index

normalization

- Normalization is the **process of organizing the data in the database**
- It is used to **minimize the redundancy** from a relation or set of relations
- It is also used to **eliminate the insertion anomaly,update anomaly and deletion anomaly**
- It divides the larger table into the smaller table and links them using relationship

Update anomalies
Deletion anomalies
Insert anomalies

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Functional dependency

- A functional dependency is an **association between two attributes of the same relational database table**.
- One of the attributes is called the **determinant** and the other attribute is called the **determined**
- If **A** is the **determinant** and **B** is the **determined** then we say that **A functionally determines B** and graphically represent this as **A -> B**

The following table illustrates A ⟹ B:

A	B
1	1
2	4
3	9
4	16
2	4
7	9

The following table illustrates that A does not functionally determine B:

A	B
1	1
2	4
3	9
4	16
3	10

Armstrong's Axioms

- Armstrong's Axiom is a **mathematical notation used to find the functional dependencies in a database**.
- Conceived by **William W. Armstrong**
- **It is a list of axioms or inference rules** that can be implemented on any relational database.
- It is denoted by the symbol **F+**.

Various Axioms Rules

A. Primary Rule

Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

B. Secondary Rules

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Closure Of Functional Dependency

- The Closure Of Functional Dependency means the **complete set of all possible attributes that can be functionally derived from given functional dependency**
- If "**F**" is a **functional dependency** then **closure of functional dependency** can be denoted using "**{F}+** ".
- There are **three steps** to calculate closure of functional dependency

Step-1 : Add the attributes which are present on Left Hand Side in the original functional dependency.

Step-2 : Now, add the attributes present on the Right Hand Side of the functional dependency.

Step-3 : With the help of attributes present on Right Hand Side, check the other attributes that can be derived from the other given functional dependencies. Repeat this process until all the possible attributes which can be derived are added in the closure.

Example:
Consider the table student_details having (Roll_No, Name,Marks, Location) as the attributes and having two functional dependencies.
FD1 : Roll_No -> Name, Marks
FD2 : Name -> Marks, Location

Step-1: {Roll_no}+ = {Roll_No}
Step-2 : {Roll_no}+ = {Roll_No,Name, Marks}
Step-3 : **{Roll_no}+ = {Roll_No, Marks, Name, Location}**

Step-1 : {Name}+ = {Name}
Step-2 : {Name}+ = {Name, Marks, Location}
Step-3 : Since, we don't have any functional dependency where "Marks or Location".
So **{Name}+ = {Name, Marks, Location}**
{Marks}+ = {Marks} and **{Location}+ = { Location}**

Equivalence of Functional Dependencies (FD)

- Two different sets of functional dependencies for a given relation may or may not be equivalent.
- If **FD1 can be derived from FD2**, we can say that FD2 ⊃ FD1.
- If **FD2 can be derived from FD1**, we can say that FD1 ⊃ FD2.
- If above two cases are true, **FD1=FD2.**

Eg:A relation R(A,B,C,D) having two FD sets FD1 = {A->B, B->C, AB->D} and FD2 = {A->B, B->C, A->C, A->D}

Step 1. **Checking whether all FDs of FD1 are present in FD2**

- A->B YES
- B->C YES
- AB->D YES For set FD2, (AB)+ = {A,B,C,D}.

FD2 ⊃ FD1 is true.

Step 2. **Checking whether all FDs of FD2 are present in FD1**

- A->B YES
- B->C YES
- A->C YES For set FD1, (A)+ = {A,B,C,D}.
- A->D YES For set FD1, (A)+ = {A,B,C,D}

FD1 ⊃ FD2 is true.

Step 3. As FD2 ⊃ FD1 and FD1 ⊃ FD2 both are true FD2 =FD1 is true. These two FD sets are semantically equivalent.

Minimal Cover

- Whenever a user updates the database, the system must check whether any of the functional dependencies are getting violated in this process. If there is a violation of dependencies in the new database state, the system must roll back. Working with a huge set of functional dependencies can cause unnecessary added computational time. This is where the minimal cover comes into play.

There are **4 rules** to find Minimal cover :

1. Break down the RHS of each functional dependency into a single attribute
2. Find redundant fds
3. Minimize LHS.
4. Group the functional dependencies that have common LHS together into a Single FD .

Q1. Minimal cover of F with dependencies F={BC->ADEF, F->DE} ?

