Database Managament IS&M Option

Mark Levene
Email: m.levene@dcs.bbk.ac.uk
Web: http://www.dcs.bbk.ac.uk/~mark/

Lecture Plan

1. Introduction to Databases
2. Data Modelling with the Entity-Relationship Model
3. The Basic Relational Model
   (a) Tables, Attributes and Domains
   (b) Primary and Foreign Keys, Entity and Referential Integrity
4. Querying a Relational Database
   (a) Querying a Single Table
   (b) Aggregating and Grouping Data
   (c) Querying Multiple Tables
   (d) Removing Duplicates
   (e) Null Values
   (f) Transaction Management and Concurrency Control

Lecture Plan - Continued

5. Designing a Relational Database
   (a) The three levels of a Database System and Data Independence
   (b) The Normalization Problem
   (c) Boyce-Codd Normal Form
6. Object-Relational Databases
7. Retrieving Information from the World-Wide-Web
8. A brief introduction to XML

There will be coursework !

Recommended books:

1. [CB02] – Comprehensive introductions to Database Management, over 1000 pages.
2. [WM00] – Introduction to Access.
3. [WM01] – Relational databases using Access.
4. [LL99] – Comprehensive treatment of the theory underpinning relational databases and recent extensions of the basic model.
5. [Cel00] – For those of you interested in a book on SQL programming.

Apart from the books I have recommended, there are many more good database books at all levels.

Introduction to Databases

- An introductory example
- What is a database?
- Why do we need Database Management Systems?
- The three levels of data abstraction
- What is a Database Management System?
- What is data independence?
- What is a data model?
- The relational data model

Examples of Applications that use a Database

- Banking
- Booking a holiday
- Shopping – Inventory control
- E-commerce – E.g. Online Book Stores
- The Invisible Web

Initial Definition of a Database.

A *database* is an organised collection of inter-connected *data items*.

Initial Definition of a Database Management System (DBMS).
A DBMS is a computer system which is responsible for efficient storage and retrieval of the data items in a database.

An introductory Example

The following concepts will be discussed, via a simple fragment of a library database:

- *tuples* (*rows*), *relations* (*tables*) and *relational databases* (*databases*).
- *entity sets* and *entities* (*objects*).
- *attributes*, *attribute values* and *domains*.
- *relation schema* (*header*) and *entity type* (*object type*).
- *relational database schema* (*database headers*).

The Books Relation

AUTHOR1	SHORT_TITLE	PUBLISHER	YEAR	ISBN
Atzeni	DB Theory	Benjamin/Cummings	1993	0-8053-0249-2
Date	Introduction to DBs	Addison-Wesley	1990	0-201-52878-9
Korth	DB Concepts	McGraw-Hill	1991	0-07-044754-3
Mannila	The Design of DBs	Addison-Wesley	1992	0-201-56523-4
Ullman	Principles of DBs	Computer Science Press	1988	0-7167-8158-1

The Loans Relation

ISBN	LOCATION	QUANTITY	LOAN
0-8053-0249-2	Science	1	0
0-201-52878-9	Main	3	2
0-07-044754-3	Main	1	1
0-201-56523-4	Science	1	0
0-7167-8158-1	Main	2	1

What is a Database?

Definition of a database.

1. A database is a collection of *persistent* data (or information).
2. A database models part of the real world, called the *enterprise*.
3. A database is, in general, a *shared* resource.

Why Do We Need DBMSs?

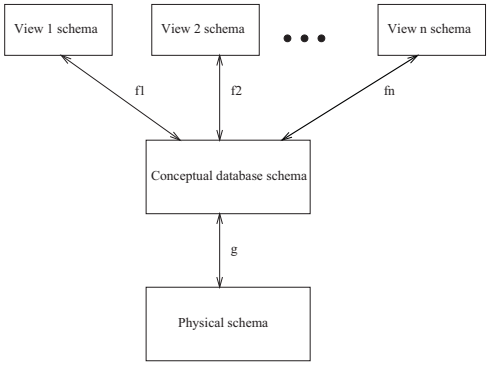
Assume that a DBMS is a software (and hardware) package that handles all the interaction of an application with the database.

1. It saves programmer time and maintenance.
2. It provides *data independence*.
3. It provide provides programmers with various tools.

The Three Levels of Data Description

This framework describes the interfaces and the information that should pass between them:

1. *physical* (or *internal*) - normally machine dependent.
2. *conceptual* (or *logical*) - based on a data model.
3. *view* (or *external*) - corresponds to application programs or user groups.



What is a Database Management System?

A DBMS is a software (and hardware) package providing the following services:

- A *data definition language* (DDL).
- A *data manipulation language* (DML).
- Efficiency in query response and storage space.
- Maintenance of integrity and consistency.
- Concurrency control and data sharing.
- Transaction management (commit and rollback).
- Recovery from failure.
- Security.
- Data administration facilities.
- Adherence to standards (e.g. SQL).

What are the disadvantages of using a DBMS ?

- Cost
- Lock-in
- Complexity

DBMS users

- End users.
- Application programmers.
- Database administrator (DBA).
- Enterprise administrator - responsible for database design.
- Application administrator - responsible for view design.

What is Data Independence?

- *physical data independence* - the physical level may be changed without affecting the conceptual level.
- *growth independence* - the independence of the view level from the addition of new attributes to relation schemas and new relation schemas to the database schema.
  - ⋆ Deletions of relations or columns from relations at the conceptual level disrupt views that reference those relations or columns.

What is a Data Model?

1. Structural part.
2. Integrity part.
3. Manipulative part - declarative or procedural.

The Relational Data Model

1. Structural part - relations (tables).
2. Integrity part - keys (entity integrity) and foreign keys (referential integrity).
3. Manipulative part - Structured Query Language (SQL) or the relational algebra.
   (E.g. *select, project, join* queries.)

Summary:

- A database is a collection of persistent data that is shared and models an enterprise.
- A DBMS is a computer system that provides the interface between database users and the database itself.
- A DBMS has three levels of abstraction: physical, conceptual and view.
- Data independence means that changing the schema at a lower level does not affect the higher level.
- A data model has structural, integrity and manipulative parts.

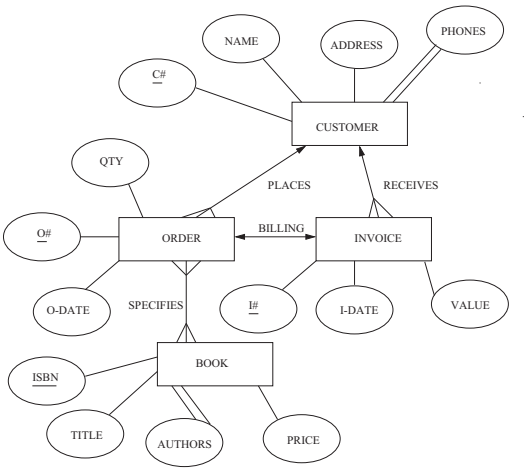
The Entity Relationship Model

- The building blocks of an Entity-Relationship Diagram (ERD)
- Cyclic relationships
- Weak entity types
- An informal algorithm for constructing an ERD

The Components of the ER Model [Che76]

- Data structures -
  ⇒ entities and relationships.
- Integrity constraints -
  ⇒ primary keys for entities and relationships, and
  ⇒ cardinality constraints for relationships.
- The ER model is only a partial data model, since it has no standard manipulative part.

Example Entity Relationship Diagram

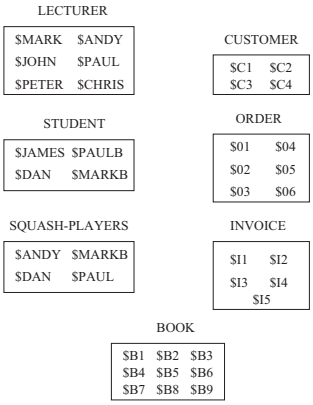


Fundamental Concepts

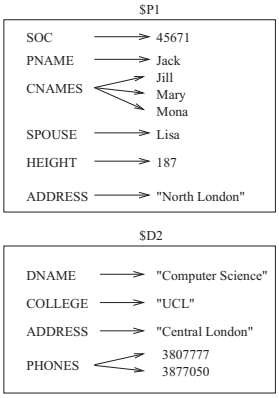
- *aggregation* -
  ⇒ a collection of attributes forms an entity type (or entity set).
  ⇒ two entity types form a relationship type.
- generalisation (specialisation, ISA) -
  ⇒ E.g. Employee ISA Person and Student ISA a Person.

Next

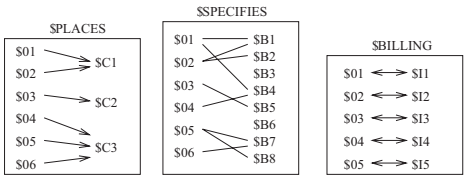
Example Entity Sets



Example Entities



Example Relationships



The Main Advantages of ERDs

- They are relatively simple
- They are user friendly
- They can provide a unified view of data, which is independent of any data model.

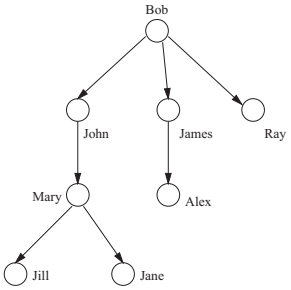
Graphs

A graph is an ordered pair (N, E) where

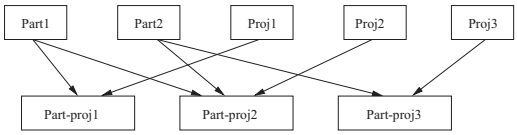
- N is a finite set of **nodes**, and
- E is a finite set of **edges**.
  Each edge e = {u, v} in E is an unordered pair of nodes in N.
  ⋆ Nodes and edges can be **labeled**.
  ⋆ In a **directed graph** (or digraph) E is a set of **arcs**.
  Each arc e = (u, v) in E is an ordered pair of nodes in N.
- Graphs and digraphs can either be cyclic, hierarchical, or acyclic.

