

COP3703 Intro to Databases

Exam 1 Notes

Furio. Digitized by Tobias Dault

October 13, 2022

Contents

1	Chapter 1	2
1.1	Why Databases?	2
1.2	Database Management System (DBMS)	2
1.3	Database System	2
1.4	Terminology/Example Database	2
1.5	Self-Describing Data Structure	3
1.6	Program-data Independence	3
1.7	Data Abstraction	3
1.8	Support for Multiple Views	3
1.9	Multi-user Transactions	3
1.10	Bad use Case	3
2	Chapter 2	4
2.1	Database Schema/Schemata	4
2.2	Database State	4
2.3	Design Goals	4
2.4	Brainstorming-Building a Databases	4
2.5	Three Schema Architecture (ANSI Architecture)	5
2.6	Internal Schema	5
2.7	Conceptual Schema	5
2.8	External Schema	5
2.9	Data Independence	5
2.10	DBMS Languages	5
2.11	DBMS	6
2.12	Evolution of Databases/DBMSs	6
3	Chapter 3	7
3.1	Basics of ER Diagrams	7
3.2	Entity Type	8
3.3	Exercise	8
3.4	Weak Entity Identification	9

4	Chapter 4	10
4.1	Extended Entity Relationship (EER) Diagrams	10
4.1.1	Super class -- > -- subclass	10
4.2	Specialization	10
4.3	Predicate vs. User Defined Subclasses	10
4.4	Disjoint vs. Overlapping	10
4.5	Complete (total) vs Partial	11
4.6	Specialization Hierarchy vs Lattice	11
4.7	UNION type	11
4.8	ERR Design Guidelines	11
5	Chapter 9	12
5.1	ER-to-Relation	12

1. Chapter 1

1.1. Why Databases?

- Goal of reliable info, storage, retrieval, and assurance at scale.
- Definition: A collection of data that is related and has an implicit meaning.
- **Implicit Properties**
 - represents "real-world" aspect (entities), where changes to this subset of the real world (UoD) show on the database.
 - data is logically coherent with purpose for users or programs.
- **Why databases > Files (in some cases)**
 - consider two offices that need similar data
 - we should give them a way to simultaneously access and update the same data, to avoid redundancies and wasted space.

1.2. Database Management System (DBMS)

- software for creation and maintenance of computer database.
- manages storage, queries, and data manipulation.
- this class uses Oracle DBMS

1.3. Database System

- The DBMS software as well as the actual data, and potentially other applications.
- Where you actually query, as well as what defines the "database definition" (metadata).

1.4. Terminology/Example Database

Name	Number	Class	Major
Tony	123	IT30	IT
Robert	124	CS20	CS

* one table in the database, also includes course, grades, pre-req tables, etc.*

- any row is a record
- any single item is an element
- all elements need a particular type, i.e. int or char.
- may need to combine info from multiple tables to satisfy a query
- queries provide information to the user, ideally useful.

1.5. Self-Describing Data Structure

- relations (the aforementioned tables) need to have their number of columns identified, and columns must have their type and associated relation specified.

1.6. Program-data Independence

- Allows changes to data structures and storage without changing DBMS access programs.
- Basically, API function access over direct access.

1.7. Data Abstraction

- conceptual data representation abstract details of data storage.

1.8. Support for Multiple Views

- Relations may be combined and presented in a relevant and useful way to users (i.e., GUIs for general works, hiding SSNs from non-HR).

1.9. Multi-user Transactions

- Transactions may be completed reliably, concurrency is accounted for (i.e. no double-booking).
- For any transaction in a complex system, there are likely to be many complex steps and queries.

1.10. Bad use Case

- Databases require extra hardware software, training, and implement costly features to help concurrency, ease of access.

2. Chapter 2

2.1. Database Schema/Schemata

- data/record structure
- NO TYPE INFO!
- no constraints/relations
- does not changes frequently (evolution)

2.2. Database State

- data of database at any given time is called its state, or a snapshot
- **Empty** - just the schema
- **Initial State** after first populated data
- state changes after each update.
- bad for static data, low-resource/embedded systems, real-time requirements, or single-user.

INSERT EXAMPLE HERE

2.3. Design Goals

- self describing
- insulation of programs and data
- multi-user access/view

2.4. Brainstorming-Building a Databases

- data is well labeled and related
- mechanism to prevent simultaneous modification
- buffer breaking connection ????? outside access and stored data

2.5. Three Schema Architecture (ANSI Architecture)

INSERT DIAGRAM

2.6. Internal Schema

- describes physical storage
- access paths
 - search structure using indices, trees, hashing, etc.
 - ideally optimized for common case.

2.7. Conceptual Schema

- hides details of physical storage
- introduces entities, attributes, relationships, to describe and model data more naturally.

