LECTURE 11: NORMALISATION

COMP2004J: Databases and Information Systems

Dr. Ruihai Dong (ruihai.dong@ucd.ie)
UCD School of Computer Science
Beijing-Dublin International College

What is Normalisation?

- **Normalisation** is the process of transforming data from a problem into relations, ensuring **data integrity** and eliminating **data redundancy**.
  - **Data Integrity**: Database is consistent and satisfies all constraint rules.
  - **Data Redundancy**: If data can be found in two places in a single database (direct redundancy) or calculated using data from different parts of the database (indirect redundancy) then redundancy exists.
- Normalisation should remove redundancy, but not at the expense of data integrity.

Problem of Redundancy

- If redundancy exists then this can cause problems during normal database operations:
  - When data is inserted into the database, the data must be duplicated wherever redundant versions of that data exists.
  - When data is updated, all redundant data must be updated at the same time.

Integrity Constraints

- An **integrity constraint** is a rule that restricts the values that may be present in the database.
- **Entity integrity:** the rows (or tuples) in a relation represent entities, and each one must be uniquely identified.
  - We must have a **primary key** that must have a unique non-null value for every row.
- **Referential integrity:** Involves the foreign keys.
  - Foreign keys tie the relations together, so it is important that the links are correct.
  - Every foreign key must either be null, or its value must be the actual value of a key in another relation.

Normal Forms

- The data in a database can be considered to be in one of a number of "**normal forms**".
- Basically, the normal form of the data indicates how much redundancy is in the data.
- The normal forms have a strict ordering:
  - 1st Normal Form
  - 2nd Normal Form
  - 3rd Normal Form
  - Boyce-Codd Normal Form (BCNF)
  - 4th Normal Form
  - 5th Normal Form

Functional Dependencies

- Sometimes the starting point for understanding the data requirements is given using **functional dependencies**.
- A function dependency is shown by two lists of attributes separated by an arrow.
  - If we know the values for the left-side attributes, we can find the values for the right.
- **Example**, a relation: student(s_number, firstname, lastname, advisor_number, advisor_name)
  - s_number → firstname, lastname, advisor_number, advisor_name
  - This is a functional dependency: if we know somebody's student number, we can find their name, their advisor's number and their advisor's name.

Functional Dependencies

- student(s_number, firstname, lastname, advisor_number, advisor_name)

- In the same way:
  - advisor_number → advisor_name
  - This is also a functional dependency: if we know the ID number of a student's advisor, we can find their name.

- But:
  - lastname, firstname →s_number
  - advisor_name → advisor_number
  - These are **NOT** functional dependencies. Two students could have the same name, so we cannot reliably find the student's ID number from a name. Similarly, two advisors could have the same name.

Example Relation

- student(<u>Snum</u>, Name, DoB, (Subject, Grade))

<u>SNum</u>	**Name**	**DoB**	**Subject**	**Grade**
12345	Smith.J	02/03/96	Java	B
			Soft Eng	C
			Databases	A
23456	White.A	04/02/94	Java	D
			Soft Eng	B
34567	Moore.T	06/10/95	Databases	A
			Soft Eng	B
			Networks	C
45678	Smith.J	02/11/98	Soft Eng	C

Example Relation

- student(<u>Snum</u>, Name, DoB, (Subject, Grade))

<u>**SNum**</u>	**Name**	**DoB**	**Subject**	**Grade**
12345	Smith.J	02/03/96	Java	B
			Soft Eng	C
			Databases	A
23456	White.A	04/02/94	Java	D
			Soft Eng	B
	Moore.T	06/10/95	Databases	A
			Soft Eng	B
			Networks	C
	Smith.J	02/11/98	Soft Eng	C

"subject" and "grade" make up a **repeating group**.

Each record has multiple entries for these attributes.

This is an "unnormalised" table (and can't actually be stored in a relational database management system).

