CS9152 – DATABASE TECHNOLOGY

UNIT – IV

DATABASE DESIGN ISSUES

TEXT BOOK

1. Elisa Bertino, Barbara Catania, Gian Piero Zarri, "Intelligent Database Systems", Addison-Wesley, 2001.

REFERENCES

1. Carlo Zaniolo, Stefano Ceri, Christos Faloustsos, R.T.Snodgrass, V.S.Subrahmanian, "Advanced Database Systems", Morgan Kaufman, 1997.
2. N.Tamer Ozsu, Patrick Valduriez, "Principles of Distributed Database Systems", Prentice Hal International Inc. , 1999.
3. C.S.R Prabhu, "Object-Oriented Database Systems", Prentice Hall Of India, 1998.
4. Abdullah Uz Tansel Et Al, "Temporal Databases: Theory, Design And Principles",Benjamin Cummings Publishers , 1993.
5. Raghu Ramakrishnan, Johannes Gehrke, "Database Management Systems", Mcgraw Hill, Third Edition, 2004.
6. Henry F Korth, Abraham Silberschatz, S. Sudharshan, "Database System Concepts", Fourth Ediion, McGraw Hill , 2002.
7. R. Elmasri, S.B. Navathe, "Fundamentals of Database Systems", Pearson Education, 2004.

DATABASE DESIGN ISSUES

Syllabus:

UNIT IV DATABASE DESIGN ISSUES 10
ER Model – Normalization – Security –Integrity – Consistency –Database Tuning –Optimization and Research Issues – Design of Temporal Databases – Spatial Databases.

Table of Contents

SL No.	Topic	Page
1	ER Model	2
2	Normalization	13
3	Security	27
4	Integrity	30
5	Consistency	32
6	Database Tuning	33
7	Optimization and Research Issues	39
8	Design of Temporal Databases	42
9	Spatial Databases	47
10	Sample Questions	53
11	University Questions	60

Topic – 1: ER Model

Introduction

A *database* can be modeled as:
☐ a collection of entities,
☐ relationship among entities.

Steps in design of database:-
• Requirements collection & analysis
• Conceptual schema design
• Implementing conceptual schema into database using Implementation model
☐logical database design or data model mapping
• Physical database design

High-level conceptual data model

ER Model Concepts

Entities, attributes and relationships

Entities are specific objects or things in the mini-world that are represented in the database.
An *entity* is an object that exists and is distinguishable from other objects.
☐ Example: specific person, company, event, plant

For example, the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT

Entity Set: A collection of similar entities. E.g., all employees.

- All entities in entity set have same set of attributes.
- Each entity set has a *key*.
- Each attribute has a *domain*.

Attributes:
Attributes are properties used to describe an entitiy.
For Example, an EMPLOYEE entity may have a Name, SSN, Address, Sex, BirthDate.

A Specific entity will have a value for each of its attributes.

For example a specific employee entity may have Name='John Smith', SSN='123456789', Address='731, Fondren Houston, TX', Sex='M', BirthDate='09-JAN-55'

Each attribute has a value set (or data type) associated with it. - e.g. integer, string, subrange, enumerated type, …..

Types of attributes

- simple Vs composite attributes
- single-valued Vs multi-valued attributes
- stored Vs derived attributes

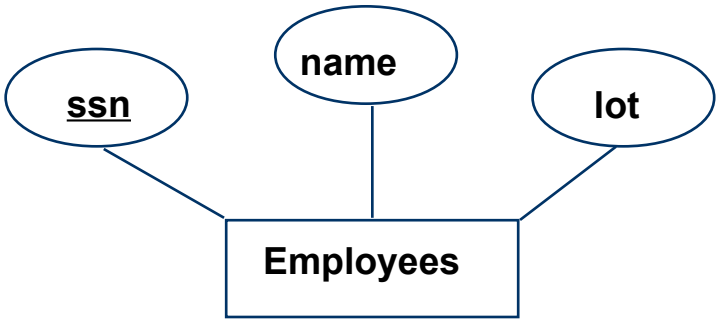
Simple – Each entity has single atomic value for the attribute.
Empid

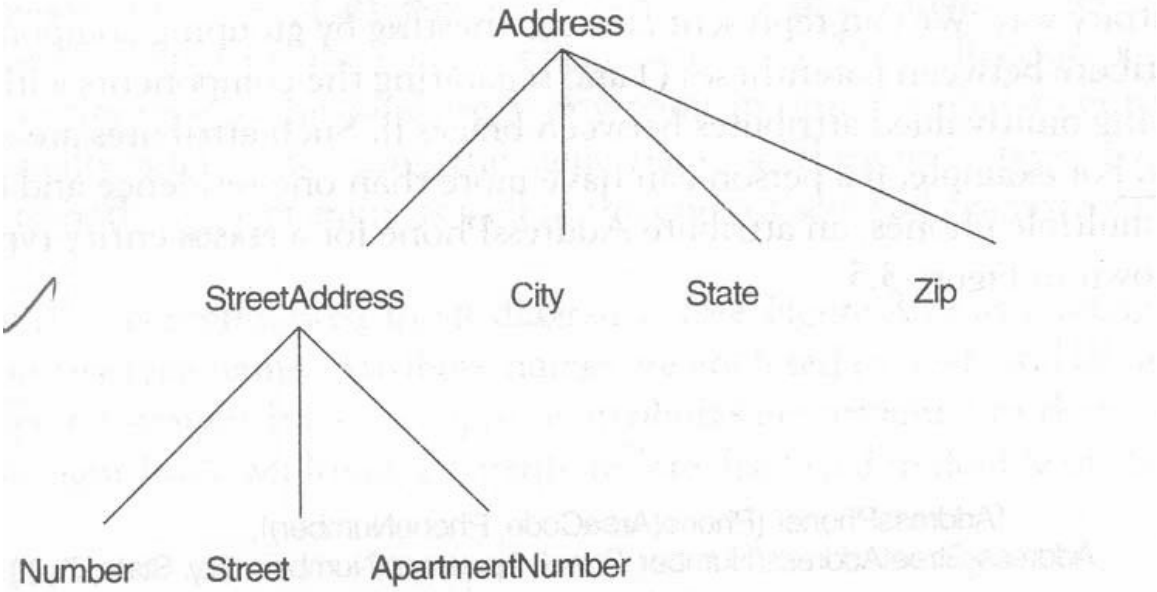
Composite - The attribute may be composed of several components. For Example, Address(Apt#, Street, City, State, ZipCode, Country) or Name (FirstName, MiddleName, LastName)

Composition may form a hierarchy where some components are themselves composite.

Multi-valued - An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT. Denoted as {Color} or {PreviousDegrees}

In general, Composite and multi-valued attributes may be nested arbitary to any number of levels although this is rare. For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Filed) }





single valued - age
multi valued - qualification
stored - date_of_birth
derived - Age

Null values
• Missing
• Available but unknown
• Inapplicable places

• **Entity types, value sets & Key attributes**
☐ Entity type – entities that share same attributes or entities with the same basic attributes are grouped or typed into an entity type
(eg) employee
• describes schema or intension for a set of entities
☐ Collection of individual entities – extension of entity type or entity set
• **Value set (domain) of attributes**
• name, type of values, format etc.

Key attribute of an entity type - An attribute of Entity type for which each entity must have a unique value is called a key attribute of the entity type.
It doesn't allow duplicate values
Example, SSN of EMPOLYEE

An entity type may have more than one key.
For example, the CAR entity type may have two keys:
- VehicleIDentificationNumber (popularly called VIN) and

- VehicleTagNumber (Number, state), alos known an license_plate number
-

• RELATIONSHIPS

Association among entity types
Relationship type – set of relationships among different entity types
E1,E2,…,En – entity types;
r1,r2,…,rn –relation instances
R – relationship type

E.g., Attishoo works in Pharmacy depart.

Relationship Set: Collection of similar relationships.

- An n-ary relationship set R relates n entity sets E1 ... En;
- each relationship in R involves entities e1 from E1, ..., en from En
  - Same entity set could participate in different relationship sets, or in different "roles" in same set.

Constraint on Relationships

- IT is alos known as ratio constraints
- Maximum Cardinality
- One-to-one (1:1)
- One-to-many (1:N) or many-to-one (N:1)
- Many-to-many

- Minimum Cardinality (also called participation constraint or existence dependency constraints)

- Zero (optional participation, not existence-dependent)

- One or more (mandatory, existence-dependent)

Relationship of Higher Degree

- Relationship of degree 2 are called binary
- Relationship of degree 3 are called ternary and of degree n are called n-ary.
- In genral, an n-ary relationship is not equivalent to n binary relationships.

Entity set Corresponding to the entity Type CAR

CAR
Registration(REgistrationNumber, State), VehicleID, Make, Model, Year(Color)

Car1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 1999, (red,black))

Car2
((ABC 123, NEW YORK), WP9872, Nissan 300ZX, 2-door, 2002, (blue))

((VSY 720, TEXAS), TD729, Buick LeSabre, 4-door, 2003, (white,blue))

.

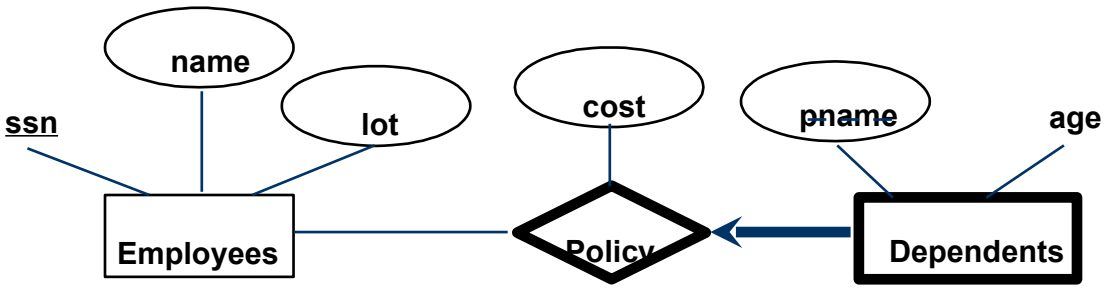
Weak Entity Types

An entity that does not have a key attribute

A Weak entity must participate in an identifying relationship type with an owner or identifying entity type.

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.

Example:



- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
- Weak entity set must have total participation in this *identifying* relationship set.

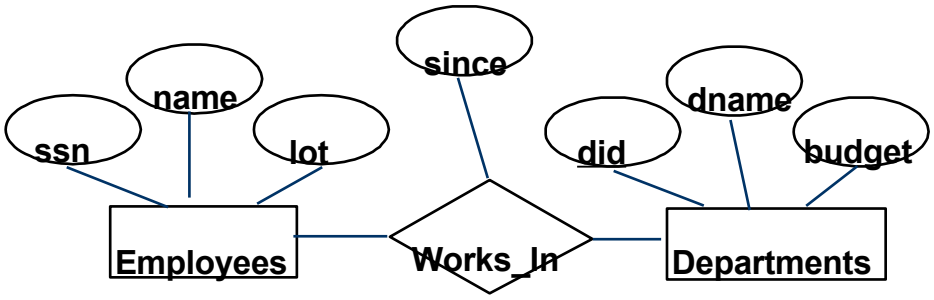
Entities are identified by the combination of:

- A partial key of the weak entity type.
- The particular entity they are related to in the identifying entity type.

