# Other Integrity Constraints

- Domain Constraint, Entity Integrity, Referential Integrity, Enterprise constraints……(Discussed in Module 1)
- Functional Dependency
- Multi-valued Dependency
- Join Dependency
- How these can be used in relational database design…? *Normalization*



# Functional Dependencies

- Frame work for systematic design and optimization of relational schema.
- Generalization over the notion of keys
- Crucial in obtaining correct normalized schema

**Definition** :

- In any relation R, if there exists a set of attributes A1,A2,…An and an attribute B such that if any tuples have the same value for A1,A2,…An then they also have the same value for B.

  A functional dependency of the above form is written as:

  A1,A2,….An → B

  i.e. A1,A2,…An uniquely determines B
- Functional Dependencies define properties of schema and not any particular tuple in the schema.



# Functional Dependencies –Example

- Movies ( title, year, length, studio_name, star_name)

| title | year | length | studio_name | star_name |
|---|---|---|---|---|
| Star Wars | 1977 | 124 | Fox | Carrie Fisher |
| Star Wars | 1977 | 124 | Fox | Harrison Ford |
| Mighty Ducks | 1991 | 104 | Disney | Emilio Estevez |
| Wayne's World | 1992 | 95 | Paramount | Dana Carvey |
| Wayne's World | 1992 | 95 | Paramount | Mike Meyers |

- FD: **title, year → length ,**
  - **title, year → studio_name**
  - **title → studio_name**
- Not a FD: **title, year → star_name**



# Functional Dependencies: Formal Definition

- Let *R* be a relation schema

$$\alpha \subseteq R \ \ and \ \ \beta \subseteq R$$

- The functional dependency (FD) $\alpha \rightarrow \beta$ holds on *R* iff
  - **for any legal relations *r* (R)**
  - **whenever any two tuples $t_1$ and $t_2$ of *r* agree on the attributes $\alpha$**
  - **they also agree on the attributes $\beta$**
  - **i.e. $\prod_\alpha(t_1) = \prod_\alpha(t_2) \Rightarrow \prod_\beta(t_1) = \prod_\beta(t_2)$**

## Functional Dependencies vs Keys

- FDs can express the same constraints we could express using keys:
- Superkeys:
  - ***K* is a superkey for relation schema *R* if and only if $K \rightarrow R$**
- Candidate keys:
  - ***K* is a candidate key for *R* if and only if**
    - $K \rightarrow R$, and
    - there is no $K' \subset K$ such that $K' \rightarrow R$
- However,FDs are more general
  - **i.e. we can express constraints that cannot be expressed using keys**



## Functional Dependencies vs Keys: Example

- Movies ( title, year, length, studio_name, star_name)

| title | year | length | studio_name | star_name |
|---|---|---|---|---|
| Star Wars | 1977 | 124 | Fox | Carrie Fisher |
| Star Wars | 1977 | 124 | Fox | Harrison Ford |
| Mighty Ducks | 1991 | 104 | Disney | Emilio Estevez |
| Wayne's World | 1992 | 95 | Paramount | Dana Carvey |
| Wayne's World | 1992 | 95 | Paramount | Mike Meyers |

- **title, year, star_name $\rightarrow$ R.**
- **Hence title, year, star_name is a key.**



## Functional Dependencies vs Keys : Example

- Example of FDs that can't be represented using keys:

*Loan-info-schema: (<u>customer-name</u>, <u>loan-number</u>, branch-name, amount).*

We expect these FDs to hold:

*loan-number* $\rightarrow$ *amount*
*loan-number* $\rightarrow$ *branch-name*

We could try to express this by making *loan-number* the key, however the following FD does *<u>not</u>* hold:

*loan-number* $\rightarrow$ *customer-name*



## Functional Dependencies (cont)

- If we only consider an instance, we can't tell if an FD holds
  - **e.g. inspecting the movies relation, we might suggest that length $\rightarrow$ title, since no two films in the table have the same length**
  - **However, we cannot assert this FD for the movies relation, since we know it is not true of the domain in general**
- Thus, identifying FDs is part of the data modelling process

# Functional Dependencies: Uses

- We use FDs to:
  - **test relations to see if they are legal under a given set of FDs**
    - If a relation *r* is legal under a set *F* of FDs, we say that *r* satisfies *F*
  - **specify constraints on the set of legal relations**
    - We say that *F* holds on *R* if all legal relations on *R* satisfy the set of FDs *F*
- Note: A specific instance of a relation schema may satisfy an FD even if the FD does not hold on all legal instances.
  - **For example, a specific instance of *Loan-schema* may, by chance, satisfy**
    ***loan-number* $\rightarrow$ *customer-name***



# Trivial Functional Dependencies

- An FD where RHS is contained within the LHS is called trivial FD. In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$
  - ***E.g.***

    ***Loan-info-schema: (<u>customer-name</u>, <u>loan-number</u>, branch-name, amount).***

    - *customer-name, loan-number* $\rightarrow$ *customer-name*
    - *customer-name* $\rightarrow$ *customer-name*
- It there is at least one element on the RHS that is not contained in the LHS, it is called non- trivial FD and if none of the element of the RHS are contained in the LHS, it is called completely non-trivial FD.



