Databases

Functional Dependencies and Normalization I

Dr Konrad Dabrowski
konrad.dabrowski@durham.ac.uk

Online Office Hour:
Mondays 13:30–14:30
See Duo for the Zoom link

Two approaches to database design

- Top-down approach: Entity-Relationship (ER) model
  - a graphical description of the DB
  - start with a set of requirements (informal system description)
  - identify the entities that you need to represent data about
  - identify the attributes of those entities
- At the next step:
  - we construct the Relational Data model (i.e. tables for the entities)
- Bottom-up approach: Normalization
  - start with the initial tables and attributes
  - analyze the relationships among the attributes
  - re-design the tables and attributes in a “better” way
  - this becomes tricky for large databases

$\Longrightarrow$ we need a formalization of this approach

Well-designed relational databases

- No redundancy: every data item is stored only once
  - e.g. keep the address of a customer only in one place!
  - exception: foreign keys (they act as pointers)

  1. minimize the amount of space required
  2. simplify maintenance of the database

- If an item is stored twice (or more), then:
  - every time we update it, we need to change it in many places
  - we may have inconsistencies (e.g. two different values for this item)

- Purpose of normalization:
  - every relation represents a “real world” entity
  - single-valued columns
  - avoid redundancy (i.e. repetitions)
  - data is easy to update correctly (i.e. avoid update anomalies)

Redundancy

- Set-valued attributes in the ER diagram:
  - result in multiple rows in corresponding table
- Example: Person (*NIN, Name, Address, Hobbies*)
  - a person entity with multiple hobbies yields multiple rows in the table "Person"

ER Model:

NIN	*Name*	*Address*	*Hobby*
1111	Joe	123 Main	{biking, hiking}

Relational Model:

NIN	*Name*	*Address*	*Hobby*
1111	Joe	123 Main	biking
1111	Joe	123 Main	hiking

. . . .

Redundancy

Redundancy

- Dependencies between attributes cause redundancy
  - e.g. all addresses in the same town have the same zip code

NIN	*Name*	*Town*	*Zip*
1234	Joe	Stony Brook	11790
4321	Mary	Stony Brook	11790
5454	Tom	Stony Brook	11790
	....		

Redundancy

Data anomalies: terminology

- Redundancy:
  - repeating data in multiple different locations
- Modification anomaly:
  - failure to maintain all existing instances of a specific value
- Deletion anomaly:
  - losing other values as a side effect when you delete data
- Insertion anomaly:
  - when new data items are inserted, we need to add much more irrelevant data
  - adding rows forces us to add information about other entities

Modification anomaly

- Update of one item value $\longrightarrow$ impacts another item value
- Example: a collection of videos at Blockbuster

Video(12, Top Gun, action, 2 hr, PG13)

Video(45, Top Gun, action, 2 hr, PG13)

- Update the genre of first video:
  - comedy
- New table:

Video(12, Top Gun, comedy, 2 hr, PG13)

Video(45, Top Gun, action, 2 hr, PG13)

- What kind of video is “Top Gun” now?

Bad data!

Insertion anomaly

- Inconsistency caused by adding a new item
- Example: a collection of videos at Blockbuster

Video(12, Top Gun, action, 2 hr, PG13)

Video(45, Top Gun, action, 2 hr, PG13)

- Add a new video: Video(46, Top Gun, mystery, 2 hr, PG13)
- Our data now looks like: table:

Video(12, Top Gun, action, 2 hr, PG13)

Video(45, Top Gun, action, 2 hr, PG13)

Video(46, Top Gun, mystery, 2 hr, PG13))

- What kind of video is “Top Gun” now?

Bad data!

Deletion anomaly

- Loss of information when last item is deleted
- Example: a collection of videos at Blockbuster

  Video(12, Top Gun, action, 2 hr, PG13)

  Video(45, Top Gun, action, 2 hr, PG13)

- Remove first copy of Top Gun

$\Rightarrow$ the database table looks like:

  Video(45, Top Gun, action, 2 hr, PG13)

- Now remove the last copy of Top Gun
- Where can you find info about Top Gun in the database?

Nowhere – bad data!

Possible action?

