CS3492 DATABASE MANAGEMENT SYSTEMS

UNIT I RELATIONAL DATABASES **10**

Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys – Relational Algebra – SQL fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL

UNIT II DATABASE DESIGN **8**

Entity-Relationship model – E-R Diagrams – Enhanced-ER Model – ER-to-Relational Mapping – Functional Dependencies – Non-loss Decomposition – First, Second, Third Normal Forms, Dependency Preservation – Boyce/Codd Normal Form – Multi-valued Dependencies and Fourth Normal Form – Join Dependencies and Fifth Normal Form

UNIT III TRANSACTIONS **9**

Transaction Concepts – ACID Properties – Schedules – Serializability – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery - Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery.

UNIT IV IMPLEMENTATION TECHNIQUES **9**

RAID – File Organization – Organization of Records in Files – Indexing and Hashing –Ordered Indices – B+ tree Index Files – B tree Index Files – Static Hashing – Dynamic Hashing – Query Processing Overview – Algorithms for SELECT and JOIN operations – Query optimization using Heuristics and Cost Estimation.

UNIT V ADVANCED TOPICS **9**

Distributed Databases: Architecture, Data Storage, Transaction Processing – Object-based Databases: Object Database Concepts, Object-Relational features, ODMG Object Model, ODL, OQL - XML Databases: XML Hierarchical Model, DTD, XML Schema, XQuery – Information Retrieval: IR Concepts, Retrieval Models, Queries in IR systems.

TOTAL: 45 PERIODS

TEXT BOOKS:

1. Abraham Silberschatz, Henry F. Korth, S. Sudharshan, —Database System Concepts||,Sixth Edition, Tata McGraw Hill, 2011
2. Ramez Elmasri, Shamkant B. Navathe, —Fundamentals of Database Systems||, Sixth Edition, Pearson, 2011.

REFERENCES:

1. C. J. Date, A.Kannan, S. Swamynathan, —An Introduction to Database Systems||, Eighth Edition, Pearson Education, 2006.
2. Raghu Ramakrishnan, —Database Management Systems||, Fourth Edition, McGraw-Hill College Publications, 2015.
3. G.K.Gupta,"Database Management Systems||, Tata McGraw Hill, 2011.

UNIT II DATABASE DESIGN

Entity-Relationship model – E-R Diagrams – Enhanced-ER Model – ER-to-Relational Mapping – Functional Dependencies – Non-loss Decomposition – First, Second, Third Normal Forms, Dependency Preservation – Boyce/Codd Normal Form – Multi-valued Dependencies and Fourth Normal Form – Join Dependencies and Fifth Normal Form

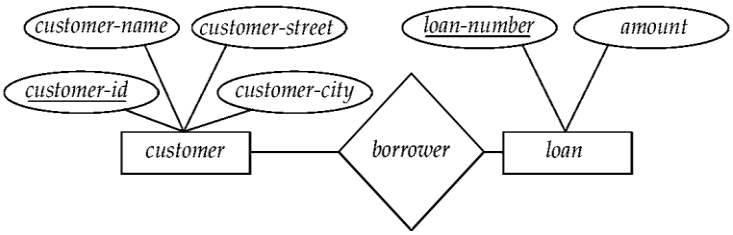
1.8 DATA MODELS

- **A collection of tools for describing**
  - **data**
  - **data relationships**
  - **data semantics**
  - **data constraints**
- **Entity-Relationship model**
- **Relational model**
- **Other models:**
  - **object-oriented model**
  - **semi-structured data models**
  - **Older models: network model and hierarchical model**

1.9 ENTITY-RELATIONSHIP MODEL

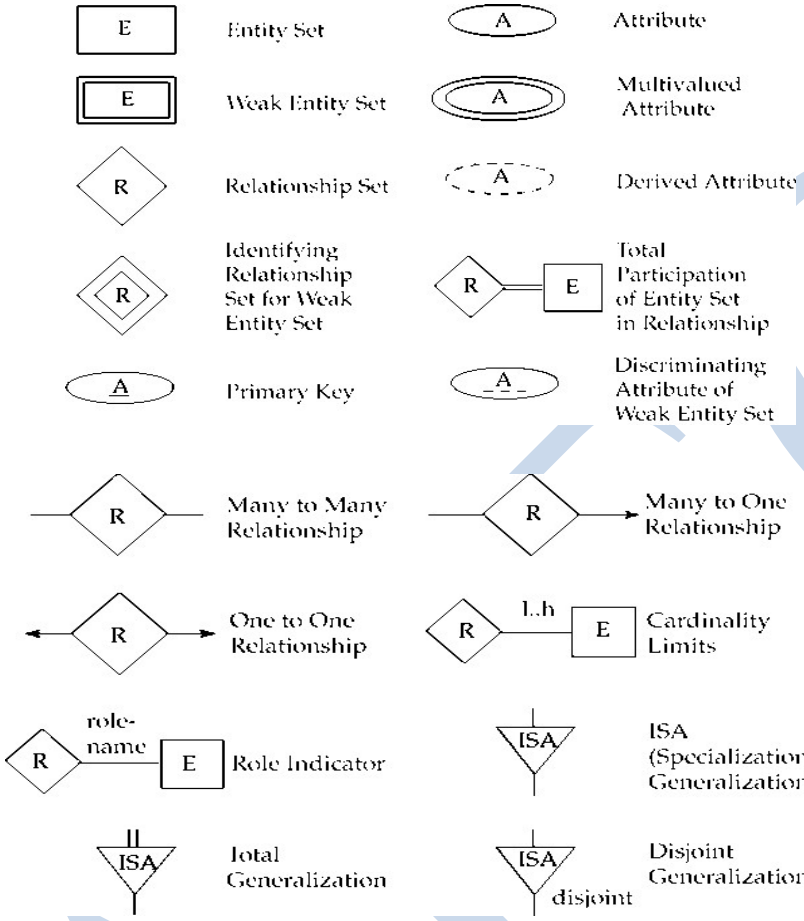
Schema in the entity-relationship model

- **A database can be modeled as:**
  - **a collection of entities,**
  - **relationship among entities.**
- **An entity is an object that exists and is distinguishable from other objects.**
  - **Example: specific person, company, event, plant**
- **Entities have attributes**
  - **Example: people have names and addresses**
- **An entity set is a set of entities of the same type that share the same properties.**
  - **Example: set of all persons, companies, trees, holidays**

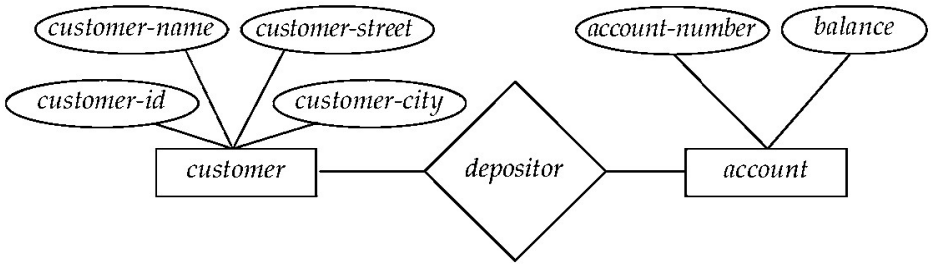


1.9.1 E-R Diagrams

Symbols Used in E-R Notation



Example:



