Contents

Key Points about Relational Models

1. Ordering of rows is not significant
2. Ordering of columns is not significant
3. Each row/column intersection contains a single attribute value
4. Each row in a table must be distinct

A table conforming to these restrictions is called a *normalised* table

Duplicated vs Redundant Data

- Must be careful to distinguish between redundant and duplicated data
- Duplictaed data occurs where an attribute (column) has two or more identical values
- Redundant data occurs is you can delete a value without information being lost

Redundancy is unnecessary duplication

Example

Part#	Part Desc
P1	bolt
P2	nut
P3	washer
P4	nut

nut is duplicated but **NOT** redundant

Part#	Part Desc
P1	bolt
P2	nut
P3	washer
P1	nut

nut is duplicated and redundant. No loss in information.

Eliminating Redundancy

- We cannot just delete values from the table in the previous example!
- Preferable to split table into 2 tables - parts and suppliers

Part#	Part Desc
P1	bolt
P6	bolt
P4	nut

S#	Part#
S2	P1
S7	P6
S2	P4
S5	P1

- Eliminated redundancy by table splitting
  - P1 description only appears once
  - Relationship is made by including part # in two tables
- So far we've assumed that table structures which permit redundancy can be recognised by inspection of table occurence
- This is *not entirely accurate* since attribute values are subject to insertion/change/deletion

S#	P#	Desc
S2	P1	bolt
S7	P6	bolt
S2	P4	nut

Inspection of table does not reveal any redundancy

Could even suggest that "no two suppliers may supply same part #"

Repeated Groups

- We stated earlier that "an attribute must only have one value in each row"

S#	SName	P#
S5	Wells	P1
S2	Heath	P1, P4
S7	Barron	P6
S9	Edwards	P8, P2, P6

Problems

1. Table is asymmetric representation of symmetical data
2. Rows can be sroted into s# but not into p#
3. Rows are different length because of variation in number of p#s
4. If rows were fixed length, they would need to be padded with null values

Elimination of Repeating Groups

- Easiest way to eliminate repeating groups is to write out the table occurrence using a vertical layout and fill in the blanks by duplicating the non-repeating data necessary
- But this can lead to 'redundency' of information

S#	Sname	P#
S5	Wells	P1
S2	Heath	P1
S2	Heath	P4
S7	Barron	P6
S9	Edwards	P8
S9	Edwards	P2
S9	Edwards	P6

Alternative Method

- Split table into two tables so that repeating group appears in one table and rest of attributes in another
- Need to provide correspondance between tables by including a key attribute the with repeating group table

S#	SName
S5	Wells
S2	Heath
S7	Barron
S9	Edwards

S#	P#
S5	P1
S2	P1
S2	P4
S7	P6
S9	P8
S9	P2
S9	P6

Eliminating Repeating groups and Redundancy

- Snapshot of table is inadequate guide to presence/absence of redundant data
- Need to know underlying rules
- DBA must discover rules which apply to conceptual model

Conclusion

- Its not possible to tell by looking at the relational tables in a DB to determine if
  - There is the potential for redundency
- But what would be a ‘correctly formed’ table?

Codd’s Normal Forms

- Codd identified some rules which govern the way we create tables so as to avoid anomolies when inserting or deleting values in these tables

- These rules are called *NORMAL forms*
- There are three and a half important levels (and two further levels which are occasionally used)

1. A relational is in first normal form is the domain of each attribute contains only atomic values and the value of each attribute contains only a single value from that domain
2. A relation is in second normal form if, in addition to satisfying the criteria for first normal form, every non-key column is *fully functionally dependent* on the entire primary key
3. A relation is in the third normal form if, in the addition to satisfying the criteria for second normal form, and no non-key attributes are *transitively dependency* upon the primary key

- Boyce Codd Normal Form (Also called thee and a half normal form)
  - "All attributes in a relation should be dependency on the key, the whole key and nothing but the key"

