CS2102 Database Systems

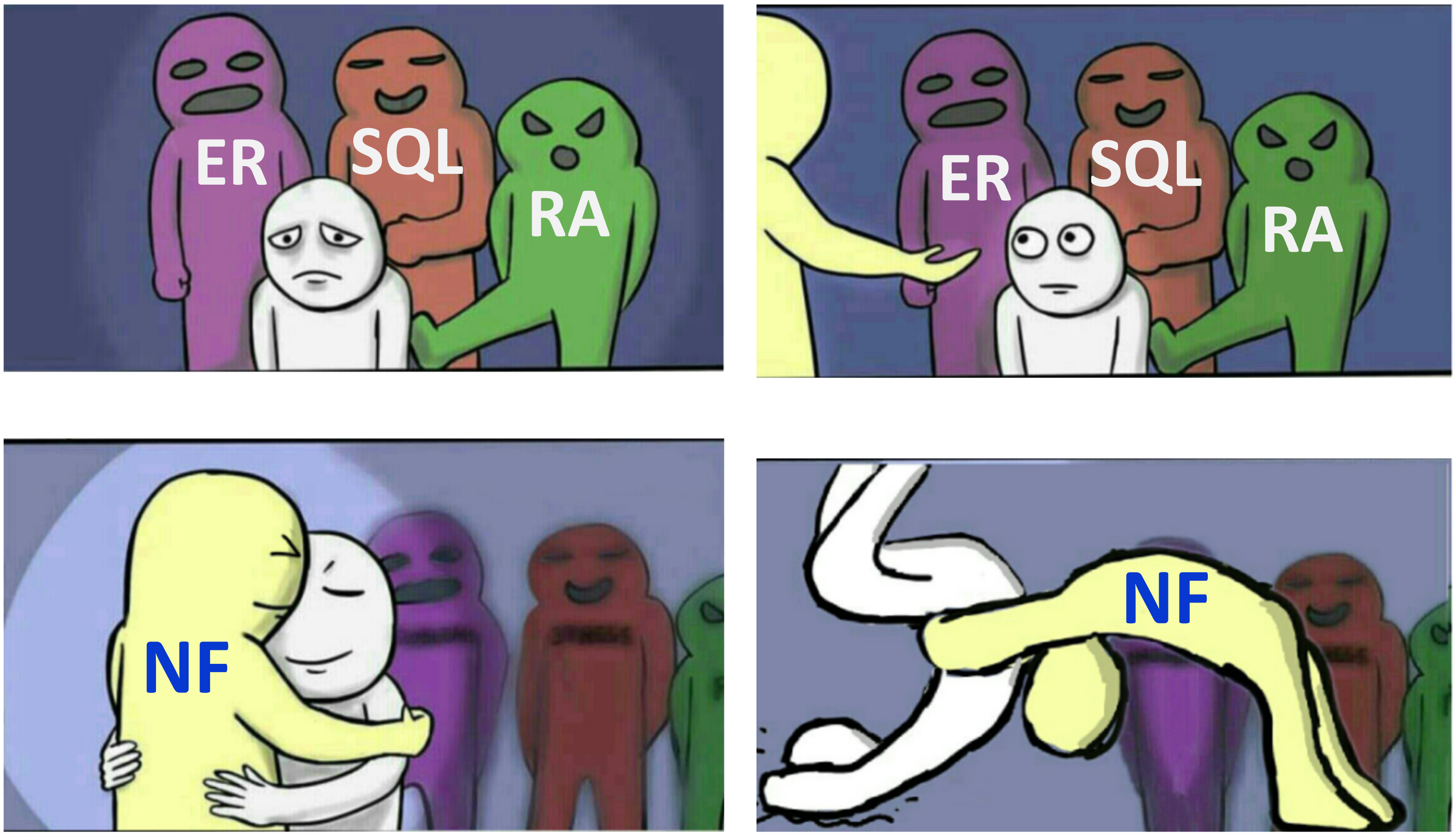
Previously in CS2102

- ER model
- Relational algebra
- SQL
- PL/pgSQL
- Triggers

What is next?

