Lecture 15 Normal Form & Schema Decomposition

CS211 - Introduction to Database

Database design optimization

- When we are given a set of tables specifying a database, they may have come from an **ER diagram which is not absolutely perfect**.

- The database designer will examine: physical storage information, enforcing data constraints, avoiding anomalies and redundancies.

- If there are problems to address, the designer may want to restructure the database, without losing any information.

Problems with a large Relation Schema

Student Relation

Sid is the Primary Key

Sid	Name	Credits	Dept	Building	Room_no	HOD	……….
1	John	5	CS	B1	101	-	
2	Adam	8	CS	B1	101	-	
3	Jiya	9	DS	B2	201	-	
4	Salim	9	DS	B2	201	-	
5	Xi	7	Civil	B1	110	-	
6	Chen	6	EC	B2	115	-	
7	Rahul	8	Civil	B1	120	-	
8	Allan	9	CS	B1	101	-	
NULL	NULL	NULL	ME	B2	120	-	

Redundant Storage

Update Anomaly

Deletion Anomaly

Insertion Anomaly

Solution by Relation Decomposition (Normalization)

Student Relation Sid is the Primary Key

Sid	Name	Credits	Dept	Building	Room_no
1	John	5	CS	B1	101
2	Adam	8	CS	B1	101
3	Jiya	9	DS	B2	201
4	Salim	9	DS	B2	201
5	Xi	7	Civil	B1	110
6	Chen	6	EC	B2	115
7	Rahul	8	Civil	B1	110
8	Allan	9	CS	B1	101
?	?	?	ME	B2	120

40 cells

- **Redundancy removed**
- **Insertion Anomaly removed**
- **Update Anomaly removed**
- **Delete Anomaly removed**

Student Sid is the Primary Key

Sid	Name	Credits	Dept
1	John	5	CS
2	Adam	8	CS
3	Jiya	9	DS
4	Salim	9	DS
5	Xi	7	Civil
6	Chen	6	EC
7	Rahul	8	Civil
8	Allan	9	CS

36 cells

Foreign Key

Department Dept is the Primary Key

Dept	Building	Room_no
CS	B1	101
DS	B2	201
Civil	B1	110
EC	B2	115
ME	B2	120

18 cells

Decomposition of Relation Schema

- The process of breaking up of a relation into smaller sub-relations is called Decomposition.

- Decomposition is required in DBMS to **convert a relation into specific normal form** which reduces redundancy, anomalies, and inconsistency in the relation.

- Database normalization is the process of "reorganizing" a relational database by, generally breaking up tables (relations) to remove various anomalies and redundancy.

- Decomposition should preserve the following three properties:
  1. Lossless decomposition
  2. Dependency Preservation
  3. Remove redundant functional dependency

Benefits of Normalization

- Reduction in redundant data
- Saves storage space
- Remove anomalies
- Avoids NULL values
- Simplifies queries
- Makes Searching, Sorting, and Creating indexes faster
- Simplifies database schema (Easy to understand purpose of table by its schema)

Normal Form

Type of Normal forms

Normalization is achieved using different **normal forms**. **A normal form applies to a table/relation schema, not to the whole database schema.**

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)
- Sixth Normal Form (6NF)

➢ The Theory of Data Normalization is still being developed further.

➢ **However, in most practical applications, normalization achieves its best in 3rd Normal Form.**

Types of data dependencies

To normalize a table, first look at **four** types of **data dependencies**:

1. **Full key dependency:** From the candidate-key (prime attributes) to outside the candidate-key (non-prime attributes).

R = {A, B, C, D, E, F} **Candidate-key is {A, B}**

{**A**, **B**} → {C, D, E, F} /* Full key dependency */

{**A**, **B**} → {B} /* Not a full key dependency */

{**A**} → {C, D} /* Not a full key dependency */

{**A**, **B, C**} → {D} /* Not a full key dependency */

Types of data dependencies

2. **Partial dependency:** From the proper subset of any candidate-key (prime attributes) to outside the candidate-key (non-prime attributes).