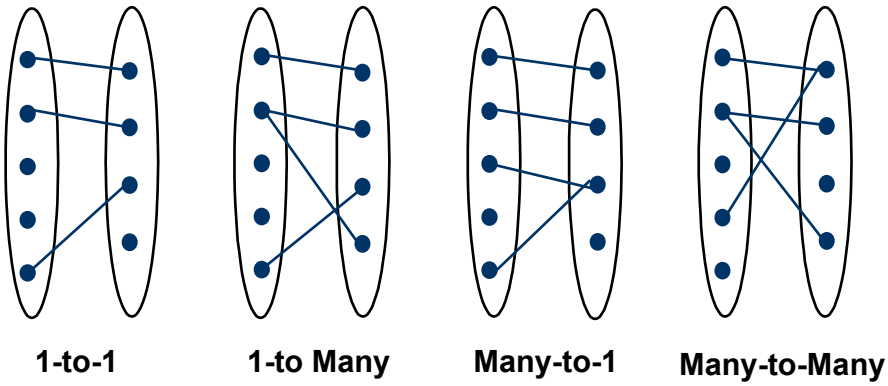
Example:

Suppose that a DEPENDENT entity is identified by the dependent's first name and birthdate, and the specific EMPLOYEE that the dependent is related to. DEPENDENT is a weak entity type with EMPLOYEE as its identifying entity type via the identifying relationship type DEPENDENT_OF

A simple ER diagram

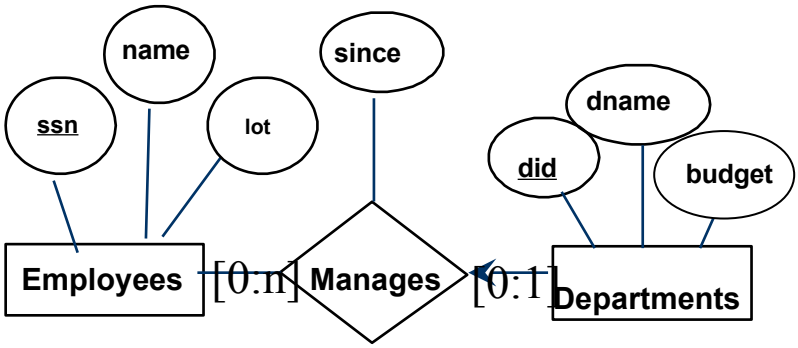


Key Constraints



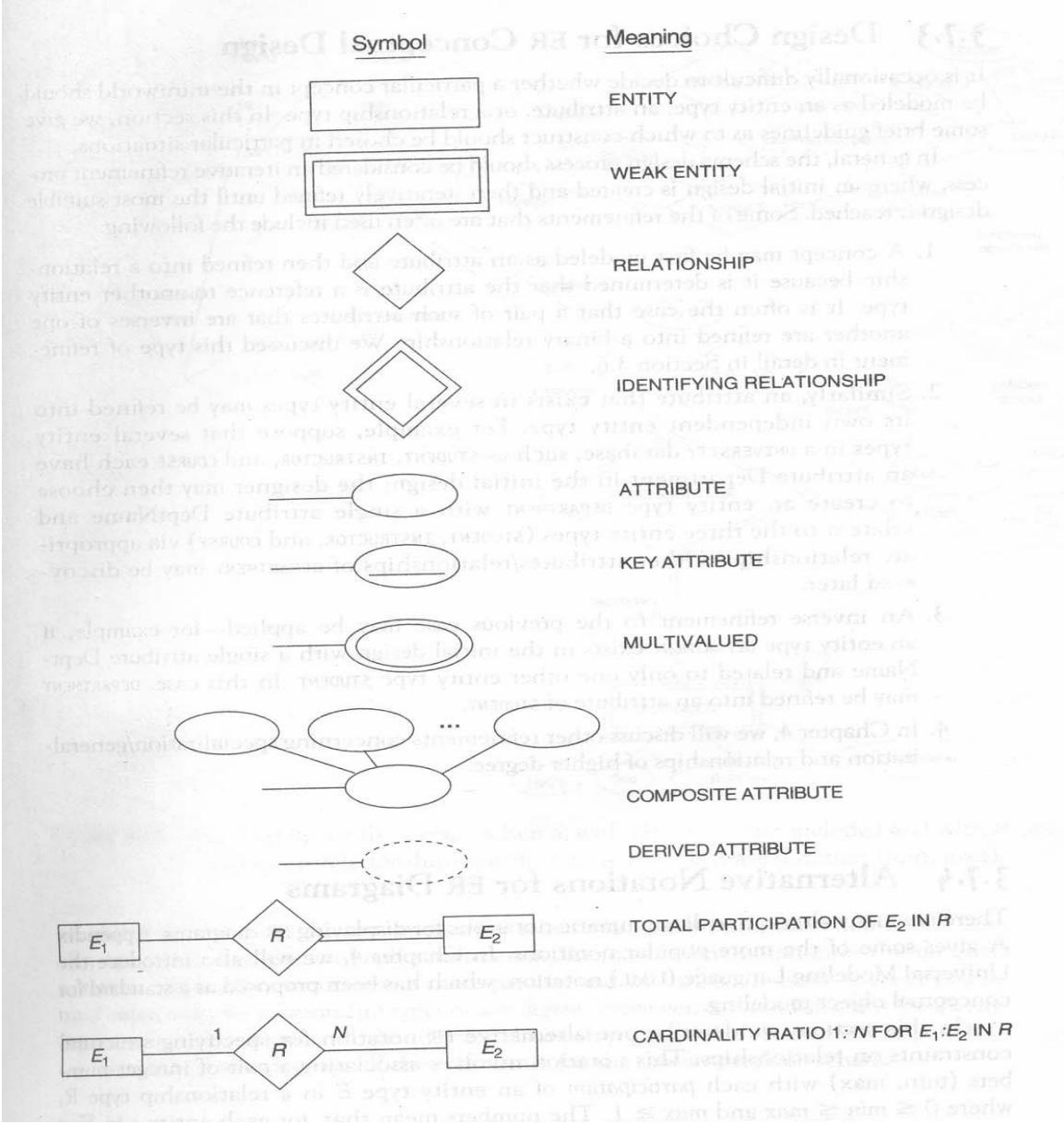
Example:

- **Consider Works_In: in the above simple ER diagram:**
  An employee can work in many departments; and
  a dept can have many employees.



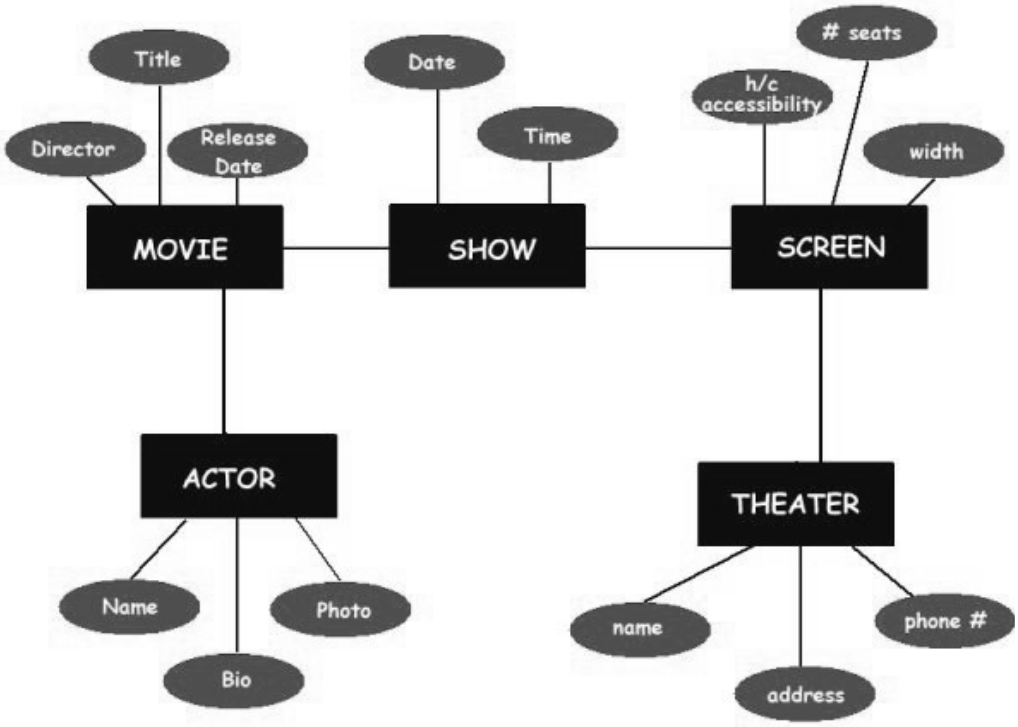
- **Each dept has at most one manager, according to the *key constraint* on Manages.**

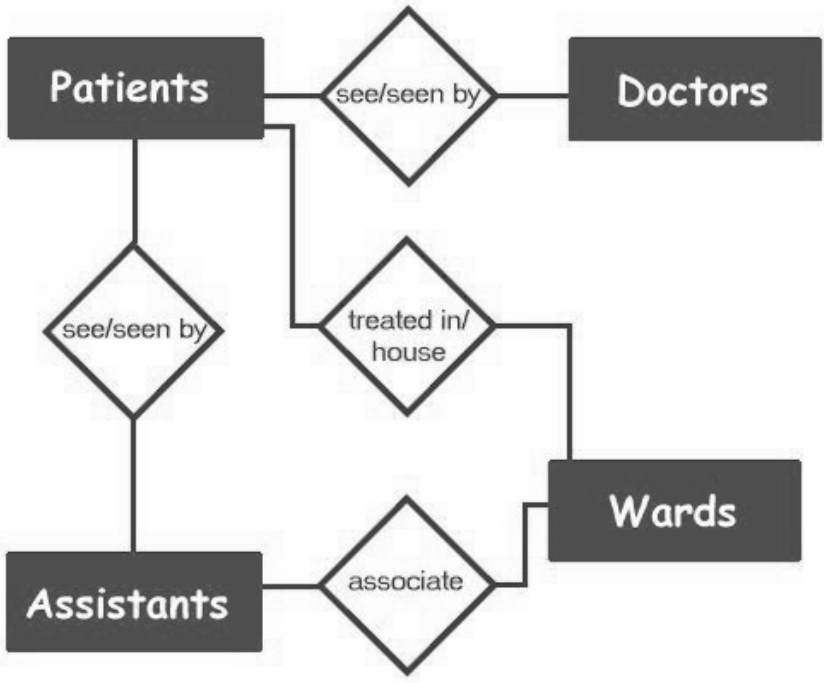
Notations for ER Diagrams



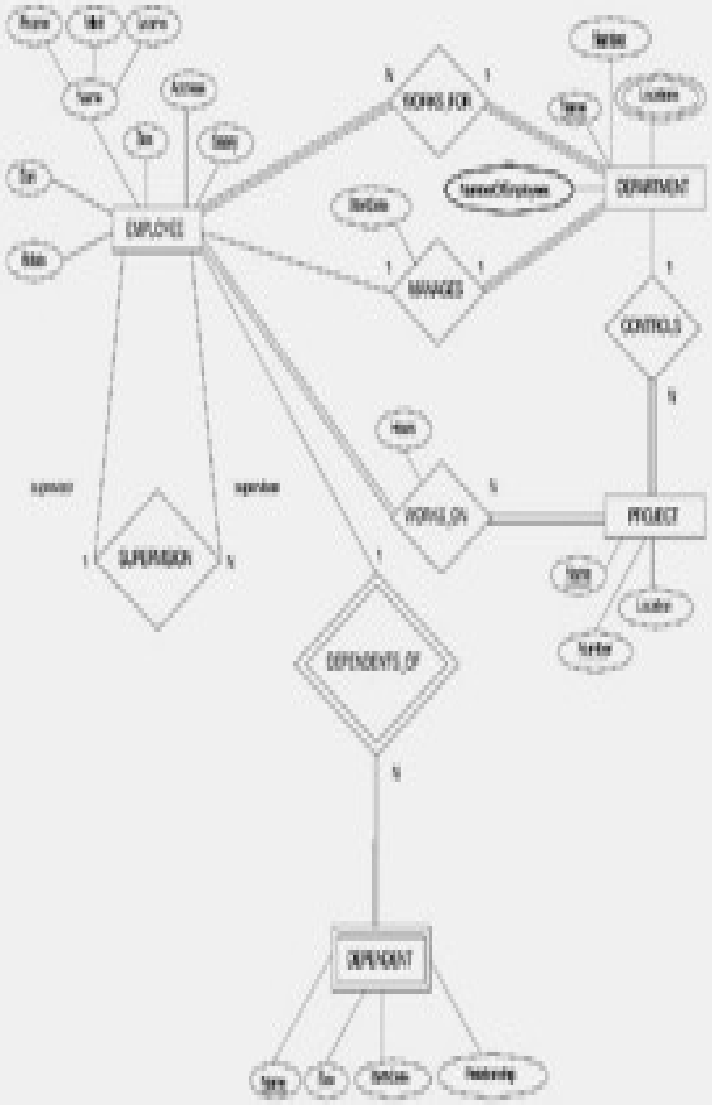
Problem with ER Notation

The Entity-Realtionship model in its original form did not support the specialization / generalization abstractions.