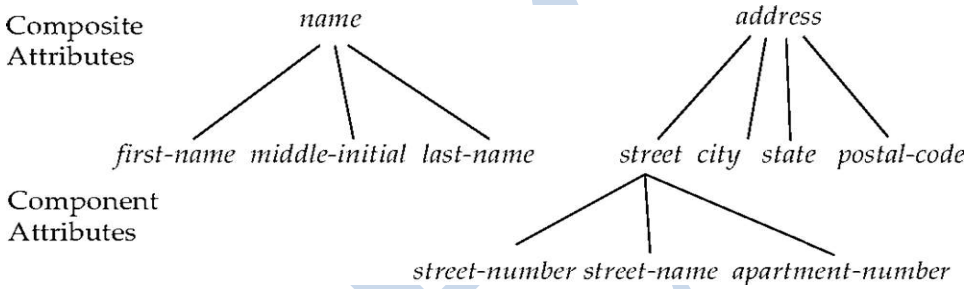
Entity-Relationship

Entity Sets

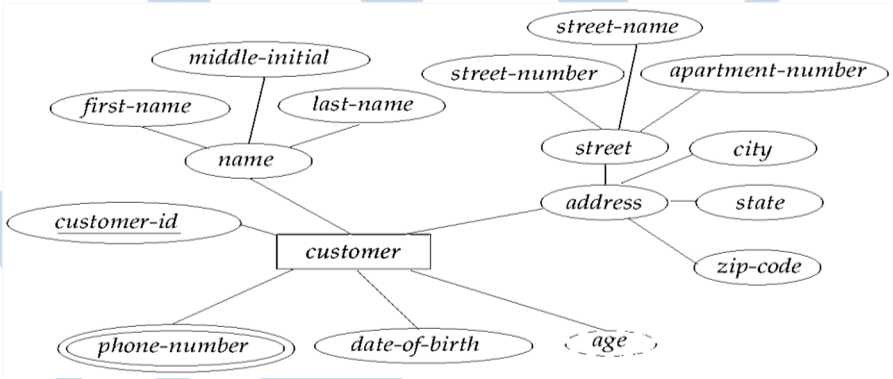
- **Rectangles** represent entity sets.
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets and entity sets to relationship sets.
- **Ellipses** represent attributes
- **Double ellipses** represent multivalued attributes.
- **Dashed ellipses** denote derived attributes.
- **Underline** indicates primary key attributes

1.9.3 ATTRIBUTES

- **An entity is represented by a set of attributes, which are descriptive properties possessed by all members of an entity set.**
- **Domain – the set of permitted values for each attribute**
- **Attribute types:**
  - **Simple and composite attributes.**
  - **Single-valued and multi-valued attributes**
    - **E.g. multivalued attribute: phone-numbers**
  - **Derived attributes**
    - **Can be computed from other attributes**
    - **E.g. age, given date of birth**
  - Composite Attributes



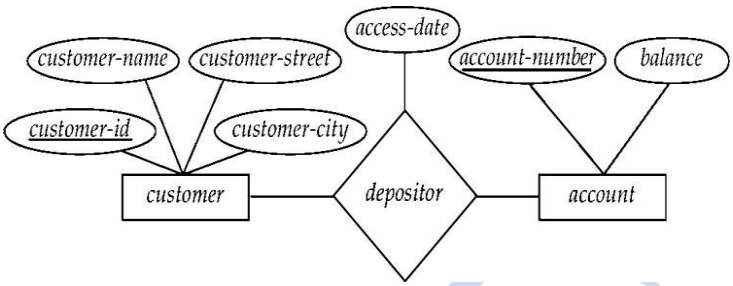
E-R Diagram with Composite, Multivalued, and Derived Attributes



1.9.4.1 Degree of a Relationship Set

- **Refers to number of entity sets that participate in a relationship set.**
- **Relationship sets that involve two entity sets are binary (or degree two). Generally, most relationship sets in a database system are binary.**
- **Relationship sets may involve more than two entity sets.**
- **Relationships between more than two entity sets are rare. Most relationships are binary.**

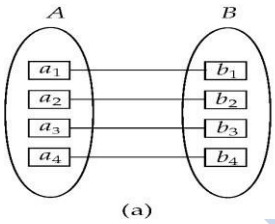
1.9.4.2 Relationship Sets with Attributes

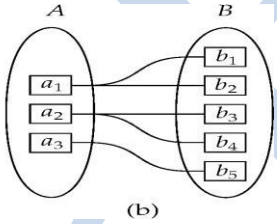


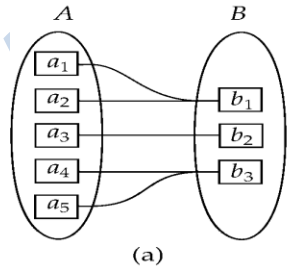
9.5 Mapping Cardinalities

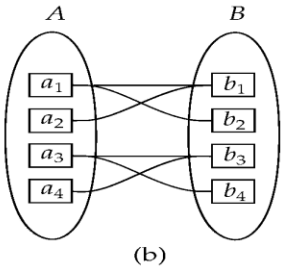
- **Express the number of entities to which another entity can be associated via a relationship set.**
- **Most useful in describing binary relationship sets.**
- **For a binary relationship set the mapping cardinality must be one of the following types:**
  - **One to one**
  - **One to many**
  - **Many to one**
  - **Many to many**

one to one **many to many**



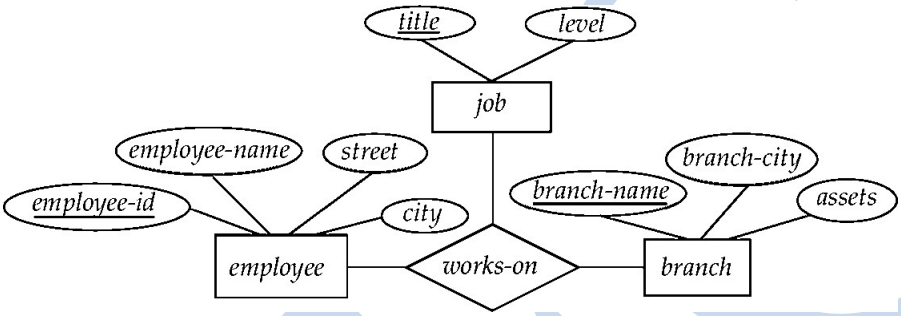






Many to One **Many to Many**

1.9.2 E-R Diagram with a Ternary Relationship



1.9.3 Cardinality Constraints on Ternary Relationship

- **We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint**
- **E.g. an arrow from works-on to job indicates each employee works on at most one job at any branch.**
- **If there is more than one arrow, there are two ways of defining the meaning.**
  - **E.g a ternary relationship R between A, B and C with arrows to B and C could mean**
  - **1. each A entity is associated with a unique entity from B and C or**
  - **2. each pair of entities from (A, B) is associated with a unique C entity, and each pair (A, C) is associated with a unique B**
  - **Each alternative has been used in different formalisms**
  - **To avoid confusion we outlaw more than one arrow**