- Normal forms

Normal Forms vs. ER, SQL, and RA



Roadmap

- We will do it step by step:
  - Functional dependencies (FD)

    ⇩
  - Closures

    ⇩
  - Keys, superkeys, and prime attributes

    ⇩
  - Normal forms and schema refinement

Motivation

- Suppose that we give an ER diagram to Alice and Bob
- Each of them translates the diagram into a relational schema
  - And claims that it is the best relational schema of all time
- How do we decide which one is better?

Motivation

- There could be many different ways to evaluate whether a relational schema is good
  - Different people may have different opinions
- But there are things that just should NOT be done
  - i.e., there are some minimum requirements to meet
- A normal form is a definition of minimum requirements to
  - reduce data redundancy, and
  - improve data integrity

Redundancy: Example

Name	NRIC	PhoneNumber	HomeAddress
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

- Primary key of the table:
  (NRIC, PhoneNumber)
- There is some redundancy in terms of Alice's address: it is unnecessarily stored twice
- In addition, the table is susceptible to several other anomalies

Update Anomalies

Name	<u>NRIC</u>	<u>PhoneNumber</u>	HomeAddress
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

- Primary key of the table:
  (NRIC, PhoneNumber)
- First, update anomalies:
  - We may accidentally update one of Alice's addresses, leaving the other unchanged

Deletion Anomalies

Name	NRIC	PhoneNumber	HomeAddress
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

- Primary key of the table:
  (NRIC, PhoneNumber)
- Second, deletion anomalies:
  - Bob no longer uses a phone
  - Can we remove Bob's phone number?
  - No. (Note: Primary key attributes cannot be NULL)

Insertion Anomalies

Name	NRIC	PhoneNumber	HomeAddress
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

- Primary key of the table:
  (NRIC, PhoneNumber)
- Third, insertion anomalies:
  - Name = Cathy, NRIC = 9394, HomeAddress = YiShun
  - Can we insert this information into the table?
  - No. (Note: Primary key attributes cannot be NULL)

Normalization

Name	NRIC	PhoneNumber	HomeAddress
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

- How do we get rid of those anomalies?
- Normalize the table (i.e., decompose it)

Name	NRIC	HomeAddress
Alice	1234	Jurong East
Bob	5678	Pasir Ris

NRIC	PhoneNumber
1234	67899876
1234	83848384
5678	98765432

Effects of Normalization

Name	NRIC	HomeAddress
Alice	1234	Jurong East
Bob	5678	Pasir Ris

NRIC	PhoneNumber
1234	67899876
1234	83848384
5678	98765432

- Redundancy?
  - No. (Alice’s address is no longer duplicated.)
- Update anomalies?
  - No. (There is only one place where we can update the address of Alice)
- Deletion anomalies?
  - No. (We can freely delete Bob’s phone number)
- Insertion anomalies?
  - No. (We can insert an individual with a phone)

Effects of Normalization

Name	NRIC	HomeAddress
Alice	1234	Jurong East
Bob	5678	Pasir Ris

NRIC	PhoneNumber
1234	67899876
1234	83848384
5678	98765432

- How do we perform such normalizations?
- Following some procedures designed based on normal forms

Roadmap

- We will do it step by step:
  - Functional dependencies (FD)

    ⇓
  - Closures

    ⇓
  - Keys, superkeys, and prime attributes

    ⇓
  - Normal forms and schema refinement

Previous Example

Name	NRIC	PhoneNumber	HomeAddress
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

- We mentioned that this table is bad because of the redundancy in HomeAddress
- What causes this redundancy?
  - Some dependency between NRIC and HomeAddress
- In particular, NRIC uniquely decides HomeAddress
- This is called a functional dependency (FD)
  - Denoted as NRIC $\rightarrow$ HomeAddress

Formal Definition of FD

- Let A_1, A_2, …, A_m, B_1, B_2, …, B_n be some attributes
- We say that $A_1A_2…A_m \rightarrow B_1B_2…B_n$, if:
  - Whenever two objects have the same values on A_1, A_2, …, and A_m,
  - they always have the same values on B_1, B_2, … , B_n
- Example: NRIC → Name
  - Read as "NIRC decides Name" or "NIRC determines Name"
- Meaning: If two tuples have the same NRIC value, then they have the same Name value

Examples

- Matric_Number $\rightarrow$ Student_Name
- Postal_Code $\rightarrow$ Building_Name

- Postal_Code $\not\rightarrow$ Unit_Number
- Matric_Number $\not\rightarrow$ Degree
  - We have double degrees

