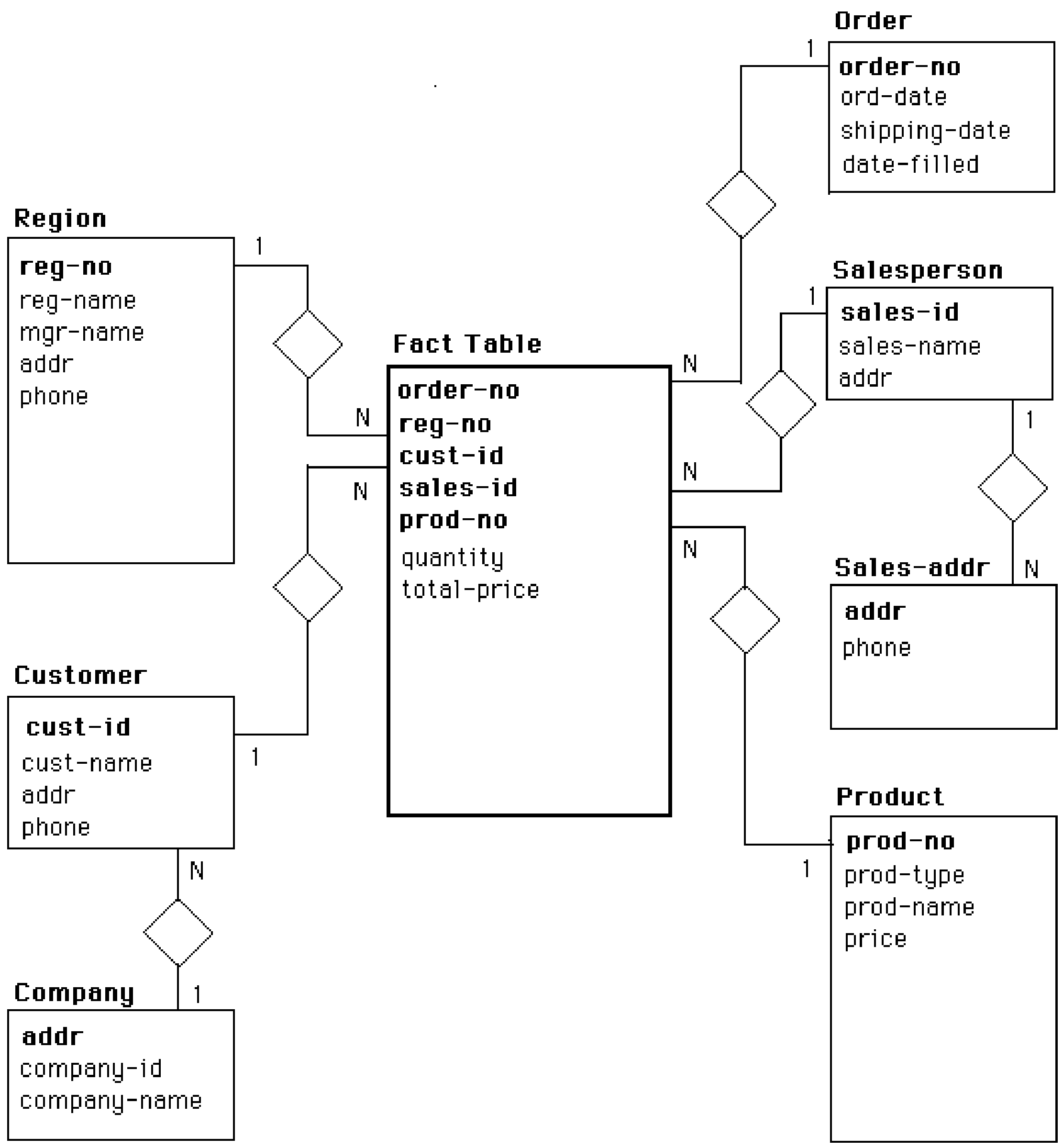


Figure 9.6 Snowflake schema for the "order" data warehouse

III. Physical database design

3.1 Indexing (access methods)

join indexes – used to map dimension tables to the fact table efficiently

bit map indexes – used for low selectivity queries

3.2 View materialization – associated with aggregation of data by one or more dimensions such as time or location

3.3 Partitioning – horizontal or vertical subsets of tables to enhance performance

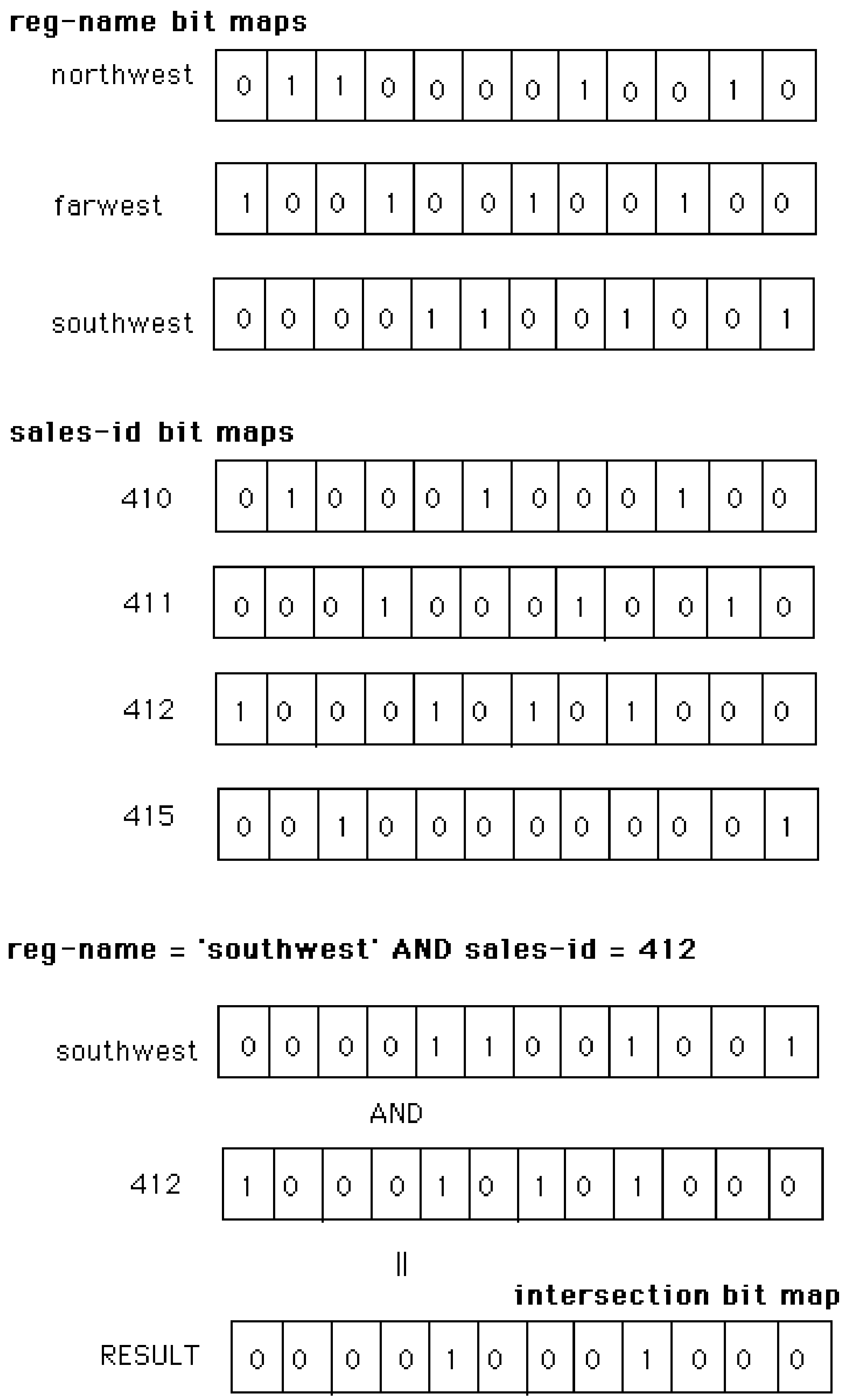

reg-name bit maps
northwest 0 1 1 0 0 0 0 1 0 0 1 0
farwest 1 0 0 1 0 0 1 0 0 1 0 0
southwest 0 0 0 0 1 1 0 0 1 0 0 1
sales-id bit maps
410 0 1 0 0 0 1 0 0 0 1 0 0
411 0 0 0 1 0 0 0 1 0 0 1 0
412 1 0 0 0 1 0 1 0 1 0 0 0
415 0 0 1 0 0 0 0 0 0 0 0 1
reg-name = 'southwest' AND sales-id = 412
