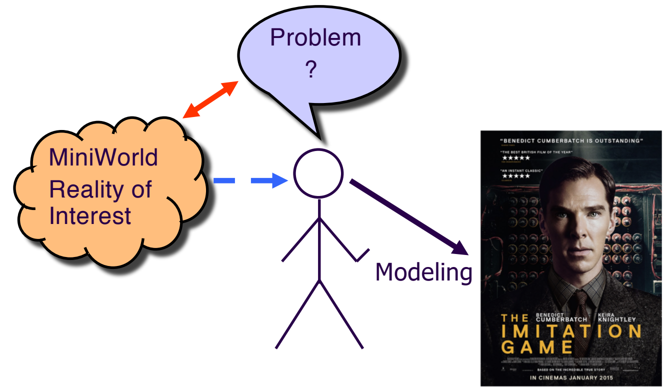
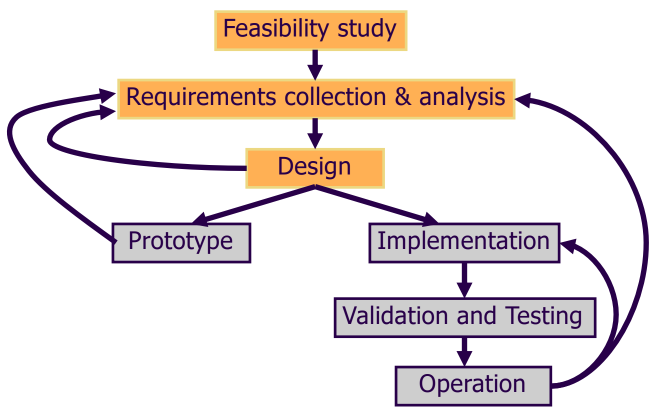
Database System Design

Data Modeling



Database System Life Cycle

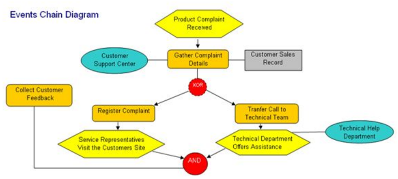


Design

- Database Design
  - Database design is the activity of specifying the schema of a database in a given data model
- Functional Design

Functional Design

- High-level specification of Transactions
  - DBMS-independent
  - Event diagrams, UML



- Application program design
  - DBMS-specific (db Schema together with DML)
  - Language and environment-specific

Relational Database Design: The Good, The Bad and The Ugly

- Bad design & anomalies
- Normal forms
- Universal Relational Approach
- Decomposition – normalization

Relational Database Design

- One single, large table
- Simple ?
- Good ? or Bad? Or just Ugly?

Goal

- Design 'good' tables
  - define what 'good' means
  - fix 'bad' tables
- in short: "we want tables where the attributes depend on the primary key, on the whole key, and nothing but the key"
- Let's see why, and how:

Bad Design

- Relation Schema:

  takes1 (ssn, cid, grade, name, address)
- 'Bad' - why?
- because: ssn → (name, address)

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

Pitfalls

- Redundancy
  - Waste of space (Repeated values, NULL)
  - Update anomalies (inconsistencies)

ssn	cid	grade	name	address
123	413	A	smith	Elm
123	415	B	smith	Main
123	211	A	smith	Main

- Insertion & Deletion anomalies

Insertion Anomaly

- "jones" registers, but takes no class
- Where do you store his address!?!

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main
234	null	null	jones	Forbes

Deletion Anomaly

- delete the last class record of 'smith'
- What about the address !?!
  - (we lose his address!)

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

Solution: decomposition

- Split offending table in two (or more)

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

? ?

Functional dependencies

- How do we split a bad table?
- How do we recognize a bad table?
- Set of rules: *normal forms*
- **Functional Dependencies** (FD)
- Let $R=(A_1, A_2, \ldots, A_n)$ and $X \subseteq R$ and $Y \subseteq R$

  $X \rightarrow Y$ if the value of X **uniquely** determines a value of Y
  - X functionaly determines Y
  - Y is functionally dependent on X

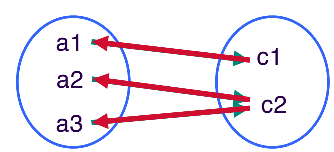
Functional Dependencies

- They are not a property of a particular relation state
  - Property of R, not r(R)
- They cannot be automatically inferred from r(R)
  - It can show which dependencies may exist
  - And show which dependencies cannot exist
  - Must have knowledge of the semantics of the attributes of R for the full story