# Full Functional Dependencies

- An FD X $\rightarrow$ A for which there is no proper subset Y of X such that Y $\rightarrow$ A (A is said to be fully functionally dependent on X)

*e.g.* *Loan-info-schema = (<u>customer-name</u>, <u>loan-number</u>, branch-name, amount).*

*customer-name, loan-number* $\rightarrow$ *amount is not a full functional dependency because of the FD loan-number* $\rightarrow$ *amount which is a full functional dependency.*



# Closure of FDs

- Given a set *F* of fds, there are other FDs logically implied by *F*
  - **E.g. If A $\rightarrow$ B and B $\rightarrow$ C, then we can infer that A $\rightarrow$ C**
- The set of all FDs implied by *F* is the *closure* of *F*, written $F^+$
- We can find all of $F^+$ by applying Armstrong's Axioms:
  - **if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$** (reflexivity)
  - **if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$** (augmentation)
  - **if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$** (transitivity)
- Additional rules (derivable from Armstrong's Axioms):
  - **If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds** (union)
  - **If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds** (decomposition)
  - **If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds** (pseudotransitivity)

# FD Closure: Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \to B$
  $A \to C$
  $CG \to H$
  $CG \to I$
  $B \to H\}$
- some members of $F^+$
  - $A \to H$
    - by transitivity from $A \to B$ and $B \to H$
  - $AG \to I$
    - by augmenting $A \to C$ with G, to get $AG \to CG$ and then transitivity with $CG \to I$
  - $CG \to HI$
    - by union rule with $CG \to H$ and $CG \to I$



# Computing FD Closure

- To compute the closure of a set of FDs F:

$F^+ = F$
**repeat**
  **for each** FD $f$ in $F^+$
    apply reflexivity and augmentation rules on $f$
    add the resulting FDs to $F^+$
  **for each** pair of FDs $f_1$ and $f_2$ in $F^+$
    **if** $f_1$ and $f_2$ can be combined using transitivity
      **then** add the resulting FD to $F^+$
**until** $F^+$ does not change any further

(NOTE: More efficient algorithms exist)



# Minimal Cover of an FD Set

- The opposite of closure: what is the "minimal" set of FDs equivalent to F, having no redundant FDs (or extraneous attributes)
- Sets of FDs may have redundant FDs that can be inferred from the others
  - **Eg: $A \to C$ is redundant in: $\{A \to B,\ B \to C,\ A \to C\}$**
  - **Parts of an FD may be redundant**
    - E.g. on RHS: $\{A \to B,\ B \to C,\ A \to CD\}$ can be simplified to $\{A \to B,\ B \to C,\ A \to D\}$
    - E.g. on LHS: $\{A \to B,\ B \to C,\ AC \to D\}$ can be simplified to $\{A \to B,\ B \to C,\ A \to D\}$
  - (We'll cover these later under the heading of extraneous attributes)
- (NB Textbook calls this "canonical" cover, though there is no guarantee of uniqueness.)



# Closure of Attribute Sets

- Given a set of attributes $\alpha$, define the *closure* of $\alpha$ under $F$ (denoted by $\alpha^+$) as the set of attributes that are functionally determined by $\alpha$ under $F$:

$$\alpha \to \beta \text{ is in } F^+ \Leftrightarrow \beta \subseteq \alpha^+$$

- Algorithm to compute $\alpha^+$, the closure of $\alpha$ under $F$

  *result* := $\alpha$;
  **while** (changes to *result*) **do**
    **for each** $\beta \to \gamma$ **in** $F$ **do**
      **begin**
        **if** $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
      **end**

# Closure of Attribute Sets: Example

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  1. ***result* = *AG***
  2. ***result* = *ABCG*** **($A \rightarrow C$ and $A \rightarrow B$)**
  3. ***result* = *ABCGH*** **($CG \rightarrow H$ and $CG \subseteq AGBC$)**
  4. ***result* = *ABCGHI*** **($CG \rightarrow I$ and $CG \subseteq AGBCH$)**
- Is *AG* a candidate key?
  1. **Is AG a superkey?**
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  2. **Is any subset of AG a superkey?**
     1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
     2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$



# Closure of Attribute Sets: Uses

- Testing for superkey:
  - **To test if $\alpha$ is a superkey, we compute $\alpha^+$, and check if $\alpha^+$ contains all attributes of *R***
- Testing FDs
  - **To check if a FD $\alpha \rightarrow \beta$ holds (or, in other words, is in $F^+$), just check if $\beta \subseteq \alpha^+$**
  - **i.e. compute $\alpha^+$ by using attribute closure, and then check if it contains $\beta$**
  - **Is a simple and cheap test, and very useful**