FDs on Tables

- An FD may hold on one table but does not hold on another
- Example:
  - Supervise(eid, <u>pid</u>)
    - pid denotes the id of the project
    - eid denotes the employee id of the supervisor for the project
    - If each project has only one supervisor, then we have pid→eid on Supervise
  - Work(<u>eid</u>, <u>pid</u>)
    - pid denotes the id of the project
    - eid denotes the id of an employee who work on the project
    - We don't have pid→eid on Work

Name	Category	Color	Department	Price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Black	Toys	99
Gizmo	Stationary	Green	Office Supplies	59

- Find the functional dependencies that are FALSE on the above table
  - Category → Department
  - Category, Color → Price
  - Price → Color
  - Name → Color
  - Department, Category → Name
  - Color, Department → Name, Price, Category

Where Do FDs Come From?

- From common sense
- From the application's requirements
- Example
  - Purchase(CustomerID, ProductID, ShopID, Price, Date)
  - Requirement: Each shop can sell at most one product
  - FD implied: ShopID $\rightarrow$ ProductID

Example

- Purchase(CustomerID, ProductID, ShopID, Price, Date)
- Requirement: No two customers buy the same product
- FD implied: ProductID → CustomerID

Example

- Purchase(CustomerID, ProductID, ShopID, Price, Date)
- Requirement: No two shops sell the same product
- FD implied: ProductID → ShopID

Example

- Purchase(CustomerID, ProductID, ShopID, Price, Date)
- Requirement: No two shops sell the same product on the same date
- FD implied: ProductID, Date $\rightarrow$ ShopID

Example

- Purchase(CustomerID, ProductID, ShopID, Price, Date)
- Requirement: No shop should sell the same product to the same customer on the same date at two different prices
- FD implied:
  CustomerID, ProductID, ShopID, Date → Price

Roadmap

- Now we know what FDs are
- Next, we will discuss how to do reasoning with FDs

FD Reasoning: Example

- We know that
  - NRIC → Matric_Number, and
  - Matric_Number → Name
- We can derive
  - NRIC → Name, by transitivity
- FD reasoning: given a set of FDs, figure out what other FDs they can imply
- This is important for normal forms

Armstrong's Axioms

- Three fundamental axioms for FD reasoning
- Axiom of Reflexivity
  - A set of attributes → A subset of the attributes
- Example
  - NRIC, Name → NRIC
  - StudentID, Name, Age → Name, Age
  - ABCD → ABC
  - ABCD → BCD
  - ABCD → AD

Armstrong's Axioms

- Three fundamental axioms for FD reasoning
- Axiom of Augmentation
  - If A → B
  - then AC → BC for any C
- Example
  - If NRIC → Name
  - Then NRIC, Age → Name, Age
  - and NRIC, Salary, Weight → Name, Salary, Weight
  - and NRIC, Addr, Postal → Name, Addr, Postal

Armstrong's Axioms

- Three fundamental axioms for FD reasoning
- Axiom of Transitivity
  - If A → B and B → C
  - then A → C
- Example
  - If NRIC → Addr, and Addr → Postal
  - Then NRIC → Postal

Additional Rules

- Reflexivity: $AB \rightarrow A$
- Augmentation: If $A \rightarrow B$ then $AC \rightarrow BC$
- Transitivity: If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$

- Rule of Decomposition
  - If $A \rightarrow BC$, then $A \rightarrow B$ and $A \rightarrow C$
- Proof:
  - By reflexivity, we have $BC \rightarrow B$ and $BC \rightarrow C$
  - By transitivity, we have
    - $A \rightarrow BC$ and $BC \rightarrow B$ ==> $A \rightarrow B$
    - $A \rightarrow BC$ and $BC \rightarrow C$ ==> $A \rightarrow C$

Additional Rules

- Reflexivity: AB→A
- Augmentation: If A→B then AC→BC
- Transitivity: If A→B and B→C then A→C
- Decomposition: If A→BC then A→B and A→C

- Rule of Union
  - If A→B and A→C, then A→BC
- Proof:
  - By augmentation, A→B ==> A→AB
  - By augmentation, A→C ==> AB→BC
  - By transitivity, A→AB and AB→BC ==> A→BC