Next

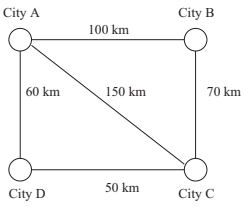
Hierarchy



Acyclic Directed Graph



Cyclic Undirected Graph



The Building Blocks of an ERD

- Entities and entity sets (entity types).
- (Binary) relationships and (binary) relationship types.
- Domains, attribute values and attributes.

Entities

Definition. An *entity* is a ''thing" that exists and can uniquely be identified.

Definition. An *entity type* (or *entity set*) is a collection of similar entities.

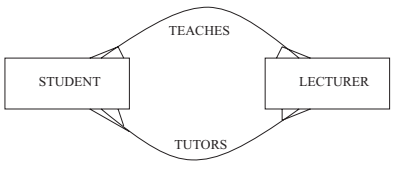
- An entity type consisting of a finite set of entities can be depicted by a graph having *no* edges.

Relationships and their Functionality

Definition. A *(binary) relationship type* is an association between two entity types.

- There may be more than one relationship type between two entity types. E.g. Tutors and Teaches between Lecturer and Student.

Example of M to M relationship Types



Example of M to One relationship Types

Example of One to One relationship Types

Definition. A *relationship* is an instance of a relationship type.

⇒ In mathematical terms a relationship is a finite *binary relation*.

• A relationship can be depicted by a **bipartite** graph between entity sets.

Example relationships

Next

Classification of Relationships

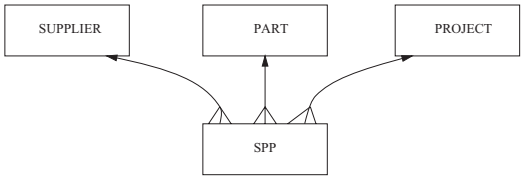
- **optional** relationship -
  An Employee may or may not be assigned to a Department
- **mandatory** relationship -
  Every Course must be Taught by at least one Lecturer.

Classification of Relationships - Continued

- many-to-one (or **one-to-many**) -
  An Employee Works in one Department or a Department has many Employees.
- one-to-one -
  A Manager Heads one Department and vice versa.
- many-to-many -
  A Teacher Teaches many Students and a Student is Taught by many Teachers.

Example relationships

Binary Versus General Relationships



Attributes and Domains

Definition of attribute. *Atrribute names* (or simply attributes) are properties of entity types.

- Attributes can be **single-valued** (e.g. pname and dname), or
- **multi-valued** (e.g. cnames and phones).

See ERD

Definition of domain. The *domain* of an attribute of an entity type is the set of constant values associated with that attribute.

- Domains can be **atomic** such as the domain of integers and the domain of strings.
- **set-valued** such as the domain of finite sets of integers or of finite sets of strings.

Definition of attribute value. An attribute, say *att* of an entity type E associates a *value* from its domain with each each entity *e* of E.

This value is denoted by *att(e)*.

• The attribute values of entities can be depicted by a bipartite graph from attributes to their values.

Keys and Superkeys

Definition of a superkey. A set of attributes of an entity type is a *superkey* if for each entity, say *e*, over that type, the set of attribute values of the attributes in the superkey *uniquely* identify *e*.

Definition of a key. A *key* for an entity type is a superkey which is *minimal*.

- **simple** keys are single attribute keys, such as E# ans SOC#.
- **composite** are keys have more than one attribute, such as {Dname, College} and {Pname, Address}.

Definition of primary key of an entity type. A *primary key* is a key, which is designated by the database designer.

• The primary key guarantees logical access to every entity.

Definition of primary key of a relationship type R.

• **R** is a relationship type between E_1 with primary key K_1 and E_2 with primary key K_2.

1. If **R** is many-to-many the primary key of **R** is $K_1 \cup K_2$; (see many to many relationship types).
2. If **R** is many-to-one the primary key of **R** is K_1; (see many to one relationship types).
3. If **R** is one-to-many the primary key of **R** is K_2; (see one to many relationship types).
4. If **R** is one-to-one the primary key of **R** is either K_1 or K_2; (see one to one relationships).

Cyclic Relationships

Definition. A cyclic relationship type (also called *recursive*) is a relationship type between two occurrences of the same entity type.

- Marriage between Person and itself.
- Parent-Child between Person and itself.
- Part-Sub-Part between Part and itself.

⋆ With each entity type in a cyclic relationship type we associate a *role*.

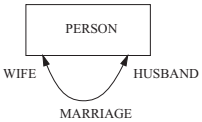
• We add the roles to the primary key attributes to form the primary key of the relationship

E.g. {Husband_Soc#, Wife_Soc#} is the primary key of Marriage assuming Soc# is the primary key of Person.

Next

Example of a Cyclic Relationship Type

Example of Roles in a Cyclic Relationship Type



Weak Entity Types

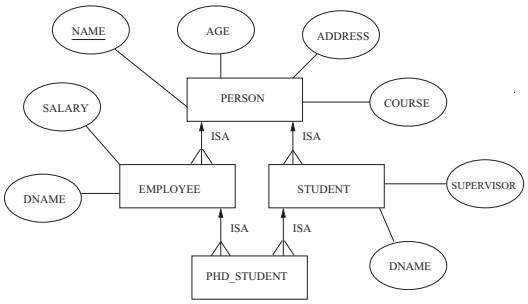
- ID relationship type -

  In an employees database Child entities *exist* only if their corresponding Parent employee entity exists.

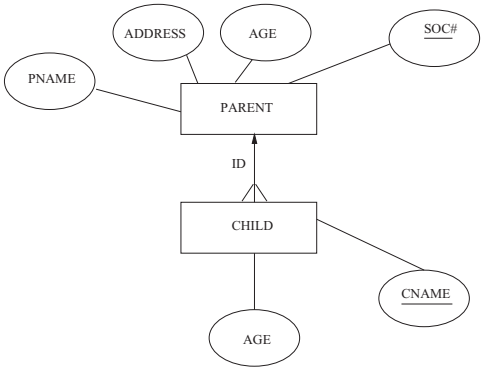
- ISA relationship type -

  An Employee is a special case of Person and therefore the *existence* of an Employee entity implies the existence of a corresponding Person entity.

ISA Relationships



ID Relationships



Let E_1 and E_2 be entity types.

Definition of ID relationship type. E_1 *ID* E_2 if the primary key of E_1 is composed of the primary key of E_2 plus one or more attributes of E_1.

Definition of ISA relationship type. E_1 *ISA* E_2 if the primary key of E_1 is also the primary key of E_2.

In addition, if in the database I_1 is the set of instances of E_1 and I_2 is the set of instances of E_2, then I_1 is a subset of I_2.

• If Employee ISA Person then Employee *inherits* all the attributes of Person.

The Universal Relation Schema Assumption (URSA)

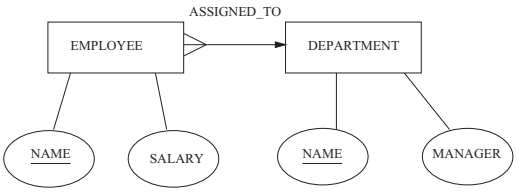
Definition. Each attribute of an entity type plays a unique role in the ERD.

I.e. all occurrences of attributes in an ERD have the same meaning.

E.g. Name can be department name or Person name and therefore needs to be modified to Dname and Pname.

⇒ The URSA is important in relational database design.

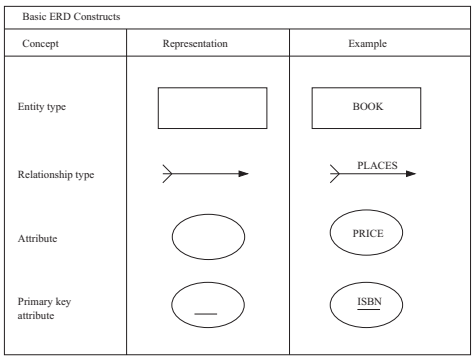
Example Illustrating the UR Problem



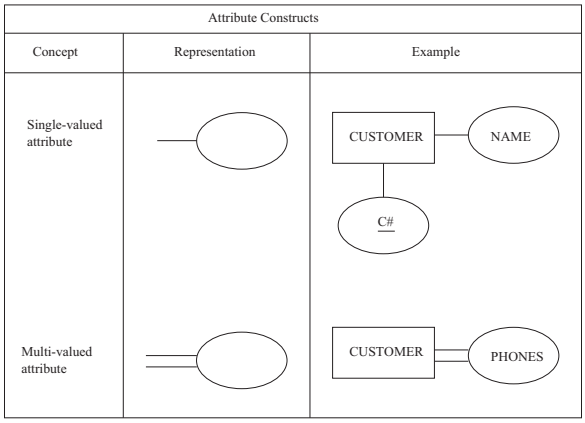
An Informal Algorithm for Constructing an ERD

1. Identify the entity types (including weak entity types) of the enterprise.
2. Draw some instances of the identified entity types.
3. Identify the relationships (including ISA and ID relationships) of the enterprise.
4. Classify each relationship identified in step 3 according to its functionality, i. e. if it is a one-to-one, many-to-one or many-to-many.
5. Draw some instances of the identified relationships.
6. Draw an ERD with the entity types and the relationships between them.
7. Identify the attributes of entity types and their underlying domain; if you are familiar with DD notation then give the DD definitions of the domains.
8. Identify a primary key for each entity type.
9. Draw some instances of attribute values of entities.
10. Add the attributes and keys to the ERD drawn in step 6.