R = {A, B, C, D, E, F} **Candidate-key is {A, B}**

{**A**} → {C, D} /* Partial dependency */

{**B**} → {D, E, F} /* Partial dependency */

{**A**} → {C, D, E, F} /* Not a partial dependency */

{**B, C**} → {D} /* Not a partial dependency */

{**A**, **B**} → {C, D, E} /* Not a partial dependency */

Types of data dependencies

3. **Transitive dependency:** A non-prime attribute depending on another non-prime attribute, which is entirely dependent on candidate key.

R = {A, B, C, D, E, F} **Candidate-key is {A, B}**

F1 = {**A**, **B**} → {C} , {C} → {D}, {D} → {E} /* Transitive dependency */

F2 = {**A**, **B**} → {C} , {C} → {B} /* Non-transitive dependency */

F3 = {F} → {D, E} /* Non-transitive dependency */

F4 = {**A**} → {E}, {E} → {F} /* Partial & Non-transitive dependencies */

Types of data dependencies

4. **Into key dependency:** From outside the candidate-key (non-prime attributes) into the candidate-key (prime attributes).

R = {A, B, C, D, E, F} **Candidate-key is {A, B}**

{D} → {**A**, **B**} /* Into key dependency */

{E, F} → {**A**, **B**} /* Into key dependency */

{C,D} → {**A**} /* Into key dependency */

{C,D} → {**A**, E} /* Not a into key dependency */

Normal Forms

- **First Normal Form (1NF):** Relation has only **singled valued attributes**

- **Second Normal Form (2NF):** Relation is in **1NF** and **no partial dependencies**

  OR

  Relation is in **1NF** and **there is full key dependency**

- **Third Normal Form (3NF):** Relation is in **2NF** and **no transitive dependencies**

- **Boyce-Codd Normal Form (BCNF):** Relation is in **3NF** and for every FD, **LHS is super key**

First Normal Form – 1NF (Essential normal form)

For a table to be in the First Normal Form, it should follow the following **5 rules**:

1. It should only have single(atomic) valued attributes.
2. Values stored in a column should be of the same domain.
3. All the columns in a table should have unique names.
4. Each record needs to be unique (there should be a primary key).
5. And the order in which data is stored, does not matter.

Student

Roll_no	Name	Addr	Subject	Subject
101	John	Changi, SN	OS, CN	Music
102	Jiya	Delhi, IN	C, C++	Drama
CS100	Salim	Lahore, PK	DBMS	Music
102	Jiya	Delhi, IN	C, C++	Drama

Original table not in 1NF

1NF Decomposition

For a table to be in the First Normal Form, it should follow the following 5 rules:

1. It should only have single(atomic) valued attributes
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. Each record needs to be unique (there should be a primary key)
5. And the order in which data is stored, does not matter

Student

Roll_no	Name	Addr	Subject	Subject
101	John	Changi, SN	OS, CN	Music
102	Jiya	Delhi, IN	C, C++	Drama
CS100	Salim	Lahore, PK	DBMS	Music
102	Jiya	Delhi, IN	C, C++	Drama

Original table not in 1NF

Table conversion

1st Method

Student

36 cells

Roll_no	Name	City	Country	**Mj_Sub**	Mi_Sub
101	John	Changi	SN	OS	Music
101	John	Changi	SN	CN	Music
102	Jiya	Delhi	IN	C	Drama
102	Jiya	Delhi	IN	C++	Drama
104	Salim	Lahore	PK	DBMS	Music

Table conversion is not a good approach

1NF Decomposition

For a table to be in the First Normal Form, it should follow the following 5 rules:

1. It should only have single(atomic) valued attributes
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. Each record needs to be unique (there should be a primary key)
5. And the order in which data is stored, does not matter

Student

Roll_no	Name	Addr	Subject	Subject
101	John	Changi, SN	OS, CN	Music
102	Jiya	Delhi, IN	C, C++	Drama
CS100	Salim	Lahore, PK	DBMS	Music
102	Jiya	Delhi, IN	C, C++	Drama

