Design Theory

SID	Class	Room	Time	Lat	Lng
4749732	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
2720942	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4823984	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4287594	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2984994	cs 145	Nvidia Aud	T/R 4:30-6	...	...
8472374	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4723663	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2478239	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4763268	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364532	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364573	cs 145	Nvidia Aud	T/R 4:30-6	...	...
3476382	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2347623	cs 145	Nvidia Aud	T/R 4:30-6	...	...
...	...	...	...	...	...
2364579	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
3476343	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
2322232	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W

~375 cs145 students

~300 cs245 students

Problems:

- Lots of data repetition.
- A single change (for example, Room change) needs a lots of update operations.
- Deletion causes unexpected data lost problem.
- Incomplete data insertion causes problem.

Here **Lat**, **Lng** are dependent on **Room**; **Room**, **Time** are dependent on **Class**.
That is, **Room** → { **Lat, Lng** } and **Class** → { **Room, Time** }

Updated version: More efficient solution if you decompose the table into 3 different tables based on the dependencies.

SID	Class
4749732	cs 145
2720942	cs 145
4823984	cs 145
4287594	cs 145
2984994	cs 145
8472374	cs 145
4723663	cs 145
2478239	cs 145
4763268	cs 145
2364532	cs 145
2364573	cs 145
3476382	cs 145
2347623	cs 145
...	...
2364579	cs 245
3476343	cs 245
2322232	cs 245

~375 cs145 students

~300 cs245 students

Class	Room	Time
cs 145	Nvidia Aud	T/R 4:30-6
cs 245	Nvidia Aud	T/R 3-4:30
cs 246	Nvidia Aud	M/W 3-4:30

Room	Lat	Lng
Nvidia Aud	37.4277° N	122.1742° W

Data Anomalies >>

- Problems that occur when we try to cram too much into a single relation are called anomalies.

1. **Redundancy:** Information may be repeated unnecessarily in several tuples.

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

If every course is in only one room, contains ***redundant*** information!

2. **Update Anomaly:** We may change information in one tuple but leave the same information unchanged in another.

Student	Course	Room
Mary	CS145	B01
Joe	CS145	C12
Sam	CS145	B01
..	..	..

If we update the room number for one tuple, we get inconsistent data
= an ***update* anomaly**

3. **Delete Anomaly:** If a set of values get deleted, we may lose other information as a side effect.

Student	Course	Room
..	..	..

If everyone drops the class, we lose what room the class was in!
= a ***delete* anomaly**

4. **Insert Anomaly:** We can't insert a new row because of some missing value whose value can't be null.

...	CS229	C12

→

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Similarly, we can't reserve a room without students
= an ***insert* anomaly**

After decomposition (without anomalies):

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

Course	Room
CS145	B01
CS229	C12

Is this form better?

- Any Redundancy?
- Any Update anomaly?
- Any Delete anomaly?
- Any Insert anomaly?

Mohammad Imam Hossain, Lecturer, dept. of CSE, UIU. Email: imambuet11@gmail.com

Normalization >>

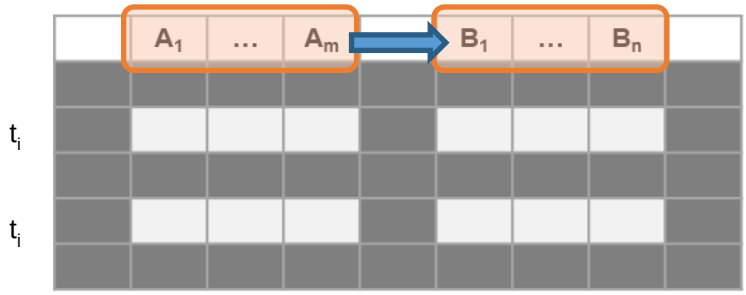
Normalization is a systematic approach of ***decomposing*** tables to eliminate data redundancy (repetition) and undesirable characteristics like insert, update and delete anomalies.