Exercise

- Reflexivity: $AB \rightarrow A$
- Augmentation: If $A \rightarrow B$ then $AC \rightarrow BC$
- Transitivity: If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$
- Decomposition: If $A \rightarrow BC$ then $A \rightarrow B$ and $A \rightarrow C$
- Union: If $A \rightarrow B$ and $A \rightarrow C$ then $A \rightarrow BC$

- Given $A \rightarrow B$, $BC \rightarrow D$
- Prove that $AC \rightarrow D$
- Proof
  - Given $A \rightarrow B$, we have $AC \rightarrow BC$ (Augmentation)
  - Given $AC \rightarrow BC$ and $BC \rightarrow D$, we have $AC \rightarrow D$ (Transitivity)

Reasoning with FD

- Reflexivity: $AB \rightarrow A$
- Augmentation: If $A \rightarrow B$ then $AC \rightarrow BC$
- Transitivity: If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$
- Decomposition: If $A \rightarrow BC$ then $A \rightarrow B$ and $A \rightarrow C$
- Union: If $A \rightarrow B$ and $A \rightarrow C$ then $A \rightarrow BC$

- Given $A \rightarrow B$, $D \rightarrow C$
- Prove that $AD \rightarrow BC$
- Proof
  - Given $A \rightarrow B$, we have $AD \rightarrow BD$ (Augmentation)
  - Given $AD \rightarrow BD$, we have $AD \rightarrow B$ (Reflexivity)
  - Given $D \rightarrow C$, we have $AD \rightarrow AC$ (Augmentation)
  - Given $AD \rightarrow AC$, we have $AD \rightarrow C$ (Reflexivity)
  - Given $AD \rightarrow B$ and $AD \rightarrow C$, we have $AD \rightarrow BC$ (Union)

Reasoning with FD

- Reflexivity: $AB \rightarrow A$
- Augmentation: If $A \rightarrow B$ then $AC \rightarrow BC$
- Transitivity: If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$
- Decomposition: If $A \rightarrow BC$ then $A \rightarrow B$ and $A \rightarrow C$
- Union: If $A \rightarrow B$ and $A \rightarrow C$ then $A \rightarrow BC$

- Given
  $A \rightarrow C$, $AC \rightarrow D$, $AD \rightarrow B$
- Prove that $A \rightarrow B$
- Proof
  - Given $A \rightarrow C$, we have $A \rightarrow AC$ (Augmentation)
  - Given $A \rightarrow AC$ and $AC \rightarrow D$, we have $A \rightarrow D$ (Transitivity)
  - Given $A \rightarrow D$, we have $A \rightarrow AD$ (Augmentation)
  - Given $A \rightarrow AD$ and $AD \rightarrow B$, we have $A \rightarrow B$ (Transitivity)

Reasoning with FD

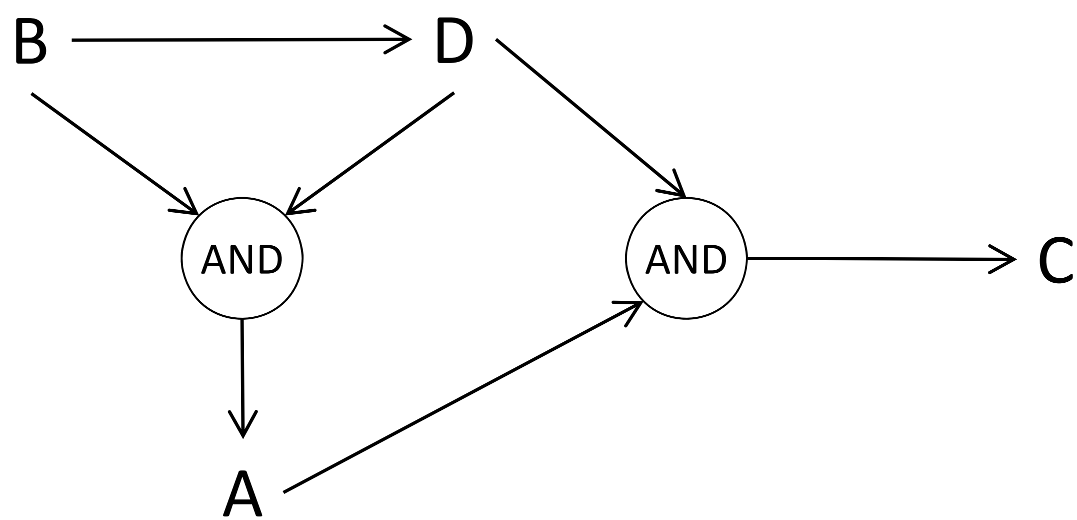
- Use Armstrong's axioms to do FD reasoning is a bit cumbersome
  - As shown in the previous slides
- We will discuss a more convenient approach: closure

