Database Design

Reading: Elmasri & Navathe Chapter 7.
Database Design Process

- stages in the design of a database:
  1. requirement analysis
  2. conceptual database design
  3. choice of the DBMS
  4. data model mapping
  5. physical design
  6. implementation

- not necessarily strictly sequential
  - feedback loops exist, i.e. may need to revisit earlier stages during a later stage
I. Requirement Collection and Analysis

- **Purpose**: to document the data requirements of the users
  - **functional requirements** are the operations that will be applied to the database, including queries and update
  - the specification will then be used as the basis for the design of the database
  - typical activities:
    - identification of application areas and user groups
    - analysis of existing documentation of application areas, e.g. policy documents, forms, reports, organisation charts
    - analysis of current operating environments and the planned use of the information, e.g. information flow, types of transactions, frequency of transaction types
    - responses to user questionnaires are analysed
... which means

- start from a description of the requirements which is:
  - poorly structured,
  - heterogeneous
  - informal
- and use a technique to transform that into a specification of the database requirements which is:
  - formal
  - homogeneous
  - consistent
  - complete
2. Conceptual Design

Two parallel activities

1. **schema design** - look at the data requirements resulting from the analysis (phase 1) and produce a conceptual schema in a DBMS-independent high level data model

2. **transaction design** - look at the database applications whose requirements were analysed in phase 1 and produce high level specifications for these transactions
2.1. Conceptual Schema Design

- Purpose: to produce a **conceptual schema** of the database
  - expressed using concepts of the high level data model
    - not including implementation details (has to be understood by non-technical users)
    - but detailed in terms of the “objects” of the domain the database will represent
  - independent of the DBMS to be used (no relational DB-oriented notions!)
- cannot be used directly to implement the database
- design is made in terms of a **semantic** or **conceptual** data model
2.2 Transaction design

- **Purpose**: to produce a design of the transactions, that will run on the database
  - retrieval: retrieve data for display or as part of a report
  - update: enter new data or amend existing data
  - mixed: more complex applications may do both retrieval and update
- **Why?**
  - need to be sure to include in the conceptual schema all information required by transactions
  - relative importance and frequency of use of transactions will influence physical database design
  - ... the software needs to be designed as well as the data!
3. choosing a DBMS

• Purpose: establish which is the best framework for implementing the produced schema:
  • type of DBMS (relational, network, deductive, ObjectOriented, ...)
  • user and programmer interfaces
  • type of query languages
• choice made on the basis of technical factors
  • the DBMS has to support the required tasks
• of economic factors
  • software acquisition/maintenance, hardware acquisition, creation/conversion, training of staff
• and of organisational factors:
  • platforms supported, availability of vendor services
4. Logical Design

- **Purpose:** to transform the generic, DBMS independent conceptual schema in the data model of the chosen DBMS *(data model mapping)*

- **two stages:**
  1. **system independent mapping:** no consideration of any specific characteristics that may apply to the specific DBMS package
  2. **tailoring to DBMS:** different DBMSs may implement the same data model in slightly different ways

- **result is a set of Data Description Language (DDL) statements in the language of the chosen DBMS**
  - some CASE tools can generate DDL statements from a conceptual design
5. Physical Design

- **Purpose:** to choose the specific storage structures and access paths for the database files
- **Attention to performances:** some relevant criteria:
  - **Response time:** may want to minimise database access time for data items referenced by frequently used transactions
  - **Space utilisation:** less frequently used data and queries may be archived
  - **Transaction throughput:** average number of transactions that can be processed per minute
6. Implementation

- Purpose: to create the database
- compile and execute DDL statements
- populate the database
  - manually/automatically (may need to convert data from a previous format)
- implement application programs (transactions)
  - programs are written with embedded DML (Data Manipulation Language) statements
- operational phase may begin
Entity-Relationship Model

- model to express the **conceptual schema** of the database
- originally proposed in 1976 by Peter Chen on the “ACM Transactions on Database Systems” journal as a means to unify the network and relational DB models
- many theoretical extensions and practical applications
  - Enhanced Entity Relationship (EER) Model
- used routinely for system analysis and design
  - simple enough to learn and understand the basic concepts
  - powerful enough to be used in the development of complex applications
- conceptual designs using the ER model are called **ER schemas**
ER Model: components