1.9.4 Binary Vs. Non-Binary Relationships

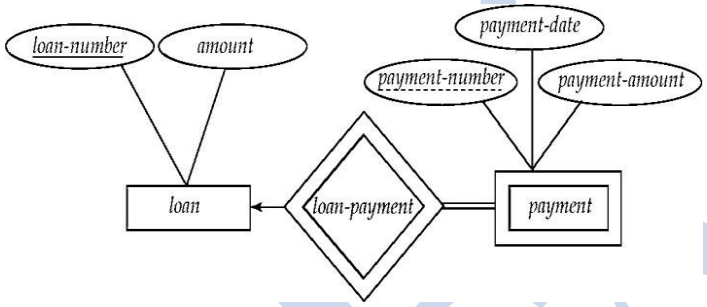
- **Some relationships that appear to be non-binary may be better represented using binary relationships**
  - **E.g. A ternary relationship parents, relating a child to his/her father and mother, is best replaced by two binary relationships, father and mother**
    - **Using two binary relationships allows partial information (e.g. only mother being know)**
  - **But there are some relationships that are naturally non-binary**
    - **E.g. works-on**

1.9.7 Weak Entity Sets

- **An entity set that does not have a primary key is referred to as a weak entity set.**
- **The existence of a weak entity set depends on the existence of a identifying entity set**

- it must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
- Identifying relationship depicted using a double diamond

- The discriminator (or partial key) of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.
- We depict a weak entity set by double rectangles.
- We underline the discriminator of a weak entity set with a dashed line.
- payment-number – discriminator of the payment entity set
- Primary key for payment – (loan-number, payment-number)

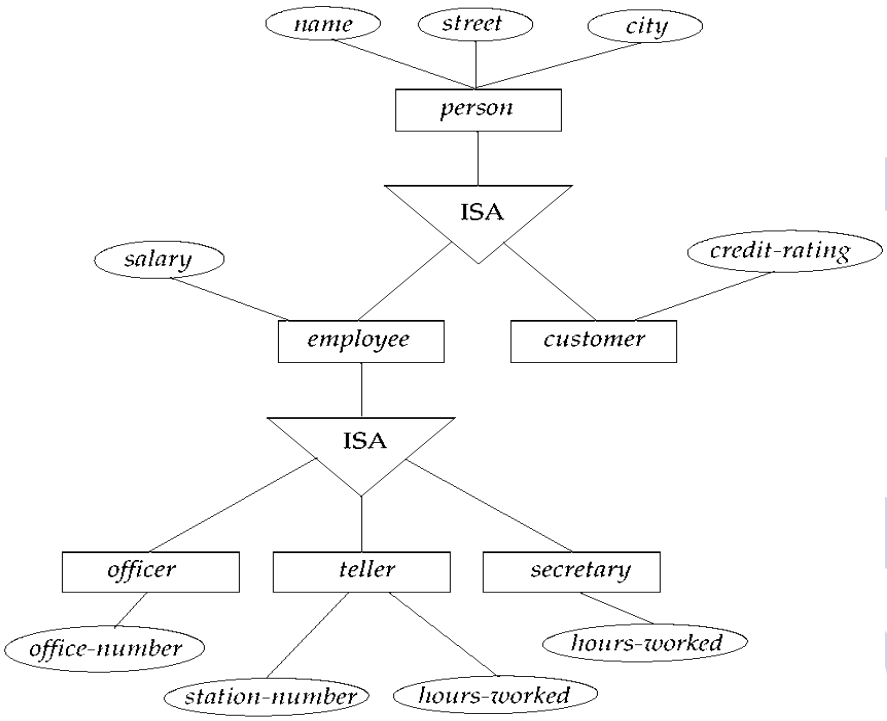


- Note: the primary key of the strong entity set is not explicitly stored with the weak entity set, since it is implicit in the identifying relationship.If loan-number were explicitly stored, payment could be made a strong entity, but then the relationship between payment and loan would be duplicated by an implicit relationship defined by the attribute loan-number common to payment and loan

1.10 Specialization

- Top-down design process; we designate subgroupings within an entity set that are distinctive from other entities in the set.
- These subgroupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a triangle component labeled ISA (E.g. customer "is a" person).
- Attribute inheritance – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

1.10.1 Specialization Example

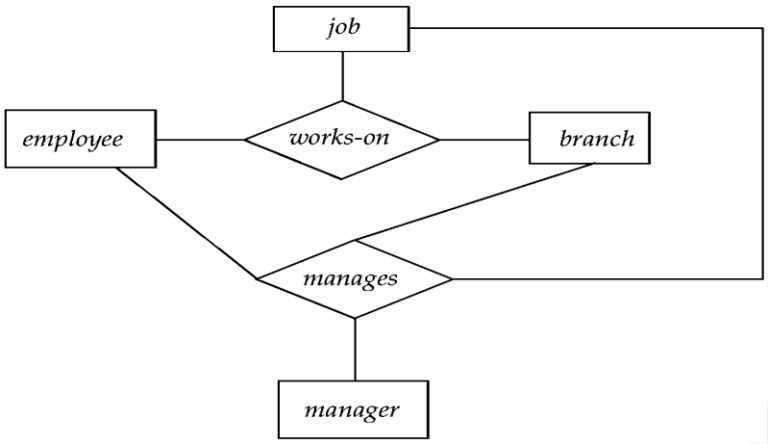


1.9 Generalization

- **A bottom-up design process – combine a number of entity sets that share the same features into a higher-level entity set.**
- **Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.**
- **The terms specialization and generalization are used interchangeably.**
- **Can have multiple specializations of an entity set based on different features.**
- **E.g. permanent-employee vs. temporary-employee, in addition to officer vs. secretary vs. teller**
- **Each particular employee would be**
  - **a member of one of permanent-employee or temporary-employee,**
  - **and also a member of one of officer, secretary, or teller**
- **The ISA relationship also referred to as superclass - subclass relationship**

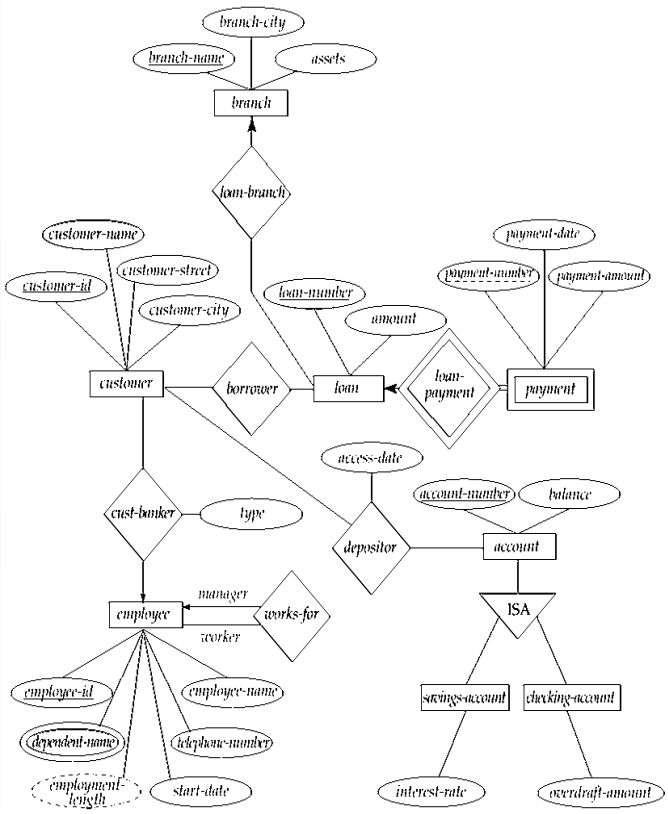
1.12 Aggregation

- **Relationship sets works-on and manages represent overlapping information**
  - **Every manages relationship corresponds to a works-on relationship**
  - **However, some works-on relationships may not correspond to any manages relationships**
    - **So we can't discard the works-on relationship**



- **Eliminate this redundancy via aggregation**
  - **Treat relationship as an abstract entity**
  - **Allows relationships between relationships**
  - **Abstraction of relationship into new entity**
- **Without introducing redundancy, the following diagram represents:**
  - **An employee works on a particular job at a particular branch**
  - **An employee, branch, job combination may have an associated manager**

1.14 E-R Diagram for a Banking Enterprise



1.15 Alternative E-R Notations

DATABASE DESIGN

3.1 Relational Database Design

The goal of the relational database design is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, and also retrieve information easily. For generating the efficient relation schemas we need to normalize the relation.