1.1) Functional Dependency:

- Let, A = { A_1, A_2, … … , A_m } and B = { B_1, B_2, … … , B_n } in R
- The functional dependency **A → B** on R holds if for any tuples t_i, t_j in R:

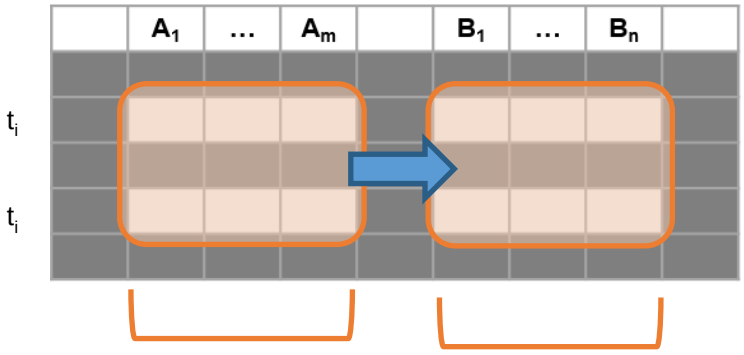
  $t_i[A] = t_j[A]$ implies $t_i[B] = t_j[B]$

  that is whenever two or more tuples in R agree on all the attributes of A, they must also agree on all the attributes of B.

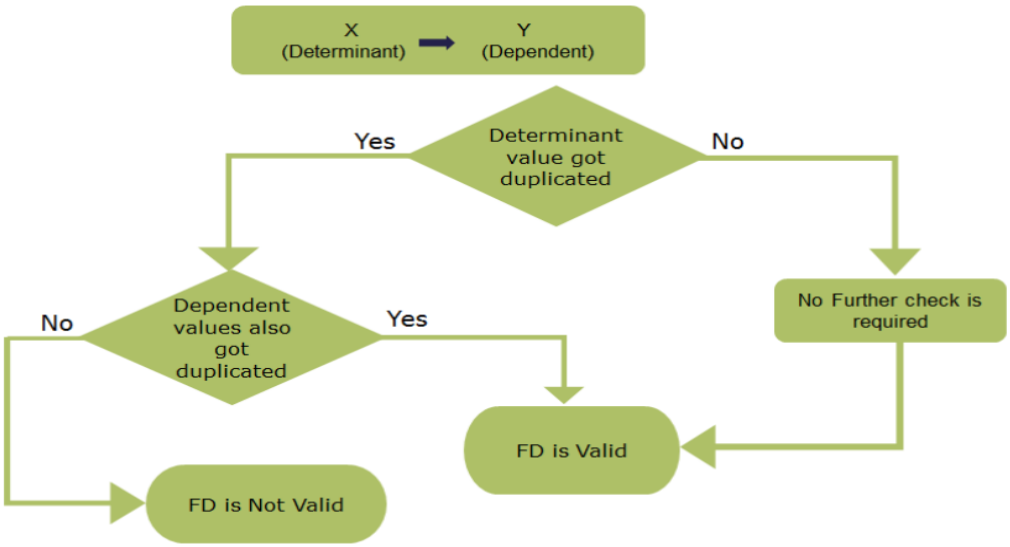


- if left side equals ti[A1] = tj[A1] , ti[A2]=tj[A2] , … … , ti[Am] = tj[Am]

  then right side also equals, ti[B1] = tj[B1] , ti[B2]=tj[B2] , … … , ti[Bn] = tj[Bn]



- **Flow diagram:**



- FD is a constraint that *holds/does not hold* on an instance.
- A particular instance of R may coincidently satisfy some FD but this FD may not hold for R in general.
- If the FD holds for every instances of Relation R, then FD becomes a part of the relational schema.
- *Example*,
    i. {position} -> {phone} holds for this instance.

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

    ii. {phone} -> {position} doesn't hold for this instance.

