Managing Data

Data storage tool must provide the following features:

- Data definition (data structuring)
- Data entry (to add new data)
- Data editing (to change existing data)
- Querying (a means of extracting data by a description)
- Persistence (data existing beyond a single operation or program)

Database Management System (DBMS)

A DBMS combines all the functions of data storage and access for a related set of tasks, for example: handling student records, stock control in a warehouse, account management in a bank. A DBMS is the software that can provide features needed by databases:

- Sharing and integration of data
- Multiple views of the same data
- Controlled concurrent access
- Management of security and integrity

The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

Database

A database is an entity in which data can be stored in a structured manner, with as little redundancy as possible. It gives users access to data, which they can view, enter or update (within the limits of the access rights granted to them). It is viewable (and writable) by many users at the same time. This is called *controlled concurrent access.*

Types of Database

- Hierarchic databases (older)
- Network databases (older)
- *Relational databases (MySQL, Postgres, Oracle, MS SQLServer)*
- Object Oriented Databases (1990s)
- NoSQL Databases (2000s | MongoDB, HBase)

Relational Database

A relational database can be thought of as a series of tables about related information.

Avoiding Redundancy

Databases avoid redundancy, such as:

Ambiguity – same thing with different names in different files

Inconsistency – if data changes in one place, it should also change in the other files it exists in

Wasted Effort – data should be shared where possible to save time and effort

Controlled Concurrent Access

Databases can have many users reading and writing at the same time, so we need to make sure that each view of the data is correct or consistent for each user to avoid incorrect updates. DBMS have concurrent control software to ensure that several users updating the same data do so in a controlled matter. This happens through transactions, which make concurrent database interactions appear to happen independently & sequentially.

Types of DBMS

Local Database – can be used on one machine by one user/application only

Distributed database – the information is stored on remote machine(s) and can be accessed over a network

Three-level Architecture

We can think about a database system at three levels/schemas:

- *External* – how users view data
- *Conceptual* – how programmers model and implement the dataset in the database
- *Internal* – how the DBMS stores the data

In designing a database, we take an *external schema*, and design a corresponding *conceptual schema.* The DBMS handles the *internal schema,* with hints from the designer.

Database Terminology

Data Model – a description of the object that could be represented by a computer system together with their properties and relationships. Typically real world objects such as products suppliers, customers and orders

Schema – a description of how a database can be designed to represent a data model. For example, tables with column definitions: suppliers have names, addresses, etc.

Database – an instance of a schema with corresponding data

Databases are created in this order.

Database Design

Databases are used by people to perform particular tasks and therefore need an interface to allow people access to the data. People may need access to the same database and we must consider the needs of the users when designing it.

Database design lifecycle:

- *Requirements analysis* – user needs; what must the database do?
- *Conceptual design* – high-level description; often using ER model
- *Logical design* – translate the ER model into a relation schema (typically)
- *Schema refinement* – check schema for redundancies and anomalies
- *Physics design/tuning* – consider typical workloads, and further optimise

People Involved

Users – access the data (casual vs expert) and need an effective way of accessing data

Database Designers – specify schema and content

Application Developers – extended functionality; provide means of data access for a particular application

Database Administrators – maintain accuracy, speed and integrity

Website Designers – designs the front end (interface)

Identifying User Requirements

Talk to the client e.g. CEO of bank

Talk to the customer e.g. those that may view the data or actual users of the system

Talk to different levels of users e.g administrators, programmers, tech staff and locate people who might need to add/update/query data

Identifying Data Requirements

Write down the physical things you need to store data on, for example: customers, branches, accounts…

Take not of the relationships between each of these things, for example: all customers must only belong to one branch.

These can be organised into data objects:

Customer	Branch
- Name	- name
- address	- address
- overdraft limit	- manager
- address	- ID
- ID	

Data Modelling

We need a way to represent the relevant data in our database:

We develop a *conceptual data model,* based on information from the users, and considering existing external views. We describe data in a high-level manner as entity types, attributes and relationships.