The bad database design may lead to:

- Repetition of information.
- Inability to represent certain information.

The design goals of relational database design are:

- Avoid redundant data.
- Ensure that relationships among attributes are represented.
- Facilitate the checking of updates for violation of database integrity constraints.

Example:

Consider following lending schema:

Lending_schema = {branchname, branchcity, assets, Custname, Loanno, amount}

Branchname	Branchcity	Assets	Custname	Loanno	amount
Downtown	Brooklyn	90000	Jones	L-17	1000
Redwood	Palo alto	21000	Smith	L-23	2000
Perryridge	Horseneck	90000	Jackson	L-14	1500
Mianus	Horseneck	80000	Jones	L-11	900
north town	Rye	37000	Hayes	L-16	1300

Suppose that we wish to add a new loan to our database. Say that the loan is made by the Perryridge branch to Adams in the amount of $1500. Let the Loanno be L-31. In our design, we need a tuple with values on all the attributes of Lending_schema. Thus, we must repeat the asset and city data for the Perryridge branch, and must add the tuple (Perryridge, Horseneck, 90000, Adams, L-31, 1500) to the Lending relation.

The repetition of information is undesirable. Repeating information wastes space. It also complicates updating the database. Suppose, for example, that the assets of the Perryridge branch change from 90000 to 100000 then many tuples of the Lending relation need to be changed. Thus, updates are more costly under the alternative design than under the original design.

3.2 Functional Dependencies

Functional dependencies play a key role in functional good database designs from bad database designs. A functional dependency is a type of constraint that is a generalization of the notion of key.

3.2.1 Definition of functional dependency

Consider a relation schema R, and let $\alpha \subseteq R$, and $\beta \subseteq \alpha$. The functional dependency

$$\alpha \rightarrow \beta$$

holds on schema R, if in any legal relation r(R), for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, it is also the case that $t_1[\beta] = t_2[\beta]$.

Using the functional dependency notation, we say that K is a superkey of R if $K \rightarrow R$. That is, K is a superkey if, whenever $t_1[K] = t_2[K]$, it is also the case that $t_1[R] = t_2[R]$ (i.e. $t_1 = t_2$).

Consider the schema:

Loan_info = (Loanno, Branch-name, Customer-name, Amount)

The set of functional dependencies that hold on this relation are:

Loanno→Branch-name

Loanno→Amount

There is no functional dependency;

Loanno→Custname

as a given loan can be made to more than one customers.

Use of the functional dependencies

- To test relations to see whether they ar1 legal under a given set of functional dependencies. If a relation r is legal under a set F of functional dependencies, we say that r satisfies F.
- To specify constraints on the set of legal relations.IfwewishtoconstrainourselvestorelationsonschemaRthatsatisfy set Functional dependencies, we say that F holds on K.

Let R be a relation on the relation scheme R. Then R satisfies the functional dependency X→Y if a given set of values for each attribute in X uniquely determines each of the values of the attributes in X→Y is said to be functionally dependent on X. The functional dependency FD is denoted as X→Y, where X is the left hand side or the determinant of the FD and Y is the right hand side of the FD.

A functional dependency X→Y is said to be trivial if $Y \subseteq X$.

In order to verify if a given FD X→Y is satisfied by a relation R on a relation scheme R we find any two tuples with the same X value; If the FD X→Y is satisfied in R then the Y values in thesetuplesmustbethesame.Werepeatthisprocedureuntilallpairsoftupleswith. ThesameXvalueareexamined.Anotherapproachinvolves ordering the tuples of n on the X values so that all tuples with the same Xvalues are together. Then it is easy to verify if the corresponding Y values are also thesame and verify if R satisfies the FD X→Y.

Schedule Relation:

Prof	Course	Room	Max_erollment	Day	Time
Smith	353	A532	40	Mon	11:45
Smith	353	A532	40	Wed	11:45
Clark	355	H940	300	Tue	1:45
Clark	355	H940	300	Thu	2:10
Turner	456	B278	45	Mon	3:45
Turner	456	B278	45	Wed	4:50

For the above relation, the FD Course→Prof is satisfied. However, the FD Prof→Course is not satisfied, as one professor can teach to more than one course.

3.2.2 Closure of a Set of Functional Dependencies

The set of functional dependencies that is logically implied by F is called the closure of F and is written as F^+.

Definition:

If F is a set of FDs on a relation scheme R, then F^+, the closure of F, is the smallest set of FDs such that $F^+ \supseteq F$ and no FD can be derived from F by using the inference axioms that are not contained in F^+.

Suppose we are given a relation scheme R = (A, B, C, G, H, I) and the set of functional dependencies:

A → B
A → C
CG → H
CG → I
B → H
The functional dependency
A → H is logically implied.

Proof:

A → B and B → H
⇨ A – H

Given F, we can compute F^+ directly from the formal definition of functional dependency. If F is large, this process would be lengthy.
Axioms, or rules of inference, provide a simpler technique for reasoning about functional dependencies.

Armstrong's axioms

We can use following three rules to find logically implied functional dependencies
This collection of rules is called Armstrong's axioms in honor of the Person who first proposed it.

i) Reflexivity Rule : If α is set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.
ii) Augmentation Rule : If $\alpha \rightarrow \beta$ holds and r is a set of attributes, then $r\alpha \rightarrow r\beta$ holds.
iii) Transitivity Rule : If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow r$ holds, then $\alpha \rightarrow r$ holds.

Armstrong's axioms are sound, because they do not generate any incorrect functional dependencies. They are complete, because for a given set F of functional dependencies they allow us to generate all F^+. Although Armstrong's axioms are complete, it is tiresome to use them directly for the computation of F^+ .1o simplify further, we list additional rules.

i) **Union rule** : If $\alpha \rightarrow \beta$ holds, and $\alpha \rightarrow r$ holds, then $\alpha \rightarrow \beta r$ holds.
ii) **Decomposition rule** : If $\alpha \rightarrow \beta r$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow r$ holds.
iii) **Pseudo transitivity rule** : If $\alpha \rightarrow \beta$ holds and $r\beta \rightarrow \delta$ holds, then $\alpha r \rightarrow \delta$ holds.