- Decompose the table "Video" into two tables:
  - "Videotape"
  - "MovieInfo"

- Replace:

  Video(12, Top Gun, action, 2 hr, PG13)

  Video(45, Top Gun, action, 2 hr, PG13)

  with this:

  Videotape (12, Top Gun)

  Videotape (45, Top Gun)

  MovieInfo(Top Gun, action, 2 hr, PG13)

$\Longrightarrow$ now we can insert / modify / delete data safely!
(check by yourself!)

Decomposition

- Schema refinement (decomposition):
  - use two (or more) relations to store the original relation

Person(*NIN, Name, Address, Hobby*)

decomposes to:

Primary key

Person (*NIN, Name, Address*)

Hobby (*NIN, Hobby*)

Primary key

Foreign key

- No update anomalies:
  - name and address stored once
  - a hobby can be separately inserted or deleted

Decomposition

• Another example:

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

• The table "StaffBranch" has redundant data:
  – the details of a branch are repeated for every member of staff

• We can decompose it into two smaller tables:

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

  – in the "Branch" table, "bAddress" appears once for each branch
  – only "branchNo" is repeated in the Staff relation (as a foreign key)

Normalization theory

- The result of ER modelling:
  - needs further refinement to reduce data redundancy
- Decomposition of the relations (tables)
  - this can be done manually for small DB
  - for larger DB we need a formalization of the approach
- Functional dependencies among data items:
  - strongly affect the data anomalies
  - the fundamentals of the underlying normalization theory
  - specify which are the candidate / primary / foreign keys (entity integrity and referential integrity)
  - specify which attributes to combine in the new tables

Reminder: relational keys

- Candidate key: (of a relation)
  - a minimal (not minimum!) set of attributes ("keys") whose values uniquely identify the tuples
- Primary key:
  - The candidate key selected to identify rows uniquely with the table
- Alternate key:
  - Those candidate key(s) not selected as primary key
- Simple key:
  - The key consists of only one attribute
- Composite key:
  - The key consists of several attributes

Functional data dependencies

- Functional data dependency:
  - describes the relationship among attributes in the same relation
  - let A and B be two sets of attributes; we say that "B is functionally dependent on A" (denoted $A \rightarrow B$) if each value of A is associated with exactly one value of B
- Informally:
  - if we know the attribute values of the set A, then we know the (unique) values for the set B
- The attribute values of B can be determined by
  1. calculation: TotalPrice = ItemUnitCost * Quantity
     (ItemUnitCost, Quantity) → TotalPrice
  2. look up: $A \rightarrow B$
     (StudentID, Term, CourseID) → grade

Functional data dependencies

- Consider the attributes “staffNo” and “sName” in this table:

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

- for every value of “staffNo”
  $\longrightarrow$ a unique value of “sName”
- for every value of “sName”
  $\longrightarrow$ a unique value of “staffNo”

- Is this always true ?
  - consider the modified table:
    two different staff members
    are both called “John White”

Staff

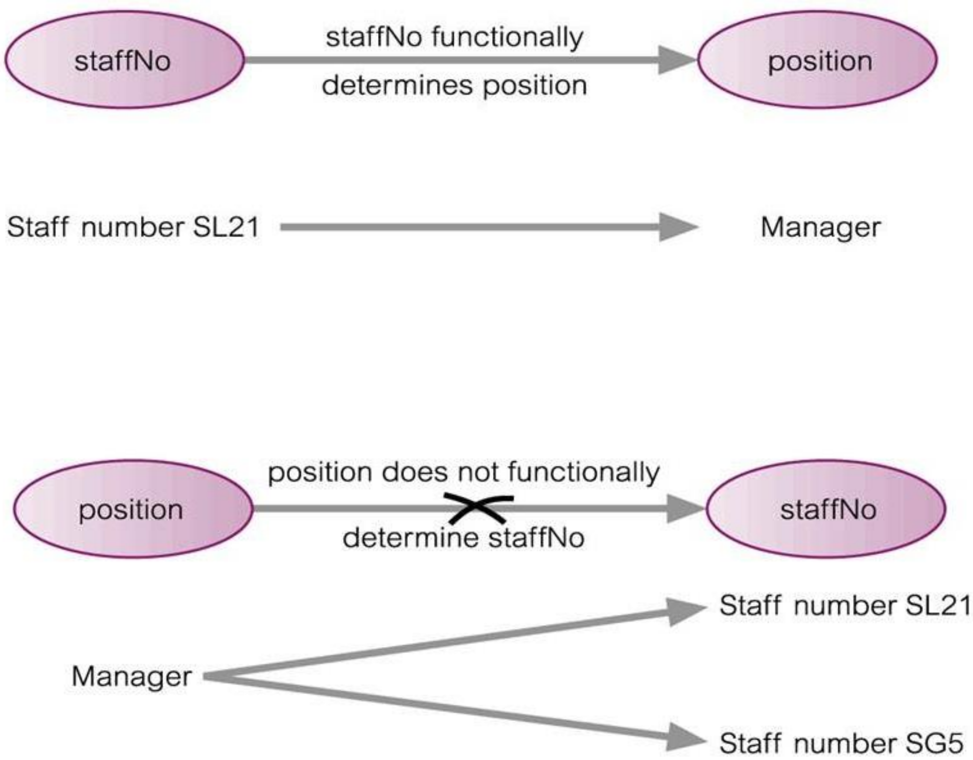
staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	John White	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