southwest 0 0 0 0 1 1 0 0 1 0 0 1
AND
412 1 0 0 0 1 0 1 0 1 0 0 0
||
intersection bit map
RESULT 0 0 0 0 1 0 0 0 1 0 0 0

Figure 9.7 Bit maps and query processing

IV. Data distribution

4.1 Define data placement, partitioning, and replication.

V. Database implementation, monitoring, and modification

5.1 Connect the data sources using gateways, ODBC drivers, etc.

5.2 Design and implement scripts for data extraction, cleaning, transformation, load, and refresh.

5.3 Populate the repository with the schema and view definitions, scripts, and other metadata.

5.4 Design and implement end-user applications. Rollout the DW and applications.

On-Line Analytical Processing (OLAP)

Common Features of Multidimensional Databases (MDD)

1. Dimensions –- perspectives or entities about the real world

2. Hypercubes — basic structure for multidimensional databases

3. Hierarchies –- certain dimensions are hierarchical in nature

4. Formulas –- derived data values can be defined by formulas (sum, average, etc.)

5. Links – links are needed to connect hypercubes and their data sources

OLAP Logical Design

Step 1 – Analyze the end-user requirements and environment

Step 2 – Define cubes, dimensions, hierarchies, and links (high level)

Step 3 – Define dimension members (low level)

Step 4 – Define aggregations and other formulas (derived data)

Aggregation Issues

1. Which data to aggregate

2. How to store aggregate data

3. When to pre-aggregate derived data

    - Pre-aggregate nothing

    - Pre-aggregate nothing, but save the materialized view (dynamic)

    - Pre-aggregate everything (static)