STEP 1: Break down the RHS

BC->A BC->D BC->E BC->F F->D F->E

STEP 2: Find redundant fds

- Assume BC->A is redundant fd and we remove this fd,now try computing (BC)+={BCDEF} but there is no A. so BC->A is not redundant
- BC->D (BC)+={BCAEFD}, D is present so BC->D is redundant
- BC->E (BC)+={BCADFE}, E is present so BC->E is redundant
- BC->F (BC)+={BCADE}, F is not present so BC->F is not redundant

so we get

BC->A BC->F F->D F->E

BC->A BC->F F->D F->E

STEP 3: Minimize LHS

BC->A BC->F

- From BC-> A, if we remove B and then we get C->A. By taking closure (C)+={C,A}, there is no B. same way remove C then B->A. By taking closure there is no C. So it can't be minimize
- BC->F ,it also can't be minimize

so we get

BC->A BC->F F->D F->E

Step 4: Group the functional dependencies that have common LHS together into a Single FD .

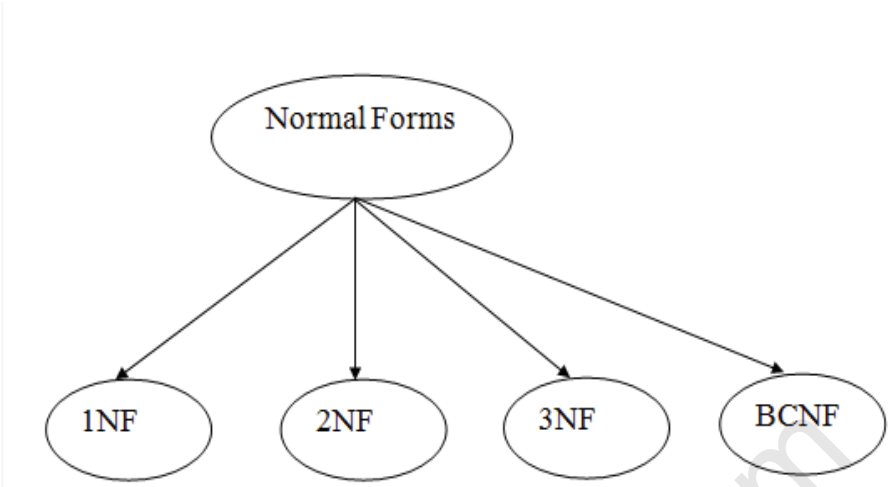
so the minimal cover is

BC->A F F->DE

index

normalization

- Normalization is the **process of organizing the data in the database**
- It is used to **minimize the redundancy** from a relation or set of relations
- It is also used to **eliminate the insertion anomaly,update anomaly and deletion anomaly**
- It divides the larger table into the smaller table and links them using relationship



First Normal Form (1NF)

- A relation will be **1NF if it contains an atomic value.**
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, **relational must be in 1NF**.
- In the second normal form, **No non-prime attribute is dependent on the proper subset of any candidate key of table**

Non-prime attribute: An attribute that is not a part of any candidate key

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- In 3NF,the relation must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed

Transitive functional dependency: **A->B** & **B->C** ,THEN **A->C**

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above: {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on
Candidate key: {EMP_ID}
Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd Normal Form (BCNF)

- BCNF is the **advance version of 3NF**.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1.EMP_ID → EMP_COUNTRY

2.EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

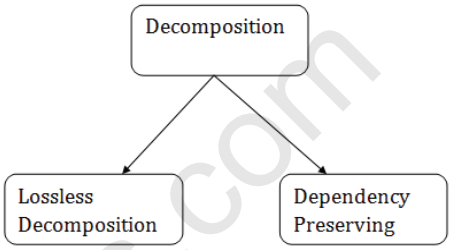
EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

index

- ❑Lossless join and dependency preserving decomposition
- ❑Algorithms for checking Lossless Join (LJ)

Lossless join and dependency preserving decomposition

- **Decomposition** of a relation is done when a relation in relational model is not in appropriate normal form.
- Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.



Lossless Join Decomposition

- If the information is **not lost from the relation that is decomposed**, then the decomposition will be **lossless**.
- ie, the relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