• The ER model describes data in terms of three primitive notions:
  1. entities
  2. attributes
  3. relationships

• an entity is a “thing”, which can be distinctly identified - e.g. a physical thing: a person, a car, a building, or an abstract thing: a university course, an event, a job

• an attribute is a property of an entity - e.g. a person has an age, a car has a colour

• a relationship is an association among entities - e.g. “a person owns a car” is an association between the entities person and car
Entities

- Entities are the “objects” the database has to store information about
- need to distinguish between
  - entities the database contains information about currently
  - world of possible entities the database might contain information about
- the conceptual schema has to capture the changing nature of data
- need to make decisions based on the world of possible entities, e.g. the “entity type”
- the entity type is an abstract description of some set of objects
- data to be actually stored form instances of such abstract description
Attributes

• common properties shared by all instances of the entity type
  • e.g. attributes of the “employee” type might be name, age, address, salary

• the ER model explicitly classifies attributes according to three criteria:
  • complexity
    • whether they have a “structure”. e.g. the component of a date
  • cardinality
    • whether the same entity may have more than one value for each of them, e.g. e-mail addresses of a person
  • primitiveness
    • whether they can be derived by other attributes e.g. the age from the date of birth
Relationships

- A relationship type defines an association among entity types.
- A relationship has a **degree** that is the number of participating entity types, for example:
  - Binary relationships (degree two): e.g. a person owns a car.
  - Ternary relationships (degree three): e.g. a lecturer teaches a course to a student.
- Relationship types can also have attributes (e.g. StartDate attribute on a “supervises” relationship).
Structural Constraints on Relationships

- Structural constraints limit the possible combinations of entities that can participate in a relationship instance:

- **Cardinality Ratio** specifies the number of relationship instances that an entity can participate in (one-to-one, one-to-many, many-to-many)

- **Participation Constraint** specifies whether the existence of an entity depends on it being in the relationship:
  - A **total participation constraint**, or existence dependency, specifies that an entity (instance) can only exist if it participates in the specified relationship (e.g. every lecturer must work in a specified department)
  - A **partial participation constraint** specifies that there may exist an entity (instance) which does not participate in the relationship (e.g. not all lecturers supervise students)
ER diagrams: basics

- entity types are represented as boxes
- relationship types are represented as diamonds connected with each participating entity types
- attributes are shown as ovals connected to the relevant entity or relation type (*key attributes* are underlined)
ER diagrams: attributes

- A simple attribute is represented as an oval
- A complex attribute can have a structure
- A multivalued attribute is a double oval
- A derived attribute is a dotted oval
ER diagrams: relationships

- Cardinality is represented on the connecting lines (an N represents the “many” side)
- Total participation is represented by a double line
- If entities participate to many relationships, a “role” may be added to the line
A special type of entity: weak entity type

- these are entity types which cannot exist in isolation
- instances are identified because they “belong” to specific entities from another entity type, known as identifying owner
  - for instance, the content of a lecture theatre (white boards, desks, etc.) cannot typically be identified as such, the lecture theatre is their identifying owner, so we can talk about “the desk which is in RB8”
- the relationship type that relates the weak entity to its owner is the weak entity’s identifying relationship
  - in the example above, the “is in” relationship
- weak entity types might have a partial key, to distinguish one weak entity from other weak entities related to the same owner
  - for example “the desk 1 (or 2, 3 etc.) which is in RB8”
ER notation: weak entity

- A weak entity type is represented as a double box
- and the identifying relation as a double diamond
- a partial key has a dotted underline
Book example: Company

- The company is organised in Departments which can be in several Locations
- A Department controls a number of Projects
- The company has Employees
- An employee is assigned to one Department, but may work on many Projects
- An employee may have one supervisor (also an employee of the company)
- A Department is managed by one employee
- For each employee, the company keeps track of their dependents (eg. spouse, children, etc) for insurance purposes
NOTE:

Alternative notation for cardinality + participation constraints

NOTE:

Relationships can have attributes too