Functional Dependency Examples I

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b2	c2	d3
a3	b3	c2	d4

- $A \rightarrow C$? YES
- $C \rightarrow A$? NO (line 5)
- $B \rightarrow C$? NO (line 3)
- $D \rightarrow B$? YES
- $AB \rightarrow D$? NO (line 4)



Functional Dependency Examples II

- Loans (loan_number, branch, customer, amount)
  - loan_number → amount   is satisfied
  - loan_number → branch   is satisfied
  - amount → branch   is not satisfied
  - loan_number → customer   ?
- takes1 (ssn, cid, grade, name, address)
  - ssn → (name, address)
  - (ssn, cid) → grade

Decompositions

- There are careless, 'bad' decompositions
- There are correct decompositions
  - lossless
  - dependency preserving

Decomposition: lossless

- Non-lossy decomposition is called lossless or non-additive decompositions
- Definition:

  Let R schema R, with FD **'F'**.

  R1, R2 is a lossless decomposition of R if for all relations r(R), r1(R1) and r2(R2)

  $$r1 \bowtie r2 = r$$

Decomposition: lossy

R1(ssn, grade, name, address)

ssn	grade	name	address
123	A	smith	Main
123	B	smith	Main
234	A	jones	Forbes

R2(cid, grade)

cid	grade
413	A
415	B
211	A

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes
123	**211**	**A**	**Smith**	**Main**
234	**413**	**A**	**Jones**	**Forbes**

ssn → (name, address)

(ssn, cid) → grade ✘

can not recover original table with a join!

Decomposition: lossless…

- What about an easier criterion?
- Theorem:

  A decomposition is lossless if the joining attribute is a part of a *superkey* in at least one of the new tables
- Formally:

$$R1 \cap R2 \rightarrow R1 \ or$$
$$R1 \cap R2 \rightarrow R2$$

Example: Loseless Decomposition

R1

ssn	cid	grade
123	413	A
123	415	B
234	211	A

(ssn, cid) → grade

R2

ssn	name	address
123	smith	Main
234	jones	Forbes

ssn → (name, address)

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn → (name, address)

(ssn, cid) → grade

Dependency Preservation & Canonical Cover

- We don't want the original FDs to span two tables.
- More specifically: … the FDs of the **canonical cover**
- **Canonical Cover** is the minimum set of FDs without any trivial, extraneous and implied FDs
- Example:

  fd.1 A → B
  fd.2 B → C
  fd.3 AB → C

  A → B & B → C
  => AB → C

  - What is the canonical cover?

    Keep only fd.1 & fd.2: A → B & B → C

Rules of Inference: Armstrong's Axioms

- IR1 - Reflexivity rule:
  - if B ⊂ A, then A → B
    - e.g., If **{ssn,name}⊇name** then **{ssn,name} → name**
- IR2 - Augmentation rule:
  - if A → B, then AC → BC
    - e.g., if **ssn → name** then **{ssn,age} → {name,age}**
- IR3 - Transitive rule:
  - if A → B and B → C, then A → C
    - e.g., if **ssn → Dno** and **Dno → Dname** then **ssn → dname**

More Rules of Inference

- IR4 - Decomposition rule:
  - if A → BC, then A → B and A → C
    - e.g., if **ssn →{name, age}** then **ssn → name** and **ssn → age**
- IR5 - Union rule:
  - if A → B and A → C, then A → BC
    - e.g., if **ssn → name** and **ssn → age** then **ssn →{name, age}**
- IR6 - Pseudotransitivity, Composition:
  - if A → B and CB → D implies CA → D
    - e.g., if **isbn → title** and **{author, title} → pubdate** then **{author, isbn} → pubdate**
- IR7 - Self-determination: A → A

Decomposition: non-dependency preserving

- Dependency preserving: the original (canonical) FDs should not span across tables
- Non-dependency preserving example:

S#	address	status
123	London	E
125	Paris	E
234	Pgh	A

S# → (address, status)

address → status

S#	address
123	London
125	Paris
234	Pgh

S#	status
123	E
125	E
234	A

S# → address **S# → status**

quiz: is it lossless?

Dependency Preserving Decomposition

- Example

S#	address	status
123	London	E
125	Paris	E
234	Pgh	A

S# → (address, status)

address → status

S#	address
123	London
125	Paris
234	Pgh

address	status
London	E
Paris	E
Pgh	A

S#→address address→status

but: S# → status ?