Consider schema:
R = (A, B, C, G, H, I), and the set of FDs (A → B, A → C, CG → H, CG → I, B →H)
Some of members of F^+ are:

❖ **A → H:**
Proof: A → B and B → H
⇨ By transitivity rule.
A → H

❖ **CC → HI:**
Proof: CG → H and CG → I
⇨ By union rule
CG → HI

❖ **AG → I:**
Proof:
A → C and CG → I
⇨ By psuedotransityvity rule
AG → I
(OR)
A → C
⇨ By augmentation rule
AG → CG and CG → I
⇨ By transitivity rule
AG → I

3.2.3 Canonical Cover

Suppose that we have a set of FDs F on a relation schema. Whenever a user performs an update on the relation, the database system must ensure that the update does not violate any FDs. The system must roll back the update if it violates any functional dependencies in the set F.

We can reduce the effort spent in checking for violations by testing a simplified set of functional dependencies that has the same closure as the given set. Simplified set is easy to test. Any database that satisfies the simplified set of FDs will also satisfy the original set, and vice versa, since the two sets have the same closure. The simplified set can be constructed by using canonical cover.

A canonical cover F_c for F is a set of dependencies such that F logically implies all dependencies in F_c. and F_c logically implies all in F. F_c must have the following properties.

- No functional dependency in F_c contains an extraneous attribute.
- Each left side of functional dependency in F_c is unique. That is no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in F_c such that $\alpha_1 = \alpha_2$.

An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies. The formal definition of extraneous attributes is as follows:

Consider a set F of functional dependencies and the functional dependency $a \rightarrow \beta$ in F.
Attributes a is extraneous in $A \in \alpha$, and F logically implies

$$(F - \{a \rightarrow \beta\} \cup \{(\alpha - A) \rightarrow \beta\}$$

Attribute A is extraneous in β *if* $A \epsilon \beta$, and the set of functional dependencies $(F - \{a \rightarrow \beta\})$ U $\{\alpha \rightarrow (\beta - A)\}$ logically implies F.

For example, suppose we have the FDs AB → C and A → C in F. Then, B is extraneous in AB → C.

Another example is, suppose the set F contains FDs AB → CD and A → C. Then C would be extraneous in AB → CD.

Canonical cover for a set of functional dependencies F can be constructed as follows:

$F_c = F$

repeat

Use the union rule to replace any dependencies in F_c of the form $a\ 1 \rightarrow \beta 1$ *and* $\alpha 1 \rightarrow \beta 2$ *with* $\alpha 1 \rightarrow \beta 1 \beta 2$.

Find a FD $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in$a$ *or in* β.

If an extraneous attribute is found, delete it from$a \rightarrow \beta$.

Until F_c does not change.

Computing canonical cover

Example 2: consider the following set F of functional dependencies on schema (A, B,C)

A → BC
B → C
A →B
AB →C

The canonical cover for F is computed as follows:

There are two FDs with the same set of attributes on the left side of the arrow:

A → BC
A → B

Combine these FDs into A → BC.

A is extraneous attribute in AB →C because F logically implies$(F - \{AB \rightarrow C\})$ U $\{B \rightarrow C\}$. This assertion is true because B → C is already in our set of FDs.

C is extraneous in A → BC because F logically implies $(F - \{A \rightarrow BC\})$ U $\{A \rightarrow B\}$. This assertion is true because A→B is already in our set of FDs.

Thus, our canonical cover is,

A → B
B → C

3.3 Data Redundancy and Update Anomalies

A major aim of relational database design is to group attributes into relations to minimize data redundancy and thereby reduce the file storage space required by the implemented base relations. Consider following three relations:

⇨ Staff (Staff_no, Sname, Position, Salary, Branch_no)
⇨ Branch (Branch_no, Baddress)
⇨ staffbranch(Staff_no, Sname, position, salary, Branch_no, Baddress)
The primary keys are underlined.

These relations contain the data given below:

1) **Staff**

Staff_no	Sname	Position	Salary	Branch_no
S1	John	Manager	10000	B005
S2	Ann	Assistant	5000	B003
S3	Susan	Supervisor	7000	B007
S4	Julie	Assistant	5000	B007
S5	Mary	Assistant	5000	B005

The Staff relation

2) **Branch**

Branch_no	Baddress
B005	22, Deccan, Pune
B003	10, Bandra, Mumbai
B007	32, Main st, Nashik

The Branch relation

3) **Staffbranch**

Staff_no	Sname	Position	Salary	Branch_no	Baddress
S1	John	Manager	10000	B005	22, Deccan, Pune
S2	Ann	Assistant	5000	B003	10, Bandra, Mumbai
S3	Susan	Supervisor	7000	B007	32, Main st, Nashik
S4	Julie	Assistant	5000	B007	32, Main st, Nashik
S5	Mary	Assistant	5000	B005	22, Deccan, Pune

The Staffbranch relation

In Staffbranch relation there is redundant data; branch address is repeated for every member of staff located at that branch. In contrast, the branch details appear only once for each branch in the Branch relation and only the Branch_no is repeated in the Staff relation.

Relations that have redundant data have problems called update anomalies, which are classified as: insertion, deletion, or modification anomalies.

1) Insertion anomalies

There are two main types of insertion anomalies:

i) To insert the details of new staff located at the branch number B007, we must enter the correct details of branch number B007 so that the branch details are consistent with values for branch number B007 in other tuples in Staffbranch relation. But Fig staff and branch does not suffer from this inconsistency.

ii) To insert details of new branch that currently has no staff into Staffbranch relation it is necessary to enter nulls into the attributes for Staff, such as Staff_no. But Staff_no is primary key of Staffbranch relation, and attempting to enter nulls for Staff_no violates entity integrity and is not allowed. The design of the relations shown in Fig. staff and branch relation avoids this problem because branch relation separately from the Staff details.

2) Deletion anomalies

If only one staff is working at a branch, and if that staff's information is deleted, then branch details are also lost, from the database. For example, if Staff_no S2's record is deleted, the branch information of Branch_no B003 is also lost. The design of the relations in Fig. staff and Branch relations avoids these problems, because Branch tuples are stored separately from Staff tuples and only attribute Branch_no relates the two relations.

3) Modification anomalies

If we want to change the value of one of the attributes of a particular branch in the Staffbranch relation, for example, the address for branch number B003, we must update the tuples of all staff located at that branch. If this modification is not carried out, the database will become inconsistent.

The above example illustrate that the Staff and Branch relations of Fig. staff and branch relations have more desirable properties than staffbranch. We can avoid update anomalies ofStaffbranch relation by decomposing the original relation into the Staff and Branchrelations.

3.4 Decomposition

We can avoid update anomalies by decomposition of the original relation.