First Normal Form (1NF)

- First Normal Form (1NF) deals with the shape of the record type.
- **A relation is in 1NF if, and only if, it contains no repeating attributes or groups of attributes.**
- **Example**:
  - The student table with the repeating group (subject, grade) is not in 1NF.
- To remove the repeating group, one of two things can be done:
  - ”Flatten” the relation (fill in the empty attribute spaces) and extend the key, or
  - “Decompose” the relation (divide into multiple relations)

Flatten and Extend Primary Key

- The original student table (with the repeating group) can be written as:
  - student(Snum, Name, DoB, (Subject, Grade))
- If the repeating group was flattened, it would look something like:
  - student(Snum, Name, DoB, Subject, Grade)
- This does not have repeating groups, but has redundancy. For every combination of SNum/Subject, the student's name and date of birth is duplicated. This can lead to errors known as **anomalies**.
- Three main types: **insertion**, **update** and **deletion** anomalies.

Insertion Anomaly

- As "subject" is now part of the primary key, we cannot add a student until they have at least one subject

This is not possible, because "subject" cannot be NULL.

SNum	Name	DoB	Subject	Grade
53342	Ford.P	2/5/99		
12345	Smith.J	02/03/96	Java	B
12345	Smith.J	02/03/96	Soft Eng	C
12345	Smith.J	02/03/96	Databases	A
23456	White.A	04/02/94	Java	D
23456	White.A	04/02/94	Soft Eng	B
34567	Moore.T	06/10/95	Databases	A
23456	White.A	04/02/94	Soft Eng	B
23456	White.A	04/02/94	Networks	C
45678	Smith.J	02/11/98	Soft Eng	C

Update Anomaly

- Changing the name of a student means finding all rows of the database where that student exists and changing each one separately.

Oops: we forgot to change this one, the data is not consistent anymore.

SNum	**Name**	**DoB**	**Subject**	**Grade**
12345	Smythe.J	02/03/96	Java	B
12345	Smythe.J	02/03/96	Soft Eng	C
12345	Smith.J	02/03/96	Databases	A
23456	White.A	04/02/94	Java	D
23456	White.A	04/02/94	Soft Eng	B
34567	Moore.T	06/10/95	Databases	A
23456	White.A	04/02/94	Soft Eng	B
23456	White.A	04/02/94	Networks	C
45678	Smith.J	02/11/98	Soft Eng	C

Deletion Anomaly

- Deleting details about one thing can also mean we delete something else.

If we delete all Soft Eng subject information, we lost all data about student 4567 also.

<u>SNum</u>	**Name**	**DoB**	**<u>Subject</u>**	**Grade**
12345	Smith.J	02/03/96	Java	B
12345	Smith.J	02/03/96	Soft Eng	C
12345	Smith.J	02/03/96	Databases	A
23456	White.A	04/02/94	Java	D
23456	White.A	04/02/94	Soft Eng	B
34567	Moore.T	06/10/95	Databases	A
23456	White.A	04/02/94	Soft Eng	B
23456	White.A	04/02/94	Networks	C
45678	Smith.J	02/11/98	Soft Eng	C

Decomposing the Relation

- The alternative approach is to split the table into two relations: one for the repeating groups and one for the non-repeating groups.
- The primary key for the original relation is included in both of the new relations.
- We can return to the original table by using a JOIN operation on these relations: **non-loss decomposition**.

No repeating groups: 1NF

Students

SNum	**Name**	**DoB**
12345	Smith.J	02/03/86
23456	White.A	04/02/84
34567	Moore.T	06/10/85
45678	Smith.J	02/11/88

Grades

SNum	**Subject**	**Grade**
12345	Databases	B
12345	VB	C
12345	Soft. Eng	A
23456	VB	D
…..	….	….
45678	Soft. Eng	C

Second Normal Form (2NF)