Determinants

- If there are rules such that duplicate values of attribute A are always associated with the *same* value of attribute B (within any given occurrence of the table) then attribute A is a *determinant* of attribute B

Note: In the special case where duplicate values of A are not allowed in a table (i.e. A is a key) then A is obviously a determinant on B

Example

- If each possible p# value has precisely one associated part description value (i.e. P4 has just one descrption nut then we can say that p# is a determinant of part description)
- Similarly, is each possible p# value has precisely one quantity in stock then we can say p# is a determinant of quantity in stock

Superfluous Attribute

- If P# determines Qty then composite attribute `{P#, P_Desc}` also determines Qty, but P_Desc is superfluous
- We assume determinants do not contain any superfluous attributes

Diagrams

- $A \to B$
  - A is determinant of B
- $A \leftrightarrow B$
  - A is determinant of B and vice versa

Transitive Determinants

- If A is determinant of B and B is determinant of C
- Then A is determinant of C

Identifiers

- Because of the rule that 'no two rows in a table can have identifical values throughout'
- Therefor individual row can always be identified by quoting the values of all its attributes
- However some values may not be needed

Example

`Employee(Employee#, Employee_name, Salary)`

Rules

- No two rows should have the same value for Employee#
  - Employee# is a row identifier of the table
- Where a composed attribute forms the identifier
  - No component part (of identifier) can be null (entity constraint)

Determinancy and Redundancy

- Given a determinancy diagram (developed from enterprise rules) we can detect and eliminate table structures which could contain redundant data

Customer# → Salesman# → Saleman name

Customer# is a determinant and identifier. Saleman# is a determinant only.

- Each customer# is associated with one salesman# but a salesman# may be associated with several different customer#
- Therefore salesman# could have ducplicate values
- But salesman# is a determinant of salesman name
- Therefore each occurrence of a duplicate salesman# values will be associated with the same salesman name
  - Table can contain redundant values of salesman
- But customer# values cannot be duplicated (because customer# is the identifier for our table) so we cannot allow redundant values of salesman#
- Potential redundancy arises because salesman# is a determinant bu not a candidiate identifier

Transforming Tables in Well-Normalised Table

- Boyce Codd rule for determining redundancy is rule "Every determinant must be a candidate identifier"
- A table which obeys this rule is said to be in Boyce Codd normal form (BCNF)

To put it another way, "all attributes in a relation should be dependency on the key, the whole key and nothing but the key"

- A determinant which is not a candidate identifier is called a *non identifying determinant*
- To transform a badly normalised table into well normalised tables:
  - Create new tables such that each non identifying determinant in the old table becomes a candidate identifier in a new table

Fully Normalised Tables

- Fully normalised tables are structured in such a way that they cannot contain redundant data
- Generally a well normalised table (i.e. one in which each determinant s a candidate identifier) is also fully normalised, but not always!
  - So further normalisation may be desirable

Example

A database storing books has the following rules

- Each book has a unique book#
- Each author has a unique author#
- Every author has a name and every book has a title
- Each subject classification has a unique subject name
- Book# does not distinguish an indivudual copy of a book, only an individual work
- A book may be written by several authors, and be classified under several subject names
- An author may write several books
- A subject name may be applied to several books

AUTHOR-BOOK

Author#	Book#
A2	B15
A5	B15
A2	B18

BOOK-SUBJECT

Book#	Subject-Name
B15	Biology
B15	Physics
B18	Physics

Advantages of Full Normalisation

- So far emphases has been placed on eliminating redundancy
- Futher benefits relate to deletion, insertion operations

C#	CName	S#	SName
C1	Brown	S4	Jones

C#	CName	S#	SName
C2	Carter	S7	Samson
C3	Cross	S4	Jones
C4	Barns	S8	Baker

- Delete C2, delete whole tuple
  - Lose salesman information
- Deleting C# on its own is not allocated as it is an identifier and cannot be null