Original table not in 1NF

Decomposition

2nd Method

Student **20 cells**

Roll_no	Name	City	Country	Mi_Subject
101	John	Changi	SN	Music
102	Jiya	Delhi	IN	Drama
104	Salim	Lahore	PK	Music

12 cells

Major

Roll_no	Mj_Subject
101	OS
101	CN
102	C
102	C++
104	DBMS

Second Normal Form (2NF)

For a table to be in the Second Normal Form, it should follow **2 rules**:

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

Employee

Employee_No	**Department_No**	Employee _Name	Department
1	101	John	Irrigation
2	102	Chen	Fishery
3	101	Rama	Irrigation

F

1. **{Employee_No, Department_No} → {Employee_Name, Department}**
2. **Department_No → Department** (it is a partial dependency)

Decomposition

Employee

Employee No	**Department No**	Employee Name
1	101	John
2	102	Chen
3	101	Rama

Department

Department No	Department
101	Irrigation
102	Fishery

- If all the attributes in R are the prime attributes, then R is always in 2NF since there can be no partial dependency.
- If R has a single attribute in the candidate-key, then R is always in 2NF since there can be no proper subset of candidate-key, and hence no partial dependency.

2NF Decomposition

Given ***R* =*{A,B,C,D}*** and ***F={A→B, B→C}***

- From the given FDs, the candidate-key is *[AD].*
- The FD *A→B* forms the partial dependency since A ⊆ candidate-key *[AD]*.

Step-1: Find the closure of partial-key *A* which participates in the partial dependency

$A^+ = \{ABC\}$

Step-2: Decompose *R* into following two relations *R1* and *R2*

(1) R1 = A^+ = **{ABC}**

(2) R2 = { super-key(R1), R-R1} = **{AD}**

Since the common attribute *A* used in the decomposition is super-key, it is a lossless decomposition

2NF Decomposition

Step-3: Check if the normal form of R1 and R2 is in 2NF after the decomposition

- Find the **FDs** of ***R1={ABC}*** and ***R2={AD}*** by finding attribute closures using given ***F={A→B, B→C}***

  F1 = { A→BC, B→C} , and F2 = {ϕ}

- Find the **candidate-key** of *F1* and *F2* using attribute closures to determine the existence of any partial dependency

  candidate-key of F1=[A] , and F2 = {ϕ}

- Since there is no partial dependency, both relations R1 and R2 are in 2NF

  R1 has only one attribute in the candidate-key, so no chance of partial key

  R2 has no candidate key at all, so no chance of partial key

Step-4: Check for dependency preservation

- **F1 ∪ F2 = {A→BC, B→C} contains all the original FDs in F = {A→B, B→C}**
- **Hence, this decomposition preserves all the original dependencies**

Third normal form (3NF)

For a table to be in the Third Normal Form, it should follow **2 rules**:

1. It is in 2NF.
2. There is **no transitive dependency** for non-prime attributes.

A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency X –> Y:

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

Student

Roll_No	Student_Name	Student_age	**Pincode**	Student_city	Student_country
1	John	20	100	Peatrh	Australia
2	Adam	19	200	Chicago	USA
3	Chen	20	100	Peatrh	Australia
4	Amy	29	200	Chicago	USA
5	Susan	19	200	Chicago	USA

F

1. Roll_No → {Student_Name, Student_age, Pincode}

2. Pincode → {Student_city, Student_country}

Decomposition

Student

Roll_No	Student_Name	Student_age	Pincode
1	John	20	100
2	John	19	200
3	Chen	20	100
4	Amy	29	200
5	Susan	19	200

City

Pincode	Student_city	Student_country
100	Peatrh	Australia
200	Chicago	USA

3NF Decomposition

Given ***R* ={A,B,C,D}** and ***F*={A →B, B →C, C →D}**

- From the given FDs, the candidate-key is ***[A].***
- The FDs *B →C* and *C →D* form the transitive dependencies.