Closure: Motivating Example

- Question:
  - Given B→D, DB→A, AD→C, check if B→C holds
- Observation: intuitively, FDs are kind of like components on a circuit board

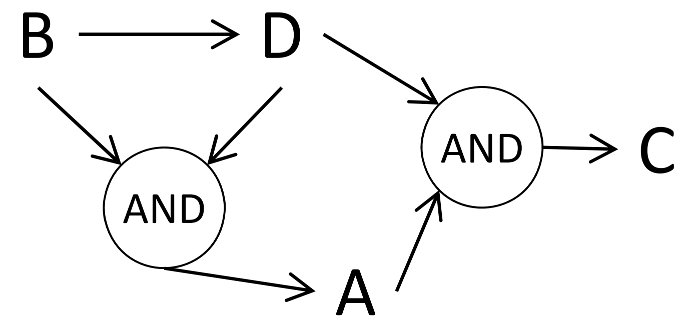
Closure: Motivating Example

- Four attributes: A, B, C, D
- Given: B→D, DB→A, AD→C
- Check if B→C holds



Closure: Motivating Example

- Four attributes: A, B, C, D
- Given: B→D, DB→A, AD→C
- Check if B→C holds



- First, activate B
  - Activated set = { B }
- Second, activate whatever B can activate
  - Activated set = { B, D }, since B→D
- Third, use all activated elements to activate more
  - Activated set = { B, D, A }, since DB→A
- Repeat the third step, until no more activation is possible
  - Activated set = { B, D, A, C }, since AD→C; done
- { B, D, A, C } is referred to as the closure of {B}

Closure

- Let $S = \{A_1, A_2, \ldots, A_n\}$ be a set of attributes
- The closure of S is the set of attributes that can be decided by $A_1, A_2, \ldots, A_n$ (directly or indirectly)
- Notation: $\{A_1, A_2, \ldots, A_n\}^+$
- Example
  - Given $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow E$
  - $\{A\}^+ = \{A, B, C, D, E\}$
  - $\{B\}^+ = \{B, C, D, E\}$
  - $\{D\}^+ = \{D, E\}$
  - $\{E\}^+ = \{E\}$

Computing Closures

- Given $A_1, A_2, \ldots, A_n$, the closure $\{A_1, A_2, \ldots, A_n\}^+$ can be computed as follows:
  1. Initialize the closure to $\{A_1, A_2, \ldots, A_n\}$
  2. If there is an FD: $A_i, A_j, \ldots, A_m \rightarrow B$, such that $A_i, A_j, \ldots, A_m$ are all in the closure, then put B into the closure
  3. Repeat step 2, until we cannot find any new attribute to put into the closure
- Example
  - A Table with five attributes A, B, C, D, E
  - $A \rightarrow B$, $C \rightarrow D$, $BC \rightarrow E$
  - $\{A\}^+ =$
  - $\{A, C\}^+ =$
  - $\{B\}^+ =$

Computing Closures

- Given $A_1, A_2, \ldots, A_n$, the closure $\{A_1, A_2, \ldots, A_n\}^+$ can be computed as follows:
  1. Initialize the closure to $\{A_1, A_2, \ldots, A_n\}$
  2. If there is an FD: $A_i, A_j, \ldots, A_m \rightarrow B$, such that $A_i, A_j, \ldots, A_m$ are all in the closure, then put B into the closure
  3. Repeat step 2, until we cannot find any new attribute to put into the closure
- Example
  - A Table with five attributes A, B, C, D, E
  - $A \rightarrow B$, $C \rightarrow D$, $BC \rightarrow E$
  - $\{A\}^+ = \{A, B\}$
  - $\{A, C\}^+ = \{A, B, C, D, E\}$
  - $\{B\}^+ = \{B\}$

Closure & FD

- To prove that X → Y holds, we only need to show that $\{X\}^+$ contains Y
- AB→C, AD→E, B→D, AF→B
- Prove that AF→D
- $\{AF\}^+$ = {AFBCDE}, which contains D
- Therefore, AF→D holds