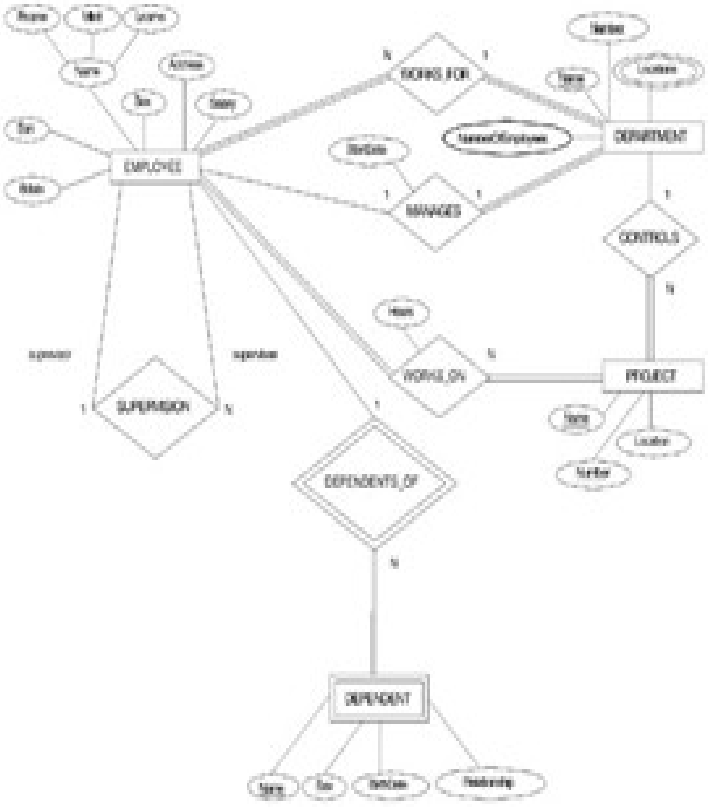

Title
Director
Release Date
MOVIE
Date
Time
SHOW
seats
h/c accessibility
width
SCREEN
ACTOR
Name
Photo
Bio
THEATER
name
phone #
address


Patients
see/seen by
Doctors
see/seen by
treated in/ house
Wards
Assistants
associate

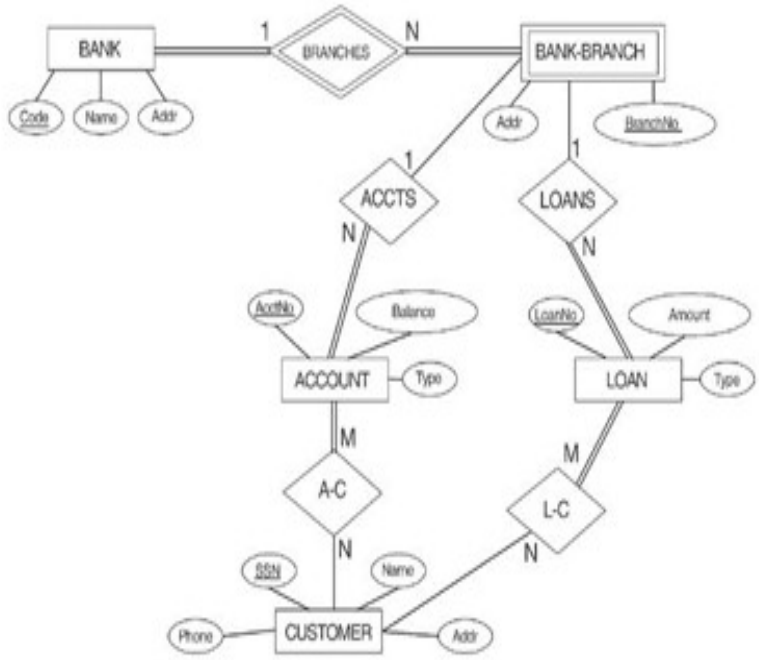
ER DIAGRAM – Entity Types are:
EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT



ER DIAGRAM – Relationship Types are: WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF



ER DIAGRAM FOR A BANK DATABASE



Some of the Currently Available Automated Database Design Tools

COMPANY	TOOL	FUNCTIONALITY
Embarcadero Technologies	ER Studio	Database Modeling in ER and IDEF1X
	DB Artisan	Database administration and space and security management
Oracle	Developer 2000 and Designer 2000	Database modeling, application development
Popkin Software	System Architect 2001	Data modeling, object modeling, process modeling, structured analysis/design
Platinum Technology	Platinum Enterprice Modeling Suite: Erwin, BPWin, Paradigm Plus	Data, process, and business component modeling
Persistence Inc.	Pwertier	Mapping from O-O to relational model
Rational	Rational Rose	Modeling in UML and application generation in C++ and JAVA
Rogue Ware	RW Metro	Mapping from O-O to relational model
Resolution Ltd.	Xcase	Conceptual modeling up to code maintenance
Sybase	Enterprise Application Suite	Data modeling, business logic modeling
Visio	Visio Enterprise	Data modeling, design and reengineering Visual Basic and Visual C++

Topic – 2: Normalization

Why Normalization?

Redundant Information in Tuples and Update Anomalies

- Mixing attributes of multiple entities may cause problems
- Information is stored redundantly wasting storage
- Problems with update anomalies
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies

Example of an update Anomaly
Consider the relation:
EMP_PROJ (Emp#, Proj#, Ename, Pname, No_hours)

Update Anomaly: Chaing the name of the project number from "Billing" to " Customer – Accounting" may cause this update to be made for all 100 employees working on project P1.

Insert Anomaly: Cannot insert a project unless an employee is assigned to.
Inversely – cannot insert an employee unless he/she is assigned to a project.

Delete Anomaly: When a project is deleted, it will result in deleting all employees who work on that project. Alternatively, if an employee is an sole employee on a project, deleting that employee would result in deleting the corresponding project.

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are **hard to understand** or to **detect**
- The database designers ***need not*** normalize to the highest possible normal form. (usually up to 3NF, BCNF or 4NF)
- **Denormalization:** the process of storing the join of higher normal form relations as a base relation—which is in a lower normal form
- 1st introduced by Codd in 1972
- Database normalization relates to the level of redundancy in a relational database's structure.

- The key idea is to reduce the chance of having multiple different versions of the same data, like an address, by storing all potentially duplicated data in different tables and linking to them instead of using a copy.

Normalization – Definition

• Normalization is the process of decomposing unsatisfactory relation schemas into smaller relation schemas which contain desirable attributes (or) properties

• For a good relation schema design, apart from normalization it should have additional properties like

☐ Lossless-join (or) Non-additive join property - It is more important & cannot be scarified

☐ Dependency preservation property - It is less important & can be scarified

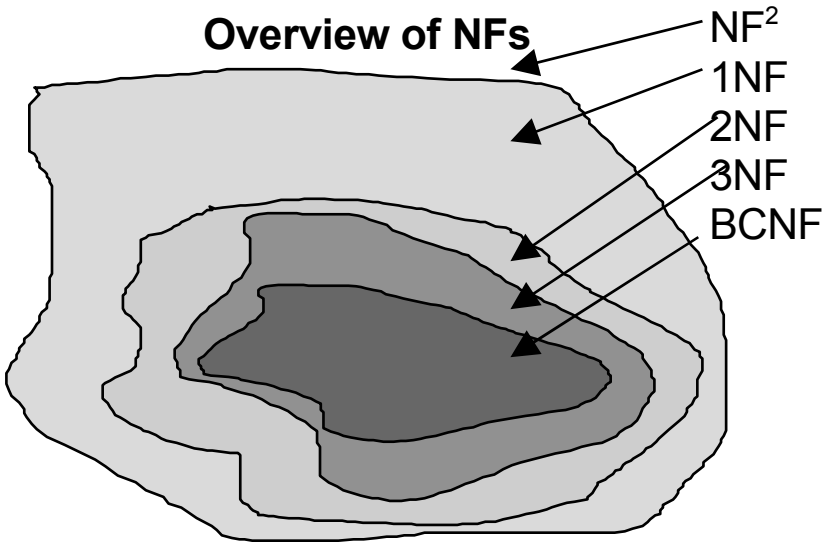
Functional Dependencies

Functional Dependencies (FDs) are used to specify formal measures of a goodness of the relational design.
FDs and Keys are used to define normal forms for relations
FDs are constraints that are derived from the meaningful and interrelationship of the data attributes.

A set of attributes X functionally determines the set of attributes Y if the value of X determines a unique value of Y

X → Y holds whenever two tuples have the same value for X they must have the same value for Y.

For any two tuples t1 and t2 in any relation insrance r(R): if t1[X] = t2[X] then t1[Y]=t2[Y].
X → Y in R specifies a constraint on all relation instances r (R)
Written as X → Y; can be displayed graphically on a relation schema as in Figures (denoted by the arrow :)
FDs are derived from the real-world constraints on the attributes.



Normal Form Types

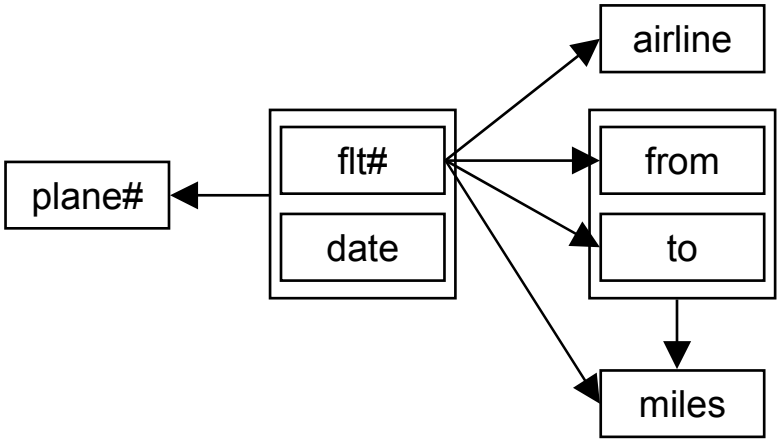
- **NF2**: non-first normal form
- **1NF**: R is in 1NF. **iff** all domain values are atomic.
- **2NF**: R is in 2. NF. **iff** R is in 1NF and every nonkey attribute is fully dependent on the key
- **3NF**: R is in 3NF **iff** R is 2NF and every nonkey attribute is non-transitively dependent on the key
- **BCNF**: R is in BCNF **iff** every determinant is a candidate key

- **Determinant**: an attribute on which some other attribute is fully functionally dependent.

Flight Relation Example:

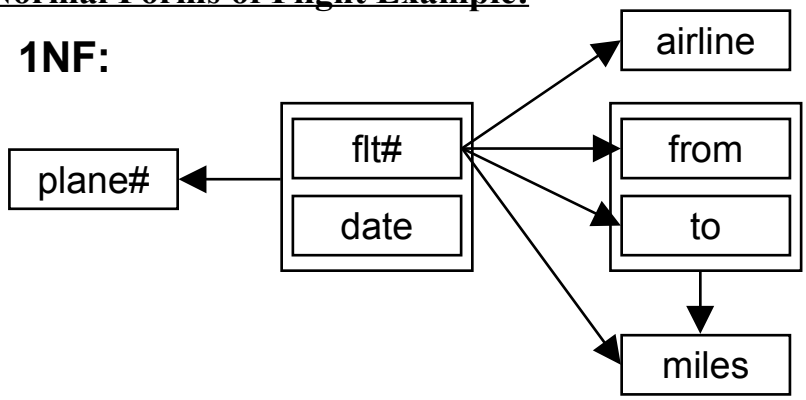
FLT-INSTANCE

flt#	date	plane#	airline	from	to	miles

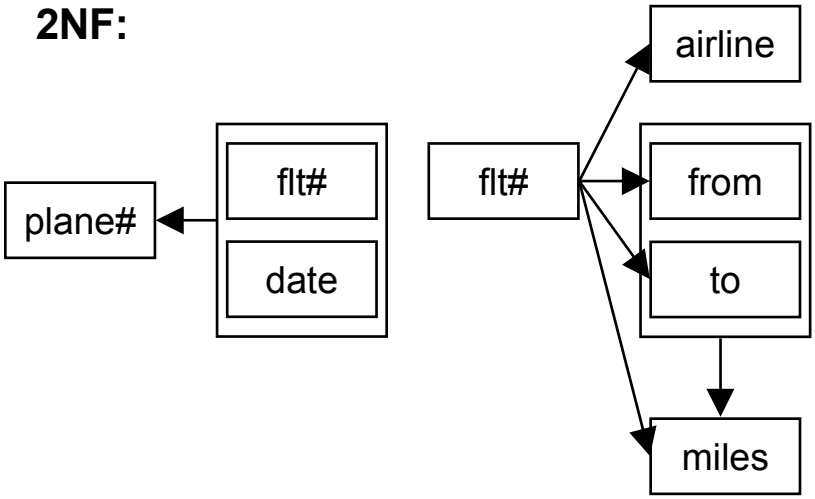


Normal Forms of Flight Example:

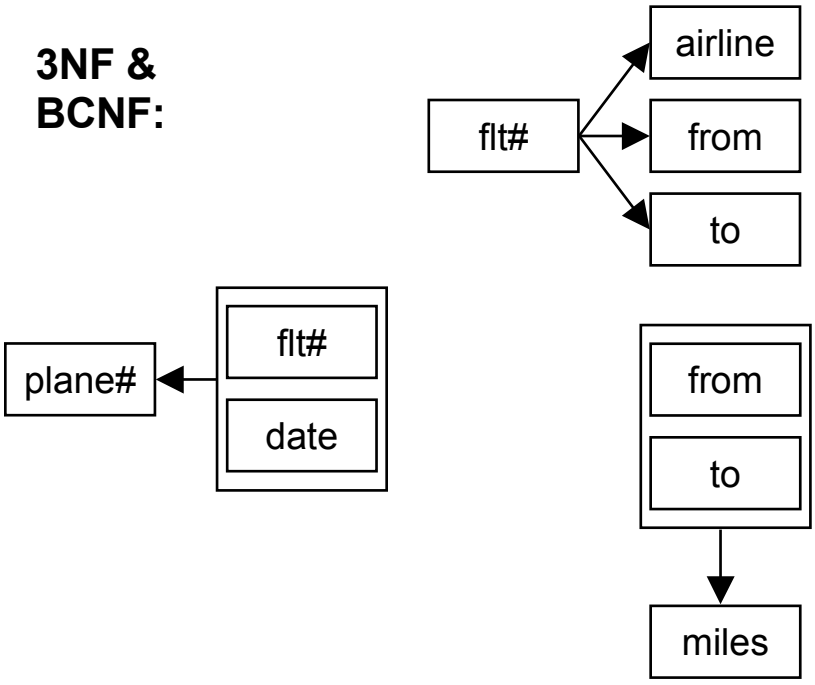
1NF:



2NF:



3NF & BCNF:



First Normal Form(1NF)

- Eliminates Repeating Groups. Make a separate table for each set of related attributes, and give each table a primary key.
- Based on the concept of multivalued & composite attributes.
- The domains of all attributes of a relation schema R are atomic, which is if elements of the domain are considered to be indivisible units.

• Disallows relation schemas to have multivalued & composite attributes
Also nested relations; attributes whose values for an individual tuple are non-atomic.

The domains of all attributes of a relation schema R are atomic, which is if elements of the domain are considered to be indivisible units.

A relational schema R is in first normal form if the domains of all attributes of R are atomic
☐ Non-atomic values complicate storage and encourage redundant (repeated) storage of data
☐ E.g. Set of accounts stored with each customer, and set of owners stored with each account
☐ We assume all relations are in first normal form

Atomicity is actually a property of how the elements of the domain are used.
☐ E.g. Strings would normally be considered indivisible
☐ Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
☐ If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
☐ Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

DEPARTMENT relation with instances

DNAME	DNO	DMGRSSN	DLOCS
Research	2	2000101	(Delhi, Mumbai, Kolcutta)
Administration	3	1000111	Bangaluru
Head Quarters	1	000111	Chennai

Consider to be part of the definition of relation.
(Eg1)
DLOCS is a multivalued attribute

Normalized Table – in I NF with redundancy

DNAME	DNO	DMGRSSN	DLOCS
Research	2	2000101	Delhi
Research	2	2000101	Mumbai
Research	2	2000101	Kolcutta
Administration	3	1000111	Bangaluru
Head Quarters	1	000111	Chennai

(Eg2)

EMP_PROJ
SSN ENAME PROJECTS
PNO HOURS

Normalized Table – in I NF with redundancy

EMP_PROJ1
SSN ENAME

EMP_PROJ2
SSN PNO HOURS

Note:
It involves that removal of redundant data from horizontal rows.
We need to ensure that there is no duplication of data in a given row, and that every column stores the least amount of information possible.

• Second Normal Form(2NF)

- Based on full functional dependency
- Eliminate Redundant Data, if an attribute depends on only part of a multi-valued key, remove it to a separate table.

It uses the concept of FD(Functional Dependency) and primary key
X → Y
Full FD
Prime, non-prime attributes

A relation schema R is in 2NF if every nonprime attribute of R is fully functionally dependent on the primary key of R

If each attribute A in a relation schema R meets one of the following criteria:

- It must be in first normal form.
- It is not partially dependent on a candidate key.
- Every non-key attribute is fully dependent on each candidate key of the relation.
-

Second Normal Form (or 2NF) deals with redundancy of data in vertical columns.

R can be decomposed into 2NF relations via the process of 2NF normalization.

Prime attribute – attribute that is member of the primary key k.

Full Functional Dependency – A FD Y → Z where removal of any attribute from Y means the FD does not hold any more.

Example:

{SSN PNO } → HOURS is a full FD
Where as SSN → HOURS PNO → HOURS does not.

The relation:

EMP_PROJ
SSN ENAME PROJECTS
PNO HOURS

is normalized into 2NF as:

SSN → ENAME

{ SSN, PNO } → HOURS

{SSN, ENAME } → HOURS

Third Normal Form (3NF)

It eliminates columns not dependent on key. If attributes do not contribute to a description of the key, remove them to a separate table.

A relation R is in Third Normal Form (3NF) if and only if it is:

- **in Second Normal Form.**
- **Every non-key attribute is non-transitively dependent on the primary key.**

An attribute C is transitively dependent on attribute A if there exists an attribute B such that A ◊ B and B ◊ C, then A ◊ C.

Transitivity Functional Dependency – a FD X → that can be derived from two FDs X → Y and Y →Z

Examples:

EMP_DEPT

ENAME SSN BDATE ADDRESS DNO DNAME DMGRSSN

SSN → DMGRSSN is a Transitive FD since

SSN → DNO and DNO → DMGRSSN holds.

SSN → DNAME is not a Transitive FD since there is no set of attributes X where SSN → X and X → ENAME

A Relation schema R is in 3NF if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key.

R can be decomposed into 3NF relations via the process of 3NF normalization.

Note: in X → Y and Y → Z with X as the primary key, we consider this a problem only if Y is not a candidate key. When Y is a candidate key, there is no problem with the transitive dependency.
Eg. Consider EMP (SSN, Emp#, Salary)
Here, SSN → Emp# → Salary and Emp# is a candidate key.

The general normal form definition- for Multiple key

A Relation schema R is in 2NF if every non-prime attribute A in R is fully functionally dependent on every key of R.

Super key of relation schema R – a set of attributes S of R that contains a key of R.
A Relation schema R is in 3NF if whenever a FD X → A holds in R then either:

a) X is a super key of R or
b) A is a prime attribute of R

Exercise:

Consider a relation called supplier-part with the following data to be processed:
{s#, status, city, p#, qty, cost}

Where,
s# -- supplier identification number (this is the primary key)
status -- status code assigned to
city -- city name of city where supplier is located
p# -- part number of part supplied
qty -- quantity of parts supplied to date

Convert the relation into 1NF, 2NF, 3NF

1NF:

FIRST

s#	status	city	p#	qty
s1	20	London	p1	300
s1	20	London	p2	200
s1	20	London	p3	400
s1	20	London	p4	200
s1	20	London	p5	100
s1	20	London	p6	100
s2	10	Paris	p1	300
s2	10	Paris	p2	400
s3	10	Paris	p2	200
s4	20	London	p2	200
s4	20	London	p4	300
s4	20	London	p5	400

Example: **1NF but not 2NF**

first (s#, status, city, p#, qty)

Functional Dependencies:
(s#, part_no) →qty
(s#) →status
(s#) →city
city →status (Supplier's status is determined by location)
Comments:
Non-key attributes are not mutually independent (city →status).

2NF:

Functional Dependency on First Normal Form:
s# —> city, status (this violated the Second Normal Form)
city —> status
(s#,p#) —>qty

Need decomposition into two tables:

SECOND

s#	status	city
s1	20	London
s2	10	Paris
s3	10	Paris
s4	20	London
s5	30	Athens

PARTS

s#	p#	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	100
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300
s4	p5	400

3NF:

Functional Dependency of the Second Normal Form:
SECOND.s# —> SECOND.status (Transitive dependency)
SECOND.s# —> SECOND.city
SECOND.city —> SECOND.status

SECOND is converted into SUPPLIER_CITY and CITY_STATUS

SUPPLIER_CITY

s#	city
s1	London
s2	Paris
s3	Paris
s4	London
s5	Athens

CITY_STATUS

city	status
London	20
Paris	10
Athens	30
Rome	50

Boyce-Codd Normal Form (BCNF)

A relation R is in Boyce-Codd normal form (BCNF) if and only if every determinant is a candidate key

A relation schema R is in BCNF if whenever an FD X → A holds in R then X is a super key of R.
Each Normal Form is strictly stronger than the previous one:

- Every 2NF relation is in 1NF
- Every 3NF relation is in 2NF
- Every BCNF relation is in 3NF

There exists relations that are in 3NF but not in BCNF.
The goal is to have each relation in BCNF (or 3NF)

To be precise, the definition of 3NF does not deal with a relation that:

has multiple candidate keys, where
those candidate keys are composite, and
the candidate keys overlap (i.e., have at least one common attribute)

Example:

1) Consider the relation schema R which has attributes
R={courseno, secno,
offeringdept, credithours, courselevel, instrctorssn, semester, year,
days_hours, roomno, noofstudents}.

The following FDs hold on R:
courseno → {offeringdept, credithours, courselevel}

{courseno, secno, semester, year} → {days_hours, roomno, noofstudents, instructorssn}

{roomno, days_hours, semester, year} → {instructorssn, courseno, secno}

2)
A relation TEACH that is in 3NF but not in BCNF

TEACH

STUDENT	COURSE	INSTRUCTOR
Narayanan	Database	Mark
Smith	Database	Navathe
Smith	Operating System	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating System	Ahamed
Wong	Database	Omicinds
Zethya	Database	Navathe

Achieving the BCNF by decomposition:

Two FDs exist in the relation TEACH:

FD1: {student, course } → instructor

FD2: instructor → course

{student, course} is a candidate key for this relation and that the dependencies shown.

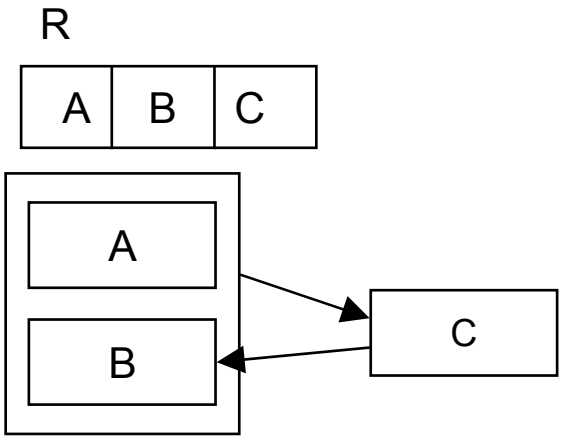
Solution:

Three possible decomposition for the relation TEACH:

i) { student , instructor } and { student, Course }

ii) { course, instructor } and { course, student }

iii) { instructor, course} and { instructor, student}



Candidate keys: {A,B} and {A,C} Determinants: {A,B} and {C}

A decomposition:

Lossless, but not dependency preserving

Multivalued Dependencies & Fourth NF

A single values of X determines many values of Y ((ie) X multidetermines Y)
A MVD is trivial MVD if
(1) Y is a subset of X, or
(2) X∪Y=R
A MVD which does not satisfy (1) & (2) is called nontrivial MVD
We use multivalued dependencies in two ways:
1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies
2. To specify constraints on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.
☐ If a relation *r* fails to satisfy a given multivalued dependency, we can construct a relations *r′* that does
satisfy the multivalued dependency by adding tuples to *r*.

Fourth Normal Form(4NF)

Isolates independent multiple relationships. No table may contain two or more 1:n (one-to-many) or n:m (many-to-many) relationships that are not directly related.