    - Pre-aggregate selectively, based on known statistics

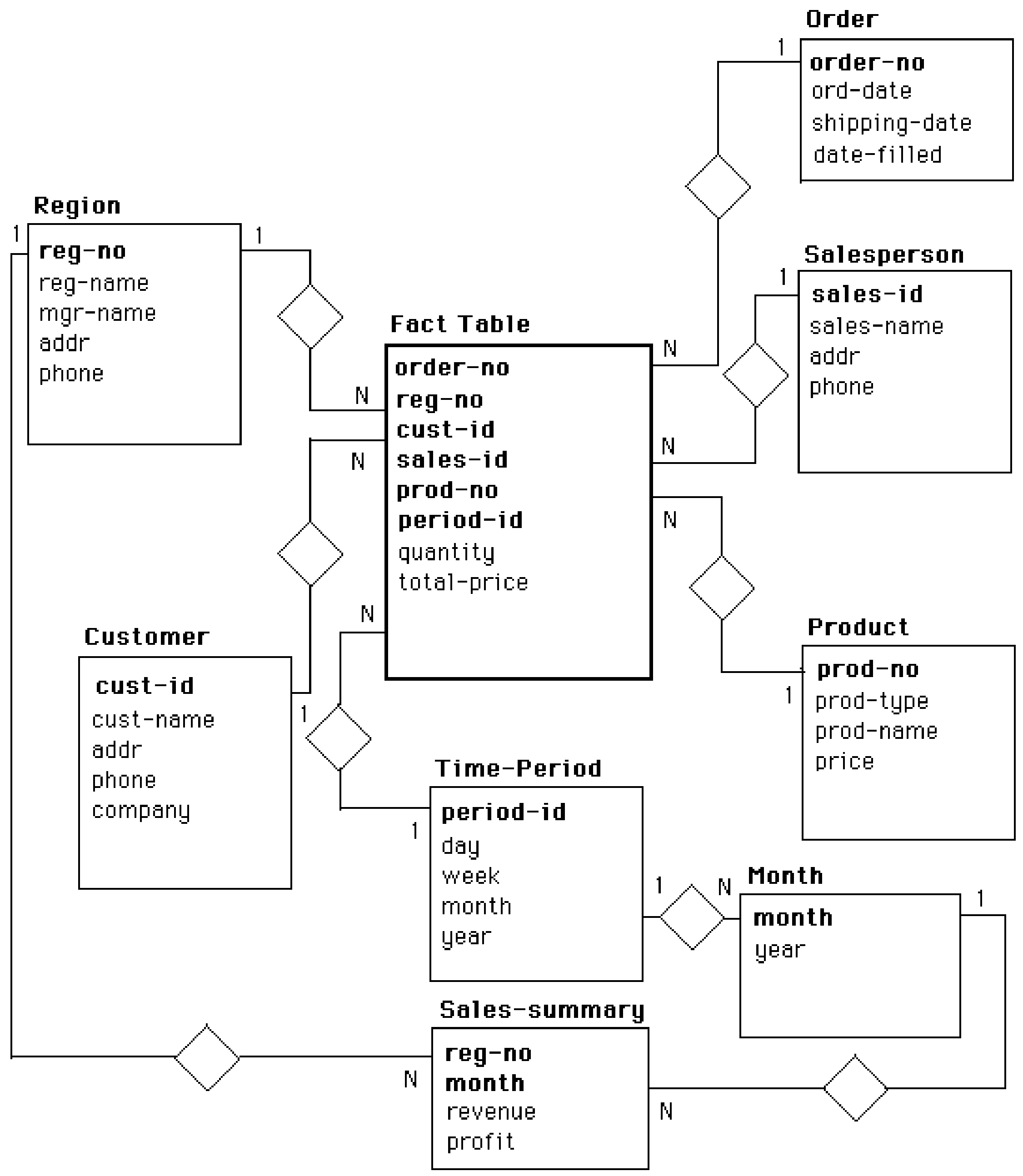


Figure 9.8 Sales-summary as an example of aggregation

time-period	region	product	variable
January1998	Southwest	Ford-Mustang	quantity-sold
February1998	Northwest	Chrysler-Eagle	total-revenue
March1998	North-central	GM-Camero	
April1998	South-central	Toyota-Camry	
May1998	Northeast		
June1998	Midwest		
1st-qtr1998	Southeast		
2nd-qtr1998			
3rd-qtr1998			
year1997			
year1998			

(a) Linear sequence of sample members from each of four dimensions

Region: Southwest		Quantity Sold	Total Revenue
January1998	Ford-Mustang	426	6317
	Chrysler-Eagle	179	3004
	GM-Camero	318	5261
	Toyota-Camry	299	4783
February1998	Ford-Mustang	451	6542
	Chrysler-Eagle	192	3119
	GM-Camero	356	6007
	Toyota-Camry	301	4936

(b) 2-dimensional layout of four dimensions of data

Figure 9.9 Display of multidimensional sales data

Report for January1998

	Southwest	Northwest	Total of regions
Ford-Mustang	426	457	883
Chrysler-Eagle	179	216	395
GM-Camero	318	245	563
Toyota-Camry	299	322	621
Total of products	1222	1240	2462

(a) Pure operations computed the same in any order (sums)

Report for January1998, Southwest region

	Quota	Quantity-sold	Quantity-sold/quota
Ford-Mustang	400	426	1.065
Chrysler-Eagle	200	179	0.895
GM-Camero	300	318	1.060
Toyota-Camry	300	299	0.997
Total	1200	1222	Ratio of sums = 1.018 Sum of ratios = 4.017