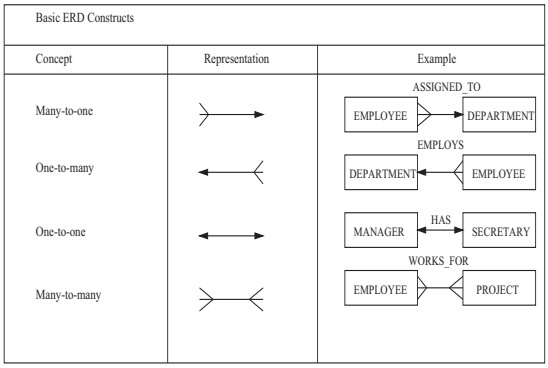
Basic ERD Constructs



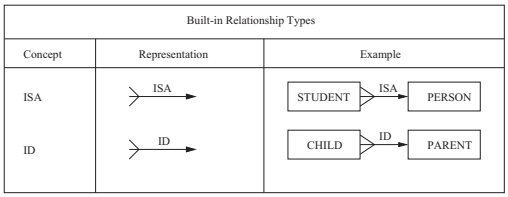
Attribute Constructs



Relationships Constructs



Built-in Relationship Constructs



The Relational Data Model
The data structures of the relational model

- Attributes and domains
- Relation schemas and database schemas (decomposition)
- The universal relation schema assumption
- Relations and databases
- First normal form (1NF)

Supplier-Parts-DB

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Supplier table

PNUM	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Part table

JNUM	JNAME	CITY
J1	Sorter	Paris
J2	Display	Rome
J3	OCR	Athens
J4	Console	Athens
J5	RAID	London
J6	EDS	Oslo
J7	Tape	London

Project table

SNUM	PNUM	JNUM	QTY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P5	J4	400
S5	P1	J4	100
S5	P3	J4	200
S5	P4	J4	800
S5	P6	J4	500
S5	P6	J2	200

Supply table

Attributes and Domains
Assumptions:

- U is a **universal** set of attributes
- D is a the **underlying database domain** of constant values.
- The domain of A in U, is DOM(A) is a subset of D.
- **Unique Name Assumption** (**UNA**): two constants c_1, c_2 in D are equal if and only if they are syntactically equal.

  E.g. "Mark" = "Mark" but "Mark" $\neq$ "John" and "Mark" $\neq$ 143.

Relation schemas and Database Schemas

A **relation schema** (or **table header**) R has the following components:

- A **relation symbol** R, which is the name of the schema.
- A **similarity type** type(R), which denotes the number of attributes of R.
- A set of **attributes** (or **column headers**) $\{att(1), \ldots, att(type(R))\}$, denoted by schema(R).

E.g. SUPPLIER is a relation symbol,
type(SUPPLIER) = 4, and
schema(SUPPLIER) = $\{att(1), att(2), att(3), att(4)\}$,
where $att(1)$ = SNUM, $att(2)$ = SNAME, $att(3)$ = STATUS, and $att(4)$ = CITY.

A **database schema R** is a collection $\{R_1, \ldots, R_n\}$ of relation schemas (table headers).

E.g. The database schema of the Supplier-parts-DB is {SUPPLIER,PART, PROJECT, SUPPLY}.

Notation.
schema(**R**) is the union of all schema(R_i), R_i in **R**.

schema(Supplier-Parts-DB) = {SNUM, SNAME, STATUS, CITY, PNUM, PNAME, COLOUR, WEIGHT, JNUM, JNAME, QTY}

- A relation schema R is in **First Normal Form** (1NF) if all the domains of attributes A_i in schema(R) are atomic.

  (I.e. non-decomposable by the DBMS.)
- A database schema **R** is in 1NF if all the relation schemas R_i in **R** are in 1NF.

A **non**-1NF version of the SUPPLY table

SNUM	PNUM	JNUM	QTY
S1	P1	J1	200
		J4	700
S2	P3	J1	400
		J2	200
		J3	200
		J4	500
		J5	600
		J6	400
		J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
		J7	300
S5	P2	J2	200
S5		J4	100
S5	P5	J5	500
		J7	100
		J4	400
S5	P1	J4	100
S5	P3	J4	200
S5	P4	J4	800
S5	P6	J2	200
		J4	500

Supply-Non-1NF table

From now on we will assume that database schemas are in 1NF.

The justification for this assumption is:

1. The semantics of 1NF are easy to understand
   (e.g. ADDRESS vs. ST_NO, ST_NAME and CITY).
2. 1NF makes it easier to formalise the relational model;
   flat relations (SUPPLY table) vs.
   nested relations (SUPPLY-NON-1NF table).

The Universal Relation Schema Assumption (URSA)

If an attribute A appears both in schema(R_i) and in schema(R_j) then it has the same meaning.

E.g. use SNAME to mean supplier name,
use PNAME to mean part name and
use JNAME to mean project name.

Question. Does CITY violate the URSA ?

Relations and Databases

A **tuple** (or **row**) over R, with schema(R) $= \{A_1, \ldots, A_m\}$ is a member of

$$\text{DOM}(A_1) \times \ldots \times \text{DOM}(A_m),$$

where $\times$ is the Cartesian product operator.

A **relation** (or **table**) over R is a finite set of tuples over R.

A **database** d over $\mathbf{R}$ is a collection $\{r_1, \ldots, r_n\}$ of relations r_i over R_i.

Cartesian Product Example

Boy		Girl
Joseph		Tamara
Nimrod		Lara
Sasha		Maria

Boy and Girl Tables

Boy	Girl
Joseph	Tamara
Joseph	Lara
Joseph	Maria
Nimrod	Tamara
Nimrod	Lara
Nimrod	Maria
Sasha	Tamara
Sasha	Lara
Sasha	Maria

Pairs Table = Boy × Girl

First Normal Form (1NF)

Relations over 1NF relation schemas are called **1NF relations** (or **flat relations** or simply relations).

1NF relations are advantageous since they have

1. A simple tabular representation.
2. Simple query languages.
3. A simple set of fundamental integrity constraints.

Synonymous notation:

- relation schema = *table header*
- database schema = *database headers*
- relation = *table*
- database = *database tables*
- attribute = *column header*
- attribute value = table cell
- tuple = row

Summary of the properties of tables:

- Tables names in a database are distinct.
- Column names in a table are distinct.
- The order of columns and rows in a table is *not* important.
- No two rows in a table are the same, i.e. a table does *not* contain duplicate rows.
- Table cells are atomic.

Projection

Let R be a relation schema, X be a subset of schema(R) be a set of attributes and t be a tuple over R.

The **projection** of t onto X, denoted by $t[X]$,
is the collection of attribute values of t under the column headers in X, i.e. the restriction of t to X.

E.g. if

$$t = \text{(S1, Smith, 20, London)}$$

then

$$t[\text{City}] = \text{(London)}$$

and

$$t[\text{SNUM, STATUS}] = \text{(S1, 20)}.$$

Null Values

We must allow for *missing* or *incomplete* information by allowing *null* values as place holders for non-null constants.

E.g. An Employee's address is *unknown*.
E.g. An Employees's spouse *does not exist*.

The distinguished place holder *null* will be used as a null value.

Superkey

Definition of a superkey for R.

A subset SK of schema(R) is a *superkey* for R if

for all relation instances r of R, the projection $t[SK]$ of any tuple t over R *uniquely* identifies a single tuple in r.

Key

Definition of a key for R. A (candidate) *key* for a relation schema R is a superkey for R having a *minimal number of column headers*.

Definition of primary key of R. A *primary key* for R is one of the candidate keys, which is designated by the database designer as being primary.

Question. What are the candidate keys for the Supplier-Parts DB tables ?

Foreign Key

Let **R** be a database schema, R_1, R_2 in **R** and assume that K is the primary key of R_2.

Definition of a foreign key.

FK a subset of R_1 is a *foreign key* for R_1 *referencing* the primary key K of R_2 if the following condition holds:

for all database instances $d = \{r_1, r_2, \ldots, r_n\}$ of **R** and for all tuples t_1 in r_1, if
$t_1[FK]$ does not contain any null values, then
there exist a tuple t_2 in r_2 such that t_1[FK] = t_2[K].

Question. What are the foreign keys for the Supplier-Parts DB tables ?

The First Fundamental Integrity Constraint of the Relational Model

Let K be the primary key of R_1.

Definition of entity integrity. Primary key values $t[K]$ of tuples t in relations over R_1 should **not** contain null values.

The Second Fundamental Integrity Constraint of the Relational Model

Let FK be a foreign key for R_2 referencing K.

Definition of referential integrity. If all of the foreign key values $t[FK]$ of a tuple t in a relation over R_2 are all *non-null*, then
$t[FK]$ are primary key values for K in the referenced relation over R_1.

Checking for Entity Integrity

The following algorithm checks whether *entity integrity* is satisfied in a relation r over R with primary key X subset of schema(R).

Algorithm 1 (Check_Primary_Key(r, X))

```
1.  begin
2.    for all tuples t in r do
3.      if there exists A in X such that t[A] is null then
4.        return NO;
5.      end if
6.      for all tuples u in r minus {t} do
7.        if u[X] = t[X] then
8.          return NO;
9.        end if
10.     end for
11.   end for
12.   return YES;
13. end.
```

Checking for Referential Integrity

The following algorithm checks whether *referential integrity* is satisfied in a database $d = \{r, s\}$ over {R, S}, with foreign key X subset of schema(R) matching primary key Y subset of schema(S).

Algorithm 2 (Check_Foreign_Key(d, X, Y))

```
1.  begin
2.    for all tuples t in r do
3.      if for all A in X, t[A] is not null then
4.        if there does not exist u in s such that u[Y] = t[X] then
5.          return NO;
6.        end if
7.      end if
8.    end for
9.    return YES;
10. end.
```

Modeling Entities in the Relational Model

An entity type E, having attributes, $A_1, \ldots, A_m$ is modeled by a relation schema R, with schema(R) = $\{A_1, \ldots, A_m\}$.

⋆ The primary key of R is the primary key of E.

Modeling Relationships in the Relational Model

A many-to-many relationship type M having primary key K_1K_2 is modeled by a relation schema R, with schema(R) = K_1K_2.

⋆ The primary key of R is K_1K_2.