A relation schema R is in 4NF with respect to a set of functional dependencies F if for every nontrivial MVD in F+, X is a super key of R

Example:

Transform the following BCNF table into a 4NF one(s).

The next table is in the BCNF form, convert it to the 4th normal form.

Employee	skill	language
Jones	electrical	French
Jones	electrical	German
Jones	mechanical	French
Jones	mechanical	German
Smith	plumbing	Spanish

The above table does not comply with the 4th normal form, because it has repetitions like this:

Jones	X	A
Jones	Y	B

So this data may be already in the table, which means that it's repeated.

Jones	B	X
Jones	A	Y

To transform this into the 4th normal form (4NF) we must separate the original table into two tables like this:

employee	skill
Jones	electrical
Jones	mechanical
Smith	plumbing

And

employee	language
Jones	French

Jones	German
Smith	Spanish

Join dependencies & Fifth Normal Form
JD(R1,R2, …, Rn)
Every legal relation instances r of R should have a lossless join decomposition into R1,R2, …, Rn
(ie)
((Π<R1>(r)), (Π<R2>(r)), … , (Π<Rn>(r)))=r

Fifth Normal Form

A relation schema R is in 5th normal form with respect to F, a set of functional, multivalued and Join dependencies, if for every nontrivial JD in F+, every Ri is a super key of R.

Topic – 3: Security

Database security:
Mechanisms used to grant and revoke privileges in relational database systems.
Mechanisms -> Discretionary access control

Mechanisms that enforce multiple levels of security -> mandatory access control

Security - protection from malicious attempts to steal or modify data.

☐ **Database system level**
☐ Authentication and authorization mechanisms to allow specific users access only to required data

☐ Assume security at network, operating system, human, and physical levels.
☐ Database specific issues:
☐ each user may have authority to read only part of the data and to write only part of the data.
☐ User authority may correspond to entire files or relations, but it may also correspond only to parts of files or relations.
☐ Local autonomy suggests site-level authorization control in a distributed database.
☐ Global control suggests centralized control

☐ **Operating system level**

☐ Operating system super-users can do anything they want to the database! Good operating system level security is required.

Protection from invalid logins
☐ File-level access protection (often not very helpful for database security)
☐ Protection from improper use of "superuser" authority.
☐ Protection from improper use of privileged machine intructions.

☐ **Network level:** must use encryption to prevent
☐ Eavesdropping (unauthorized reading of messages)
☐ Masquerading (pretending to be an authorized user or sending messages supposedly from authorized users)

Each site must ensure that it communicate with trusted sites (not intruders).
☐ Links must be protected from theft or modification of messages
☐ Mechanisms:
☐ Identification protocol (password-based),
☐ Cryptography.

Physical level
☐ Physical access to computers allows destruction of data by intruders; traditional lock-and-key security is needed
☐ Computers must also be protected from floods, fire, etc.
☐ Protection of disks from theft, erasure, physical damage, etc.
☐ Protection of network and terminal cables from wiretaps noninvasive electronic eavesdropping, physical damage, etc.

Solutions:
☐ Replicated hardware:
☐ mirrored disks, dual busses, etc.
☐ multiple access paths between every pair of devises
☐ Physical security: locks,police, etc.
☐ Software techniques to detect physical security breaches.

☐ **Human level**
☐ Users must be screened to ensure that an authorized users do not give access to intruders
☐ Users should be trained on password selection and secrecy
Protection from stolen passwords, sabotage, etc.

☐ Primarily a management problem:
☐ Frequent change of passwords
☐ Use of "non-guessable" passwords
☐ Log all invalid access attempts
☐ Data audits
☐ Careful hiring practices

Issues:
• Legal & ethical issues
• Policy issues
• System related issues
• Multiple security levels - secret, top secret, confidential & unclassified
Database security and authorization subsystem – responsible for ensuring the security of portions of a database against unauthorized access.

Introduction to Database Security Issues

- Types of Security
- Database Security and the DBA
- Access Protection, User Accounts, and Database Audits

Discretionary Access Control Based on Granting and Revoking Privileges

- Types of Discretionary Privileges
- Specifying Privileges Using Views
- Revoking Privileges
- Propogation of Privileges Using the GRANT OPTION
- An Example
- Specifying Limits on Propagation of Privileges

Mandatory Access Control and Role-Based Access Control for Multilevel Security

- Comparing Discretionary Access Control and Mandatory Access Control
- Role-Based Access Control
- Access Control Policies for E-Commerce and the Web

Types of security mechanisms:
☐ Discretionary security mechanisms
☐ Mandatory security mechanisms
Statistical database security
Data encryption
Database security & DBA:
DBA – responsible for overall security of the database system
Privileged account – system account
Actions performed by DBA:

☐ Account creation
☐ Privilege granting
☐ Privilege revocation
☐ Security level assignment

Database log - audit trail

Discretionary Access Control:
• 2 levels of assigning privileges:
☐ *Account level*
☐ *Relation level*
• Specifying Authorization by using Views
• Revoking Privileges
• Propagation of Privileges and the GRANT OPTION
GRANT SELECT ON EMPLOYEE TO A5;
REVOKE SELECT ON EMPLOYEE FROM A3;
Mandatory Access Control for Multilevel Security:
• ***Simple security property***
• ***Read access***
S, Object O class(S) ≥ class(O)
• ***-property (or star property)**
• **Write access**
S, Object O class(S) ≤ class(O)

Topic – 4: Integrity

Integrity here refers to the CORRECTNESS & CONSISTENCY of the data stored in the database

Database Integrity
CONSISTENCY
Implies that the data held in the tables of the database is consistent in terms of the Relational Data Model
Entity integrity
Referential Integrity

Entity integrity
Each row in the table
Represents a single instance of the entity type modelled by the table
Has a UNIQUE and NON-NULL primary key value
Each column in the table
Represents the occurrences of a single attribute type

Has entries of the appropriate data type

Referential Integrity

Concerned with relationships between tables in the database

i.e. that the data in 1 table does not contradict the data in another

e.g. every FOREIGN KEY value in a table must have a matching PRIMARY KEY value in the corresponding table

Data Validation

Database Management System (DBMS) provides features to help ensure data integrity

Usually implemented using Database Constraints

Specified in data dictionary table definition

Usually specified on creation of table

May be altered/added/dropped later

Constraints

Column Constraints

e.g.

Not Null

Specifies that when a new row is inserted into table

This column must not contain only null values

Default

Allows a default value to be specified

Any time a row with a null value for this column is entered the default value is inserted

Constraints

Table Constraints

e.g.

Primary Key specifies that when a new row is inserted the value of this column must be

NOT NULL &

UNIQUE

DBMS creates an INDEX on primary key columns

Constraints

Table Constraints

Foreign Key specifies that when a new row is inserted the value of this column MUST match

VALUE of the corresponding PRIMARY KEY in the master table

No corresponding master table entry

Row not inserted

Error message

Creating Tables

each column has a column-type indicating

the size of the column and

the datatype of values that are acceptable

initially we will use data types
VARCHAR2 for Alphanumeric
DATE for dates and
NUMBER for numeric

Creating Tables

e.g. The customer table could be defined as

```
Create Table Customer (
    CustomerNo Varchar2(5) NOT NULL,
    Name Varchar2(20) NOT NULL,
    Address Varchar2(60) NOT NULL,
    TelNo Varchar2(15) NOT NULL,
    Email Varchar2(30),
    Constraint Customer_pk
      Primary Key (CustomerNo))
```

Integrity Constraints
- ☐ Domain constraint
- ☐ Key constraint
- ☐ Entity Integrity
- ☐ Referential Integrity

Topic – 5: Consistency

It ensures the truthfulness of the database.

The consistency property ensures that any transaction the database performs will take it from one consistent state to another.

The consistency property does not say how the DBMS should handle an inconsistency other than ensure the database is clean at the end of the transaction. If, for some reason, a transaction is executed that violates the database's consistency rules, the entire transaction could be rolled back to the pre-transactional state - or it would be equally valid for the DBMS to take some patch-up action to get the database in a consistent state.

Thus, if the database schema says that a particular field is for holding integer numbers, the DBMS could decide to reject attempts to put fractional values there, or it could round the supplied values to the nearest whole number: both options maintain consistency.

The consistency rule applies only to integrity rules that are within its scope. Thus, if a DBMS allows fields of a record to act as references to another record, then consistency implies the DBMS must enforce referential integrity: by the time any transaction ends, each and every reference in the database must be valid. If a transaction consisted of an attempt to delete a record referenced by another, each of the following mechanisms would maintain consistency:

- abort the transaction, rolling back to the consistent, prior state;
- delete all records that reference the deleted record (this is known as *cascade delete*); or,
- nullify the relevant fields in all records that point to the deleted record.

These are examples of propagation constraints; some database systems allow the database designer to specify which option to choose when setting up the schema for a database.

Application developers are responsible for ensuring *application level* consistency, over and above that offered by the DBMS. Thus, if a user withdraws funds from an account and the new balance is lower than the account's minimum balance threshold, as far as the DBMS is concerned, the database is in a consistent state even though this rule (unknown to the DBMS) has been violated.

Topic – 6: Database Tuning

- When is tuning necessary?
  - Only if you feel that application is not running fast enough
- What is to be tuned?
  - Oracle database
  - Application
  - Operating system
  - Network

Tuning Goals

- To optimize the performance of database
- To make database available to users without making them wait for resources
- To perform maintenance operations without interrupting users

Tuning Parameters

- Response time
- Database availability
- Database hit percentages
- Memory utilization

Tuning Steps

a) Tune the design
b) Tune the application
c) Tune memory
d) Tune IO
e) Tune contention
f) Tune operating system

Tuning Considerations

- Different for
  - OLTP databases
  - DSS databases
  - Hybrid databases
- Our database
  - Hybrid database
  - Data entry and Report generation done simultaneously

Performance Tuning

Adjusting various parameters and design choices to improve system performance for a specific application.

Tuning is best done by
identifying bottlenecks, and
eliminating them.

Can tune a database system at 3 levels:

Hardware -- e.g., add disks to speed up I/O, add memory to increase buffer hits, move to a faster processor.

Database system parameters -- e.g., set buffer size to avoid paging of buffer, set checkpointing intervals to limit log size. System may have automatic tuning.

Higher level database design, such as the schema, indices and transactions

Bottlenecks

Performance of most systems (at least before they are tuned) usually limited by performance of one or a few components: these are called bottlenecks

E.g. 80% of the code may take up 20% of time and 20% of code takes up 80% of time

Worth spending most time on 20% of code that take 80% of time

Bottlenecks may be in hardware (e.g. disks are very busy, CPU is idle), or in software

Removing one bottleneck often exposes another

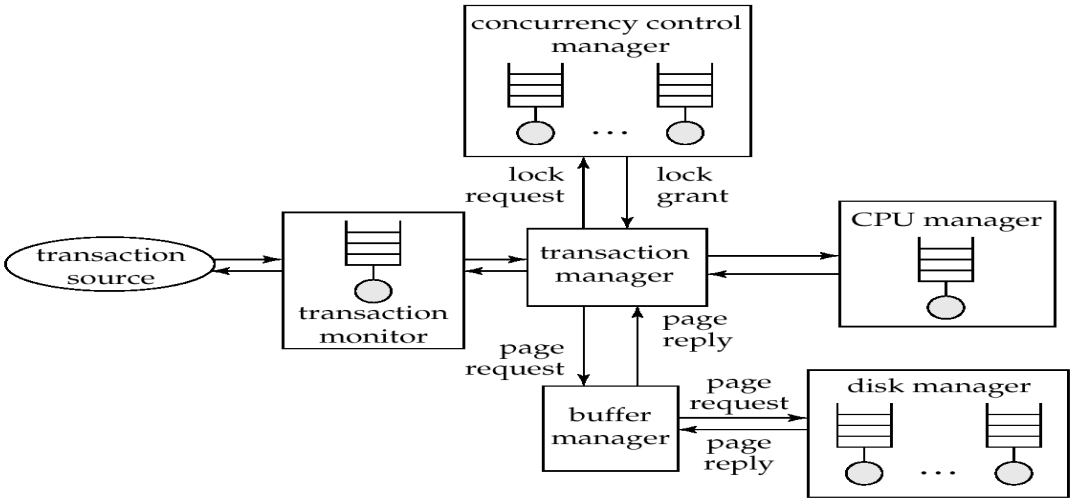
De-bottlenecking consists of repeatedly finding bottlenecks, and removing them