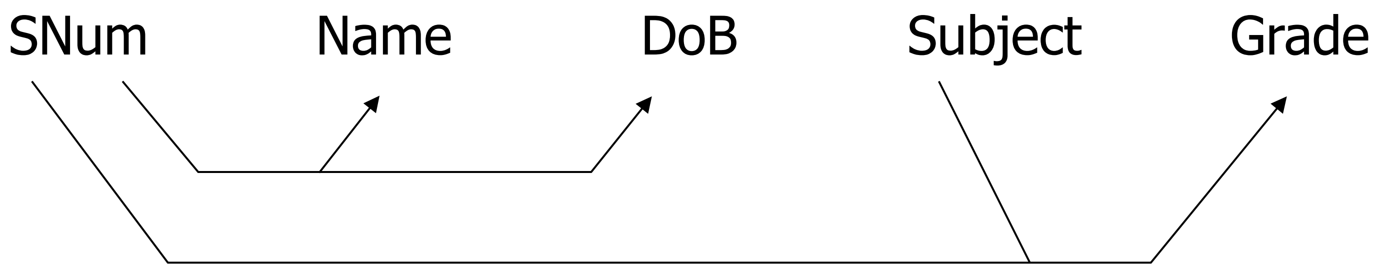
- **A relation is in 2NF if, and only if, it is in 1NF and every non-key attribute is fully functionally dependent on the whole key**.
- The relation must be in 1NF and all non-key attributes must depend on the whole key, not just part of it.
  - In other words: there must be no **partial key dependencies.**
- The problem arises when there is a **compound key**, e.g. in the Grades relation: <u>SNum</u>, <u>Subject</u>
- In this case it is possible for non-key attributes to depend on only part of the key (i.e. on only one of the key attributes).

2NF Example

- Consider the flattened student relation:
  - student(<u>SNum</u>, Name, DoB, <u>Subject</u>, Grade)
- There are no repeating groups: already in 1NF.
- However, there is a compound primary key, so we must check that the non-key attributes depend on the whole key.
  - SNum determines the Name and DoB but not Grade.
    - **SNum → Name, DoB**
  - Subject together with SNum determines Grade, but not Name or DoB
    - **SNum, Subject→ Grade**
- There is a problem with potential redundancies.

Dependency Diagram

- We can use a **dependency diagram** to show how non-key attributes relate to each part or combination of parts of the primary key.



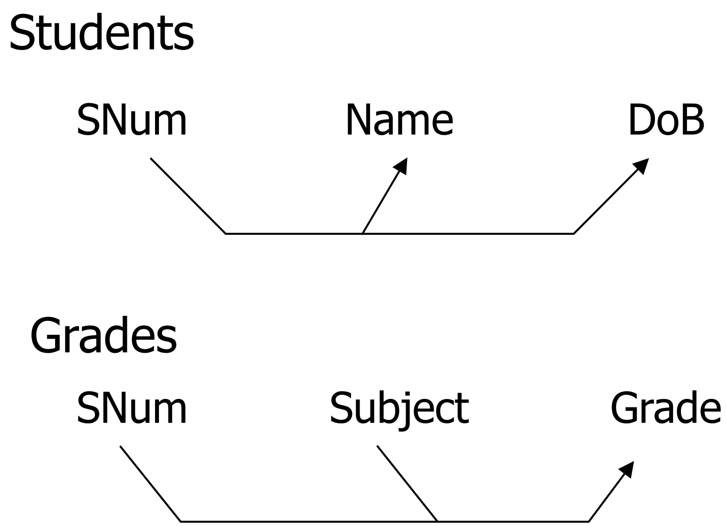
- This relation is not in 2NF. It appears to be two tables squashed into one.
- Solution: decompose the relation.

2NF

Procedure:

1. Create a new relation that contains all of the attributes that are solely dependent on SNum (SNum is the primary key)
2. Create a new relation that contains all the attributes that are solely dependent on Subject.
   - In this example, there are no attributes that depend only on Subject.
3. Create a new relation with all the attributes that depend on both SNum and Subject.
   - The primary key is SNum, Subject

2NF



- All attributes in each relation are fully functionally dependent on the primary key.
- Both relations are now in 2NF.

- Interesting: this is the same set of relations we got when we decomposed to remove the repeating group.