# Extraneous Attributes

- Recall that we could have redundant FDs. *Parts* of FDs can also be redundant
- Consider a set *F* of FDs and the FD $\alpha \rightarrow \beta$ in *F*.
  - **Attribute A is extraneous in $\alpha$ if $A \in \alpha$ and *F* logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.**
  - **Attribute *A* is extraneous in $\beta$ if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies *F*.**
- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - ***B* is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping *B* from $AB \rightarrow C$).**
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - ***C* is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting *C***



# Multivalued Dependencies

Given a relation schema R (A, B, C), the multivalued dependency A→→B holds in r(R) iff the set of B values matching a given (A, C) value pair in r depend only on the A-value and is independent of the C-value. A, B and C may be composite attributes.

Example : Information about f**light** ***Service :***
***(Flight No, Day, plane Type)***

- {106} X {Monday, Thursday} x {747, 1011} and
  {204} X {Wednesday} x {807, 827}

| Flight | Day | Plane-Type |
|---|---|---|
| 106 | Monday | 747 |
| 106 | Thursday | 747 |
| 106 | Monday | 1011 |
| 106 | Thursday | 1011 |
| 204 | Wednesday | 807 |
| 204 | Wednesday | 827 |

# Multivalued Dependencies- Example

Example: **Relation *Service:***

| Flight | Day | Plane-Type |
|---|---|---|
| 106 | Monday | 747 |
| 106 | Thursday | 747 |
| 106 | Monday | 1011 |
| 106 | Thursday | 1011 |
| 204 | Wednesday | 807 |
| 204 | Wednesday | 827 |

- If a certain plane type can be used for a flight on one day it flies, that plane type can be used on any day the flight flies.
- {106} X {Monday, Thursday} x {747, 1011} and

  {204} X {Wenesday} x {807, 827}

**Flight $\rightarrow\rightarrow$ Day** and **Flight $\rightarrow\rightarrow$ Plane-Type** or

**MVD: Flight $\rightarrow\rightarrow$ Day | Plane-Type**



# Multivalued Dependencies (Cont.)

- **If we have tuples < f, d, p > and < f, d ', p ' > in relation service, then we must also have tuple**

  **< f , d', p > and < f, d, p'> .**
- MVDs can exist only if a relation has at least 3 attributes.
- Also in a given relation R(A, B, C) if A $\rightarrow\rightarrow$B holds iff A $\rightarrow\rightarrow$ C also holds. This fact can be expressed as

  A $\rightarrow\rightarrow$B|C



# Decomposition of a Relation

- The operation of breaking up a table into multiple tables is called a decomposition.
- Let $R$, $R_1$, $R_2$ be schemas such that $R = R_1 \cup R_2$ where R1 and R2 should not be disjoint.
- *For any relation r (R), can be decomposed* into relations *$r_1$ and $r_2$* with schema R1 and R2 respectively as $\prod_{R1}(r)$ and $\prod_{R2}(r)$.

  **Then** $r \subseteq \prod_{R1}(r) \bowtie \prod_{R2}(r)$
- *General Form of decomposition:*

*Let* $R = R_1 \cup R_2 \ldots\ldots \cup R_{m}$..

*Decompose r into* $\prod_{R1}(r), \prod_{R1}(r), \ldots\ldots \prod_{Rm}(r)$.

*Property:* $r \subseteq \prod_{R1}(r) \bowtie \prod_{R2}(r) \ldots\ldots \bowtie \prod_{Rm}(r)$



# Decomposition of a Relation-Example

Lending-schema :

(branch-name, branch-city, assets, customer-name, loan-number, amount)

Decompositions:

Branch-schema: (branch-name, branch-city, assets)

Loan-info-schema: (customer-name, loan-number, branch-name, amount)

How to ensure that the original data is recoverable?

1. **all attributes of the original schema R must appear in the decomposition (R1, R2), i.e. R = R1 $\cup$ R2**
2. **decomposition must be a lossless-join decomposition**

   $r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$

# Lossless and Lossy decompositions

How to ensure that the original data is recoverable?