This is a heuristic

Identifying Bottlenecks

Transactions request a sequence of services
e.g. CPU, Disk I/O, locks
With concurrent transactions, transactions may have to wait for a requested service while other transactions are being served
Can model database as a **queueing system** with a queue for each service
transactions repeatedly do the following
request a service, wait in queue for the service, and get serviced
Bottlenecks in a database system typically show up as very high utilizations (and correspondingly, very long queues) of a particular service
E.g. disk vs CPU utilization
100% utilization leads to very long waiting time:
Rule of thumb: design system for about 70% utilization at peak load
utilization over 90% should be avoided

Queues In A Database System



Tunable Parameters

Tuning of hardware
Tuning of schema
Tuning of indices
Tuning of materialized views
Tuning of transactions

Tuning of Hardware

Even well-tuned transactions typically require a few I/O operations
Typical disk supports about 100 random I/O operations per second
Suppose each transaction requires just 2 random I/O operations. Then to support *n* transactions per second, we need to stripe data across *n*/50 disks (ignoring

skew)

Number of I/O operations per transaction can be reduced by keeping more data in memory

If all data is in memory, I/O needed only for writes

Keeping frequently used data in memory reduces disk accesses, reducing number of disks required, but has a memory cost

Hardware Tuning: Five-Minute Rule

Question: which data to keep in memory:

If a page is accessed *n* times per second, keeping it in memory saves

$$n * \frac{\text{price-per-disk-drive}}{\text{accesses-per-second-per-disk}}$$

Cost of keeping page in memory

$$\frac{\text{price-per-MB-of-memory}}{\text{ages-per-MB-of-memory}}$$

Break-even point: value of *n* for which above costs are equal

If accesses are more then saving is greater than cost

Solving above equation with current disk and memory prices leads to:

5-minute rule: if a page that is randomly accessed is used more frequently than once in 5 minutes it should be kept in memory

(by buying sufficient memory!)

Hardware Tuning: One-Minute Rule

For sequentially accessed data, more pages can be read per second. Assuming sequential reads of 1MB of data at a time:

1-minute rule: **sequentially accessed data that is accessed once or more in a minute should be kept in memory**

Prices of disk and memory have changed greatly over the years, but the ratios have not changed much

so rules remain as 5 minute and 1 minute rules, not 1 hour or 1 second rules!

Hardware Tuning: Choice of RAID Level

To use RAID 1 or RAID 5?

Depends on ratio of reads and writes

RAID 5 requires 2 block reads and 2 block writes to write out one data block

If an application requires *r* reads and *w* writes per second

RAID 1 requires $r + 2w$ I/O operations per second

RAID 5 requires: $r + 4w$ I/O operations per second

For reasonably large r and w, this requires lots of disks to handle workload

RAID 5 may require more disks than RAID 1 to handle load!

Apparent saving of number of disks by RAID 5 (by using parity, as opposed to the mirroring done by RAID 1) may be illusory!

Thumb rule: RAID 5 is fine when writes are rare and data is very large, but RAID 1 is preferable otherwise

If you need more disks to handle I/O load, just mirror them since disk capacities these days are enormous!

Tuning the Database Design

Schema tuning

Vertically partition relations to isolate the data that is accessed most often -- only fetch needed information.

E.g., split *account* into two, (*account-number*, *branch-name*) and (*account-number*, *balance*).

Branch-name need not be fetched unless required

Improve performance by storing a **denormalized relation**

E.g., store join of *account* and *depositor*; branch-name and balance information is repeated for each holder of an account, but join need not be computed repeatedly.

Price paid: more space and more work for programmer to keep relation consistent on updates

better to use materialized views (more on this later..)

Cluster together on the same disk page records that would match in a frequently required join,

compute join very efficiently when required.

Index tuning

Create appropriate indices to speed up slow queries/updates

Speed up slow updates by removing excess indices (tradeoff between queries and updates)

Choose type of index (B-tree/hash) appropriate for most frequent types of queries.

Choose which index to make clustered

Index tuning wizards look at past history of queries and updates (the **workload**) and recommend which indices would be best for the workload

Materialized Views

Materialized views can help speed up certain queries

Particularly aggregate queries

Overheads

Space

Time for view maintenance

Immediate view maintenance:done as part of update txn

time overhead paid by update transaction

Deferred view maintenance: done only when required

update transaction is not affected, but system time is spent on view maintenance

until updated, the view may be out-of-date

Preferable to denormalized schema since view maintenance

is systems responsibility, not programmers
Avoids inconsistencies caused by errors in update programs
How to choose set of materialized views
Helping one transaction type by introducing a materialized view may hurt others
Choice of materialized views depends on costs
Users often have no idea of actual cost of operations
Overall, manual selection of materialized views is tedious
Some database systems provide tools to help DBA choose views to materialize
"Materialized view selection wizards"

Tuning of Transactions

Basic approaches to tuning of transactions
Improve set orientation
Reduce lock contention
Rewriting of queries to improve performance was important in the past, but smart optimizers have made this less important
Communication overhead and query handling overheads significant part of cost of each call
Combine multiple embedded SQL/ODBC/JDBC queries into a single set-oriented query
Set orientation -> fewer calls to database
E.g. tune program that computes total salary for each department using a separate SQL query by instead using a single query that computes total salaries for all department at once (using **group by)**
Use stored procedures: avoids re-parsing and re-optimization of query
Reducing lock contention
Long transactions (typically read-only) that examine large parts of a relation result in lock contention with update transactions
E.g. large query to compute bank statistics and regular bank transactions
To reduce contention
Use multi-version concurrency control
E.g. Oracle "snapshots" which support multi-version 2PL
Use degree-two consistency (cursor-stability) for long transactions
Drawback: result may be approximate
Long update transactions cause several problems
Exhaust lock space
Exhaust log space
and also greatly increase recovery time after a crash, and may even exhaust log space during recovery if recovery algorithm is badly designed!
Use **mini-batch** transactions to limit number of updates that a single transaction can carry out. E.g., if a single large transaction updates every record of a very large relation, log may grow too big.
* Split large transaction into batch of ``mini-transactions,'' each performing part

of the updates
Hold locks across transactions in a mini-batch to ensure serializability
If lock table size is a problem can release locks, but at the cost of serializability
* In case of failure during a mini-batch, must complete its remaining portion on recovery, to ensure atomicity.

Performance Simulation

Performance simulation using queuing model useful to predict bottlenecks as well as the effects of tuning changes, even without access to real system
Queuing model as we saw earlier
Models activities that go on in parallel
Simulation model is quite detailed, but usually omits some low level details
Model **service time**, but disregard details of service
E.g. approximate disk read time by using an average disk read time
Experiments can be run on model, and provide an estimate of measures such as average throughput/response time
Parameters can be tuned in model and then replicated in real system
E.g. number of disks, memory, algorithms, etc

Topic – 7: Optimization and Research Issues

Understanding the Query Optimizer

A SQL statement can be executed in many different ways, such as full table scans, index scans, nested loops, and hash joins.

The output from the optimizer is a plan that describes an optimum method of execution.

The query optimizer determines the most efficient way to execute a SQL statement after considering many factors related to the objects referenced and the conditions specified in the query.

This determination is an important step in the processing of any SQL statement and can greatly affect execution time.

The query optimizer determines which execution plan is most efficient by considering available access paths and by factoring in information based on statistics for the schema objects (tables or indexes) accessed by the SQL statement.

The query optimizer also considers hints, which are optimization suggestions placed in a comment in the statement.

The query optimizer performs the following steps:

1. The optimizer generates a set of potential plans for the SQL statement based on available access paths and hints.

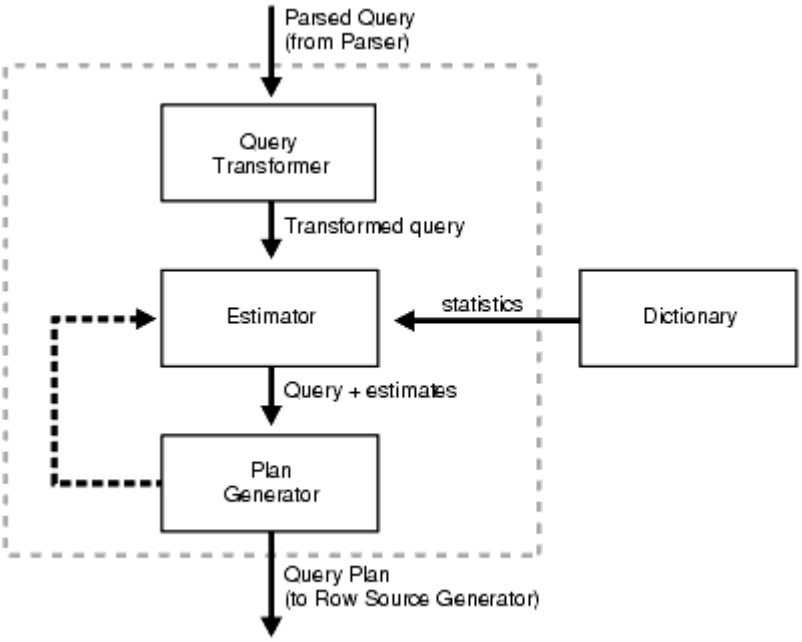
2. The optimizer estimates the cost of each plan based on statistics in the data dictionary for the data distribution and storage characteristics of the tables, indexes, and partitions accessed by the statement.

   The **cost** is an estimated value proportional to the expected resource use needed to execute the statement with a particular plan. The optimizer calculates the cost of access paths and join orders based on the estimated computer resources, which includes I/O, CPU, and memory.

   Serial plans with higher costs take more time to execute than those with smaller costs. When using a parallel plan, however, resource use is not directly related to elapsed time.

3. The optimizer compares the costs of the plans and chooses the one with the lowest cost.

Query optimizer components are illustrated in



Components of the Query Optimizer

The query optimizer operations include:

- Transforming Queries
- Estimating
- Generating Plans

Transforming Queries

The input to the query transformer is a parsed query, which is represented by a set of query blocks. The query blocks are nested or interrelated to each other. The form of the query determines how the query blocks are interrelated to each other. The main objective of the query transformer is to determine if it is advantageous to change the form of the query so that it enables generation of a better query plan.

Estimating

The end goal of the estimator is to estimate the overall cost of a given plan. If statistics are available, then the estimator uses them to compute the measures. The statistics improve the degree of accuracy of the measures.

The estimator generates three different types of measures:

- Selectivity
- Cardinality

- Cost

These measures are related to each other, and one is derived from another.

Generating Plans

The main function of the plan generator is to try out different possible plans for a given query and pick the one that has the lowest cost. Many different plans are possible because of the various combinations of different access paths, join methods, and join orders that can be used to access and process data in different ways and produce the same result.

Research Issues

Multi-Query Optimization
Scenario: Multiple related, but slightly different queries
Goal: Save power and communication
Challenge: Combining multiple queries, finding common query parts
Two approaches:
Materialization
Pipelining

(syntactic) optimizer Vs syntactic optimizer
SQL query text is first semantically optimized then passed to the conventional (syntactic) optimizer.

Any advantage bestowed by the semantic optimizer can only be manifested by the syntactic optimizer.

The syntactic optimizer will typically look to indexes to enhance query efficiency.

Topic – 8: Design of Temporal Databases

What are temporal databases?

Temporal Databases
Temporal DBMS manages time-referenced data, and times are associated with database entities.
It encompasses database applications that require some aspect of time when organizing their information.