The conceptual data model can then be converted into a *conceptual schema* describing how data is stored – as tables and records, for instance. These are Implementation-level/logical data models.

Low-level or *Physical Data Models* describe how data is internally stored on the computer: files, storage structures, etc. This is handled by the DBMS (with occasional help from the DBA).

The choice of data model to use depends on the type of database. We use the *Entity Relationship Model* which easily maps the Relational DBMS. Once we have a conceptual data model for a problem in terms of an Entity Relationship diagram, we can easily generate a conceptual schema for the database.

Types of Database

Relational – Information is represented by a collection of two dimensional tables with rows and columns.

Network – information is represented by a directed graph of records and links between the records.

Hierarchical – the information is represented as a tree of records organised in sets connected by ownership links

Objects oriented – collections of classes with certain behaviours or methods

NoSQL – trade redundancy for speed/scalability

ER Modelling

1. We identify entities – real world objects such as products, customers, orders…
2. We identify attributes – information on entity, such as name, address, id…
3. We identify relationships between entities – one bank has many customers

This can later be mapped to a logical data model or *schema* and, in turn, mapped to a physical model by the DBMS.

Entities

An entity is a uniquely identifiable object and is capable of an independent existence. The are grouped together into categories called *entity types* or *entity sets* such as employee, department, project. An entity is an instance of a given entity-type and there are usually many instances of an entity-type.

Entity Types

- Can be a physical object such as a house or a car
- Can be an event such as a house sale or a car service
- Can be a concept such as a customer transaction…

Attributes

These are properties that describe an entity. It is expected that all instances of a given entity type will have the same attributes.

Simple attributes are composed of a single value. For example *Gender* may be male, female or other.

Composite attributes are composed of a set of component values. For example, *Address* may contain other attributes such as *Number*, *Street*, *City* – each with their own values.

Single-valued attributes hold a single value.

Multi-valued attributes store a set of values. For example, locations for a department.

Primary Key

An entity type will usually have one or more attributes which have unique values.

Key Attributes

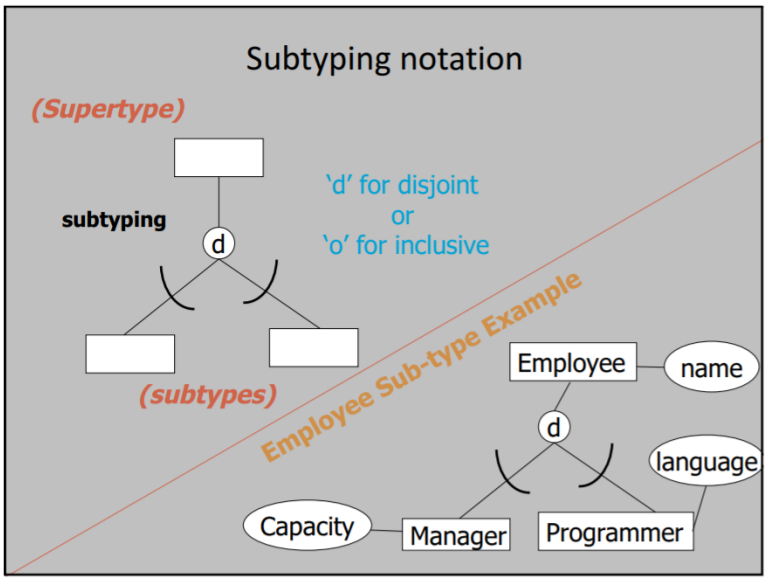
The primary key attributes of an entity type is an attribute whose values are distinct for each entity. Sometimes several attributes (a composite attribute) form a key together.