Step-1: Find the closure of B which participates in the transitive dependency *B→C*. Form a new relation *R1* from this closure.

B^+ = {BCD} , So **R1={B,C,D}**

Step-2: Find the closure of C which participates in the transitive dependency *C →D*. Form a new relation *R2* from this closure.

C^+ = {CD} , So **R2={C,D}**

Step-3: Form a new relation *R3* as follows*:*

R3 = { (R - {R1 ∪ R2}), any super-key of either R1 or R2 }

R3 = { A, B} **OR** **R3 = {A, C}**

Since the common attribute used in the decomposition is super-key, it is a lossless decomposition

3NF Decomposition

Step-4: Check if the normal form of R1, R2 and R3 is in 3NF after the decomposition

- Find the **FDs** of ***R1={BCD} , R2={CD} , and R3={AB}*** by finding attribute closures using given ***F={A→B, B→C, C→D}***

  F1 = { B→CD, C→D} , F2 = {C→D} , F3 = {A→*B*}

- Find the **candidate-key** of *F1*, F2 and *F3* using attribute closures to determine the existence of any transitive dependency.

  candidate-key of F1=[B] , F2=[C] and F3 = [A]

- There is no transitive dependency in relations R2 and R3 since their FDs have candidate-key on LHS.
- But in relation R1, there is a transitive dependency in **C→D**.

Step-4.1: **a)** Find the closure of **C** which participates in the transitive dependency *C→D* using ***F1 = { B→CD, C→D}***

C^+ = {CD} , super-key is C

b) Form a new relation *R11* as follows*:*

R11= { (R1 - {CD}), super-key C}

R11={B,C} and **F11 = {B →C}**

- There is no transitive dependency in relation R11 since its FD have candidate-key on LHS.

- We got the final 3NF decomposition as **R2={CD} , R3={AB}, R11={B,C}**

3NF Decomposition

Step-5: Check for dependency preservation

- **$\mathbf{F2} \cup \mathbf{F3} \cup$ F11= {C→D, A→B, B→C} contains all the original FDs in *F={A →B, B →C, C →D}***
- **Hence, this decomposition preserves all the original dependencies**

An algorithm for (almost) 3NF Lossless-Join Decomposition

1. Compute F_m, the minimal cover for F
2. For each $X \rightarrow Y$ in F_m, create a new relation schema XY
3. For every relation schema that is a subset of some other relation schema, remove the smaller one.
4. The set of the remaining relation schemas is an almost final decomposition

Boyce-Codd Normal Form (BCNF)

For a table to be in the BCNF Normal Form, it should follow **2 rules**:

1. It is in 3NF.
2. The LHS of every functional dependency is a super-key.

A relation is in BCNF if in every **non-trivial** functional dependency X –> Y, **X is a super key**.

- A table in BCNF is automatically in 3NF as no **transitive dependencies** are possible.
- It is an extension to 3NF and hence cannot be said 4NF.
- Table in 3NF but not in BCNF is **unusual.**
- BCNF can handle **overlapping candidate-keys** which is not possible in 3NF.
  For example where all {AB}, {BC}, {DC} are candidate-keys.

Example:

R={A,B,C}, F={(A, B)→C , C→B}, candidate key is (AB)

1. (A, B)→C and C→B meet 3NF as there are no partial or transitivity dependencies
2. C→B violate BCNF as C is not a super key

Relation in 1NF, 2NF, 3NF but not in BCNF

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Cshrp
104	Java	P.Java

Student

F

{ student_id, subject} → professor , professor → {subject}

- One student can enroll for multiple subjects. For example, student with **student_id** 101, has opted for subjects - Java & C++
- For each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like we have for Java.