Third Normal Form (3NF)

- 3NF is an even stricter normal form that removes almost all redundant data.
- **A relation is in 3NF if, and only if, it is in 2NF and there are no transitive functional dependencies**.
- Transitive functional dependencies are when one non-key attribute is functionally dependent on another non-key attribute.
- By definition, a transitive functional dependency can only happen if there is more than one non-key attribute.
  - Any relation in 2NF with < 2 non-key attributes must automatically be in 3NF.

3NF: Example

Projects

Proj_No	Manager	Address
P1	Black. B	32 High St
P2	Smith. J	11 New St
P3	Black. B	32 High St
P4	Black. B	32 High St

- “Projects” has more than one non-key field (Manager and Address) so we must check for transitive dependency.

3NF: Example

- In this example, we are told that Address depends on the value of Manager.
  - If we know a project's manager, we can find the address.

- Projects(Proj_No, Manager, Address)
  - Manager → Address

- In this case, Address is **transitively dependent** on manager.

- The primary key is Proj_No, but the functional dependency makes no reference to this key.

3NF: Example

- Data redundancy can come from this:
  - We duplicate the address if a manager is in charge of more than one project.
  - Causes problems if we have to change the address, because it must be changed in several places.
- Solution: **decompose**:
  - Create two relations: one with the transitive dependency in it, and another for all of the remaining attributes.
  - Split Projects into Projects and Managers
  - In the Projects relation, we keep the same primary key: <u>Proj_no</u>
  - In the Managers relation we use the left side of the functional dependency as the primary key: <u>Manager</u>

3NF: Example Result

- Now we need to store the address only once.
- If we need to know a manager's address, we can look it up in the Managers relation.
- The "manager" attribute is the link between the two tables: in the Projects relation it is now a foreign key.
- These relations are now in 3NF

Projects

Proj_No	Manager
P1	Black.B
P2	Smith.J
P3	Black.B
P4	Black.B

Managers

Manager	Address
Black.B	32 High St
Smith.J	11 New St

Summary: 1NF

- A relation is in 1NF if it has no repeating groups.
- To convert an unnormalised relation to 1NF, either:
  - Flatten the table and extend the primary key or
  - Decompose the relation into smaller relation: one for the repeating groups and one for the non-repeating groups.
    - Remember to put the primary key from the original relation into both new relations.
- Decomposition often gives the best results:
  - R($\underline{a}$, b, ($\underline{c}$, d)) becomes:
    - R($\underline{a}$, b)
    - R1($\underline{a}$, $\underline{c}$, d)

Summary: 2NF

- A relation is in 2NF if it contains no repeating groups (1NF) and no partial key functional dependencies.
  - Rule: A relation in 1NF with a single key attribute must be in 2NF.
- To convert a relation with partial key functional dependencies to 2NF, create a new set of relations:
  - One relation for the attributes that are fully dependent on the key.
  - One relation for each part of the key that has partially dependent attributes.

- R($\underline{a}$, $\underline{b}$, c, d) and a$\rightarrow$c becomes:
  - R($\underline{a}$, $\underline{b}$, d)
  - R1($\underline{a}$, c)

Summary: 3NF

- A relation is in 3NF if it contains no repeating groups (1NF), no partial key functional dependencies (2NF), and no transitive functional dependencies.
- To convert a relation with transitive functional dependencies to 3NF, remove the attributes involved in the transitive dependency and put them in a new relation.
  - Rule: a relation in 2NF with only one non-key attribute must be in 3NF.
- In a normalised relation, a non-key field must provide data about “the key, the whole key and nothing but the key”.

Summary: 3NF

- Relations in 3NF are sufficient for most practical database design problems. However, 3NF does not guarantee that all anomalies have been removed.