Closure & FD

- To prove that X → Y does not hold, we only need to show that $\{X\}^+$ does not contain Y
- AB→C, AD→E, B→D, AF→B
- Prove that AD→F does not hold
- $\{AD\}^+ = \{ADE\}$, which does not contain F
- Therefore, AD→F does not hold

Exercise

- Given: $C \rightarrow D$, $AD \rightarrow E$, $BC \rightarrow E$, $E \rightarrow A$, $D \rightarrow B$
- Check if $C \rightarrow A$ holds
- We start with {C}
- Since $C \rightarrow D$, we have {C, D}
- Since $D \rightarrow B$, we have {C, D, B}
- Since $BC \rightarrow E$, we have {C, D, B, E}
- Since $E \rightarrow A$, we have {C, D, B, E, A}
- So A must be in $\{C\}^+$, hence, $C \rightarrow A$ holds

Roadmap

- We will do it step by step:
  - Functional dependencies (FD)

    $\Downarrow$
  - Closures

    $\Downarrow$
  - Keys, superkeys, and prime attributes

    $\Downarrow$
  - Normal forms and schema refinement

Superkeys of a Table

Name	NRIC	Postal	Address
Alice	1234	939450	Jurong East
Bob	5678	234122	Pasir Ris
Cathy	3576	420923	Yishun

- Definition: A set of attributes in a table that decides all other attributes
- Example:
  - {NRIC} is a superkey
  - Since NRIC → Name, Postal, Address
  - {NRIC, Name} is a superkey
  - Since {NRIC, Name} → Postal, Address

Keys of a Table

Name	NRIC	Postal	Address
Alice	1234	939450	Jurong East
Bob	5678	234122	Pasir Ris
Cathy	3576	420923	Yishun

- Definition: A superkey that is minimal
- i.e., if we remove any attribute from the superkey, it will not be a superkey anymore
- Example:
  - {NRIC} is a superkey
  - Since NRIC → Name, Postal, Address
  - {NRIC, Name} is a superkey
  - Since {NRIC, Name} → Postal, Address
  - NRIC is a key, but {NRIC, Name} is not a key

Keys of a Table

Name	NRIC	Postal	Address
Alice	1234	939450	Jurong East
Bob	5678	234122	Pasir Ris
Cathy	3576	420923	Yishun

- Note: Not to be confused with the keys of entity sets

Keys of a Table

Name	NRIC	StudentID	Postal	Address
Alice	1234	1	939450	Jurong East
Bob	5678	2	234122	Pasir Ris
Cathy	3576	3	420923	Yishun

- A table may have multiple keys
- Example:
  - {NRIC} is a key
  - Since NRIC → Name, StudentID, Postal, Address
  - {StudentID} is a key
  - Since StudentID → Name, NRIC, Postal, Address
  - Both {NRIC} and {StudentID} are keys

Keys of a Table: Exercise

- We have
  - A table T(A, B, C) with three attributes A, B, C
  - Two FDs: A→BC and BC→A
- Find the key(s) of T
- Answer: there are two keys
  - A
  - BC
- Note: BC is a key even though it contains more attribute than A
  - Because BC is a minimal superkey

Why are we talking about keys?

- Because they are needed in our discussion of normal forms
  - Whether or not a table T has redundancy and anomalies would partially depend on what the keys of T are
- Question: how do we know the keys of T?
- Answer:
  - Check the FDs on the T, and use closures to derive the keys

Algorithm for finding keys

- Definition: a key is a minimal set of attributes that decides all other attributes
- Given: a table T(A, B, C, …) and a set of FDs on T
- Algorithm for finding keys:
  - Consider every subset of attributes in T:
    - A, B, C, …, AB, BC, CA, …, ABC, …
  - Derive the closure of each subset:
    - $\{A\}^+$, $\{B\}^+$, $\{C\}^+$, …, $\{AB\}^+$, $\{BC\}^+$, $\{AC\}^+$, …, $\{ABC\}^+$, …
  - Identify all superkeys based on the closures
  - Identify all keys from the superkeys