The relation schema R is decomposed into relation schemas:

{R1, R2,...,Rn} in such a way that $R1 \cup R2 \cup \ldots Rn = R$

Example:

Consider following Lending schema:

Lending_schema = (Branch_name, Branch_city, Assets, customer-name, Loan_number,Amount)

The Lending_schema is decomposed into following two relations:

1) Branch_customer_schema = (Branch-name, Branch-city, Assets, Customer-name)

2) Customer_loan_schema:(Customer_narne, Loan_No, Amount)

The new relations Branch_customer_schema and Customer_loan-schema are constructed using the lending relation as given bellow:

1) Branch_customer = $\pi_{\text{Branch_name, Branch_city, Assets, customer_name}}$(Lending)

2) Customer_loan = $\pi_{\text{Customer_name, Loan_No, Amount}}$ (Lending)

Branch_name	Branch_city	Assets	Customer_name
Downtown	Brooklyn	90000	Jones
Redwood	Palo Alto	21000	Smith
Perryridge	Horseneck	90000	Jackson
Mianus	Horseneck	80000	Jones
Northtown	Rye	37000	Hayes

Branch_customer relation

Customer_name	Loan_No	Amount
Jones	L-17	10000
Smith	L-23	2000
Kackson	L-14	1500
Jones	L-11	900
Hayes	L-16	1300

Customer_loan relation

We can reconstruct the original lending schema using
Branch_customer ⋈ Customer_loan
The result of this join is shown below:

Branch_name	Branch_city	Assets	Customer_name	Loan_No	Amount
Downtown	Brooklyn	90000	Jones	L-17	1000
Downtown	Brooklyn	90000	Jones	L-11	9000
Redwood	Palo Alto	21000	Smith	L-23	2000
Perryridge	Horseneck	90000	Jackson	L-14	1500
Mianus	Horseneck	80000	Jones	L-11	900
Mianus	Horseneck	80000	Jones	L-17	1000
Northtown	Rye	37000	Hayes	L-16	1300

The relation Branch_customer ⋈ Customer_loan

1. If the relation Branch-customer ⋈ Customer_loan is compared with the originallending relation, we observe that:
   - Every tuple in lending relation is in Branch-customer ⋈ Customer_loanrelation.
   - The relation Branch_customer ⋈ Customer_loan contains followingadditional tuples:
     (Downtow4, Brooklyn, 90000, Jones, L-11, 900)
     (Mianus, Horseneck,80000, Jones, L-17, 1000)

2. Consider the query:
"Find all bank branches that have made a loan in an amount less than Rs. 1000"
   - The result of this query using the lending relation is: Mianus.
   - The result of the same query using the relation
     Branch-customer ⋈ Customer_loan is : Downtown and Mianus.

3. From the above two cases we c6rn say that the decomposition of lendingrelation into Branch_customer and Customer-loan results in bad databasedesign. It results in loss ofinformation and repetition of information.

4. Because of loss of information we call the decomposition of lending relation into Branch_customer and Customer_loan a *lossy*decomposition or a*lossy -join* decomposition.

3.5 Desirable Properties of Decomposition

There are two important properties associated with decomposition.

i) **Lossless join**:

This property ensures that any instance of the original relation can be identified from corresponding instances in the smaller relations.

ii) Dependency preservation:

This property ensures that a constraint on the original relation can be maintained by simply enforcing some constraint on each of the smaller relations.

3.5.1 Non-loss (Lossless-Join) Decomposition

- Let R be a relation schema, and let F be a set of functional dependencies on R. Let R1 and R2 form a decomposition of R. This decomposition is a lossless-join decomposition of R if at least one of the following functional dependencies is in F^+ :

$$R1 \cap R2 \rightarrow R1$$
$$R1 \cap R2 \rightarrow R2$$

- In other words, if R1 n R2 forms a super key of either R1 or R2, the decomposition of R is lossless-join decomposition.

Example 1:

- Consider following lending relation:

Lending_schema= (Branch_name, Branch_city, Assets, Customer_name, Loan_No, Amount)

- The decomposition of lending relation given below is lossless-join decomposition. It is described below:
  - ⇨ We now decompose the lending relation into two relations:
    1. Branch_schema: (Branch_name, Branch_city, Assets)
    2. Loan_info_schema = (Branch-name, Customer_name, Loan_No, Amount)
  - ⇨ Branch_schema ∩Loan_info_schema is Branch_name which is common attribute and

    Branch_name → Branch_city, Assets. (By augmentation rule)
    Branch_name → Branch_name, Branch_city, Assets
  - ⇨ This is nothing but a Branch_schema relation. Thus the first clause is satisfied. Hence, the decomposition of lending relation into:
    Branch_schema and **Loan_info_schema is lossless-join decomposition.**
  - ⇨ Next, we decompose the Loan_info_schema into:
    Loan_schema = (Loan_No, Branch_name, Amount)
    Borrower_schema = (Customer_name, Loan_No)
  - ⇨ This decomposition is also lossless-join decomposition since Loan_No is a common attribute and Loan_No → Amount, Branch_name.

3.5.2 Dependency Preservation

Given a relation schema R and set of functional dependencies associated with it is F. It is decomposed into the relation schemas R1, R2,....,Rn with functional dependencies F1, F2,..... Fn. Then the decomposition is dependency preserving if the closure of F' (where F' = F1 F2 …Fn) is identical to F^+ i.e.

$$F'^+ = F^+$$

Example:

Let R (A, B, C, D) and F^+ = (A → B, A → C, C → D). Show the decomposition of R is dependency preserving.

Solution: R is decomposed into

R1 = (A, B, C) with the FDs

F1 = (A → B, A → C)

R2 = (C, D) with FDs

F_2 = (C→D)

F' = F1UF2 .....UFn = (A → B, A → C, C →D)

Thus, $F'^+ = F^+$

Hence the decomposition is dependency preserving.

3.6 Normalization

Normalization is an essential part of database design. A good understanding of the semantics of data helps the designer to build efficient design using the concept of normalization.

3.6.1 Purpose of Normalization

- ⇨ Minimize redundancy in data.
- ⇨ Remove insert, delete and update anamoly during database activities.
- ⇨ Reduce the need to reorganize data when it is modified or enhanced.
- ⇨ Normalization reduces a complex user view to a set of small and stable subgroups of fields/relations. This process helps to design a logical data model known as conceptual data model.

3.6.2 Normalization Forms

Different normalization forms are:

1) First normal Form {1NF):

A relation is said to be in the first normal form if it is already in unnormalized form and it has no repeating group.

2) Second normal form (2NF):

A relation is said to be in the second normal form if it is already in the first normal form and it has no partial dependency.

3) Third normal form (3NF):

A relation is said to be in the third normal form ifit is already in second normal form and it has no transitive dependency.

4) Boyce-Codd normal form (BCNF):

A relation is said to be in Boyce-Coddnormal form if it is already in the third normal form and every determinant isa candidate key. It is a stronger version of 3NF.

5) Fourth normal form (4NF):

A relation is said to be in the fourth normal formif it is already in BCNF and it has no multivalued dependency.

5) Fifth normal form (5NF):

A relation is said to be in 5NF if it is already in3NF and it has nojoin dependency.

Terminologies used in normal forms:

The different terminologies used in various normal forms are explained below:

- ❖ **Partial dependency**

  In a relation having more than one key field, a subset of non-key fields maydepend on all the key fields but another subset/a particular non-key field maydepend on only one of the key fields (i.e. may not depend on all the keyfields). Such dependency is called partial dependency.

❖ **Transitive dependency**

In a relation, there may be dependency among non-key fields. Such dependency is called as transitive dependency.

❖ **Determinant**

A determinant is any field (simple field or composite field) on which some other field is fully functionally dependent.

❖ **Multivalued dependency**

Consider three fields X, Y and Z in a relation. If for each value of X, there is well-defined set of values of Y and a well-defined set of values of Z and the set of values of Y is independent of the set of values of Z, then multivalued dependency exists.
I.e. X →→Y/Z

❖ **Join dependency**

A relation which has a join dependency cannot be decomposed by projection into other relations without any difficulty and undesirable results.

3.7 First, Second, Third Normal Forms

Alpha Book House

Pune - 413001

Customer No. : 1052

Customer Name : Beta school of computer science

Address : Shivaji Nagar, Pune - 01.