◆ why is dependency preservation good?

Why Dependency Preservation?

- Insert the status for Athens to be A:

S#	address	status
123	London	E
125	Paris	E
234	Pgh	A

S# → (address, status)

address → status

S#	address
123	London
125	Paris
234	Pgh
333	Athens

address	status
London	E
Paris	E
Pgh	A
Athens	A

S#→address address→status

Decomposition - conclusions

- decompositions should always be lossless
  - joining attribute -> superkey
- Decompositions should be dependency preserving whenever possible

Testing for Loseless Decomposition

- Given
  - relation schema R (A1, A2, …, An),
  - a set of functional dependencies F and
  - decomposition p = {R1, R2,…, Rm}
- Steps:

1. Construct an m x n table S,

   with a column for each of the n attributes in R and

   row for each of the m relations in the decomposition.

Testing for Loseless Decomposition…

2. For each cell S(i,j) of S,

   If the attribute for the column Aj is in the relation for the row Ri,

   then set S(i,j) = k(j) to indicate *known* value

   else set S(i,j) = u(i,j) to indicate *unknown* value

3. Consider each FD, X → Y F until no more changes can be made to S.
   – Look for rows whose X-column agrees
   – EQUATE Y-column

4. After all possible changes have been made to S,
   – If a row is made up entirely of symbols k(1), k(2), .., k(n) the join is lossless.
   – Otherwise, if there is no such row, the join is lossy.

Example

- R (A,B,C,D,E)
- Decomposition: R1(A,C), R2(A,B,D), R3(D,E)
- FDs: A → C, AB → D, D → E

	A	B	C	D	E
R1(A,C)					
R2(A,B,D)					
R3(D,E)					

Example

- R (A,B,C,D,E)
- Decomposition: R1(A,C), R2(A,B,D), R3(D,E)
- FDs: A → C, AB → D, D → E
- Textbook notation

	A	B	C	D	E
R1(A,C)	α(1,1)	β(1,2)	α(1,3)	β(1,4)	β(1,5)
R2(A,B,D)	α(2,1)	α(2,2)	β(2,3)	α(2,4)	β(2,5)
R3(D,E)	β(3,1)	β(3,2)	β(3,3)	α(3,4)	α(3,5)

Example

- R (A,B,C,D,E)
- Decomposition: R1(A,C), R2(A,B,D), R3(D,E)
- FDs: A → C, AB → D, D → E

	A	B	C	D	E
R1(A,C)	K	U	K	U	U
R2(A,B,D)	K	K	U	K	U
R3(D,E)	U	U	U	K	K

Example

- R (A,B,C,D,E)
- Decomposition: R1(A,C), R2(A,B,D), R3(D,E)
- FDs: A → C, AB → D, D → E

	A	B	C	D	E
R1(A,C)	K	U	K	U	U
R2(A,B,D)	K	K	K	K	U
R3(D,E)	U	U	U	K	K

Example

- R (A,B,C,D,E)
- Decomposition: R1(A,C), R2(A,B,D), R3(D,E)
- FDs: A → C, AB → D, D → E

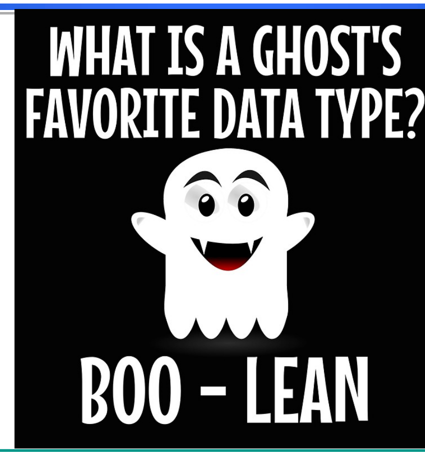
	A	B	C	D	E
R1(A,C)	K	U	K	U	U
R2(A,B,D)	K	K	K	K	K
R3(D,E)	U	U	U	K	K

Example (2)

- R (A,B,C,D,E)
- Decomposition: R1(A,C), R2(A,B,D), R3(D,E)
- FDs: A $\rightarrow$ C, AB $\rightarrow$ D, E $\rightarrow$ D

	A	B	C	D	E
R1(A,C)	K	U	K	U	U
R2(A,B,D)	K	K	K	K	U
R3(D,E)	U	U	U	K	K

Halloween Question?



Happy Halloween