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

- **Practice:**

A	B	C
1	2	3
2	2	3
3	2	3
4	3	2
5	2	3
6	3	2

A → A	Valid	AB → A	Valid
A → B	Valid	AB → B	Valid
A → C	Valid	AB → C	Valid
B → A	Invalid	BC → A	Invalid
B → B	Valid	BC → B	Valid
B → C	Valid	BC → C	Valid
C → A	Invalid	CA → A	Valid
C → B	Valid	CA → B	Valid
C → C	Valid	CA → C	Valid

- Why we need FDs in Database Design:
    i. First we will start with some relational schema (received from ERD)
    ii. **[Task 1]** Then we will find out its Functional Dependencies.
    iii. **[Task 2]** Finally by using these FDs we will design a better schema that will minimize the possibility of anomalies.

1.2) Task 1 (Discover all FDs):

- **Armstrong's Axioms:**
    i. **Reflexivity rule**: If α is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.
    Ex: AB → B, here B is a subset of AB.
    ii. **Augmentation rule**: If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.
    Ex: if **AB → C** holds then, **AB D → C D** holds
    iii. **Transitivity rule**: If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.
    Ex: if **A → B** and **B → C**, then **A → C**

- Additional Rules:
  i. **Union rule**: If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.
     Ex: if **A → B** and **A → C**, then **A → BC**
  ii. **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.
     Ex: if **A → BC** then **A → B** and **A → C**
  iii. **Pseudo-transitivity rule**: If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.
     Ex: if **A → B** holds and **CB → D** then **CA → D**

- Let, **R = (A, B, C, G, H, I)** and **F = { *A → B, A → C, CG → H, CG → I, B → H* }**
  Then:
  - *A → H*. Since *A → B* and *B → H* hold, we apply the transitivity rule.
  - *CG → HI*. Since *CG → H* and *CG → I* , the union rule implies that *CG → HI*
  - *AG → I*. Since *A → C* and *CG → I*, the pseudo-transitivity rule implies that *AG → I* holds.

- Functional Dependency Closure: [out of syllabus]

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

Figure A procedure to compute F^+.

- Example:
  Let, **R = (A, B, C, D)** and **F = {*A → B, B → C*}**
  F^+ = {

$A \rightarrow A$, $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow B$, $B \rightarrow C$, $C \rightarrow C$, $D \rightarrow D$, $AB \rightarrow A$,
$AB \rightarrow B$, $AB \rightarrow C$, $AC \rightarrow A$, $AC \rightarrow B$, $AC \rightarrow C$, $AD \rightarrow A$, $AD \rightarrow B$,
$AD \rightarrow C$, $AD \rightarrow D$, $BC \rightarrow B$, $BC \rightarrow C$, $BD \rightarrow B$, $BD \rightarrow C$, $BD \rightarrow D$,
$CD \rightarrow C$, $CD \rightarrow D$, $ABC \rightarrow A$, $ABC \rightarrow B$, $ABC \rightarrow C$, $ABD \rightarrow A$,
$ABD \rightarrow B$, $ABD \rightarrow C$, $ABD \rightarrow D$, $BCD \rightarrow B$, $BCD \rightarrow C$, $BCD \rightarrow D$,
$ABCD \rightarrow A$, $ABCD \rightarrow B$, $ABCD \rightarrow C$, $ABCD \rightarrow D$.

  }

- **Inefficient process!!!!!**

- **Closure of Attribute Set:**

Let α be a set of attributes. We call the set of all attributes functionally determined by α under a set F of functional dependencies the **closure** of α under F. we denote it by α+.

Algorithm:

```
result := α;
while (changes to result) do
    for each functional dependency β → γ in F do
        begin
            if β ⊆ result then result := result ∪ γ;
        end
```

Figure An algorithm to compute α^+, the closure of α under F.

Example:

Let, **R = (A, B, C, G, H, I)** and **F = {*A → B, A → C, CG → H, CG → I, B → H*}**

Now, attribute closure of AG that is $(AG)^+$:

Initially,	$(AG)^+$	= AG	
		= AG B	[using A → B rule, as A is a part of AG]
		= AGB C	[using A → C rule, as A is a part of AGB]
		= AGBC H	[using CG → H rule, as CG is a part of AGBC]
		= AGBCH I	[using CG → I rule, as CG is a part of AGBCH]
		= AGBCHI	[using B → H rule, as B is a part of AGBCHI, no change]
		= AGBCHI	[no more check is needed as every FDs is checked]