Insertion side effect

- Add S2 whose name is Hall
- You cannot do this until the salesman is associated with a customer, otherwise identifier C# will be null

Conceptual Design

- The process of *conceptual design* uses a high-level *conceptual data model* to create the *conceptual schema* for a database
  - In this case, the Relational Model
  - Part of the requirements analysis process
- Conceptual Schema
  - Concise description of the data requirements of the user, including descriptions of entity types, relationships and constraints
- The conceptual schema does not include any implementation details
  - Hence, can be used to communicate with non-technical users
  - Used as a reference to ensure all data requirements are met, and that there are no conflicts
- Part of physical and logical data independence

Entity Relationship Model

- An abstract and high-level conceptual representation of information

- Entity Relationship Diagrams
  - Diagrammatic Notation of the ER Model
- Used to support the conceptual design of databses and help procude the conceptual schema
- Describes data as entities, relationships and attributes

Entities

- The basic object that an ER diagram represents is an *entity*
- An entity is a real world object with an independent existence
  - Physical of conceptual
- Each entity has attributes which are the particular properties that describe the real world object

Attributes

- Several types of attributes occur and need to be modeled in the Entity-Relationship Model
  - *Simple* versus *composite*
  - *Single-valued* versus *multi-valued*
  - *Stored* versus *derived*

Simple and Composition Attributes

- Composite attributes can be decided into smaller sub-parts
  - Address
- Attributes that are not divisible are called simple, or atomic, attributes
  - Movie Certificate, Age…
- Composite attributes can be hierarchical
  - Apartment number, Building number, Street

Single and Multi-Valued Attributes

- Most attributes have a single attribute for each entity
  - PPS number, Age
  - Such attributes are called *single-valued*

- In some cases an attribute can have a set of values for an entity
  - Genre for a Movie, Colour for a Car
  - Some entities may have one value, others multiple
  - Such attributes are called multi-valued

Stored versus Derived Attributes

- In some cases, two or more attributes are related
  - Age and Birth Date
- Age can be calculated using today's date and a person's date of birth
  - Age is therefore called a derived attribute
  - It is said to be derivable from Birth Date
  - Birth Date is called a stored attribute
- Some attributes can be derived from information in related entities, rather than attributes
  - If a number of employees attribute was associated with a CINEMA entity
  - This could be derived by totaling the number of employee entries stored in the EMPLOYEE entity

Entity Types and Sets

- ER diagrams don't show singple instances of entities (or relations)
- They show *entity types*
  - An entity which is identified by its name and attributes
  - In a cinema database, MOVIE could be an entity type
  - all movie entities share the same attributes but each instance has its own values for each attribute
- The collection of all instances of a particular entity type in a database is called an *entity set*

Key Attributes

- Each entity type usually has one or more attributes whose values are unique for each instance in the entity set
  - An attribute whose values are used to uniquely identify each entity is called the *key attribute*
    * ISBN, PPS, Student Number

  - More than one attribute can be used to form the key
  - In this case the combination must be unique
    * Composite key attribute

- Specifying the key attribute places a uniqureness constraint on the entity type
  - This uniqueness constraint must hold for every instance of an entity in the entity set
- The key constraint is derived from the real world requirements that the database represents
- Some entity types have more than one key attribute

Relationships

- A *relationship* captures how two or more entity types are related to one another
- Whenever an attribute of an entity type refers to another entity type, a relationship exists
  - There are a number of implicit relationships in a movie example - SCREEN and THEATRE, SCREENING and MOVIE, SCREENING and SCREEN
  - In the ER model, these references should not be represented as attributes, but as relationships
- A Relationship can be informally thought of as a verb, linking two or more nouns from the world you are trying to model
  - A "manages" relationship between an employee and a department
  - A "performs" relationship between an artist and a song
  - A "bores" relationship between a lecturer and a student
  - A "proved" relationship between a mathematician and a theorem