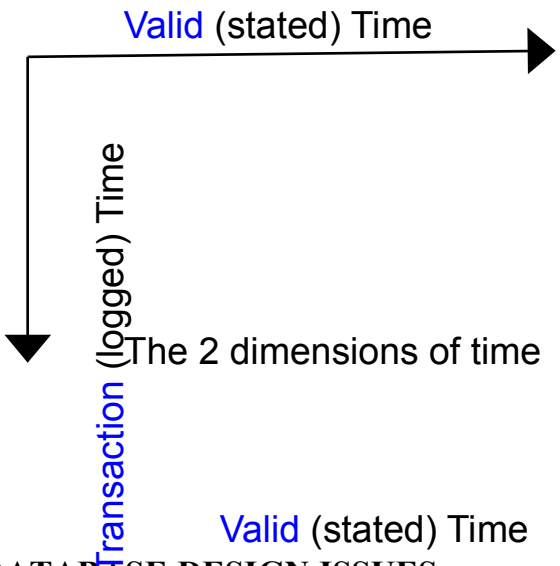
Most applications of database technology are temporal in nature:
Financial apps.: portfolio management, accounting & banking
Record-keeping apps.: personnel, medical record and inventory management

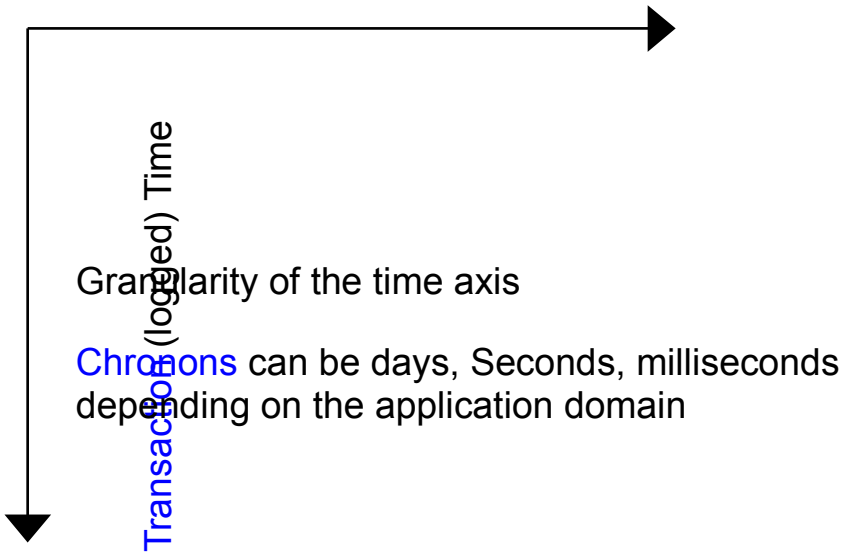
- Scheduling apps.: airline, car, hotel reservations and project management
- Scientific apps.: weather monitoring
- Definition:
-

Applications:
health-care system insurance reservation systems, scientific databases
• Time Representation, Time Dimensions
time- ordered sequence of **points** in some granularity that is determined by application
Calendar- organizes time into different time units
(eg) 60 secs. -> 1 min etc.

- **Non Temporal**
  - store only a single state of the real world, usually the most recent state
  - classified as snapshot databases
  - application developers and database designers need to code for time varying data requirements eg history tables, forecast reports etc
- **Temporal**
  - stores upto two dimensions of time i.e VALID (stated) time and TRANSACTION (logged) time
  - Classified as historical, rollback or bi-temporal
  - No need for application developers or database designers to code for time varying data requirements i.e time is inherently supported

Temporal Data types:
1) DATE 2) TIME 3) TIMESTAMP 4) INTERVAL 5) PERIOD





We can use these two dimensions to distinguish between different forms of temporal database

- A rollback database stores data with respect to transaction time e.g. Oracle 10g has flashback query
- A historical database stores data with respect to valid time
- A bi-temporal database stores data with respect to both valid time and transaction time.

What is time varying data?

- You want a reprint of a customer's invoice of August 12, 1999.
- What was the stock value of the Oracle shares on June 15th, last year?
- What was the lowest stock quantity for every product last year? How much money will you save, if you keep the stocks at those levels?
- Where do you enter the new address of this customer as from the first of next month?
- What will your profits be next month, given the price list and cost prices by then?

And combinations of the situations can be very complex

- You offered these goods to the customer on January 10 this year. What were the billing prices and what was his discount level when you sent him this offer? He has not accepted yet. Is it smart to offer him an actualized discount now?

- Given the final settlements for all the insurance claims of the last three years, what will be the minimum insurance premium your customers have to pay next year**?**

Examples of application domains dealing with time varying data:

- Financial Apps (e.g. history of stock market data)
- Insurance Apps (e.g. when were the policies in effect)
- Reservation Systems (e.g. when is which room in a hotel booked)
- Medical Information Management Systems (e.g. patient records)
- Decision Support Systems (e.g. planning future contigencies)
- CRM applications (eg customer history / future)
- HR applications (e.g Date tracked positions in hierarchies)

In fact, time varying data has ALWAYS been in business requirements – but existing technology does not deal with it elegantly!

Event Information Versus Duration (or State) Information:

Point events or facts
single time point
time series data
Duration events or facts
time period [start-time, end_time]

Valid Time and Transaction Time Dimensions:

Interpretation of events available in temporal databases
valid time
transaction time
valid time database, transaction time database
Bitemporal database
User-defined time
90
Time dimensions
Time, semantics & program
applications

Incorporating Time in Relational Databases

Valid Time Relations

Granularity → Day, data type

Valid Start Time(VST), Valid End Time(VET)

Temporal variable → now

Update operations in temporal relations:
current version, old version
proactive update (updation before implementation)
reactive update (updation after implementation)
simultaneous update

EMP_BT

SSN ENAME DNO VST VET TST TET

Incorporating Time in Relational Databases

Transaction Time Relations
Transaction Start Time(TST), Transaction End Time(TET)
Transaction time relations
Rollback database

Bitemporal Time Relations
< VST,VET,TST,TET>

Time Series Data
Data values recorded according to a specific predefined sequence of time points.
Usage:
financial, sales & economics applications
Typical queries involve temporal aggregation
Time series management systems

Implementation Approaches
Several implementation strategies are available

- Use a date type supplied in a non-temporal DBMS and build temporal support into applications (traditional)
- Implement an abstract data type for time (object oriented)
- Provide a program layer (api) above a non-temporal data model (stratum)

- Generalise a non-temporal data model into a temporal data model (Temporal Normal Form)
- Re-design core database kernel (Temporal Database)

Topic – 9: Spatial Databases

Introduction

Many applications in various fields require management of *geometric, geographic or spatial* data (data related to space)

- A geographic space: surface of the earth
- Man-made space: layout of VLSI design
- Model of the human brain
- 3-D space representation of the chains of protein molecules
- The Common challenge:
- Dealing with large collections of relatively simple geometric objects: e.g., 100,000 polygons

Spatial Database

What is a SDBMS ?

- A SDBMS is a software module that
  - can work with an underlying DBMS
  - supports spatial data models, spatial abstract data types (ADTs) and a query language from which these ADTs are callable
  - supports spatial indexing, efficient algorithms for processing spatial operations, and domain specific rules for query optimization
- Example: Oracle Spatial data cartridge, ESRI SDE
  - can work with Oracle 8i DBMS
  - Has spatial data types (e.g. polygon), operations (e.g. overlap) callable from SQL3 query language
  - Has spatial indices, e.g. R-trees

A spatial database system:

- Is a database system (with additional capabilities for handling spatial data)
- Offers spatial data types (SDTs) in its data model and query language
- Structure in space: e.g., POINT, LINE, REGION
- Relationships among them: e.g., *a* intersects *b*
- Supports SDT in its implementation
- Spatial indexing: retrieving objects in particular area without scanning the whole space
- Efficient algorithm for spatial joins

Example:
Assume 2-D GIS application, two basic things need to be represented:

☐ Objects in space: cities, forests, or rivers
distinct entities arranged in space, each of which has its own geometric description
=>modeling single objects
☐ Space: describe the space itself say something about every point in space
=>modeling spatially related collections of objects

SDBMS Example

- Consider a spatial dataset with:
  - County boundary (dashed white line)
  - Census block - name, area, population, boundary (dark line)
  - Water bodies (dark polygons)
  - Satellite Imagery (gray scale pixels)
  - Storage in a SDBMS table:

```
create table census_blocks (
        name  string,
        area          float,
  population      number,
       boundary   polygon );
```

Spatial Databases
Concepts about objects in a multidimensional space.
n-dimensional space.
(eg) maps
Police vehicles, ambulances

Techniques for spatial indexing:
1) **R-trees**
Rectangle areas
Leaf node
Internal nodes->rectangles whose area covers all the rectangles in its subtree
2) **Quadtrees**
divides each space or subspace into equally sized areas & proceed with the subdivisions of each subspace to identify the positions of various objects.

Spatial Data Types and Traditional Databases

- Traditional relational DBMS
  - Support simple data types, e.g. number, strings, date
  - Modeling Spatial data types is tedious
- Example: modeling of polygon using numbers
  - Three new tables: polygon, edge, points

        - Note: Polygon is a polyline where last point and first point are same
    - A simple unit sqaure represented as 16 rows across 3 tables
    - Simple spatial operators, e.g. area(), require joining tables
    - Tedious and computationally inefficient

Question. Name post-relational database management systems which facilitate modeling of spatial data types, e.g. polygon

Spatial Data Types and Post-relational Databases

- *Post-relational DBMS*
    - *Support user defined abstract data types*
    - *Spatial data types (e.g. polygon) can be added*
- *Choice of post-relational DBMS*
    - *Object oriented (OO) DBMS*
    - *Object relational (OR) DBMS*
- *A spatial database is a collection of spatial data types, operators, indices, processing strategies, etc. and can work with many post-relational DBMS as well as programming languages like Java, Visual Basic etc.*

How is a SDBMS different from a GIS?

- *GIS is a software to visualize and analyze spatial data using spatial analysis functions such as*
    - *Search Thematic search, search by region, (re-)classification*
    - *Location analysis Buffer, corridor, overlay*
    - *Terrain analysis Slope/aspect, catchment, drainage network*
    - *Flow analysis Connectivity, shortest path*
    - *Distribution Change detection, proximity, nearest neighbor*
    - *Spatial analysis/Statistics Pattern, centrality, autocorrelation, indices of similarity, topology: hole description*
    - *Measurements Distance, perimeter, shape, adjacency, direction*
- *GIS uses SDBMS*
    - *to store, search, query, share large spatial data sets*

- **SDBMS focuses on**
    - Efficient storage, querying, sharing of large spatial datasets
    - Provides simpler set based query operations
    - Example operations: search by region, overlay, nearest neighbor, distance, adjacency, perimeter etc.
    - Uses spatial indices and query optimization to speedup queries over large spatial datasets.
- SDBMS may be used by applications other than GIS
    - Astronomy, Genomics, Multimedia information systems, ...

- Will one use a GIS or a SDBM to answer the following:
  - How many neighboring countries does USA have?
  - Which country has highest number of neighbors?

Components of a SDBMS

- Recall: a SDBMS is a software module that
  - can work with an underlying DBMS
  - supports spatial data models, spatial ADTs and a query language from which these ADTs are callable
  - supports spatial indexing, algorithms for processing spatial operations, and domain specific rules for query optimization
- Components include
  - spatial data model, query language, query processing, file organization and indices, query optimization, etc.
  - Figure 1.6 shows these components
  - We discuss each component briefly in chapter 1.6 and in more detail in later chapters.

Three Layer Architecture

Spatial Applications → Spatial DB → DBMS

Spatial Taxonomy, Data Models

- Spatial Taxonomy:
  - multitude of descriptions available to organize space.
  - Topology models homeomorphic relationships, e.g. overlap
  - Euclidean space models distance and direction in a plane
  - Graphs models connectivity, Shortest-Path
- Spatial data models
  - rules to identify identifiable objects and properties of space
  - Object model help manage identifiable things, e.g. mountains, cities, land-parcels etc.
  - Field model help manage continuous and amorphous phenomenon, e.g. wetlands, satellite imagery, snowfall etc.

Spatial Query Language

Types of spatial queries:
1) Range query 2) Nearest neighbor query 3) Spatial Joins

- Spatial query language
  - Spatial data types, e.g. point, linestring, polygon, …
  - Spatial operations, e.g. overlap, distance, nearest neighbor, …

- Callable from a query language (e.g. SQL3) of underlying DBMS

```
SELECT  S.name
FROM    Senator S
WHERE S.district.Area() > 300
```

- Standards
  - SQL3 (a.k.a. SQL 1999) is a standard for query languages
  - OGIS is a standard for spatial data types and operators
  - Both standards enjoy wide support in industry

Two main issues:
☐ 1. Connecting the operations of a spatial algebra to the facilities of a DBMS query language.
☐ 2. Providing graphical presentation of spatial data (i.e. results of queries), and graphical input of SDT values used in queries.