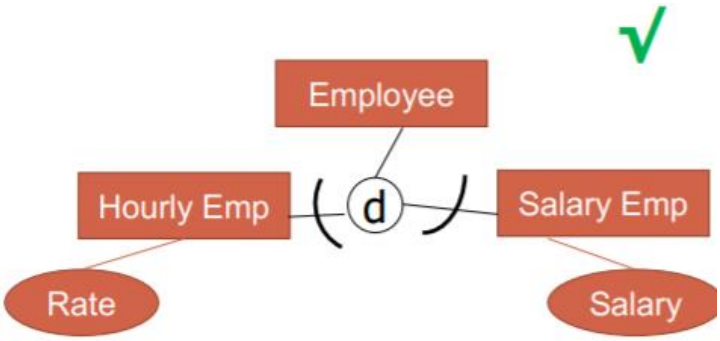
Subtyping

A subtype is an entity type that inherits the properties of it parent type. For example, programmer and manager can be represented as subtypes of employee.

Employee attributes belong to programmer and manager since they are a subtype of employee.

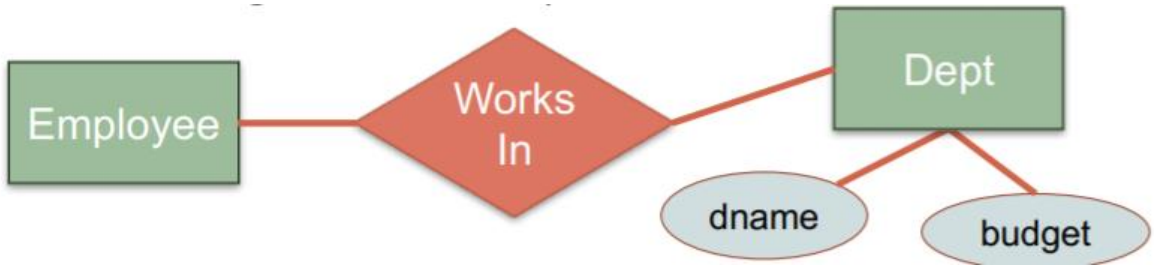
Subtypes may be disjoin (belong to exactly one subtype) or inclusive (belong to either or both).





Relationships

Relationship types represent the interaction between entity types. For example, the entities in types *employee* and *dept* can interact through the relationship *worksIn*



Relationships can also have attributes. For example, worksIn could have an attribute *since* which could store when they started working there.

Relationship Degrees

The degree of a relationship is the number of entity types participating.

Binary relationships – 2 participating entity types. For example Employee *works for* Department

Ternary relationships – 3 participating entity types. For example a Manager *manages* a project in a Department

Cardinality Constraints

The cardinality specifies the number of entity instances that can participate from each side of the relationship of a binary relationship:

One to one (1:1)

One to many (1:N)

Many to many (M:N)

Participation Constraints

A double line indicates the total participation constraint. All entities in the set must participate in at least one of the relationships in the set.

Weak Entity Types

Depend on other entities to guarantee uniqueness. They do not have primary key attributes of their own. They must have total participation in the identifying relationship.

Recursive relationship

An entity type may be in a relationship with itself.

Constructing an ER Diagram

1. Identify the entity types
2. Identify the properties of each entity type
3. Decide which properties are attributes
4. Select primary key
5. Determine which properties infer relationships
6. Decide on the cardinality and participation of the relationship

Conceptual Schema

Now that we have an ER Model, we need to arrange the data into a logical structure. The logical structure can then be mapped into the tables. This is the process of creating a conceptual schema from the conceptual design.

Entities to Tables

A table (relation) is constructed for each item of interest in a database. A relation approximately equates to an entity type or some relationships in an ER Diagram. All relations must have a *heading* and a *body*.

Matric #	Firstname	Surname	…
2019582	Joe	Bloggs	
2058292	Jane	Bloggs	
…			

Each table has *rows* (tuples) and *columns* (attributes)

The Heading

Name of relation: Student

Names of columns (attributes): Name, StudentID, Exam1, Exam2

STUDENT(Name, StudentID, Exam1, Exam2)

The number of attributes determines the *degree* of the relation.