- If we decompose a relation R into relations R1 and R2

1. Decomposition is lossy if R1 ⋈ R2 is not R
2. Decomposition is lossless if R1 ⋈ R2 is equal to R

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

⋈

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

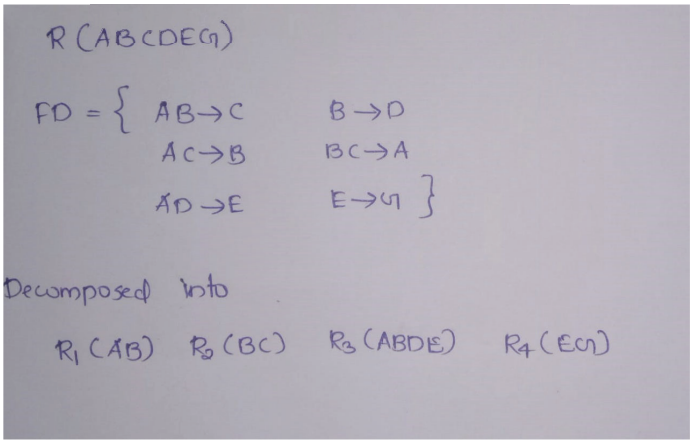
Employee

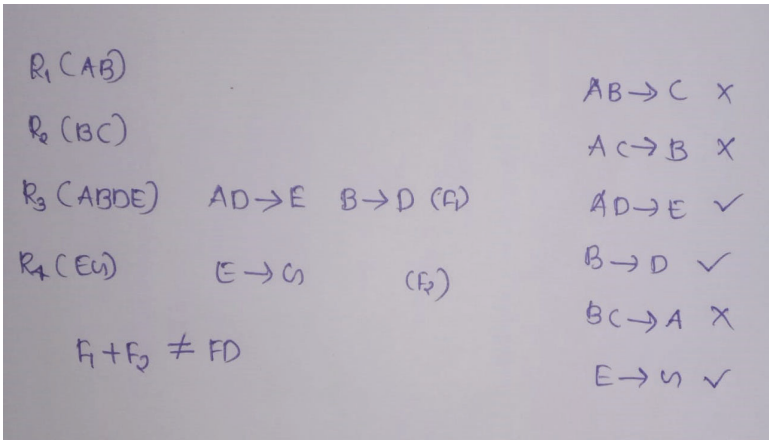
DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Department

Dependency Preserving Decomposition

- The dependency preservation property, which ensures that each functional dependency is represented in some individual relation resulting after decomposition
- In the dependency preservation, **at least one decomposed table must**





Algorithms for checking Lossless Join (LJ)

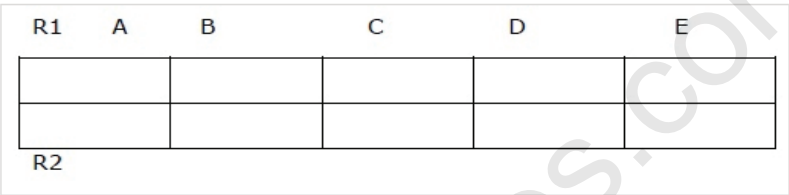
R(A,B,C,D,E)

F:{A->B, BC->E, ED->A}

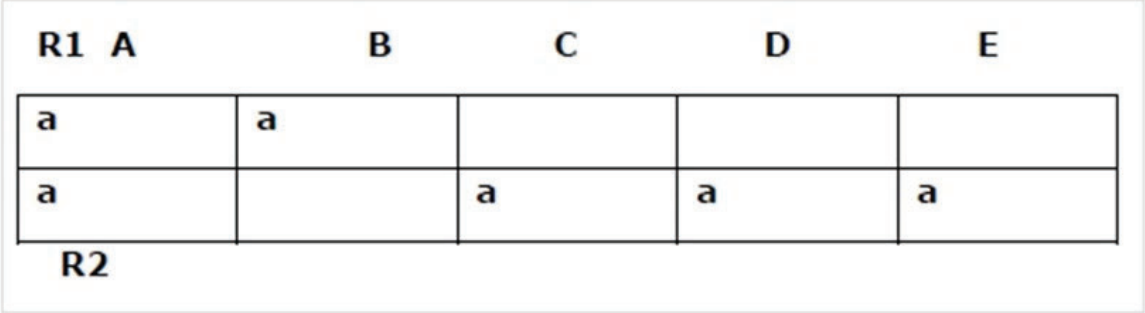
R is decomposed into R1(AB) and R2(ACDE)

Step 1 – Create a table with M rows and N columns

- M= number of decomposed relations.
- N= number of attributes of original relation.



Step 2 – If a decomposed relation Ri has attribute A then Insert a symbol (say 'a') at position (Ri,A)



Step 3 – Consider each FD X->Y

If column X has two or more symbols then

Insert symbols in the same place (rows) of column Y.

Now let us insert symbol ‘a’ for A->B in second column, second row

R1

A	B	C	D	E
a	a			
a	a	a	a	a

R2

Step 4 – If any row is completely filled with symbols then

Decomposition is lossless.

Else

Decomposition is lossy.

R2 is completely filled => decomposition is lossless.