- R(<u>a</u>, b, c, d) and c → d becomes
  - R(<u>a</u>, b, c)
  - R1(<u>c</u>, d)

bcnf: all attributes must dependent on the whole key including other key. exist key-key.
3nf: non-key must dependent on the key

Boyce-Codd Normal Form

- Boyce-Codd Normal Form (BCNF) is named after Raymond Boyce and Edgar Codd who developed it in 1974 to address types of anomaly not addressed in 3NF.
  - Sometimes referred to as 3.5NF
- BCNF relies on the concept of a **candidate key**.
- A candidate key is an attribute (or group of attributes) that are capable of uniquely identifying any row in a relation.
  - In other words, attributes that could be used as a primary key.

BCNF: Candidate Keys

- Frequently, a relation has only one candidate key, which is used as the primary key.
- However, there are sometimes multiple candidate keys. When this happens, we choose one to be the primary key.
- For example, a relation named “Students” that contains the following attributes:
  - `UCD_ID`
  - `BJUT_ID`
- Both are candidate keys, as they uniquely identify a student, though we choose only one to be the primary key.

BCNF

- When a relation has more than one candidate key, anomalies can result even though the relation is in 3NF.
- 3NF does not deal with **overlapping candidate keys**.
  - i.e. composite candidate keys with at least one attribute in common.
- BCNF is based on the concept of a **determinant**.
  - A determinant is any attribute (or group of attributes) that some other attribute is fully functionally dependent on (i.e. the left side of a functional dependency).
- A relation is in BCNF if, and only if, every determinant is a candidate key.

BCNF: Hospital Example

PatNo	PatName	AppSlot	Time	Doctor
1	Eamonn	0	09:00	Octopus
2	Eoin	0	09:00	Evil
3	Arnold	1	10:00	Octopus
4	Stephen	0	13:00	Evil
5	Patricia	1	14:00	Octopus

- **Extra information:**
  - Every patient has a unique patient number.
  - Patients with names beginning with a letter before ‘P’ get morning appointments.
  - The Appointment slots start at 0 for the first appointment of the morning or afternoon, 1 for the second and so on.

BCNF: Hospital Example

- Appointments(PatNo, PatName, AppSlot, Time, Doctor)
- We are given some functional dependencies (mostly based on the extra information):
  - PatNo → PatName
  - PatNo, AppSlot → Time, Doctor
  - Time → AppSlot
- We have two options for selecting a primary key:
  - Appointments(PatNo, PatName, AppSlot, Time, Doctor): example A
  - Appointments(PatNo, PatName, AppSlot, Time, Doctor): example B

BCNF: Example A

- Appointments(<u>PatNo</u>, PatName, <u>AppSlot</u>, Time, Doctor)
- No repeating groups, so in 1NF.
- 2NF – eliminate partial key dependencies:
  - Appointments(<u>PatNo</u>, <u>AppSlot</u>, time, Doctor)
  - Patients(<u>PatNo</u>, PatName)
- 3NF – no transitive dependencies so it's already in 3NF.
- Now try BCNF.

BCNF: Every determinant must be a candidate key.

- Appointments(<u>PatNo</u>, <u>AppSlot</u>, time, Doctor)
- Patients(<u>PatNo</u>, PatName)

- For Appointments: is determinant a candidate key for each of the functional dependencies?

- PatNo → PatName
  - PatNo is present in Appointments, but not PatName, so this is not relevant.

BCNF: Every determinant must be a candidate key.

- Appointments(PatNo, AppSlot, time, Doctor)
- Patients(PatNo, PatName)

- For Appointments: is determinant a candidate key for each of the functional dependencies?

- PatNo, AppSlot → Time, Doctor
  - All of these attributes are present in Appointments, so this functional dependency is relevant. Is this a candidate key?
  - PatNo, AppSlot **is** the key, so this is a candidate key: OK.

BCNF: Every determinant must be a candidate key.

- Appointments(<u>PatNo</u>, <u>AppSlot</u>, time, Doctor)
- Patients(<u>PatNo</u>, PatName)

- For Appointments: is determinant a candidate key for each of the functional dependencies?

- Time → AppSlot
  - Time is present in Appointments and so is AppSlot, so it's relevant.
  - If this is a candidate key, then we could rewrite Appointments as:
    - Appointments(PatNo, Appslot, <u>Time</u>, Doctor).
  - This won't work, so it is not in BCNF. "Time" is not a candidate key.