- We are only interested in dependencies that hold
  for all instances of the relation

$\Longrightarrow$ we only have the functional dependency: staffNo $\longrightarrow$ sName

Functional data dependencies

- Another example of functional dependency:



Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Functional data dependencies

- In a functional data dependency $(A \rightarrow B)$:
  - determinant: the set of all attributes on the left hand side (i.e. A)
  - dependent: the set of all attributes on the right hand side (i.e. B)

- Full functional dependency $A \rightarrow B$:
  - B is functionally dependent on A
  - B is not functionally dependent on any proper subset of A

- Partial functional dependency $A \rightarrow B$:
  - B is functionally dependent on A
  - B remains functionally dependent on at least one proper subset of A

- Transitive functional dependency:
  - if there exist functional dependencies $A \rightarrow B$ and $B \rightarrow C$
  - then the functional dependency $A \rightarrow C$ also exists and is said to be transitive

Functional data dependencies

Examples:

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

- Full functional dependencies:
  - staffNo → sName
  - staffNo → position
- Partial functional dependencies:
  - staffNo, sName → branchNo

    (it suffices: staffNo → branchNo)
- Note: any dependency on a single attribute is full!
- Transitive functional dependencies:
  - staffNo → branchNo, bAddress
  - branchNo → bAddress

Functional data dependencies

- By the definition of relational keys:
  – a candidate key is a minimal set of attributes,
    which functionally determine all attributes in a relation

- How can we determine all functional dependencies?
  – some of them are obvious from the semantics, e.g. staffNo $\longrightarrow$ sName
    (easy if the problem constraints are well understood)
  – some others follow from specification / discussions with customers
    (if necessary, use your experience / common sense)
  – what about all other dependencies?

- Let F be a set of functional dependencies

- The closure of F (denoted F^+):
  – the set of all functional dependencies that are implied
    by the dependencies in F

The closure of a set F of dependencies

- To compute the closure F^+ of F:
  - we need a set of inference rules

- Armstrong's axioms:
  1. Reflexivity: if $B \subseteq A$, then $A \rightarrow B$
  2. Augmentation: if $A \rightarrow B$, then $A, C \rightarrow B, C$
  3. Transitivity: if $A \rightarrow B$ and $\mathrm{B} \rightarrow C$, then $A \rightarrow C$

- These inference rules are complete:
  - given a set X of functional dependencies, all dependencies implied by X can be derived from X using these rules

  and sound:
  - no additional functional dependencies (which are not implied by X) can be derived using these rules