Relationship Types and Sets

- As with entities, relationships have a *relationship type*, which is illustrated in an ER diagram
  - The collection of all instances of a particular relationship type in a database is called a *relationship set*
- Related entity types are said to *participate* in a relationship type
  - Each relationship instance, r_i, is an association of entities, where the association includes exactly one entity from each of the participating entity types

Binary and Tenary Relationships

- The degree of a relationship type is the number of entity types that participate
  - The SHOW relationship is of degree two
- Relationship types of degree two are called *binary*, relationship types of degree three are called *ternary*

Relationship Roles

- Each entity type that participates in a relationship type plays a particular role
- A role name can be optionally added to an ER diagram to clearly identify what the relationship means

Recursive Relationships

- In some cases the same entity type participates more than once in a relationship type, in different roles
  - Such relationships are called *resursive relationsips*
- In this case the relationship roles are key in distinguishing the role each participating entity plays

Relationship Constraints

- Constraints limit the possible combination of entities that can participate in a relationship
- These constraints are determined by the real world requirements that are being modeled
- Two main types of relationship constraint
  - Cardinality constraints
  - Participation constraints

Cardinality Constraints

- Specify the *maximum number of relationship instances that an entity can participate in
- Consider our SHOW relationship type
  - Cardinality ratio for MOVIE:SCREENINGS is 1:N

  - Each movie can be shown in many screenings
  - Each screening shows a maximum of one movie
- Possible cardinality ratios for binary relationships
  - 1:1 - One to one
  - 1:N - One to many
  - M:N - Many to many

Participation Constraints

- Specify the *minimum* number of relationship instances that an entity can participate in
- Two types of participation constraint
  - *Total participation*
  - *Partial participation*
- Formally, participation constraints specify if the existence of an entity depends on it being related to another entity via the relationship type
- Company policy states that every Employee MUST work for a department
- Then, an employee entity can only exist if they participate in at least one WORKS relationship between EMPLOYEE and DEPARTMENT
- The participation of EMPLOYEE in WORKS is called *total participation*
  - Every entity in the total set of employee eneities must be related to a department entity via the WORKS relationship type
- Total participation is also called *existence dependency*
- In the same company, every employee will not be responsible for managing a department
- Therefore, an employee entity can exist if they don't participate in the MANAGE relationship between EMPLOYEE and DEPARTMENT
- The participation of EMPLOYEE in MANAGE Is called *partial participation*
  - Some, but not all, of the total set of employee enetities are related to a department entityt via the MANAGE relationship type
- Together, cardinality consttraints and participation constraints are referred to as the *structual constraints* of a relationship type
- In ER diagrams, total paricipation is represented by a double line connecting the participating entity type to the relationship type
- Partial participation is represented by a single line

Relationship Type Attributes

- In the entity relationship model, relationship types can have attributes, similar to entity types

- These relationship type attributes can be migrated to participating entities for relationships with cardinality of 1:1 or 1:N
- In relationship types with 1:N cardinality ratios, the relationship attribute can only be migrated to the N side of the relationship
- In relationship types with N:M cardinality ratios, the attribute cannot be migrated and remains specified as a relationship attribute

Relationships as Attributes

- It is sometimes convenient to think of a relationship type in terms of attributes
  - As when initially modelling the entity types
- With binary relationships there are always two options for representing it as an attribute
  - Consider the “shows” relationship
  - We could use an attribute in Screening that refers to the Movie ID for the movie in question
  - Or, we could use a multi-valued attribute in movie that refers to all the svreenings of that movie

Weak Entity Types

- In ER diagrams, entity types which don’t have a key attribute are called *weak entity types*
- Entities of this type are identified by their relationship to specific entities from other entity types
  - This other entity type is called the identifying entity type
  - The relationship type that related a weak entity type to its identifier is called the identifying relationship
- A weak entity type always has a total participation constraints with relation to its identifying relationship
  - As a weak entity cannot be identified without its identifier entity