ISBN	Book title	Author's name	Author's country	Qty	Price (Rs.)	Amount (Rs.)
81-203-5	DOS	P.K. Sinha	India	5	250	1250
0-112-6	DBMS	Korth	U.S.A.	6	300	1800
1-213-9	Simulation	Gordon	U.S.A.	5	100	500
					Grand Total	3550

Invoice of a book company

Normalization of invoice report upto 3NF

The above figure represented in a form of relation. The relation is named as invoice. This is in unnormalized form.

1) Invoice (Cust_no, Cust_Name, Cust_Add, (ISBN, Title, Author_Name, Author_Country, Oty, Unit_price))

The amount in the last column of the invoice for each book can be computed bymultiplying the number of copies (Qty) with respective unit price. Similarly, the grandtotal at the bottom of the invoice representing the invoice amount can be calculated bysumming the entries in the last column of the invoice. Similarly, the data at the topright comer of the invoice can be printed using the system clock. Hence, these threeentries are not included in the invoice relation.

3.8 First Normal Form (1NF)

The relation is in 1NF if it has no repeating groups.

In the invoice relation, the field in the inner most of parentheses put together is known as a repeating group. This will result in redundancy of data for the first three fields. This redundancy will

lead to inconsistency of data. Hence, it is divided into two sub-relations shown below to remove the repeating group.

2) Customer (Cust_No, Cust_Name, Cust_Add)

3) Customer_Book (Cust_No, ISBN, Title, Author_Name, Author_Country, Qty, Unit_price)

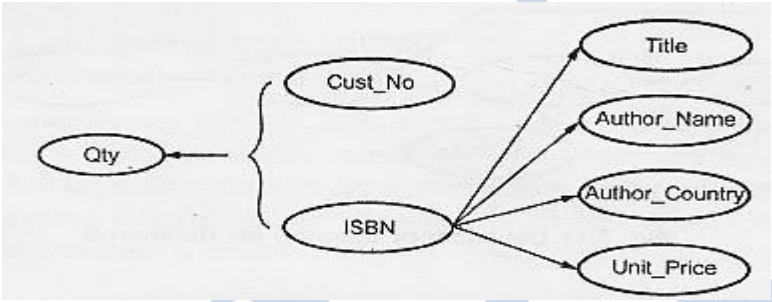
Now each of the above relations (i.e. relation 2 and relation 3) is in 1NF.

3.9 Second Normal Form (2NF)

2NF removes partial dependency among attributes of relation. In Relation 2, the number of key fields is only one and hence there is no scope for partial dependency.

The absence of partial dependency in Relation 2 takes it into 2NF without any modification.

In Relation 3, the number of key fields is two. The dependency diagram of Relation 3 is shown below.



Dependency diagram of relation3

The above diagram, Qty depends on Cust_No and ISBN, but the remaining non-key fields(Title, Author_Name, Author_Country, and Unit_price) depend only on ISBN. Thus, there exists partial dependency.

The existence of partial dependency will result into insertion update, and deletion anomaly.

Insertion anomaly

In Relation 3, if we want to insert data of a new book (ISBN, Title, Author_Name, Author_Country, Unit-price) there must be at least one customer. This means that the data of a new book can be entered into Relation 3 only when the first customer buys the book.

Update anomaly

In Relation 3, if we want to change any of the non-key fields, like Title, Author_Name, it will result into inconsistency because the same is available in more than one record.

Deletion anomaly

In Relation 3, if book is purchased by only one customer, then the book data will be lost when we delete that record after fully satisfying that customer order.

Hence, Relation 3 is divided into two relations as shown below:

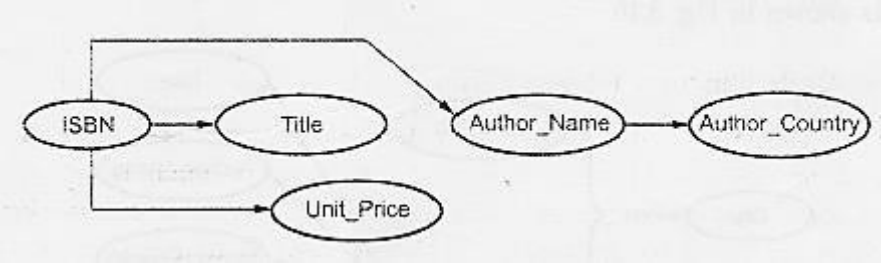
4) Sales (Cust_No, ISBN, Oty)

5) Book-Author (ISBN, Title. Author-Name, Author-Country, Unit-Price)

Above two relations (i.e. Relation 4 and Relation 5) are now in 2NF.

3.10 Third Normal Form (3NF)

Third normal form removes transitive dependency among attributes of relation.



In Relation 4, there is only one non-key field. So, there is no question of dependency between non-key fields. Thus, there is no transitive dependency. Hence Relation 4 is in 3NF.

In Relation 2, there is no dependency between the non-key fields. This means that ii has no transitive dependency. Hence, Relation 2 is also in 3NF.

In Relation 5, authors' country depends on author name. This means that Relation 5 has transitive dependency.

The existence of transitive dependency will result into insert, update and delete anomaly:

Insertion anomaly

Consider the book company has resident authors who are in the process ofdeveloping new books; it will be difficult to include the author's details inRelation 5. This means that there should be at least one published book toinsert the details of a resident author.

Update anomaly

If author's country is to be modified, then it is necessary to modify number of tuples as the same data is in number of tuples.

Deletion anomaly

If the only one book of a resident author is not reprinted, then the respective author's data will be lost. Hence, to overcome all these anomalies, Relation 5 is subdivided into two relations:

6) Book (ISBN, Title, Unit_price, Author_Name)

7) Author (Author_Name, Author_Country)

Le Relation 6, Author-Name is underlined with dotted line to indicate that it is a foreign key.

3.11 Boyce-Codd Normal Form (BCNF)

BCNF is more rigorous form of 3NF" It deals with relational tables which has multiple candidate keys, composite candidate keys, and candidate keys that overlapped, BCNF is based on the concept of determinant" A determinant is a column on which some of the columns are fully functional dependent. A relational table is in BCNF if every determinant is candidate key.

Case 1: Multiple candidate keys

Consider following relational table

A#	Aname	Aqualification	Astatus	TitleId	Royalty
100	Arora	Ph.D	10	T1	3000
110	Sharma	M.Tech	20	T2	4000

A relational table

Suppose author's status (Astatus) depends on his/her qualification. If author is Ph.D his status (Astatus) is 10, if M.Tech it is 20 and so on. This relation has three determinants: A#, TitleId, Aqualification. But only (A#, Titleld) combination is a candidate key; so this relation is not in BCNF. For a relation to be in BCNF each determinant must be a candidate key.

Case 2: Composite candidate keys

Suppose there are three relations as given below:

A#	TitleId	Royalty
A1	T1	5000
A2	T2	7000

Author_title table

A#	Aname	Aqualification
A1	John	Ph.D
A2	Tom	M.Tech

Author table

Aqualification	Astatus
Ph.D	10
M.Tech	20
B.Tech	30
Others	40

Author_qualification_status

- ⇨ Author_title table has one determinant (A#, Titleld). It is also a candidate key so the relation is in BCNF.
- ⇨ Author table has one determinant A# and it is also a candidate key so Author is in BCNF.
- ⇨ Similarly Author_qualification_status is also in BCNF.