- These properties can be proved by definition of a functional dependency

The closure of a set F of dependencies

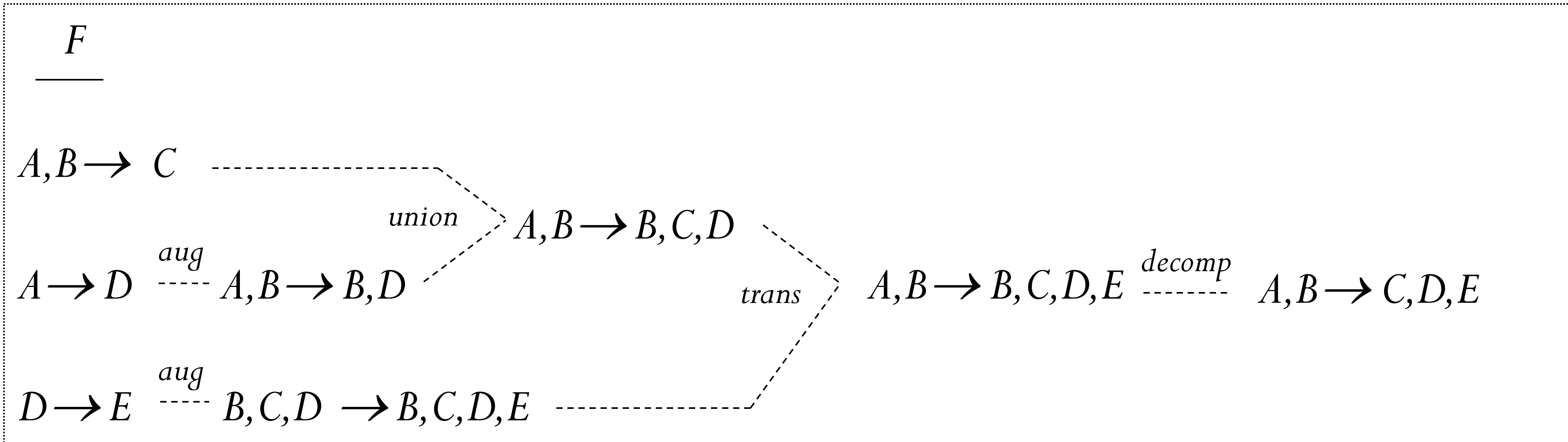
- Further rules can be derived from Armstrong's axioms:
  4. Decomposition: if $A \to B, C$, then $A \to B$ and $A \to C$
  5. Union: if $A \to B$ and $A \to C$, then $A \to B, C$
  6. Composition: if $A \to B$ and $\mathrm{C} \to D$, then $A, C \to B, D$

- For example, proof of the Union rule using the axioms:
  – $A \to B$, augmentation with $C \Rightarrow A, C \to B, C$
  – $A \to C$, augmentation with $\mathrm{A} \Rightarrow A, A \to A, C$, (i.e. $A \to A, C$)
  – transitivity from last two dependencies: $A \to A, C \to B, C$

- To compute the closure F^+ of F:
  – apply repeatedly Armstrong's axioms (or the above 3 extra rules) to get the closure of F

The closure of a set F of dependencies

- Example:



Thus: $A,B \rightarrow B,D$, $A,B \rightarrow B,C,D$, $A,B \rightarrow B,C,D,E$, and $A,B \rightarrow C,D,E$ are (some) elements of F^+

2. Augmentation: if $A \rightarrow B$, then $A, C \rightarrow B, C$
3. Transitivity: if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$
4. Decomposition: if $A \rightarrow B, C$, then $A \rightarrow B$ and $A \rightarrow C$
5. Union: if $A \rightarrow B$ and $A \rightarrow C$, then $A \rightarrow B, C$

The closure of a set F of dependencies

A pseudo-code for computing the closure F^+ :

$F^+ = F$ *(initialization)*

repeat

for every func. dep. f in the set F^+

apply the rules of reflexivity and augmentation to f

add these new func. dep. to the set F^+

for each pair f_1, f_2 of func. dep. in the set F^+

if f_1, f_2 imply a func. dep. f_3 using transitivity

then add f_3 to the set F^+

until the set F^+ does not change any more

Functional data dependencies

- Let F be a set of functional dependencies
- Let A be a set of attributes in a relation

- The closure of A under F (denoted A^+):
  - the set of all attributes that can be implied by the attributes of A, using functional dependencies from F

- We can compute the closure A^+ (under F):
  - similarly to the closure F^+
  - start with the functional dependencies of F, which have attributes of A on the left hand side
  - apply repeatedly Armstrong's axioms

- By the definition of relational keys:
  - a candidate key is a minimal set of attributes, such that A^+ (under F^+) includes all attributes in a relation

Summary of the lecture

- Redundancy in the Relational Data Model
- Data anomalies
  - modification anomaly
  - deletion anomaly
  - insertion anomaly
- Decomposition of relations (schema refinement)
- Functional Dependencies
  - full dependencies
  - partial dependencies
  - transitive dependencies
- Armstrong's axioms
  - computation of the closure of a set of functional dependencies
  - computation of the closure of a set of attributes