The Body

The rows of a relation comprise its body, these are referred to as *tuples*.

A tuple is an ordered list of values. The meaning of each value is determined by its position in the tuple.

The number of tuples in a relation determine its *cardinality*.

Degree and Cardinality of a Relation

Degree – number of attributes/columns

Cardinality – number of rows/tuples

STUDENT

name	matric	exam1	exam2
Gedge	891023	12	58
Kerr	892361	66	90
Fraser	880123	50	65

The Student relation has a degree of 4 and a cardinality of 3.

WARNING: Do not confuse this with the cardinality of a relationship type in an ER Diagram!

Domains

Domains are a lot like data types in programming. They define the set of values that can be assigned to an attribute and determines the range of allowable operations on each value (add, subtract, concatenate).

A *domain* is a set of atomic values that can be assigned to an attribute.

A domain has two aspects:

- Its meaning (for example, set of student matriculation numbers)
- Its format (for example, an integer from 0 to 999999)

Different DBMS offer different sets of domains:

- *MS Access* – Text, Number, Memo, Date/Time, Currency, AutoNumber, Yes/No, etc (these are not SQL standard)
- *MySQL* – CHAR, VARCHAR (strings), INT, FLOAT, DATE… (these are SQL standard)

Here are some more standard SQL examples:

Data Type	Description	Examples
INT	Integer number	1, 5, -100
FLOAT, DOUBLE	Floating point number	-1.1, 5, 6e10
BOOLEAN	Boolean	1, 0
(MySQL doesn't have BOOLEAN) TINYINT(1)	*Integer with only 1 bit*	*1, 0*
CHAR(x)	Fixed length string of length x	'A '
VARCHAR(x)	Variable length string upto length x	'A'
DATE, TIME, TIMESTAMP	Various date/time data types	'2016-01-01 00:00:00.000000'

More examples here: http://dev.mysql.com/doc/refman/5.7/en/data-type-overview.html

Translating ER Models to a Relational Schema

1. Entities and their simple attributes
2. Weak entities and their simple attributes
3. 1-1 relationships (and their attributes)
4. 1-M relationships (and their attributes)
5. M-N relationships (and their attributes)
6. Composite attributes
7. Multivalued attributes

Relations to Schema

A *tuple* (record) is a row of a relation – a set of values which are instances of the attributes.

e.g *< 'Fraser', 880123, 50, 65 >*

A *relation schema* is a set of attributes written like:

Student(name: Test, matric: Number, Exam1: Number, Exam2: Number)

A *relational database schema* is a set of these relation schemas.

A *relational database* is just a relational database schema with data.

Attributes to Columns

A column of a relation is an attribute having:

- A name (indicates the role the column has in this relation)
- A domain (indicates the set of values it may take)

e.g Student(StudentID: **INT**, Address: **VARCHAR(100)**, DOB: **DATE**)

Primary Keys (schema)

Primary keys are underlined. Multiple attributes are underlined if they are a composite key.

Student(**StudentId: INT**, Address: **VARCHAR(100),** DOB: **DATE**)

Weak Entities Mapping (schema)

Create primary key for a weak entity type from foreign keys of identifying relationship types or partial keys of the weak entity.

Foreign Keys (schema)

A *foreign key* is an attribute (or set of attributes) that exist in one or more tables and which is the primary key for one of those tables. A value in a foreign key must exist in the referenced primary key attribute.

A foreign key is used to cross-reference tables. It is a referential constraint between two tables and a value in the foreign key must exist in the referenced primary key.

A table can have multiple foreign keys and each foreign key can reference a different table.

Foreign keys don't need to have the same attribute name across tables.