Case 3: Candidate keys that overlapped

Consider following relation

A#	Aname	TitleId	Royalty
A1	john	T1	3000
A2	Tom	T2	4000

Author table

Suppose that each Author's name is unique. Then in above relation, the keys are: (A#, Titleld), and (ANarne, Titleld).

These are two determinants A# and Aname. The attributes of each possible relation which we can make out of the original relation(A#, Aname, Titleld, Royalty) must depend on these two determinants (A#,Aname).

But the attributes of relation (A#, Aname) does not depend on the determinant sinceeach A# and AName is independent of the other.

Hence the relation Author is not inBCNF. But this relation is in Third Normal Form. So a relation which is in ThirdNormal Form need not be in BCNF, but the converse is hue. So BCNF is strongerthan the 3NF.

- **BCNF decomposition algorithm**

```
result : = {R}
done : = false;
compute F+;
while (not done) do
  if (there is a schema Ri in result that is not in BCNF} then
  begin
     let α → β be a nontrival FD that holds on Ri such that α → Ri is not in F+
     and α ∩ β = ϕ;
     result : = (result – Ri) ∪ (Ri – β) ∪ (α, β);
  end
else done : = true;
```

- **Comparison of BCNF and 3NF**

1) 3NF design is always dependency preserving and lossless.
   Dependency preserving is difficult to achieve in BCNF sometimes.
2) BCNF strictly removes transitive dependency.
3) BCNF relation is in 3NF, but reverse is not possible.

3.12 Multivalued Dependencies and Fourth Normal Form

Consider an unnormalized relation that contains information about vehicles, dealers and models. Each tuple contains vehicle name/ dealer's name repeated for each vehicle name and Model vehicle repeated for each dealer name.

Vehicle	Dealer	Mode
Maruti car	Competent Aganil Link	Maruti 800 Baleno
Scooter	ABC	Bajaj LML

Unnormalized data

When we say dealer's name is repeated over Vehicle name, it means each dealer: Competent, Aganil or Link Motor can sell cars of all the indicated models.

Here we make an observation: Dealer and Model are independent of each other, i.e. it is immaterial which dealer sells a particular vehicle.

First, we convert this unnormalized relation into normalized one as there is no functional dependency. Recall the definition of functional dependency which says an attribute B of relation R is functional Dependent on attribute A of R if there exists one value of B of for a given value of A. Here there are many values of Dealer (and Model) for a given vehicle there is no functional dependency in the data.

We rewrite the unnormalized data of Table 3.1 after "flattering" the data as shown below.

Vehicle	Dealer	Model
Maruti	Competent	Maruti 800

Maruti	Competent	Baleno
Maruti	Aganil	Maruti 800
Maruti	Aganil	Baleno
Maruti	Link motor	Maruti 800
Maruti	Link motor	Baleno
Scooter	ABC	Bajaj
Scooter	ABC	LML

Flattened data

Above relation involves a lot of redundancy, leading to problems for insert, update and delete operations. To normalize the unnormalized relations we have to use the principle of projection. We use the same principle to obtain two projections: **Vehicle-Dealer and Vehicle-Model.**

Vehicle	Dealer
Maruti car	Competent
Maruti car	Aganil
Maruti car	Link motor
Scooter	ABC
Scooter	ABC

`Vehicle_Dealer

Vehicle	**Model**
Maruti car	Maruti car
Maruti car	Baleno
Scooter	Bajaj
scooter	LML

Vehicle_Model

Here decomposition of Table 3.2 is made not on the basis of dependencies; but on the basis of multi-valued dependency (MVD).

Vehicle_Dealer_Model.Vehicle→→Vehicle_Dealer_Model. Dealer

Vehicle_Dealer_Model.Vehicle→→Vehicle_Dealer_Model. Model

Consider the first MVD :

Vehicle_Dealer_Model.Vehicle →→ Vehicle_Dealer_Model. Dealer

Dealer is functionally dependent on Vehicle, i.e. for each vehicle there is a well-defined set of dealers.

For a Vehicle (say Maruti Car) and a Model (Maruti 800) the set of Dealers depend only on the Vehicle (Maruti) and it is independent of the value Model But the only condition is that Dealer and Model must hppear together in some tuple of relation Vehicle_Dealer_Model.

Therefore, we say the set of Dealers-values matching a pair of (vehicle_values, Model_values) in relation (Vehicle-dealer-Mode1) depends only on Vehicle values and it is independent of Model_values.

Consider the second MVD:

Vehicle_Dealer_Model. Vehicle→→Vehicle_Dealer_Model.Model

The interpretation of the second MVD is also similar' Model-value matching the pair (vehicle_values, Dealer_value) depends only on vehicle_values and it is independent ofDealer_value.The above discussion laid down the foundation for the definition ofMultivalued Dependency.

3.13 Join Dependencies and Fifth Normal Form

So far we have been normalizing the unnormalized data using the principledecomposition, i.e. by replacing the unnormalized relation into two of its projections. This principle of taking projection has worked very well until 4NF. There are some relations, which cannot be decomposed into two projections. But these relations can be decomposed into three projections, this is in 5NF.

Suppose there is a relation Dealer_Parts_Customer (DPC) as shown below which contains "all key", and it is in 4NF (i.e. there are no functional dependencies and Multi_Valued Dependencies).

Example

We decompose it in three projections as shown in Table 3.5 (a), 3.5 (b) and 3.5 whenwe join Table 3.5 (a), 3.5 (b) and Table 3.6 (a) and Table 3.5 (c), do we get back the original relation shown in Table Dealer_Parts_Customer (DPC) relation we get original relation: Dealer-Parts-Customer.If we join all three projections DP, PC and CD.

Dealer-Parts-Supplier is obtainable if all the three projections are joined. Two projections if joined do not give back the original Dealer-Parts-Customer relations. This is called join dependency.The statement that Dealer_Parts_Customer is equal to join of its projections Dealer_Parts, Parts_Customer and Customer_Dealer is tant amount to saying:

Dealer	Parts	Customer
D1	P1	C1
D1	P1	C2
D1	P2	C1
D2	P1	C1

Dealer_Parts_Customer (DPC) relation

If the pair <D1, P1> appears in Dealer_Parts
and the pair <P1, C1>, appears in Customer_Dealer
The pair <C1, P1, C1> appears in Customer_Dealer
then the hiple<D1, P1, C1> appears in Dealer_Parts_Customer.

Adding in instance that is not in 5NF gives spurious results when the table are decomposed and then joined.

Dealer	Parts
D1	P1
D1	P2
D2	P1

Table 3.5 (a) Dealer_parts (DP)

Part	Customer
P1	C1
P1	C2
P2	C1

Table 3.5 (b) Parts_customer (PC)

Customer	Dealer
C1	D1
C2	D1
C1	D2

Table 3.5 (c) Customer_dealer (CD)

Join over parts

Dealer	Parts	Customer
D1	P1	C1
D1	P1	C2
D1	P2	C1
D2	P1	C1
D2	P1	C2

Extra tuple

Join over customer and dealer

Table 3.6 (a) DP-PC

Dealer	Parts	Customer
D1	P1	C1
D1	P2	C1
D1	P1	C2
D2	P1	C1

Table 3.6 (b) Join DP-PC with CD over Customer and Dealer together