1. **all attributes of the original schema R must appear in the decomposition (R1, R2), i.e. R = R1 ∪ R2**
2. **decomposition must be a lossless-join decomposition**

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- **If** $r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$ **then the decomposition is called Lossless Decomposition (ensuring that original data is recoverable),**

  ***otherwise the decomposition is lossy decomposition.***



# Lossy Decomposition: Example

*R = (A, B, C) , $R_1$ = (A,C) and $R_2$ = (A, B)*

r

| A | B | C |
|---|---|---|
| R | 1 | A |
| R | 2 | C |
| S | 1 | B |

$\Pi_{A,C}(r_1)$

| A | C |
|---|---|
| R | A |
| R | C |
| S | B |

$\Pi_{A,B}(r_2)$

| A | B |
|---|---|
| R | 1 |
| R | 2 |
| S | 1 |

$\Pi_{A,C}(r_1) \bowtie \Pi_{A,B}(r_2)$

| A | B | C |
|---|---|---|
| R | 1 | A |
| R | 2 | A |
| R | 1 | C |
| R | 2 | C |
| S | 1 | B |

Spurious Tuples

Thus, r is different to $\Pi_{A,C}(r1) \bowtie \Pi_{A,B}(r2)$ and so (R1, R2) is not a lossless-join decomposition of R.



# Lossless Decomposition: Example

- `Relation R (S,P, J)`

| S | P | J |
|---|---|---|
| S1 | P1 | J2 |
| S1 | P2 | J1 |
| S2 | P1 | J1 |
| S1 | P1 | J1 |

| S | P |
|---|---|
| S1 | P1 |
| S1 | P2 |
| S2 | P1 |

| P | J |
|---|---|
| P1 | J2 |
| P2 | J1 |
| P1 | J1 |

| J | S |
|---|---|
| J2 | S1 |
| J1 | S1 |
| J1 | S1 |

`is decomposed into`

`R1(S,P) , R2 (P,J) and R3 (J,S) .`



# Testing for Loss Less Decomposition

- The decomposition of R into R1 and R2 is lossless wrt F iff either $R1 \cap R2 \rightarrow (R1 - R2)$ belongs to $F^+$ or $R1 \cap R2 \rightarrow (R2 - R1)$ belongs to $F^+$
- E.g. gradeInfo (rollNo,Adhar,course,grade)

  Given set of FDs = {rollNo,course → grade;
  Adhar, course → grade;
  rollNo → Adhar;
  Adhar → rollNo}

  decomposed into grades (rollNo, course, grade) and studInfo (rollNo, Adhar) is lossless

  because rollNo → Adhar

# Join Dependencies

- A relation r(R) satisfies the join dependency, depicted as $*(r_1, r_2, \ldots r_n)$, if r can be nonloss decomposed into $r_1(R_1), r_2(R_2), \ldots r_n(R_n)$.
- In the relation r(S, P, J).

there is a JD *(r1, r2, r3)

| S# | P# | J# |
|---|---|---|
| S1 | P1 | J2 |
| S1 | P2 | J1 |
| S2 | P1 | J1 |
| S1 | P1 | J1 |

- Since <S1,P1> appear together in R(S,P,J) it will appear in the projection R1(S,P) and similarly <p1,J1> in R2 and <J1,S1> in R3. Hence, <S1,P1,J1> appears in the join of these projections.
- In other words, *if <s1,p1,J2> ,<s2,p1,J1> and <S1,P2,J1> appear in R (S,P,J) then <S1,P1,J1> also appears in R(S,P,J).*



# Multivalued Dependencies as a special case of Join Dependency

- Let r be a relation on schema R, and let R1, R2, and R3 be subsets of R such that R3 = R - (R1, R2)
- Relation r satisfies the MVD R1 $\rightarrow\rightarrow$ R2|R3 if only if r decomposes nonloss (Join Dependency) onto the relation schemas r1 (R1, R2) and r2 (R1, R3). R1 ,R2 and R3 may be composite
- So MVD is a special type of JD in which a relation can be nonloss decomposed into two relations as shown above
- In other words, JD is generalization of MVD.
- Similarly, an FD is an MVD in which the set of dependent values actually consists of a single value.



# Example (Cont.)

- The relation Service can be decomposed losslessly onto Service-Day (Flight, Day) and Service-Type (Flight , Plane-Type)
- Service-Day

| Flight | Day |
|---|---|
| 106 | Monday |
| 106 | Thursday |
| 204 | Wednesday |

Service-Type

| Flight | Plane-Type |
|---|---|
| 106 | 747 |
| 106 | 1011 |
| 204 | 707 |
| 204 | 727 |

| Flight | Day | Plane-Type |
|---|---|---|
| 106 | Monday | 747 |
| 106 | Thursday | 747 |
| 106 | Monday | 1011 |
| 106 | Thursday | 1011 |
| 204 | Wednesday | 807 |
| 204 | Wednesday | 827 |



# Example (Cont.)

- Consider a different relation *service_1* shown as follows. There is no MVD and it can not be decomposed losslessly onto Service-Day (Flight, Day) and Service-Type (Flight ,plane-type).

| Flight | Day | Plane-Type |
|---|---|---|
| 106 | Monday | 747 |
| 106 | Thursday | 747 |
| 106 | Thursday | 1011 |
| 204 | Wednesday | 707 |
| 204 | Wednesday | 727 |