Question. What happens if a relationship type is many-to-one or one-to-one ?

Answer. No new relation schemas need to defined but foreign keys need be present in the appropriate relation schemas.

$\star$ For a many-to-one relationship type from E_2 to E_1, where K_1 is the primary key of E_1,
K_1 must be a subset of schema(R_2), where R_2 models E_2.

$\Rightarrow$ K_1 is a foreign key for R_2 referencing K_1 in R_1, where R_1 models E_1.

Querying a Relational Database

Three query languages:

- **SQL** (Structured Query Language) - declarative,
  used in commercial DBMSs [IBM mid 1970's]
- **Relational Algebra** - procedural,
  used as a theoretical tool, operations incorporated into SQL [Codd 1970, IBM]
- **QBE** (Query By Example) - graphical,
  used in MS ACCESS [IBM mid 1970's]

All three query languages are equivalent !

Definition of a computable query:

A *program* which takes a collection of tables as input as returns a table as its output.

Definition of a query:

A computable query which is restricted by disallowing *iteration* and *recursion* programming structures.

Alternative definition of a query:

The set of computable queries that can express *logical operations* on tables.

Types of operation:

- Projection
- Selection
- Join
- Union, Intersection and Difference
- Aggregation

Also,

- Create a table
- Modify a table
- Transaction management

Supplier-Parts-DB (Draw the ERD for the Database)

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Supplier table

PNUM	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Part table

SNUM	PNUM	QTY
S1	P1	900
S2	P3	3100
S2	P5	100
S3	P3	200
S3	P4	500
S4	P6	600
S5	P1	100
S5	P2	300
S5	P3	200
S5	P4	800
S5	P5	1000
S5	P6	700

Supply table

Projection

`Select` SNAME, CITY `From` Supplier

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

⇓

SNAME	CITY
Smith	London
Jones	Paris
Blake	Paris
Clark	London
Adams	Athens

Projection

`Select` CITY `From` Supplier

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

⇓

CITY
London
Paris
Athens

Selection

`Select` * `From` Supplier `Where` City = 'London'

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

⇓

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S4	Clark	20	London

Selection

`Select` SNAME, CITY `From` Supplier `Where` STATUS > 20

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

⇓

SNAME	CITY
Blake	Paris
Adams	Athens

Selection

`Select` SNAME, CITY `From` Supplier
`Where` STATUS > 20 `AND` CITY = 'Paris'

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

⇓

SNAME	CITY
Blake	Paris

Selection Condition

A boolean expression may contain the following operators:

(1) `AND`: CITY = 'London' `AND` COLOUR = 'Red'

(2) `NOT`: `NOT` COLOUR = 'Red'
Equivalently: COLOUR <> 'Red'

(3) `OR`: COLOUR = 'Red' `OR` COLOUR = 'Blue'

Join

`Select` * `From` Small-Supplier, Small-Supply
`Where` Small-Supplier.SNUM = Small-Supply.SNUM

Small-Supplier Table

SNUM	SNAME	CITY
S1	Smith	London
S2	Jones	Paris

Small-Supply Table

SNUM	PNUM	QTY
S1	P1	900
S2	P3	3100
S2	P5	100

⇓

SNUM	SNAME	CITY	PNUM	QTY
S1	Smith	London	P1	900
S2	Jones	Paris	P3	3100
S2	Jones	Paris	P5	100

Join

`Select` * `From` Small-Part, Small-Supply
`Where` Small-Part.PNUM = Small-Supply.PNUM

Small-Part Table

PNUM	PNAME	COLOUR
P1	Nut	Red
P3	Screw	Blue
P5	Cam	Blue

Small-Supply Table

SNUM	PNUM	QTY
S1	P1	900
S2	P3	3100
S2	P5	100

⇓

PNUM	PNAME	COLOUR	SNUM	QTY
P1	Nut	Red	S1	900
P3	Screw	Blue	S2	3100
P5	Cam	Blue	S2	100

Join

`Select` * `From` Small-Supplier, Small-Part, Small-Supply
`Where` Small-Supplier.SNUM = Small-Supply.SNUM
`AND` Small-Part.PNUM = Small-Supply.PNUM

Small-Supplier Table

SNUM	SNAME	CITY
S1	Smith	London
S2	Jones	Paris

Small-Part Table

PNUM	PNAME	COLOUR
P1	Nut	Red
P3	Screw	Blue
P5	Cam	Blue

Small-Supply Table

SNUM	PNUM	QTY
S1	P1	900
S2	P3	3100
S2	P5	100

⇓

SNUM	SNAME	CITY	PNUM	PNAME	COLOUR	QTY
S1	Smith	London	P1	Nut	Red	900
S2	Jones	Paris	P3	Screw	Blue	3100
S2	Jones	Paris	P5	Cam	Blue	100

Nested Queries

`SELECT` SNUM, QTY
`FROM` Supply
`WHERE` PNUM `in`
(`SELECT` PNUM `FROM` Part `WHERE` PNAME = 'Screw')

Part and Supply Tables

⇓

SNUM	QTY
S2	3100
S3	200
S3	500
S5	200
S5	800

`SELECT` SNUM, QTY
`FROM` Parts p, Supply sp
`WHERE` p.PNUM = sp.PNUM `and` PNAME = 'Screw'

Not the lowest weight

`SELECT` PNAME
`FROM` Part
`WHERE` WEIGHT > `any`
(`SELECT` WEIGHT `FROM` Part)

Part Table

⇓

PNAME
Bolt
Screw
Cog

The lowest weight

```
SELECT PNAME
FROM Part
WHERE WEIGHT <= all
      (SELECT WEIGHT FROM Part)
```

Part Table

⇓

PNAME
Nut
Cam

Union

```
Select * From London-Supplier
         Union
Select * From Paris-Supplier
```

London-Supplier		
SNUM	SNAME	STATUS
S1	Smith	20
S4	Clark	20

Paris-Supplier		
SNUM	SNAME	STATUS
S2	Jones	10
S3	Blake	30

⇓

London-Union-Paris		
SNUM	SNAME	STATUS
S1	Smith	20
S4	Clark	20
S2	Jones	10
S3	Blake	30

Intersect

```
Select * From London-Rome-Colour-Part
         Intersect
Select * From Paris-Colour-Part
```

London-Rome-Colour-Paris
COLOUR
Red
Blue

Paris-Colour-part
COLOUR
Green
Blue

⇓

London-Rome-Intersect-Paris
COLOUR
Blue

Difference

```
Select * From London-Rome-Colour-Part
         Minus
Select * From Paris-Colour-Part
```

London-Rome-Colour-Paris
COLOUR
Red
Blue

Paris-Colour-part
COLOUR
Green
Blue

⇓

London-Rome-Minus-Paris
COLOUR
Red

Symmetric Difference

(`Select` * `From` London-Rome-Colour-Part
`Minus`
`Select` * `From` Paris-Colour-Part)
`Union`
(`Select` * `From` Paris-Colour-Part
`Minus`
`Select` * `From` London-Rome-Colour-Part)

London-Rome-Colour-Paris
COLOUR
Red
Blue

Paris-Colour-part
COLOUR
Green
Blue

⇓

London-Rome-Symmetric-Paris
COLOUR
Red
Green

Aggregation

Q1 How many suppliers are listed?

Q2 How many suppliers are listed in each city?

Q3 What is the overall average and standard deviation of the weight of parts?

Q4 What is the average weight of parts of a given colour?

Q5 What is the maximum, respectively minimum, quantity a supplier has in stock?

Q6 What is the sum of quantities of parts each supplier has in stock?

The most common aggregate functions are:
`min`, `max`, `sum`, `avg`, `stddev` and `count`

⋆ Beware of `null`s in numeric aggregates !

`SELECT sum`(QTY) TOTAL
`FROM` Supply

⇓

TOTAL
8500

`SELECT avg`(QTY) TOTAL
`FROM` Supply

⇓

TOTAL
708.33

```
SELECT count(*)
FROM Supply
```

⇓

count(*)
12

```
SELECT count(SNUM)
FROM Supply
```

⇓

count(SNUM)
5

```
SELECT sum(QTY * WEIGHT) TOT-WEIGHT
FROM Part, Supply
WHERE Part.PNUM = Supply.PNUM
```

⇓

TOT-WEIGHT
132700

```
SELECT PNAME
FROM Part
WHERE WEIGHT >
      (SELECT avg(WEIGHT) FROM Part)
```

Part Table

⇓ avg = 15.2

PNAME
Bolt
Screw
Cog

```
SELECT PNUM, sum(QTY) TOTAL
FROM Supply
GROUP BY PNUM
```

⇓

PNUM	TOTAL
P1	1000
P2	300
P3	3500
P4	1300
P5	1100
P6	1300

```
SELECT PNUM, sum(QTY) TOTAL
FROM Supply
GROUP BY PNUM
HAVING sum(QTY) >= 1000
```

⇓

PNUM	TOTAL
P3	3500
P4	1300
P5	1100
P6	1300

```
SELECT PNUM, sum(QTY) TOTAL
FROM Supply
GROUP BY PNUM
HAVING sum(QTY) >
        (SELECT avg(QTY) FROM Supply)
ORDER BY sum(QTY)
```

avg(QTY)

⇓

PNUM	TOTAL
P1	1000
P5	1100
P4	1300
P6	1300
P3	3500

Supplier-Parts-DB

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Supplier table

PNUM	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Part table

SNUM	PNUM	QTY
S1	P1	900
S2	P3	3100
S2	P5	100
S3	P3	200
S3	P4	500
S4	P6	600
S5	P1	100
S5	P2	300
S5	P3	200
S5	P4	800
S5	P5	1000
S5	P6	700