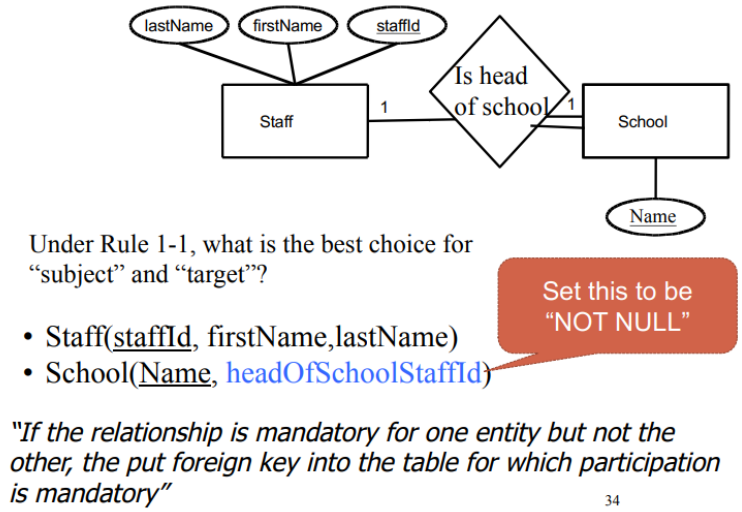
They must have the same data type.

Relationships (schema)

1:N – the primary key on the 'one side' of the relationship is added to the 'many side' as a foreign key. For example Stars(Name, Address, **Studio, Salary**). Studio and Salary are both foreign keys. A studio can have many stars. A Salary can be given to multiple stars.

1:1 – For each one-to-one relationship type between two entity types, choose one entity type to be the subject and one to be the target type. Add the key attributes of the subject class to the target schema as foreign key attributes and add the attributes of the relationship to the target schema.

Not got a clue what that word-spaghetti means, so here's his slide:



N:M – A new relation is produced which contains the primary keys from both sides of the relationship as foreign keys. These attributes form a composite primary key for the relation.

Subtypes and Supertypes (schema)

You cannot directly map to a relational schema, so you can model the supertype and its subtypes as a single table (and leave attributes null if they don't apply) or turn each subtype into its own table and set up 1:1 relationships between the super-entity types.

Composite Attributes (schema)

Create an attribute in the relation schema for each component attribute. Use the name of the composite attribute as a prefix for each of the component names.

Multi-valued Attributes (schema)

Represent each multi-valued attribute as if it were a weak entity class.

Characteristics of the Relational Model

A relation is a set of tuples. There are no duplicate tuples because the tuples form a set and this must be checked when a new tuple is added, a value is modified, or a new relation is created as a restriction of an old one.

This implies that a *primary key will always exist*. Worst case would be a key composed of all the attributes.

Tuples are unordered.

Attributes are also an unordered set.

Relations represent:

- An entity type and its attributes
- A relationship
- A set of values

Unknown values must be represented using NULL

NULL Values

We use NULL values when we don't have the data needed. Primary keys must not be NULL.

Integrity

Inherent integrity constraints must hold for all relational databases. They are typically enforced by the DBMS.

Enterprise constraints are specific to a particular application.

Integrity Constraints

Primary key values must be unique.

Primary key values cannot be NULL.

Foreign key values must exist in the primary key of the referenced relations – we call this *referential integrity*. It may be NULL (if it's not a mandatory participation).

Enterprise Constraints

Application dependent.

Specified non-key attributes must not be NULL. For examples, all students must have a name, even if the primary key is the student number.

Values of one attribute must be less than values in another attribute. For example, the age of parent must be greater than age of child.

Referential Integrity

Concerns the use of foreign keys.

Guarantees that relationships between tuples are coherent. Every non-NULL value in a foreign key must also exist in the relation for which it is the Primary Key.

Restrict – ban any alterations to a primary key if there are foreign key references to it

Cascade – cascade the effect to all relations in all tables that refer to it

Set to NULL – allow update in the original table, set all corresponding foreign key values to NULL