Algorithm for finding keys: Example

- A table R(A, B, C), with A→B, B→C
- Steps for finding keys:
  - Consider every subset of attributes in T:
    - A, B, C, AB, BC, CA, ABC
  - Derive the closure of each subset:
    - $\{A\}^+=$ $\{B\}^+=$ $\{C\}^+=$
    - $\{AB\}^+=$ $\{BC\}^+=$ $\{AC\}^+=$ $\{ABC\}^+=$
  - Identify all superkeys based on the closures
  - Identify all keys from the superkeys

Algorithm for finding keys: Example

- A table R(A, B, C), with A→B, B→C
- Steps for finding keys:
  - Consider every subset of attributes in T:
    - A, B, C, AB, BC, CA, ABC
  - Derive the closure of each subset:
    - $\{A\}^+=\{ABC\}$, $\{B\}^+=\{BC\}$, $\{C\}^+=\{C\}$
    - $\{AB\}^+=\{ABC\}$, $\{BC\}^+=\{BC\}$, $\{AC\}^+=\{ABC\}$, $\{ABC\}^+=\{ABC\}$
  - Identify all superkeys based on the closures
    - A, AB, AC, ABC
  - Identify all keys from the superkeys
    - A

Exercise: Find the keys

- A table R(A, B, C, D)
- With AB→C, AD→B, B→D
- First, enumerate all attribute subsets:
  - {A}, {B}, {C}, {D}
  - {AB}, {AC}, {AD},
  - {BC}, {BD}, {CD},
  - {ABC}, {ABD},
  - {ACD}, {BCD},
  - {ABCD}

Exercise: Find the keys

- A table R(A, B, C, D)
- With AB→C, AD→B, B→D
- Second, compute the closures of the subsets:
  - {A}, {B}, {C}, {D}
  - {AB}, {AC}, {AD},
  - {BC}, {BD}, {CD},
  - {ABC}, {ABD},
  - {ACD}, {BCD},
  - {ABCD}

Exercise: Find the keys

- A table R(A, B, C, D)
- With $AB \rightarrow C$, $AD \rightarrow B$, $B \rightarrow D$
- Second, compute the closures of the subsets:
  - $\{A\}^+ = \{A\}$, $\{B\}^+ = \{BD\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$
  - $\{AB\}^+ = \{ABCD\}$, $\{AC\}^+ = \{AC\}$, $\{AD\}^+ = \{ABCD\}$
  - $\{BC\}^+ = \{BCD\}$, $\{BD\}^+ = \{BD\}$, $\{CD\}^+ = \{CD\}$
  - $\{ABC\}^+ = \{ABCD\}$, $\{ABD\}^+ = \{ABCD\}$
  - $\{ACD\}^+ = \{ABCD\}$, $\{BCD\}^+ = \{BCD\}$
  - $\{ABCD\}^+ = \{ABCD\}$

Exercise: Find the keys

- A table R(A, B, C, D)
- With AB→C, AD→B, B→D
- Third, identify the superkeys:
  - $\{A\}^+ = \{A\}$, $\{B\}^+ = \{BD\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$
  - $\{AB\}^+ = \{ABCD\}$, $\{AC\}^+ = \{AC\}$, $\{AD\}^+ = \{ABCD\}$
  - $\{BC\}^+ = \{BCD\}$, $\{BD\}^+ = \{BD\}$, $\{CD\}^+ = \{CD\}$
  - $\{ABC\}^+ = \{ABCD\}$, $\{ABD\}^+ = \{ABCD\}$
  - $\{ACD\}^+ = \{ABCD\}$, $\{BCD\}^+ = \{BCD\}$
  - $\{ABCD\}^+ = \{ABCD\}$

Exercise: Find the keys

- A table R(A, B, C, D)
- With AB→C, AD→B, B→D
- Third, identify the superkeys:
  - $\{A\}^+ = \{A\}$, $\{B\}^+ = \{BD\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$
  - $\{AB\}^+ = \{ABCD\}$, $\{AC\}^+ = \{AC\}$, $\{AD\}^+ = \{ABCD\}$
  - $\{BC\}^+ = \{BCD\}$, $\{BD\}^+ = \{BD\}$, $\{CD\}^+ = \{CD\}$
  - $\{ABC\}^+ = \{ABCD\}$, $\{ABD\}^+ = \{ABCD\}$
  - $\{ACD\}^+ = \{ABCD\}$, $\{BCD\}^+ = \{BCD\}$
  - $\{ABCD\}^+ = \{ABCD\}$