Supply table

Creating Relation Schemas

```
CREATE TABLE Supplier (
  SNUM char(5),
  SNAME varchar(20) NOT Null unique,
  STATUS number(2) default 0
         check(STATUS ≥ 0),
  CITY varchar(15),
  primary key(SNUM))
```

```
CREATE TABLE Part (
  PNUM char(5),
  PNAME varchar(20) NOT Null
  COLOUR varchar(7) default 'Blue',
  WEIGHT number(2) default 10
         check(WEIGHT IS Null OR WEIGHT ≥ 10),
  CITY varchar(15) default 'London',
  primary key(PNUM))
```

```
CREATE TABLE Supply (
  SNUM char(5),
  PNUM char(5),
  QTY number(5) default '0'
        check(QTY ≥ 0),
  primary key(SNUM, PNUM),
  foreign key(SNUM) references Supplier,
  foreign key(PNUM) references Part)
```

Changing a Relation Schema

```
ALTER TABLE Supply ADD(
  ORDERED number(5))
```

```
ALTER TABLE Supply MODIFY(
  ORDERED number(4),
          check(ORDERED ≥ 0))
```

```
ALTER TABLE Supply DROP(ORDERED)
```

```
DROP TABLE Supply
```

Supplier-Parts-DB (Draw the ERD for the Database)

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Supplier table

PNUM	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Part table

SNUM	PNUM	QTY
S1	P1	900
S2	P3	3100
S2	P5	100
S3	P3	200
S3	P4	500
S4	P6	600
S5	P1	100
S5	P2	300
S5	P3	200
S5	P4	800
S5	P5	1000
S5	P6	700

Supply table

Modifying a Relational Database

```
CREATE TABLE Supplier (
  SNUM char(5),
  SNAME varchar(20) NOT Null unique,
  STATUS number(2) default 0
         check(STATUS ≥ 0),
  CITY varchar(15),
  primary key(SNUM))

INSERT INTO Supplier values ('S6', 'Bloggs', 20, 'London')
INSERT INTO Supplier (SNUM, SNAME) values ('S6', 'Bloggs')
```

```
CREATE TABLE Light-Part (
  PNUM char(5),
  PNAME varchar(20) NOT Null
  WEIGHT number(2) default 0,
         check (WEIGHT IS Null OR WEIGHT <= 15),
  CITY varchar(15) default 'London',
  primary key(PNUM))

INSERT INTO Light-Part
      SELECT *
      FROM Part
      WHERE PRICE <= 15 OR WEIGHT IS Null
```

```
UPDATE Supplier
SET STATUS = 20,
       CITY = 'London'
WHERE SNAME = 'Blake'

UPDATE Part
SET WEIGHT = WEIGHT - 3
WHERE PNAME = 'Cog'

DELETE FROM Parts
WHERE WEIGHT IS Null
```

Creating Views

A **view** is a *virtual relation* that is computed at run-time as the result of a relational query.

```
CREATE VIEW Sup-Qty as
SELECT s.SNUM, SNAME, CITY,
        sum(QTY) TOTAL
FROM Supplier s, Supply sp
WHERE s.SNUM = sp.SNUM
GROUP BY SNUM

SELECT SNAME, TOTAL
FROM Sup-Qty
WHERE CITY = 'London'

DROP VIEW Sup-Qty
```

Transactions

A **transaction** is a sequence of one or more SQL queries that act as an *atomic* unit with respect to *recovery*.

Example: transferring a sum of money from one account to another.

Example: recording election results in a database.

Commands:

- `COMMIT` makes all changes to the database permanent.
- `ROLLBACK` undoes all changes since the last `COMMIT`.

Notes:

- When you exit an SQL session the changes are committed.
- `CREATE`, `ALTER` and `DROP TABLE` are automatically committed.

Integrity Constraints in the Relational Model

Definition. Integrity constraints are logical statements that restrict the set of allowable relations in a database.

Example.

- database schema, $\mathbf{R} = \{\text{EMP, DEPT}\}$, with
- schema(EMP) = {ENAME, DNAME, ADDRESS, POSTCODE, LOC}
- schema(DEPT) = {DNAME, MNAME, NO_EMPS, LOC}.
- database, d = $\{r_1, r_2\}$ OVER $\mathbf{R}$,
- r_1 is a relation over EMP, and
- r_2 is a relation over DEPT.

Functional Dependencies

- Stating that ENAME is a **key** of EMP, means that no two distinct tuples in r_1 have the same ENAME.
- Stating that DNAME is a **key** of DEPT, means that no two distinct tuples in r_2 have the same DNAME.
  - ⋆ Keys are special cases of **Functional Dependencies** (**FDs**).
- An example of an FD which is not the result of a key, is the constraint that an ADDRESS has a unique POSTCODE.

Inclusion Dependencies

- Stating that DNAME in EMP is a foreign key referencing the key DNAME in DEPT, means that whenever there is a tuple in r_1 with a nonnull DNAME-value, say *mark*, then there is a corresponding tuple in r_2 whose DNAME-value is also *mark*.
  - ⋆ Foreign keys are special cases of **Inclusion Dependencies** (**INDs**).
- An example of an IND which is not the result of a foreign key, is the constraint that the LOCation an employee works in is included in the LOCations of the departments.

Definition. Constraints that depend on the equality or inequality of values in tuples of relations are called **data dependencies**.

• FDs and INDs are data dependencies.

Definition. Constraints that restrict the allowable domain values are called **domain dependencies** (**DDs**).

• An example of a DD is that SALARY ranges between 15 and 40.

• Another example of a DD is that ENAME is a string of at most 25 characters.

Definition. Constraints that restrict the cardinality of a projection of a relation onto a set of attributes are called **cardinality constraints** (**CCs**).

- An example of a CC is that there should not be more managers than employees.
- Another example of a CC that the number of students doing the MSc course should not exceed 100.

An example database.

R is a relation schema, with
schema(R) = {ENAME, DNAME, MNAME}
Relation r over R is given by

ENAME	*DNAME*	*MNAME*
Mark	*Computing*	*Steve*
Angela	*Computing*	*Steve*
Graham	*Computing*	*Steve*
Paul	*Math*	*Donald*
George	*Math*	*Donald*

S is a relation schema, with
schema(S) = {ENAME, CNAME, SAL}
Relation s over S is given by

ENAME	*CNAME*	*SAL*
Jack	*Jill*	25
Jack	*Jake*	25
Jack	*John*	25
Donald	*Dan*	30
Donald	*David*	30

Functional Dependencies

Definition. A FD over R is a statement of the form

$$R : X \rightarrow Y \text{ (or simply } X \rightarrow Y)$$

where X and Y are subsets of schema(R).

- R : {ENAME} → {DNAME,MNAME},
  an employee works in one department and has one manager.
- S : {ENAME} → {SAL},
  an employee has one salary.

Definition. A FD X → Y is **satisfied** in a relation r, if whenever two rows in r have the same X-value they also have the same Y-value.

Alternative definition. An FD X → Y is **satisfied** in a relation r, if for each X-value of r there is at most one Y-value.

NAME → AGE is satisfied in the following relation:

NAME	*AGE*	*CHILD*
Jack	20	*John*
Jack	20	*Jane*

NAME → AGE is violated in the following relation:

NAME	*AGE*	*CHILD*
Jack	20	*John*
Jack	30	*Jane*

Definition. A relation r over R **satisfies** a set of FDs F over R, if all the FDs in F are satisfied in r.

Let F_1 = {{ENAME} → {DNAME},
{DNAME} → {MNAME}} be a set of FDs over R_1.

It can be verified that r satisfies F_1.

Let F_2 = {ENAME → SAL} be a set of FDs over R_2.

It can be verified that s satisfies F_2.

An example of one FD satisfied and the other violated.

NAME → AGE is satisfied in the following relation, but
AGE → CHILD and NAME → CHILD are violated:

NAME	*AGE*	*CHILD*
Jack	20	*John*
Jack	20	*Jane*

Definition. A set of attributes X contained in schema(R) is a **superkey** for a relation r over R if r satisfies X → schema(R).

Definition. A set of attributes X contained in schema(R) is a **key** for r if

(i) X is a superkey for R, and

(ii) for **no** proper subset Y of X, is Y a superkey for r.

⋆ What are the superkeys and keys for r and s ?

Let schema(R) = SPJ :

- S stands for student,
- J stands for subject and
- P stands for position.

Let F be the following set of FDs over R :

- SJ → P, i.e. every student has one position in each subject.
- PJ → S, i.e. every position has one student in each subject.

⋆ What are the superkeys and keys of relations over R that satisfy F ?

A comprehensive example.

Let $\mathcal{R}$ be a relation schema with schema($\mathcal{R}$) = CTHRSG :

- C stands for a course,
- T stands for a teacher,
- H stands for hour,
- R stands for room,
- S stands for student and
- G stands for grade.

$\mathcal{F}$ is the following a set of FDs over $\mathcal{R}$:

- C → T, i.e. a course has one teacher.
- HR → C, i.e. a room can only only have one course at any time.
- HT → R, i.e. a teacher can only be in one room at any time.
- CS → G, i.e. a student has one grade per course.
- HS → R, i.e. a student can only be in one room at any time.

⋆ What are the superkeys and keys for this example ?

Definition. The **closure** of X with respect to F, denoted by CLOSURE(X, F), is the sets of all attributes Y such that Y can be shown to be dependent on X.

The closure of X is computed by the following algorithm:

Algorithm 3 (CLOSURE(X, F))

```
1.  begin
2.    Cl := X;
3.    Done := false ;
4.    while not Done do
5.      Done := true;
6.      for each W → Z in F do
7.        if W is a subset of Cl and Z is not a subset of Cl then
8.          Cl := Cl union Z;
9.          Done: = false;
10.       end if
12.     end for
13.   end while
14.   return Cl;
15. end.
```

For our example we have:

- CLOSURE(C, $\mathcal{F}$) = CT.
- CLOSURE(HR, $\mathcal{F}$) = HRCT.
- CLOSURE(RSG, $\mathcal{F}$) = RSG.
- CLOSURE(HRSG, $\mathcal{F}$) = HRSGCT.