(b) Mixed sums and ratios give inconsistent results

Figure 9.10 Examples of mixing formulas for derived data values

Data Mining

Definition – data mining is the activity of sifting through large files and databases to discover useful, nonobvious, and often unexpected trends and relationships

The Knowledge Discovery in Databases (KDD) Process

1. Data selection and cleaning
2. Data transformation and reduction
3. Data mining
4. Interpretation and evaluation
5. Taking action on the discovery

Data Mining Methods

1. Predictive modeling
2. Database segmentation
3. Data summarization and link analysis
4. Dependency analysis or modeling
5. Change and deviation analysis
6. Optimization searching

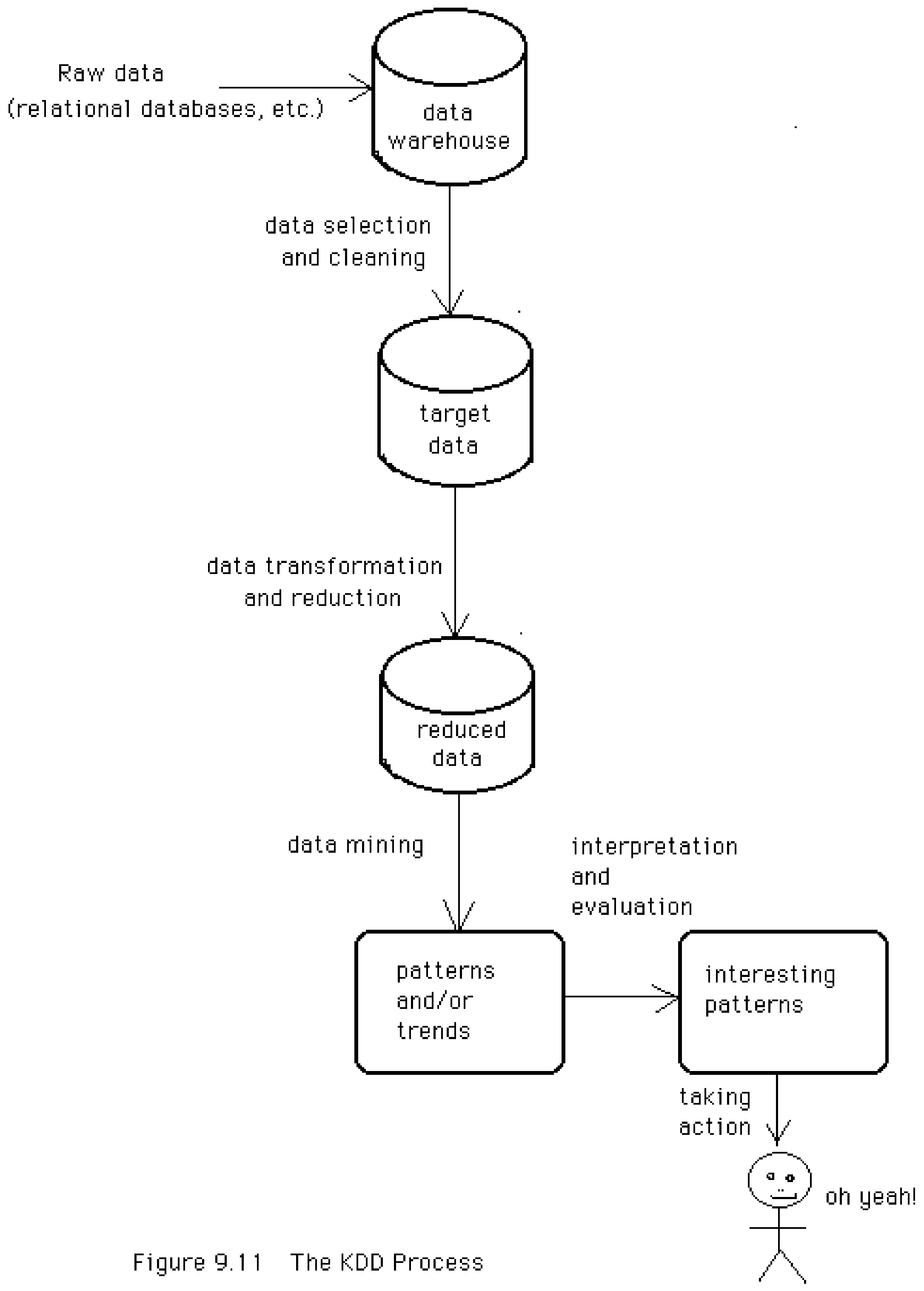


Figure 9.11 The KDD Process

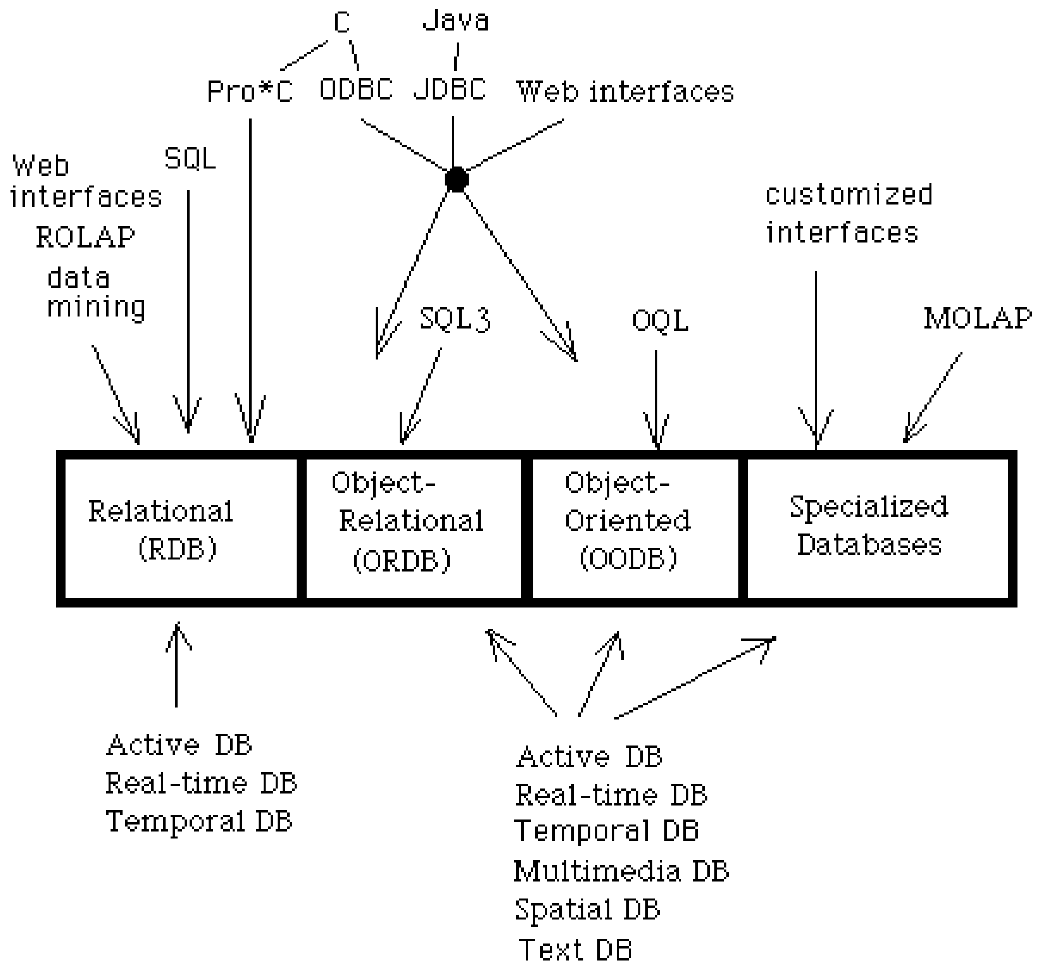


Figure 10.1 Advanced Database Architecture