Now,	$(AB)^+$	= AB	
		= AB	[using A → B rule, as A is a part of AB, no change]
		= AB C	[using A → C rule, as A is a part of AB]
		= ABC H	[using B → H rule, as B is a part of ABC]
		= ABCH	[couldn't use CG → H rule, as CG is not a part of ABCH]
		= ABCH	[couldn't use CH → I rule, as CG is not a part of ABCH]
		= ABCH	[no more changes is possible]

Practice:

If **R = (A, B, C, D, E)** and **F = {B → AC, C → AB, ABC → D, BD → A, AD → C, E → D}**
a) Find all the attribute closures with single element of R.
b) Find all the attribute closures for all the sets with two attributes from relation R.

Uses:

- **Superkey check:**

  To test if α is a superkey, we compute α+, and check if α+ contains all attributes of R. **Ex:** $(AG)^+$ = ABCGHI

- **FD validity checking:**

  We can check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), by checking if $\beta \subseteq \alpha^+$ ***Ex.*** *AG → I* is valid as **$(AG)^+$ = ABCGHI**

- **Determine all FDs: [No need]**

  For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

1.3) Different types of Keys:

- **Superkey:**

  Let *R* be a relation schema. A subset *K* of *R* is a **superkey** of *R* if, in any legal relation *r*(*R*), for all pairs t_1 and t_2 of tuples in *r* such that $t_1 \neq t_2$, then $t_1[K] \neq t_2[K]$.

  A set X of attributes in R is a superkey of R if and only if X^+ contains all attributes of R. In other words, X is a superkey if and only if it determines all other attributes.

- **Candidate key:**

  X is a candidate key if and only if it is a superkey, but none of its proper subset is a superkey.

All candidate key finding **algorithm**:

Observation 1: any candidate key must contain attributes that have not appeared on the RHS of any functional dependency. (RHS keys are those keys that need help from others to be determined).

Observation 2: if an attribute has occurred on the RHS of some FD, but not on the LHS of any FD, then it cannot be in any candidate key. (These keys are determined by others and no other keys are dependent on them).

Final Algorithm:

1) Find all the attributes that have not appeared on the RHS of any FD. Denote this set by $\boldsymbol{\alpha}$
2) Denote the set of attributes that appear on the RHS of some FD, but not on the LHS of any FD by $\boldsymbol{\beta}$
3) Compute the closure set α^+, if α^+= R, then α is the **only candidate key**.
4) If $\alpha^+ \neq$ R, then for each attribute x in R - β, test whether α U { x } is a candidate key. If not, try to add another attribute from R- β to α and test whether it is candidate key.
5) Repeat step 4, until all candidate keys have been found.

Example 1:

If **R = (A, B, C, D, E)** and **F = {A → C, CD → B}**
then, α = { A, D, E} , β = { B }

Now α^+ = ABCDE = R
So α is the only candidate key.

Example 2:

If **R = (A, B, C, D, E)** and **F = {A → C, C → BD, D → A}**
then, α = { E }, β = { B }

Now α^+ = { E } , not a superkey/candidate key. We will test each of {C, E} , {A, E}, {D, E} next (not {B, E}).

$\{C, E\}^+$ ={ C, E, B, D, A } . Therefore { C, E } is a superkey. { C, E } is also a candidate key since neither { E } nor { C } is a superkey.

{ A, E }$^{+}$ ={ A, E, C, B, D }. Similar to the above, { A, E } is a candidate key.

Similarly we can verify { D, E } is a candidate key.

Therefore {C,E}, {A,E}, {D,E} are all of the candidate keys.

Practice 1:

If **R=(A,B,C,D,E)** and **F = {A-->BC, CD-->E, B-->D ,E-->A}**
a) compute closure for each β in $\beta \rightarrow \gamma$ in F.
b) List candidate keys of R.

Practice 2:

If **R = (A, B, C, D, E)** and **F = {A→C, B→D, AC→D, CD→E, E→A}** then list the candidate keys of R

Practice 3:

If **R = (P, Q, R, S, T, U)** and **F = {PQ→RTU, PR→S, U→P, R→S, ST→PU}** then list the candidate keys of R.