Alternative Definition. A set of attributes X contained in schema(R) is a **superkey** for R with respect to F if

$$\text{CLOSURE(X, F)} = \text{schema(R)}.$$

⋆ what are the keys and superkeys of our comprehensive example?

Let R be a relation schema with schema(R) = $\{A_1, A_2, A_3, B_1, B_2, B_3, C\}$.

Let F = $\{A_1 \rightarrow B_1, A_2 \rightarrow B_2, A_3 \rightarrow B_3, B_1 \rightarrow A_1, B_2 \rightarrow A_2, B_3 \rightarrow A_3, \{B_1, B_2, B_3\} \rightarrow C\}$,

⋆ How many keys does R have with respect to F and what are they?

Definition. A set of FDs F over R is a **cover** of a set of FDs G over R if they are equivalent with respect to the closures of the left-hand sides of their FDs.

Informal Definition. A set of FDs F is a **cover** of a set of FDs G if their semantics are equivalent, i.e. either F and G can be used to model the application.

Example.

Let schema(R) = EPL :

- E stands for employee,
- P stands for project and
- L stands for location.

Let F_1 be the following set of FDs over R :

- E $\rightarrow$ P, ie. an employee works on one project.
- P $\rightarrow$ L, i.e. a project is situated in one location.

$G_1 = \{\text{E} \rightarrow \text{P}, \text{P} \rightarrow \text{L}, \text{E} \rightarrow \text{L}\}$ is a cover of F_1.

$\star$ G_1 has the **redundant** FD E $\rightarrow$ L, which can be derived from F_1.

Let $F_2 = \{\text{E} \rightarrow \text{PL}\}$ be a set of FDs over R.

$\star$ F_2 is *not* a cover of F_1, why?

$G_3 = \{\text{E} \rightarrow \text{P}, \text{E} \rightarrow \text{L}\}$ is a cover of F_2.

$\star$ G_3 has *more* FDs than F_2, i.e. it is *not* **minimum**.

$G_4 = \{\text{E} \rightarrow \text{P}, \text{EP} \rightarrow \text{L}\}$ is a cover of F_2.

$\star$ The FD EP $\rightarrow$ L is *not* **reduced**, since E $\rightarrow$ L can be derived from G_4.

Let schema(R) = TNOML :

- T stands for teacher social security number,
- N stands for teacher name,
- O stands for office number,
- M stands for day and hour ,
- L stands for lecture theatre code.

Let F be the following set of FDs over R :

- NO $\rightarrow$ T, a name and office is associated with one teacher.
- T $\rightarrow$ NO, i.e. a teacher has one name and office.
- TM $\rightarrow$ L, i.e. a teacher is in one lecture at a given time.

G = {NO $\rightarrow$ T, T $\rightarrow$ NO, NOM $\rightarrow$ L} is a cover of F.

$\star$ The number of attributes in G is *greater* than the number of attributes in F, i.e. it is *not* **optimum**.

Relational Database Design

Update Anomalies in Relational Databases

There are two interconnected problems which lead to a bad database design:

- Update anomalies.
- Redundancy problems.

Example 1.
Let $F_1 = \{\text{E} \rightarrow \text{D}, \text{D} \rightarrow \text{M}, \text{M} \rightarrow \text{D}\}$.

- E stands for ENAME,
- D stands for DNAME, and
- M stands for MNAME,

$\star$ E is the only key for EMP_1 with respect to F_1.

A relation r_1 over EMP_1:

$ENAME$	$DNAME$	$MNAME$
$Mark$	$Computing$	$Peter$
$Angela$	$Computing$	$Peter$
$Graham$	$Computing$	$Peter$
$Paul$	$Math$	$Donald$
$George$	$Math$	$Donald$

Problems with EMP_1 and F_1:

1. Due to entity integrity we cannot insert a tuple with a null ENAME; such a *problem* is called an **insertion anomaly**.
2. For the same reason as (1) we cannot delete all the employees and keep just the department information;
such a *problem* is called a **deletion anomaly**.

3. E.g. modifying "Peter" to "Philip" in the second tuple, does not violate any FD resulting from a key but D $\rightarrow$ M would be violated (D is not a key for EMP_1 with respect to F_1);
such a *problem* is called a **modification anomaly**.
4. E.g., same as (3) but this time we modify "Computing" in the first tuple to "Math".

$\star$ In (3) and (4) it is **not** sufficient to check that r_1 satisfies the FDs resulting from the keys of R with respect to F.

$\Rightarrow$ Ideally, we would like **all** the FDs of a relation schema to be inferred from **key dependencies**, i.e. FDs of the form

$$\text{K} \rightarrow \text{schema(R)},$$

where K is a key for R with respect to F.

5. There is **redundancy** in r_1, i.e. for every employee in a given department MNAME is repeated.

$\star$ "Peter" appears three times for "Computing" and "Donald" twice for "Math".

Example 2.

Let $F_2 = \{\text{E} \rightarrow \text{S}\}$.

1. E stands for ENAME,
2. S stands for SAL, and
3. C stands for CNAME.

$\star$ EC is the only key for EMP_2 with respect to F_2.

A relation r_2 over EMP_2:

$ENAME$	$CNAME$	SAL
$Jack$	$Jill$	25
$Jack$	$Jake$	25
$Jack$	$John$	25
$Donald$	Dan	30
$Donald$	$David$	30

Problems with EMP_2 and F_2:

1. An insertion anomaly is present, since we cannot insert an employee without any children.
2. A deletion anomaly is present, since if there is a mistake and "Donald" does not have any children, we cannot record this fact, with deleting the two tuples for "Donald".
3. A modification anomaly occurs if we try and modify the salary of "Jack" to be 27 instead of 25, since **no** FD resulting from a key will be violated, **but** E $\rightarrow$ S would be violated.
4. A redundancy problem is present, since the salary of every employee is repeated for every child.

Example 3.

Let $F_3 = \{\text{SC} \rightarrow \text{P}, \text{P} \rightarrow \text{C}\}$.

1. S stands for Street,
2. C stands for City, and
3. P stands for Postcode.

$\star$ SC and PS are the two keys for R w.r.t.with respect to F; assume PS is the primary key.

A relation r_3 over ADDRESS:

$Street$	$City$	$Postcode$
$Hampstead\ Way$	$London$	$NW11$
$Falloden\ Way$	$London$	$NW11$
$Oakley\ Gardens$	$London$	$N8$
$Gower\ St$	$London$	$WC1E$
$Amhurst\ Rd$	$London$	$E8$

Problems with ADDRESS and F_3:

1. An insertion anomaly is present, since we cannot insert an address which has not been assigned a postcode.
2. A deletion anomaly is present, since if there is a mistake and a postcode is erroneous, we cannot remove this error without deleting the full address.
3. A modification anomaly occurs if we try and modify the city of the 1st tuple to be "Bristol" instead of "London", since **no** FD resulting from a key will be violated, **but**
   P $\rightarrow$ C would be violated.
4. A redundancy problem is present, since the postcode of every city is repeated for each street.

Formalising Redundancy Problems.

Let R be a relation schema and F be a set of FDs over R.

Definition. The FD resulting from a key K for R with respect to F is K $\rightarrow$ schema(R). Such an FD is called a **key dependency**.

Definition. R has a **redundancy problem** if

1. there exists a relation r over R that satisfies F, and
2. there exists a FD X $\rightarrow$ A in F and two distinct tuples in r that have equal XA values.

• It can be shown that redundancy problems, give rise to update anomalies and vice versa.

⋆ Verify that the schemas of Examples, 1, 2 and 3 have redundancy problems.

Normal Forms

We assume that R is a (1NF) relation schema and that F is a set of FDs over R.

⋆ The normal forms we define restrict the *allowable* FDs in F.

1. An FD X $\rightarrow$ Y is **trivial** if Y is a subset of X; otherwise it is **nontrivial**.
2. An FD X $\rightarrow$ A is **canonical** if it is nontrivial and A is a single attribute.

We will assume that all FDs are canonical.

Note that there is no loss of generality in this approach.

Definition. R is in *Boyce-Codd Normal Form* (BCNF) with respect F if for every FD X $\rightarrow$ A in F, X is a superkey for R with respect F.

BCNF Example 1.

Let schema(R_1) = {STUDENT, POSITION, SUBJECT};

1. S stands for STUDENT,
2. J stands for SUBJECT, and
3. P stands for POSITION.

Let F_1 = {SJ $\rightarrow$ P, PJ $\rightarrow$ S}.

1. What are the superkeys of R_1 with respect to F_1 ?
2. Is R_1 in BCNF with respect to F_1 ?

BCNF Example 2.

Let schema(R_2) = {STREET, CITY, POSTCODE};

1. S stands for STREET,
2. C stands for CITY, and
3. P stands for POSTCODE.

Let F_2 = {SC $\rightarrow$ P, P $\rightarrow$ C}.

1. What are the superkeys of R_2 with respect to F ?
2. Is R_2 in BCNF with respect to F_2 ?

BCNF Example 3.

Let schema(R_3) = {ENAME, DNAME, MNAME};

1. E stands for ENAME,
2. D stands for DNAME, and
3. M stands for MNAME.

Let F_3 = {E $\rightarrow$ D, D $\rightarrow$ M}.

1. What are the superkeys of R_3 with respect to F_3 ?
2. Is R_3 in BCNF with respect to F_3 ?

BCNF Example 4.

Let schema(R_4) = {ENAME, CNAME, SAL};

1. E stands for ENAME,
2. C stands for CNAME, and
3. S stands for SAL.

Let F_4 = {E $\rightarrow$ S}.

1. What are the superkeys of R_4 with respect to F_4 ?
2. Is R_4 in BCNF with respect to F_4 ?

Recall the concept of a **cover** of a set of FDs.

The following result shows that BCNF is cover insensitive.

Result. R is in *Boyce-Codd Normal Form* (BCNF) with respect to F if and only if R is in BCNF with respect to G, where F is a cover of G.