Rewrite to BCNF

- Appointments(PatNo, AppSlot, time, Doctor)
- Patients(PatNo, PatName)

- Rewrite to BCNF:
  - Appointments(PatNo, Time, Doctor)
  - Patients(PatNo, PatName)
  - Slots(Time, AppSlot)

- "Time" is enough to work out the appointment slot of a patient.
- Now BCNF is satisfied, and the final relations shown are in BCNF.

BCNF: Example B

- Appointments(PatNo, PatName, AppSlot, Time, Doctor)
- No repeating groups, so it's in 1NF
- 2NF – eliminate partial key dependencies:
  - Appointments(PatNo, Time, Doctor)
  - Patient(PatNo, PatName)
  - Slots(Time, AppSlot)
- 3NF – no transient dependencies, so it's in 3NF
- Now try BCNF.

BCNF Check

- Appointments(<u>PatNo</u>, <u>Time</u>, Doctor)
- Patient(<u>PatNo</u>, PatName)
- Slots(<u>Time</u>, AppSlot)

- For Appointments: is determinant a candidate key for each of the functional dependencies?

- PatNo $\rightarrow$ PatName
  - PatNo is present in Appointments, but no PatName, so it's not relevant.

BCNF Check

- Appointments(<u>PatNo</u>, <u>Time</u>, Doctor)
- Patient(<u>PatNo</u>, PatName)
- Slots(<u>Time</u>, AppSlot)

- For Appointments: is determinant a candidate key for each of the functional dependencies?

- PatNo, AppSlot → Time, Doctor
  - PatNo and AppSlot are not both present in Appointmetns, so this is not relevant.

BCNF Check

- Appointments(PatNo, Time, Doctor)
- Patient(PatNo, PatName)
- Slots(Time, AppSlot)

- For Appointments: is determinant a candidate key for each of the functional dependencies?

- Time →AppSlot
  - Time is present in Appointments, but not AppSlot, so this is not relevant.
- Relations are in BCNF

Example Summary

This example demonstrates three things:

1. BCNF is stronger than 3NF: relations in 3NF are not necessarily in BCNF.
2. BCNF is needed in certain situations to get a full understanding of the data model.
3. There are several routes to take to arrive at the same set of relations in BCNF.
   - Unfortunately there are no rules to guarantee the easiest route.

What problem does BCNF overcome?

Student_No	Major	Supervisor
123	Physics	Einstein
123	Music	Mozart
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

- We have the following functional dependencies:
  - Student_No, Major → Supervisor
  - Supervisor → Major

What problem does BCNF overcome?

Student No	**Major**	**Supervisor**
123	Physics	Einstein
123	Music	Mozart
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

- No repeating groups, so it's in 1NF
- No partial key dependencies, so it's in 2NF
- There's only one non-key attribute (Supervisor) so it must be in 3NF.

What problem does BCNF overcome?

<u>**Student_No**</u>	<u>**Major**</u>	**Supervisor**
123	Physics	Einstein
123	Music	Mozart
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

- If the record for student 456 is deleted we lose not only information about that student, but also the fact that Darwin advises in Biology.
- We cannot record the fact that Watson can advise on Computing until we have a student doing a project on Computing that has Watson as an advisor.

What problem does BCNF overcome?

- In BCNF we have two tables, and these problems are eliminated:

Student_No	**Supervisor**
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

Supervisor	**Major**
Einstein	Physics
Mozart	Music
Darwin	Biology
Bohr	Physics

Summary

- A relation is in 1NF if, and only if, it contains no repeating groups.
- A relation is in 2NF if, and only if, it is in 1NF and every non-key attribute is fully functionally dependent on the whole key.
- A relation is in 3NF if, and only if, it is in 2NF and has no transitive functional dependencies.
- A relation is in BCNF if, and only if, it is in 3NF and every determinant is a candidate key.