2.8. External Schema

- provides different views as appropriate to various user type (access control)
- what process type of users are permitted to execute
- get form data

2.9. Data Independence

- **Logical data independence:** ability to change conceptual schema without change to external schema
- **Physics data independence:** change internal without change to conceptual schema

2.10. DBMS Languages

- Data Definition Language (DDL)
- Storage Definition Language (SDL)
- Data Manipulation Language (DML)
- in modern systems, all in one language but separately compiled.

2.11. DBMS

- common (keyboard, forms, speech to text) to less considered (embedded system, cameras)

2.12. Evolution of Databases/DBMSs





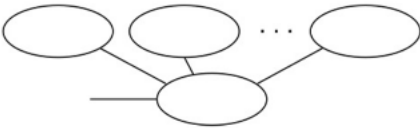

- intrinsically linked to growth of computers
- **Centralized:** early on, runs as a normal program on non-networked computer → singlepoint failure.
- **Two-tier:** client linked to server via network, but no buffer → no program/data insulation
- **Three-tier:** modern external/conceptual/internal design

3. Chapter 3

3.1. Basics of ER Diagrams

- **Entity:** something in the miniworld with an independent existence.
- **Attribute:** properties that describe the entity
 - Atomic: an attribute which cannot be further divided
 - Compound: an attribute consisting of other compound/atomic attributes (design choice)
 - single vs multivalued possible (i.e. dropdown vs checkboxes, also a design decision)
 - Stored (i.e. directly storing age) derived (i.e. getting age from birthday)
 - NULL values (interpret as N/A, missing, etc)
 - Complex attributes, such as nested, composite, or multivalued attributes.
 - Key Attribute Can ID each entity uniquely as efficiently as possible due to two properties.
 - * Uniqueness (within entity type)
 - * Minimality (i.e. why store name and n-number when n-number is already unique)

ER diagrams

Symbol	Meaning
	Entity
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute

3.2. Entity Type

- a name representing the set of all entities of the same type (always singular, all caps)
- **Intention/schema**: attribute types associated with entity
- **Extension/entity set**: all entities in the DB of this particular type.
- **n-tier architecture**: scale increases, complex platform support, difficult for one layer to handle all abstraction, provide useful APIs for each layer

3.3. Exercise

: Web system for airline tickets/reservation

INSERT VISUAL

3.4. Weak Entity Identification

- double line
- a weak entity can be uniquely identified by combining its partial key with the key of the entity on the other side of the double-lined relationship this weak entity is connected to, by double lines.

4. Chapter 4

4.1. Extended Entity Relationship (EER) Diagrams

- Superset of ER diagrams, primarily adding subclasses and superclasses, plus union and category types with inheritance.

4.1.1. Super class -- > -- subclass

- subclass inherits all attributes of parent, and may define additional ones
- relationships may be defined on subclasses and not superclasses as well
 - but, a subclass inherits relationships from the parent

4.2. Specialization

- **specialization** is the process of defining a set of subclasses of an entity type
- defined on basis of distinguishing characteristic in entity type.

4.3. Predicate vs. User Defined Subclasses

- **Predicate Defined** if membership is determined by placing condition on value of attribute of superclass
- **User Defined** if membership is specified for all entities by the user.

4.4. Disjoint vs. Overlapping

- **Disjoint** if entity may not belong to > 1 subclass
 - indicated by (d) branch off superclass
- **Overlapping** if there is no disjointness constraint
 - indicated by (o)

4.5. Complete (total) vs Partial

- **complete** if entity must belong to a subclass. The union of all entities in its super-classes.
indicated by double line
- **partial** if there is not a completeness requirement. Subset of the union

4.6. Specialization Hierarchy vs Lattice

- **hierarchy** if every subclass is allowed to be a subclass of only one parent class → tree/strict hierarchy
- **lattice** if each subclass is permitted to be a subclass of > 1 parent
 - same idea as multiple inheritance, and subclass must have parent from all classes it inherits

4.7. UNION type

- represents collection of entities from different types
- a union of entities is called a **uniontype** or **category**
- ex. $\text{bank} \cup \text{person} \cup \text{company} \rightarrow \text{owner}$
- category inherits attributes only of the subclass it belongs to
- needed for situations where "is all" relationship does not exist

4.8. ERR Design Guidelines

- only use subclasses when deemed necessary
- union/categories should generally be avoided
- choice of disjoint/overlapping and user/predicate defined should be based on the mini-world requirements.
 - default to overlapping/partial

5. Chapter 9

5.1. ER-to-Relation

- **Step 1: Map Regular Entity Type**
 - for each regular (strong) entity type E , create relation R with all of E 's sample attributes (flatten composites)
 - called "entity relations", and each tuple is "entity instance"
 - choose ONE key in E to be the Private Key in R .
- **Step 2: Map Weak Entity Types**