Assume that we do **not** allow FDs of the form $\emptyset \rightarrow$ Y, i.e. *the left-hand of an FD is assumed to be nonempty.*

What is the meaning of such an FD ?

Result. If schema(R) has two or less attributes, then R is in BCNF with respect to F.

Definition. An attribute A in schema(R) is said to be **prime** with respect to F if A is a member of one of the keys of R with respect to F.

Definition. R is in *Third Normal Form* (3NF) with respect to F if for every FD X $\rightarrow$ A in F either X is a superkey for R with respect to F or A is prime.

Result. 3NF is cover insensitive.

3NF Example 1.

Let schema(R_1) = {STUDENT, POSITION, SUBJECT};

1. S stands for STUDENT,
2. J stands for SUBJECT, and
3. P stands for POSITION.

Let F_1 = {SJ $\rightarrow$ P, PJ $\rightarrow$ S}.

1. What are the superkeys and prime attributes of R_1 with respect to F_1 ?
2. Is R_1 in 3NF F with respect to F_1 ?

3NF Example 2.

Let schema(R_2) = {STREET, CITY, POSTCODE};

1. S stands for STREET,
2. C stands for CITY, and
3. P stands for POSTCODE.

Let F_2 = {SC $\rightarrow$ P, P $\rightarrow$ C}.

1. What are the superkeys and prime attributes of R_2 with respect to F ?
2. Is R_2 in 3NF with respect to F_2 ?

3NF Example 3.

Let schema(R_3) = {ENAME, DNAME, MNAME};

1. E stands for ENAME,
2. D stands for DNAME, and
3. M stands for MNAME.

Let F_3 = {E → D, D → M}.

1. What are the superkeys and prime attributes of R_3 with respect to F ?
2. Is R_3 in 3NF with respect to F_3 ?

3NF Example 4.

Let schema(R_4) = {ENAME, CNAME, SAL};

1. E stands for ENAME,
2. C stands for CNAME, and
3. S stands for SAL.

Let F_4 = {E → S}.

1. What are the superkeys and prime attributes of R_4 with respect to F ?
2. Is R_4 in 3NF with respect to F_4 ?

Definition. R is in *Second Normal Form* (2NF) with respect to F if for every FD X → A in F either X is not a proper subset of a key for R with respect to F or A is prime.

Result. 2NF is cover insensitive.

2NF Example 1.

Let schema(R_1) = {STUDENT, POSITION, SUBJECT};

1. S stands for STUDENT,
2. J stands for SUBJECT, and
3. P stands for POSITION.

Let F_1 = {SJ → P, PJ → S}.

1. What are the superkeys and prime attributes of R_1 with respect to F_1 ?
2. Is R_1 in 2NF F with respect to F_1 ?

2NF Example 2.

Let schema(R_2) = {STREET, CITY, POSTCODE};

1. S stands for STREET,
2. C stands for CITY, and
3. P stands for POSTCODE.

Let F_2 = {SC $\rightarrow$ P, P $\rightarrow$ C}.

1. What are the superkeys and prime attributes of R_2 with respect to F ?
2. Is R_2 in 2NF with respect to F_2 ?

2NF Example 3.

Let schema(R_3) = {ENAME, DNAME, MNAME};

1. E stands for ENAME,
2. D stands for DNAME, and
3. M stands for MNAME.

Let F_3 = {E $\rightarrow$ D, D $\rightarrow$ M}.

1. What are the superkeys and prime attributes of R_3 with respect to F ?
2. Is R_3 in 2NF with respect to F_3 ?

2NF Example 4.

Let schema(R_4) = {ENAME, CNAME, SAL};

1. E stands for ENAME,
2. C stands for CNAM,E and
3. S stands for SAL.

Let F_4 = {E $\rightarrow$ S}.

1. What are the superkeys and prime attributes of R_4 with respect to F ?
2. Is R_4 in 2NF with respect to F_4 ?

Some fundamental Results.

Result0. R is in BCNF with respect to F if and only if R has no redundancy problems.

Result1. If R is in BCNF with respect to F, then R is in 3NF with respect to F.

Result2. If R is in 3NF with respect to F, then R is in 2NF with respect to F.

Some interesting results.

Recall A key is **simple** if it consists of a single attribute.

Result1. If R is in 3NF with respect to F and all the keys of R with respect to F are simple, then R is also in BCNF.

Result 2. If R is in 3NF with respect to F and there is a unique key for R with respect to F (i.e. R has only one key with respect to F) then R is in BCNF with respect to F.

A BCNF Normalisation Algorithm

Input:

- A *specification* containing:
  1. a relation schema, R, and
  2. a set of Functional Dependencies (FDs), F over R.
- An Entity-relationship Diagram (ERD) conforming to the specification.

Notes:

- schema(R) is called the *universal set of attributes*.
- We assume the ERD satisfies the URSA.

Output: A database schema **R** which is in BCNF with respect to F.

Notes:

- $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$, where each R_i is a subset of schema(R).
- The union of all schema(R_i) is schema(R).
- **R** is called a **decomposition** of schema(R).

Strategy

Step 1. Convert ERD into a database schema, **S**.

Step 2. If any of the relation schemas in **S** are *not* in BCNF with respect to F, then decompose them further, using the DECOMPOSE algorithm, given in this lecture.

Algorithm 4 (ERD-TO-BCNF(ERD, F))

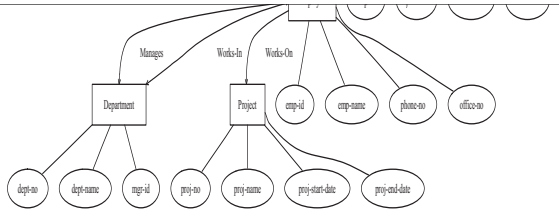
1. **begin**
2. Convert ERD into a database schema $\mathbf{S} = \{S_1, S_2, \ldots, S_m\}$;
3. **let** the output database schema $\mathbf{R}$ be empty;
4. **for each** S_i in $\mathbf{S}$ **do**
5. **if** S_i is in BCNF with respect to F
6. **add** S_i to $\mathbf{R}$;
7. **else**
8. **merge** DECOMPOSE(S_i, F) and $\mathbf{R}$;
9. **end for**;
10. **return** the decomposition $\mathbf{R}$;
11. **end.**

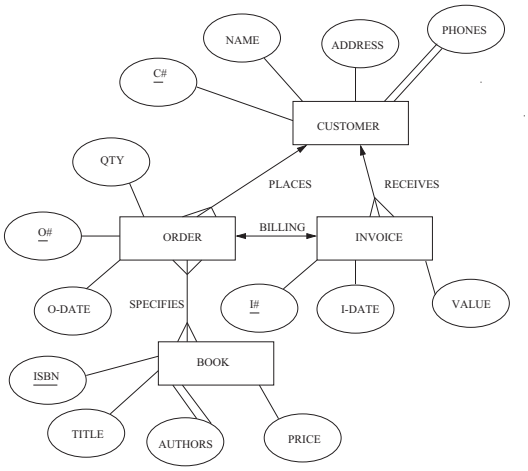
Example Entity Relationship Diagrams

ERD for Employee Database

ERD for Online-Bookshop Database

Convert these to database schemas each with a set of FDs.





NEXT

Testing whether a database schema is in BCNF

Algorithm 5 (TEST-BCNF(R, F))

```
1. begin
2.   for each X → Y in F do
3.     if X is not a superkey with respect to F
4.       return NO;
5.     end if;
6.   end for;
7.   return YES;
8.  end.
```

Decomposition Condition used by the Algorithm

- Let schema(PHONE) = {cust-name, phone-num, phone-network}.
- Let F = {cust-name → phone-num, phone-num → phone-network}.

Is PHONE in BCNF with respect to F ?

The Idea

So phone-num → phone-network **violates** BCNF.

- phone-num is the **left-hand** side of the violating FD and
- phone-network is the **right-hand** side of the violating FD.

Split PHONE into two relation schemas:

1. R_1 = NETWORK, with schema(NETWORK) = {phone-num, phone-network}, containing the attributes in the left and right hand sides of the **violating** FD, and

   F_1 containing phone-num → phone-network.

2. R_2 = CUST, with schema(CUST) = {cust-name, phone-num}, containing the attributes in schema(PHONE) **except** those in the right-hand side of the **violating** FD (and not in its left-hand side), and

   F_2 containing cust-name → phone-num.

Algorithm 6 (DECOMPOSE(R, F))

```
1. begin
2.   let the output database schema Out be empty;
3.   if TEST-BCNF(R, F) = YES then
4.     add R to Out;
5.   else
6.     let X → Y in F be nontrivial (i.e. Y is not a subset of X)
         such that X is not a superkey with respect to F;
7.     let R1 be a relation schema,
         with schema(R1) = X merged with Y;
8.     merge DECOMPOSE(R1, F) and Out;
9.     let R2 be a relation schema,
         with schema(R2) = schema(R) except
         the attributes that are in Y and not in X;
10.    merge DECOMPOSE(R2, F) and Out;
11.  end if
12.    return Out;
13. end.
```

Result. DECOMPOSE(R, F) returns a decomposition of schema(R).

Result. The relations over DECOMPOSE(R, F) can be joined naturally during query processing; this is known as the **lossless join** criterion.

An Example

Let STUD be a relation schema, with schema(STUD) = {SNUM, POSTCODE, CITY, COUNTRY}, with FDs
{SNUM → POSTCODE, POSTCODE → CITY,
CITY → COUNTRY}

- CITY → COUNTRY **violates** BCNF in STUD, so *decompose* STUD into

  CC, with schema(CC) = {CITY, COUNTRY}, and

  STUD1, with schema(STUD1) = {SNUM, POSTCODE, CITY}
- CC is in BCNF while POSTCODE → CITY violates BCNF in STUD1, so *decompose* STUD1 into

  PC, with schema(PC) = {POSTCODE, CITY}, and

  SINFO = {SNUM, POSTCODE}.
- All the relation schemas in the database schema {CC, PC, SINFO} is now in BCNF with respect to the specification.