Exercise: Find the keys

- A table R(A, B, C, D)
- With AB→C, AD→B, B→D
- Fourth, identify the keys from the superkeys
  - $\{A\}^+ = \{A\}$, $\{B\}^+ = \{BD\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$
  - $\{AB\}^+ = \{ABCD\}$, $\{AC\}^+ = \{AC\}$, $\{AD\}^+ = \{ABCD\}$
  - $\{BC\}^+ = \{BCD\}$, $\{BD\}^+ = \{BD\}$, $\{CD\}^+ = \{CD\}$
  - $\{ABC\}^+ = \{ABCD\}$, $\{ABD\}^+ = \{ABCD\}$
  - $\{ACD\}^+ = \{ABCD\}$, $\{BCD\}^+ = \{BCD\}$
  - $\{ABCD\}^+ = \{ABCD\}$

Exercise: Find the keys

- A table R(A, B, C, D)
- With AB→C, AD→B, B→D
- Fourth, identify the keys from the superkeys
  - $\{A\}^+ = \{A\}$, $\{B\}^+ = \{BD\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$
  - $\{AB\}^+ = \{ABCD\}$, $\{AC\}^+ = \{AC\}$, $\{AD\}^+ = \{ABCD\}$
  - $\{BC\}^+ = \{BCD\}$, $\{BD\}^+ = \{BD\}$, $\{CD\}^+ = \{CD\}$
  - $\{ABC\}^+ = \{ABCD\}$, $\{ABD\}^+ = \{ABCD\}$
  - $\{ACD\}^+ = \{ABCD\}$, $\{BCD\}^+ = \{BCD\}$
  - $\{ABCD\}^+ = \{ABCD\}$

A Small Trick

- Always check small attribute sets first
- A table R(A, B, C, D)
- A→B, B→C, C→D, D→A
- Compute the closures:
  - $\{A\}^+ = \{ABCD\}$, $\{B\}^+ = \{ABCD\}$, $\{C\}^+ = \{ABCD\}$, $\{D\}^+ = \{ABCD\}$
  - No need to check others
  - The others are all superkeys but not keys
- Keys: {A}, {B}, {C}, {D}

Another Small Trick

- A table R(A, B, C, D)
- AB→C, AD→B, B→D
- Notice that A does not appear in the right hand side of any functional dependencies
- In that case, A must be in every key
- Keys of R: AB, AD (see the previous exercises)
- In general, if an attribute that does not appear in the right hand side of any FD, then it must be in every key

Exercise (Find the Keys)

- A table R(A, B, C, D)
- $A \rightarrow B$, $A \rightarrow C$, $C \rightarrow D$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{ABCD\}$
  - No need to check others
- Keys: {A}

Exercise (Find the Keys)

- A table R(A, B, C, D, E)
- $AB \rightarrow C$, $C \rightarrow B$, $BC \rightarrow D$, $CD \rightarrow E$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{A\}$
  - $\{AB\}^+ = \{ABCDE\}$
  - $\{AC\}^+ = \{ACBDE\}$
  - $\{AD\}^+ = \{AD\}$, $\{AE\}^+ = \{AE\}$
  - $\{ADE\}^+ = \{ADE\}$
- Keys: AB, AC

Exercise (Find the Keys)

- A table R(A, B, C, D, E, F)
- AB→C, C→B, CBE→D, D→EF
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{A\}$
  - $\{AB\}^+ = \{ABC\}$
  - $\{AC\}^+ = \{ACB\}$
  - $\{AD\}^+ = \{ADEF\}$
  - $\{AE\}^+ = \{AE\}$, $\{AF\}^+ = \{AF\}$
  - $\{ABC\}^+ = \{ABC\}$
  - $\{ABD\}^+ = \{ABE\}^+ = \{ACD\}^+ = \{ACE\}^+ = \{ABCDEF\}$
  - $\{ADE\}^+ = \{ADEF\}$
- Keys: ABD, ABE, ACD, ACE

Prime Attributes

- If an attribute appears in a key, then it is a prime attribute
- Otherwise, it is a non-prime attribute
- This concept will be used when we talk about normal forms

Roadmap

- We will do it step by step:
  - Functional dependencies (FD)

    $\Downarrow$
  - Closures

    $\Downarrow$
  - Keys, superkeys, and prime attributes

    $\Downarrow$
  - Normal forms and schema refinement