Fundamental spatial algebra operations:

☐ **Spatial selection**: returning those objects satisfying a spatial predicate with the query object
Example: All big cities no more than 300Kms from Lausanne
☐ SELECT cname FROM cities c WHERE dist(c.center, Lausanne.center) < 300 and c.pop > 500K

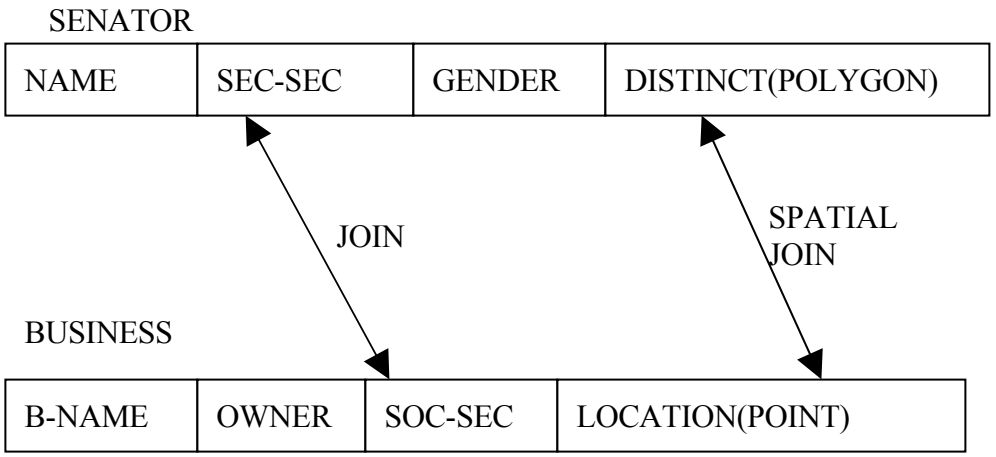
☐ **Spatial join:** A join which compares any two joined objects based on a predicate on their spatial attribute values For each river pass through Switzerland, find all cities within less than 50KMs

☐ SELECT c.cname FROM rivers r, cities c
WHERE r.route intersects Switzerland. .area and
dist(r.route, c.area) < 50KM

Requirements for spatial querying
☐ Spatial data types
☐ Graphical display of query results
☐ Graphical combination of several query results
☐ Display of context
☐ A facility for checking the context of display
☐ Extended dialog
☐ Varying graphical representations
☐ Legend
☐ Label placement

- ☐ Scale Selection
- ☐ Subarea for queries

Multi-scan Query
Spatial join Example
SELECT S.name FROM Senator S, Business B WHERE S.dsitinct.Area() > 300 AND Within(B.location, S.distinct)
Non-Spatial Join Example:
SELECT S.name FROM Senator S, Business B WHERE S.soc.Sec AND S.gender =’Female’ AND Within(B.location, S.distinct)



Sample Questions

Topic – 1:

1) What are the two ways of modeling a Database? (2M)
2) What are the steps in designing database? (2M)
3) What are entities? Describe about Entity set (2M)
4) What are attributes? (2M)
5) What are the different types of attributes? (2M)
6) What is relationship? Describe about Relationship set. (2M)
7) Describe each of the following:
   Entity types, value sets & Key attributes (8M)
8) What is cardinality? List the benefits of it.
9) Explain all the four types of Cardinalities with an example. (8M)
10) List and describe the notations used for ER models. (16M)
11) Draw ER Model diagram for the following problem statement:
    The problem area is Company environment.
    a) Each employee data such emp#, name, date-of-birth, address, city, state, country should be stored.
    b) Employee must work in particular department.
    c) Each department information such dept#, name, location should be stored.

Topic – 2:

1) What is Normalization? (2M)
2) Why we need to select and apply Normalization? (2M)
3) What are redundant data ? How they influences different anomalies and explain them with an example. (8M)
4) Compare and contrast Normalization with Denormalization. (2M)
5) What are Functional Dependencies? (FDs). Explain briefly. (8M)
6) Briefly describe 3 Basic normal forms with an example for each. (8M)
7) List and describe the basic rule(s) behind First Normal Form(1NF). Explain with an example.

8) List and describe the basic rule(s) behind First Normal Form(1NF). Explain with an example.
9) List and describe the basic rule(s) behind Second Normal Form(2NF). Explain with an example.
10) List and describe the basic rule(s) behind First Normal Form(3NF). Explain with an example.
11) List and describe the basic rule(s) behind Boyce-Codd Normal Form(BCNF). Explain with an example.
12) List and describe the basic rule(s) behind Fourth Normal Form(4NF). Explain with an example.
13) List and describe the basic rule(s) behind Fifth Normal Form(5NF). Explain with an example.
14) "All 3NF relations need not be BCNF" – Explain with an example. (2M)
15) What are Multivalued dependencies? Explain with an example. (2M)
16) What are Join dependencies? Explain with an example. (2M)
17) What is Normalization? Explain the various normalization techniques with suitable examples. (16M)
18) Given the Comparison between BCNF and 3NF. (8M)
19) Choose a key and write the dependencies for the following Grades: relation:
GRADES(Student_ID, Course#, Semester#, Grade)

Answer:
Key is :
Student_ID, Course#, Semester#,
Dependency is:
Student_ID, Course#, Semester# -> Grade

20) Choose a key and write the dependencies for the LINE_ITEMS relation:
LINE_ITEMS (PO_Number, ItemNum, PartNum, Description, Price, Qty)

Answer:
Key can be: PO_Number, ItemNum
Dependencies are:
PO_Number, ItemNum -> PartNum, Description, Price, Qty
PartNum -> Description, Price

21) What normal form is the above LINE_ITEMS relation in?

Answer:
First off, LINE_ITEMS could not be in **BCNF** because:
not all determinants are keys.
next: it could not be in **3NF** because there is a transitive dependency:

PO_Number, ItemNum -> PartNum
and
PartNum -> Description

Therefore, it must be in **2NF**, we can check this is true because:
the key of PO_Number, ItemNum determines all of the non-key attributes however, PO_Number by itself and ItemNum by itself can not determine any other attributes.

22) What normal form is the following relation in?
STORE_ITEM(SKU, PromotionID, Vendor, Style, Price)
SKU, PromotionID -> Vendor, Style, Price
SKU -> Vendor, Style

Answer:
STORE_ITEM is in **1NF** (non-key attribute (vendor) is dependent on only part of

the key.

23) Normalize the above (Q4) relation into the next higher normal form.

Answer:
STORE_ITEM (SKU, PromotionID, Price)
VENDOR ITEM (SKU, Vendor, Style)

24) Choose a key and write the dependencies for the following SOFTWARE relation (assume all of the vendor's products have the same warranty).
SOFTWARE (SoftwareVendor, Product, Release, SystemReq, Price, Warranty)
SoftwareVendor, Product, Release -> SystemReq, Price, Warranty

Answer:
key is: SoftwareVendor, Product, Release
SoftwareVendor, Product, Release -> SystemReq, Price, Warranty
SoftwareVendor -> Warranty
.:. SOFTWARE is in 1NF

25) Normalize the above Software relation into 4NF.

Answer:
SOFTWARE (SoftwareVendor, Product, Release, SystemReq, Price)
WARRANTY (SoftwareVendor, Warranty)

26) What normal form is the following relation in?
only H,I can act as the key.
STUFF (H, I, J, K, L, M, N, O)
H, I -> J, K, L
J -> M
K -> N
L -> O

Answer:
2NF (Transitive dependencies exist)

25) What normal form the following relation in?
STUFF2 (D, O, N, T, C, R, Y)
D, O -> N, T, C, R, Y
C, R -> D
D -> N

Answer:
1NF (Partial Key Dependency exist)

26) Is this relation in 1NF? 2NF? 3NF?
Convert the relation to 3NF.

Invoice relation

Inv#	date	custID	Name	Part#	Desc	Price	#Used	Ext Price	Tax rate	Tax	Total
14	12/63	42	Lee	A38	Nut	0.32	10	3.20	0.10	1.22	13.42
14	12/63	42	Lee	A40	Saw	4.50	2	9.00	0.10	1.22	13.42
15	1/64	44	Pat	A38	Nut	0.32	20	6.40	0.10	064	7.04

Table not in 1NF because
- it contains derived values
EXT PRICE(=Price X # used)
3.2 = 0.32 X 10

- Tax (=sum of Ext price of same Inv# X Tax rate)
1.22 = (3.2 + 9.00) X 0.10

- Total (=sum of Ext price + Tax)
13.42 = (3.20 + 9.00) + 1.22

To get 1NF, identify PK and remove derived attributes

Inv#	date	custID	Name	**Part#**	Desc	Price	#Used	Tax rate
14	12/63	42	Lee	A38	Nut	0.32	10	0.10
14	12/63	42	Lee	A40	Saw	4.50	2	0.10
15	1/64	44	Pat	A38	Nut	32	20	0.10

To get 2NF
- Remove partial dependencies

- Partial FDs with key attributes.
- Inv# -> Date, CustID, Name, Tax Rate
- Part# -> Desc, Price

Remove Partial FDs

|–K1-||————————D1————————————||—K2—||———-D2————|

Inv#	date	custID	Name	Tax rate	**Part#**	Desc	Price	#Used
14	12/63	42	Lee	0.10	A38	Nut	0.32	10
14	12/63	42	Lee	0.10	A40	Saw	4.50	2
15	1/64	44	Pat	0.10	A38	Nut	32	20

=

Inv#	date	custID	Name	Tax rate
14	12/63	42	Lee	0.10
~~14~~	~~12/63~~	~~42~~	~~Lee~~	~~0.10~~
15	1/64	44	Pat	0.10

Inv#	**Part#**	#Used
14	A38	10
14	A40	2
15	A38	20

Part#	Desc	Price
A38	Nut	0.32
A40	Saw	4.50
~~A38~~	~~Nut~~	~~32~~

Remove transitive FD

Inv#(PK) -> CustID -> Name

Inv#	date	custID	Name	Tax rate
14	12/63	42	Lee	0.10
15	1/64	44	Pat	0.10

=

Inv#	date	custID	Tax rate
14	12/63	42	0.10
15	1/64	44	0.10

+

custID	Name
42	Lee
44	Pat

All relations in 3NF

Inv#	**Part#**	#Used
14	A38	10
14	A40	2
15	A38	20

Part#	Desc	Price
A38	Nut	0.32
A40	Saw	4.50

Inv#	date	custID	Tax rate
14	12/63	42	0.10
15	1/64	44	0.10

custID	Name
42	Lee
42	Pat

27) Given an Unnormalized Data Items for Puppies

- puppy number
- puppy name
- kennel code
- kennel name

- kennel location
- trick ID
- trick name
- trick where learned
- skill level

Convert the relation from NNF to !NF, 2NF, 3NF.

Topic – 3:

1. Define Database security. (2M)
2. Explain Database system level security. (2M)
3. Explain Operating system level security. (2M)
4. Explain Network level security. (2M)
5. Explain Physical level security. (2M)
6. Explain Human level security. (2M)
7. Briefly explain the Database Security Issues. (8M)
8. Briefly explain on Types of security mechanisms. (8M)

Topic – 4:

What are Database Integrity? (2M)
How consistency is related to Integrity? Explain. (8M)
Explain Entity integrity in detail. (8M)
Explain about Integrity Constraints. (8m)

Topic – 5:

1. Explain in detail on Database Consistency. (8M)

Topic – 6:

1. When is tuning necessary? (2M)
2. What is to be tuned? (2M)
3. What is DB Tuning? (2M)
4. List the Tuning Goals. (2M)
5. What are the Tuning Parameters considered in DB tuning? (8M)
6. List and describe the Tuning Steps in DB tuning. (8M)
7. Explain briefly on Performance Tuning. (8M)
8. What are Tunable Parameters? (2M)
9. Explain briefly on Tuning of Hardware. (8M)
10. How Tuning the Database Design is achieved? Explain. (8M)

Topic – 7:

1. Explain Query Optimization in detail. (8M)
2. How do you understand the Query Optimizer? (2M)
3. What are the steps performed in the query optimizer? Describe. (8M)
4. Illustrate Query optimizer components with a neat diagram. (8M)

5. Explain the three basic query optimizer operations? (4M)
6. List and describe the Research Issues briefly. (8M)

Topic – 8:

1. Explain the design issues of Temporal databases (8M)

Topic – 9:

1. Explain in detail the features of spatial databases. (8M)

University Questions

1. Discuss about the design issues involved in temporal databases. (8M).

********** End of Unit – IV **********