1. This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

2. This table also satisfies the **2nd Normal Form** as their is no **Partial Dependency**.

3. And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.

4. But this table is not in **Boyce-Codd Normal Form** since **professor is not a super-key in FD**

   professor → {subject}

Relation in 1NF, 2NF, 3NF but not in BCNF

Student

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Cshrp
104	Java	P.Java

Decomposition

Student

student_id	professor
101	P.Java
101	P.Cpp
102	P.Java2
103	P.Cshrp
104	P.Java

Professor

professor	subject
P.Java	Java
P.Cpp	C++
P.Java2	Java
P.Chash	C#

BCNF Decomposition Algorithm

Input: relation *R*, and set of FDs *F* for R

Output: decomposition of R into BCNF relations with “lossless join”

Compute candidate-keys for R

Repeat until all relations are in BCNF:

> Pick any *R’* with $\alpha \rightarrow \beta$ that violates BCNF
>
> Decompose *R’* into $R1(\alpha, \beta)$ and $R2(\alpha, rest)$
>
> Compute *FDs* for *R1* and *R2*
>
> Compute *candidate-keys* for *R1* and *R2*

Student

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Cshrp
104	Java	P.Java

Decomposition

Student

student_id	professor
101	P.Java
101	P.Cpp
102	P.Java2
103	P.Cshrp
104	P.Java

Professor

professor	subject
P.Java	Java
P.Cpp	C++
P.Java2	Java
P.Chash	C#

BCNF decomposition example-1

***R*={A,B,C,D,E}** and ***F* = { AC → BE, C → D }**, Candidate key is **{A, C}**

C → D violates BCNF

Pick C → D , $\alpha = C$ and $\beta = D$

- Decompose R into $R1(\alpha, \beta)$ and $R2(\alpha, rest)$

  R_1 = CD, R_2 = ABCE

  F_1 = C → D, F_2 = {AC → BE}

  CK_1 = C, CK_2 = {AC}

- Both R_1 and R_2 are in BCNF

- Result: R_1 R_2

Input: relation R, and set of FDs F for R

Output: decomposition of R into BCNF relations with "lossless join"

Compute candidate-keys for R

Repeat until all relations are in BCNF:

  Pick any R' with $\alpha \rightarrow \beta$ that violates BCNF

  Decompose R' into $R1(\alpha, \beta)$ and $R2(\alpha, rest)$

  Compute *FDs* for $R1$ and $R2$

  Compute *candidate-keys* for $R1$ and $R2$

BCNF decomposition example-2

R={A,B,C,D,E} and F = { A → BE, C → D } , Candidate key is {A, C}

Both A → BE, C → D violates BCNF

Pick A → BE. So $\alpha = A$ and $\beta = BE$ We can also pick C→D which will give the same decomposition result

- Decompose R into $R1(\alpha, \beta)$ and $R2(\alpha, rest)$

  R_1 = ABE, R_2 = ACD
  F_1 = A → BE, F_2 = C → D
  CK_1 = A, CK_2= {∅}

- R_1 is in BCNF and R_2 is not in BCNF

- Decompose R_2 into:

  R_3 = CD, R_4 = AC
  F_3= C → D, F_4 = {∅}
  CK_3 = C, CK_4 = {∅}

- Both R_3 and R_4 are in BCNF

- Result: R_1 R_3 R_4

Input: relation R, and set of FDs F for R

Output: decomposition of R into BCNF relations with "lossless join"

Compute candidate-keys for R

Repeat until all relations are in BCNF:

  Pick any R' with $\alpha \rightarrow \beta$ that violates BCNF

  Decompose R' into $R1(\alpha, \beta)$ and $R2(\alpha, rest)$

  Compute *FDs* for $R1$ and $R2$

  Compute *candidate-keys* for $R1$ and $R2$

Key Points:

- A relation in a Relational Database is always and at least in 1NF form.
- BCNF is free from redundancy.
- If a relation is in BCNF, then it is also in 3NF.
- If all attributes of relation are prime attributes, then the relation is always in 3NF.
- Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
- If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
- Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
- Generally, normalize up to 3NF and not up to BCNF.
- There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it’s generally not required to go beyond BCNF. So the database is sometimes not fully normalized.