Another Example

Let $\mathcal{R}$ be a relation schema, with schema($\mathcal{R}$) = {C,T,H,R,S,G}.

- C stands for a course,
- T stands for a teacher,
- H stands for hour,
- R stands for room,
- S stands for student and
- G stands for grade.

An example set of FDs $\mathcal{F}$ over $\mathcal{R}$:

1. C → T,
2. HR → C,
3. HT → R,
4. CS → G and
5. HS → R.

Decompose $\mathcal{R}$ into BCNF.

Object-Relational Databases

Evolution of Database Systems

Graph-Based	**Relational**	**Object-Relational**
1960's-1970's	1980's-1990's	1990's

- Pure object-oriented database systems have failed to capture the database market !!
- The object-relational approach is an evolutionary approach

Why is the relational model so successful

- Data independence
- High level query language - SQL
- Query optimisation
- Support for integrity constraints
- Well-understood database design
- Transaction management and concurrency

Why do we need Object-Oriented Databases

- For some applications 1NF is too strict.
  - ◇ Document management and web site management
  - ◇ Geographic and Statistical database management
  - ◇ Biological data management
- SQL is not a complete programming language such as C++ and Java - cannot implement abstract data types

Object-Oriented Approach

- Couple an Object-Oriented programming language with a DBMS
  - ◇ One approach is to implement on top of a relational DBMS.
- Object-Oriented databases failed to deliver robust and versatile systems which can compete with relational database technology
- Object-Relational approach -
  integrate Object-Oriented features into existing relational DBMSs

Object-Relational Evolution

- *Object identity* - the property that distinguishes between objects
  Object identity should be
  1) Location independent - e.g. physical addresses are *not* location independent
  2) Value independent -
  e.g. relational keys are *not* value independent
  (an object exists independently of its values)
- Object identity allows data sharing
  e.g. a couple having the same children
  - ◇ *Surrogates* are system-generated unique identifiers which are location independent

Simulate Object Identity via Sequences

```
CREATE sequence S#-SEQ
START WITH 1
INCREMENT BY 1
NOCYCLE

INSERT INTO suppliers values
(S#-SEQ.nextval, 'Bloggs', 20, 'London')

DROP sequence S#-SEQ

ALTER sequence S#-SEQ
MAXVALUE 1000
```

Complex Objects

- An object which may violate 1NF is called a *complex object*

```
CREATE TABLE person (
  PID#       number(5),
  PNAME      varchar(20) not null,
  PDOB       date,
  PADDRESS   address,
  CHILDREN   SET OF(children),
  primary key(PID#))
```

```
CREATE TABLE address (
  HOUSE#     number(4),
  STREET     varchar(20),
  POSTCODE   char(8),
  CITY       varchar(15),
  primary key(HOUSE#, STREET))


CREATE TABLE child (
  CID#    number(5),
  CNAME   varchar(20) not null,
  CDOB    date,
  primary key(CID#))
```

Inheritance

- Is a natural way of organising knowledge
- Economical - we can reuse objects

```
CREATE TABLE employee UNDER person (
  SAL          number(5),
  WORKS_FOR    company)


CREATE TABLE student UNDER person (
  DEGREE    varchar(10),
  DNAME     varchar(20),
  PLACE     college)
```

Retrieving Information from the Web

Database and Information Retrieval (IR) Systems both manage data !

• The data of an IR system is a *collection of documents* (or *pages*)

User tasks:

◇ *Browsing* - examining documents

◇ *Retrieval* - searching for documents

	Database System versus IR System	
Category	**SQL**	**Search Engine**
Matching	exact answer	ranked answer
Language	sophisticated	simple
Algorithm	deterministic	probabilistic
Database	structured	semistructured
Query	complete	incomplete
Error	sensitive	insensitive

The Web is a Hypertext System

• *Content* - collection of pages

• *Structure* - links (directed graph)

Additional user task:

◇ *Navigation* - traversing links and following a *trail* of associated links

Quote from Bush 1945 "As We May Think" (download from my web links): "the process of tying two items together is an important thing .. when numerous items have been thus joined together to form a trail they can be reviewed in turn"

Nelson's vision of a *universal hypertext database* - *Xanadu* (1960's)

The Basic Information Retrieval Algorithm

1. Remove stopwords such as: of, the, a ...
2. Apply stemming to *terms* (or words),

i.e. remove prefixes and suffixes

E.g. connected, connecting, connection and connections ⇒ connect

3. Weight the terms in the query and in pages
4. Rank the pages according to similarity with the query

Term Weighting

N - total no. of pages in the system

n_j - no. pages in which term j appears

term frequency

tf_{ij} = frequency of term j in page i

inverse document frequency

$idf_j = -\log \frac{n_j}{N} = \log \frac{N}{n_j}$

(self-information of term j)

normalised term weight

$w_{ij} = \frac{tf_{ij} \times idf_j}{\max_k \ tf_{ik}}$

Query Weighting

$w_{qj} = \left(0.5 + \frac{0.5 \times tf_{qj}}{\max_k \ tf_{qk}}\right) \times idf_j$

Similarity
m - no. of terms considered
Represent page i as a vector
$\mathbf{i} = \langle w_{i1}, w_{i2}, \ldots, w_{im} \rangle$
Represent query q as a vector
$\mathbf{q} = \langle w_{q1}, w_{q2}, \ldots, w_{qm} \rangle$
$sim(i, q) = \sum_{k=1}^{m} w_{ik} \times w_{qk}$
(dot product of $\mathbf{i}$ and $\mathbf{q}$)
(Other similarity measures exist)

Measures of Information Retrieval
R_F - no. relevant pages returned
R_N - no. relevant pages *not* returned
I_F - no. of irrelevant pages returned
I_N - no. of irrelevant pages *not* returned
$recall = \frac{R_F}{R_F + R_N}$
• Proportion of relevant pages returned.
$precision = \frac{R_F}{R_F + I_F}$
• Proportion of returned pages which are relevant.
◇ Precision versus Recall curve

Searching the Web
♣ over 2 billion pages (2000) growing at 1 million pages per-day.
♣ Each page has on average 7 out-links.
♣ Over 600 GB of text changes every month.
♣ Largest crawlers cover 30-40% of the indexable web during several months.
♣ 10 percent redundancy in mirrored sites.
♠ Most users type in short queries on average less than 3 terms.
♠ Most users only look at the top ten results.
♠ Most users do not modify their original query.

Using Link Structure in Search
$L_{ij} = 1$ if there is a link from i to j and 0 otherwise.
Structured Weighting
$sw_{qj} = w_{qj} + \sum_{k \neq j} \alpha L_{kj} \times w_{qk}$
• α is between 0 and 1 (0.2 seems to be optimal)
• the sum is over all pages that have a link to page j

HITS - Hypertext Induced Topic Search
Given a query such as "XML" distinguish between:
• **authorities** - pages which focus on the topic of XML such as various publications on the XML standard.
• **hubs** - pages that contain many useful links to relevant pages
• A densely linked focused subgraph of hubs and authorities is called a *community*.
• Over 100,000 emerging web communities have been discovered from a web crawl (a process called *trawling*).

The HITS Algorithm
1. Collect the top t pages (say $t = 200$) based on similarity, called the *root set*.
2. Extend the root set into a *base set* as follows, for all pages p in the root set:
2.1. add to the root set all pages that p points to, and
2.2. add to the root set up-to d pages that point to p (say $d = 50$).
3. Delete all links between the same web site in the base set resulting in a *focused subgraph*.
4. Assign to each page p a non-negative *hub weight* y_p and a non-negative *authority weight* x_p.
5. Iteratively reinforce hubs and authorities as follows, until convergence:

$$x_p := \sum_q \text{ where } {}_{q \to p}\, y_q$$
$$y_p := \sum_q \text{ where } {}_{p \to q}\, x_q$$

PageRank - Google
Model of a "random surfer":
1. The surfer given a web page at random.
2. The surfer follows "forward" links without going "back".
3. When the surfer gets bored a random page is chosen as the next page.
• The PageRank of a page is the probability that a random surfer visit a page
P - a page which has incoming links from pages $P_1, P_2, \ldots, P_n$
r - a positive number between 0 and 1
$O(P_i)$ - the number of links going out of page P_i
$PR(P) = r + (1 - r) \sum_{i=1}^{n} \frac{PR(P_i)}{O(P_i)}$

Metasearch
Problem: Search engines have limited coverage and overlap (Nature 1999)
♣ Relative coverage of major search engines about 20%.
♣ The overall coverage is small, less than 16% are indexed by all engines, not taking into account the *deep web*.
Solution: Select and merge results from several data sources
• Not easy to do well due to heterogeneity of local search engines

The Navigation Problem in Hypertext
The steps in searching for information:

1) **Query** - user provides the context
2) **Information Retrieval** - ranked list of pages returned
3) **Navigation** - user *repeats* :
   (a) choose a page to *browse*
   (b) follow a *link*
4) **Query Modification** - user *returns* to (1)

Problem: "getting lost in hyperspace" -
navigation (link following) leads to disorientation in terms of the *goals* and *relevance* of the currently browsed page to the query.
Solution: Trails are first-class citizens
• We develop algorithms which maximise the expected trail relevance.

References

[CB02] T. Connolly and C. Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Addison-Wesley, Harlow, Essex, 3rd edition, 2002.

[Cel00] J. Celko. *Joe Celko's SQL for Smarties: Advanced SQL Programming*. Morgan Kaufmann, San Francisco, Ca., 2nd edition, 2000.

[Che76] P.P.-S. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[LL99] M. Levene and G. Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-Verlag, London, 1999.

[WM00] M. Whitehorn and B. Marklyn. *Accessible Access 2000*. Springer-Verlag, London, 2000.

[WM01] M. Whitehorn and B. Marklyn. *Inside Relational Databases*. Springer-Verlag, London, 2nd edition, 2001.