Practice 4:

If **R = (U, V, X, Y, Z)** and **F = {UV→XZ, UX→Y, X→Y, VZ→YX, Z→UV}** then list the candidate keys of R.

1.4) Extraneous Attribute Detection:

An attribute of a functional dependency is said to be **extraneous** if we can remove it without changing the closure of the set of functional dependencies.

Let *R* be the relation schema, and let *F* be the given set of functional dependencies that hold on *R*. Consider an attribute *A* in a dependency $\alpha \rightarrow \beta$.

- If $A \in \beta$, to check if *A* is extraneous consider the set ***F'***= $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ and compute α+ (the closure of α) under ***F'***; if α+ includes *A*, then *A* is extraneous in β.

  Example:
  $F = \{ AB \rightarrow CD, A \rightarrow E, E \rightarrow C \}$, check if C is extraneous in AB → CD or not?

  formula, if F = { P → QR, Q → R } then R is extraneous in P → QR

- If $A \in \alpha$, to check if *A* is extraneous, let $\gamma = \alpha - \{A\}$, and compute γ+ (the closure of γ) under *F*; if γ+ includes all attributes in β, then *A* is extraneous in α.

  Example:
  F = { P→Q, PQ→R }, check if Q is extraneous in PQ→R?

1.5) Minimal Cover(No redundancy):

Given a set F of FDs, we say another set **E** of FDs is a minimal cover of **F** if

- Every FDs in **E** has a single attribute on the RHS.
- **F** and **E** are equivalent, that is, every FD in **E** can be inferred from the FDs in **F**, and every FD in **F** can be inferred from the FDs in **E**.
- Every FD ***A → b*** in **E** is minimal in its LHS, that is, there is no proper subset C of A such that ***C → b***
- There is no redundant FD in E. That is removing any FD from E will result in a set of FD that is not equivalent to F.

Algorithm:

Initially E=F
Step 1: rewrite each FD that has *m* attributes on the RHS into *m* FDs where the RHS is a single attribute.
Step 2: remove trivial FDs.
Step 3: minimize LHS of each FD. For each FD $X \rightarrow y$ in E, and for each attribute x in X, if $X-\{x\} \rightarrow y$ is implied by E, then replace $X \rightarrow y$ with $X-\{x\} \rightarrow y$.
Step 4: remove redundant FDs. For each FD in E, if it is implied by other FDs in E, then remove it from E.

Example:

If **R=(A, B, C, D, E, F)** and **F={ ABC → CDEF, C → E, A → B, D → F }**

Step 1

ABC → C
ABC → D
ABC → E
ABC → F

C → E
A → B
D → F

Step 2

ABC → C (cancel)
ABC → D
ABC → E
ABC → F

C → E
A → B
D → F

Step 3

C → E
A → B
D → F

AC → D
AC → F
ABC → E (cancel)

Step 4

C → E
A → B
D → F

AC → D
AC → F (cancel)

Final minimal cover, $F = \{AC \rightarrow D, C \rightarrow E, A \rightarrow B, D \rightarrow F\}$

Practices:

1. *F = { AB→CD, B→C, BC→D, CD→EF, E→F}*. Find minimal cover for this FD set.
Solution: F = {B→CD, CD→E, E→F} is minimal cover.

2. F = {A→BC,CD→E, E→C, D→AEH, ABH→BD, DH→BC}. Find minimal cover for this FD set.
Solution: F = {A→BC, D→AEH, AH→D, E→C} is minimal cover

3. F = { AB -> C, C -> A, BC -> D, ACD -> B, D -> E, D -> G, BE -> C, CG -> B, CG -> D, CE -> A, CE -> G}
Solution 1: {AB -> C, C -> A, BC -> D, CD -> B, D -> E, D -> G, BE -> C, CG -> D, CE -> G}

Solution 2: {AB -> C, C -> A, BC -> D, D -> E, D -> G, BE -> C, CG -> B, CE -> G}