Principles of Databases

Normalisation

David Sinclair

What is a Good Database Design?

- Deciding on a suitable robust logical structure for the tables is a key element of a good relational database design.
- A good design has the following features:
  - *Efficient*: Adding a number of entities should not require the additional of a large number of redundant information or assuming unassigned data values. For example, consider the following table:

Student

sID	sName	address	SSName	SSID	SSCity	score	grade
123	John	1 May St.	Belvedere	123	Dublin	55	3
234	Amy	5 Town St.	Loretto	234	Cork	75	1
345	Paul	3 May St.	Belvedere	123	Dublin	62	2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Adding a large number of new students would require a values (possibly NULL) for the other fields.

What is a Good Database Design? (2)

- Avoids possible anomalies.
  - *Update Anomalies*: Changing the information of one tuple may result in the same information in another tuple remaining unchanged. If we changed the Secondary School ID from Belvedere from 123 to 555 for student ID 123, we might forget to do it for student 345.
  - *Insertion Anomalies*: Inserting a tuple requires all the fields to be completed which may result in NULL values being entered (e.g. Score and Grade) or information being unnecessarily repeated (e.g. Secondary School Name, ID and City).
  - *Deletion Anomalies*: Deleting all the tuple with a specific values in a field may result in other information being lost. For example, if student 234 was the only student in the database that went to the Loretto Secondary School deleting student 234 would also remove any information about this school from the database.

Normalisation

- Normalisation theory is based on some simple ideas and helps guide us in developing a suitable set of tables for a given problem.
- Normalisation theory allows us to recognise relations with possibly undesirable properties and design better relations.
- Normalisation theory is built on a hierarchy of normal forms.
  - Codd defined 1NF, 2NF and 3NF in 1972.
  - 3NF has some inadequacies, so Codd revised it in 1974 to BCNF (Boyce-Codd Normal Form) also called 3.5NF.
  - 4NF was introduce by Fagin in 1977 to address multivalued dependencies.
  - There are higher normal forms. 5NF is used but higher forms are rarely used.
- Normalisation is generally a great idea, but we should always ask if the resulting tables enable the envisaged queries to be efficiently executed.

Functional Dependency

- Consider two tuples of R. There is a *functional dependency* if when they have same values in their attributes $A_1, A_2, \ldots, A_n$, they must also have the same values in their attributes $B_1, B_2, \ldots, B_m$.
- $A_1, A_2, \ldots, A_n$ *functionally determines* $B_1, B_2, \ldots, B_m$
- Formally, $\forall t, u \in R$:
  $t[A_1, A_2, \ldots, A_n] = u[A_1, A_2, \ldots, A_n]$
  $\Rightarrow t[B_1, B_2, \ldots, B_m] = u[B_1, B_2, \ldots, B_m]$
- This is written
  $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$
  or $\overline{A} \rightarrow \overline{B}$

R

$\overline{A}$	$\overline{B}$	$\overline{C}$
$\overline{a}$	$\overline{b}$	$\overline{c_1}$
$\overline{a}$	$\overline{b}$	$\overline{c_2}$
⋮	⋮	⋮

Functional Dependency (2)

- A functional dependency is based on the real world we are representing.
- All instances of the relation must adhere to this functional dependency.

Consider the Student relation

Student

sID	sName	address	SSName	SSID	SSCity	score	grade

$sID \rightarrow sName$
$sID \rightarrow address$
$SSID \rightarrow SSName, SSCity$
$SSName, SSCity \rightarrow SSID$ (?)
$sID \rightarrow score$
$score \rightarrow grade$
$sID \rightarrow grade$

Functional Dependencies (3)

- If $\overline{A} = \{A_1, A_2, \ldots, A_n\}$ is a *key* then:
  1. $\overline{A}$ determines all other attributes in the relation.
  2. No proper subset of $\overline{A}$ determines all other attributes in the relation, i.e. the key is *minimal*
- *Trivial Functional Dependency*: $\overline{A} \rightarrow \overline{B}$ & $\overline{B} \subseteq \overline{A}$
- *Non-Trivial Functional Dependency*: $\overline{A} \rightarrow \overline{B}$ & $\overline{B} \not\subseteq \overline{A}$
- *Completely Non-Trivial Functional Dependency*: $\overline{A} \rightarrow \overline{B}$ & $\overline{B} \cap \overline{A} = \emptyset$
- *Splitting Rule*: $\overline{A} \rightarrow B_1, B_2, \ldots, B_m$ $\Rightarrow \overline{A} \rightarrow B_1, \overline{A} \rightarrow B_2, \ldots, \overline{A} \rightarrow B_m$
- *Combining Rule*: $\overline{A} \rightarrow B_1, \overline{A} \rightarrow B_2, \ldots, \overline{A} \rightarrow B_m$ $\Rightarrow \overline{A} \rightarrow B_1, B_2, \ldots, B_m$
- *Transitivity Rule*: $\overline{A} \rightarrow \overline{B}$ & $\overline{B} \rightarrow \overline{C} \Rightarrow \overline{A} \rightarrow \overline{C}$

Closure of Attributes

- Given a set of functional dependencies S and a set of attributes $\overline{A}$ the *closure of* $\overline{A}$ finds all the attributes $\overline{B}$ such that $\overline{A} \rightarrow \overline{B}$.
- The *closure of* $\overline{A}$ is denoted $\overline{A}^+$.
- Starting with the closure set $= \{A_1, A_2, \ldots, A_n\}$
- Repeat until no change
  - If $\overline{A} \rightarrow \overline{B}$ and $\overline{A}$ is in the closure set, then add $\overline{B}$ to the closure set.
- Example:
  `Student(sID,sName,address,SSName,SSID,SSCity,score,grade)`
  `sID` $\rightarrow$ `sName,address,score` `score` $\rightarrow$ `grade`
  `SSID` $\rightarrow$ `SSName,SSCity`
  $\{$`sID,SSID`$\}^+$ =
  $\{$`sID,SSID,sName,address,score,grade,SSName,SSCity`$\}$

Closure of Attributes (2)

- If $\overline{A}^+$ = all attributes, then $\overline{A}$ is a key.
- How do we find all the keys from a given set of functional dependencies?
  - Consider every subset $\overline{A}$ (of increasing size) of the attributes.
  - If $\overline{A}^+$ = all attributes, then $\overline{A}$ is a key.

- Example: Find the keys of $\{\texttt{title,year,studioName,president,presAddress}\}$ given
  $\texttt{title,year} \rightarrow \texttt{studioName}$
  $\texttt{studioName} \rightarrow \texttt{president}$
  $\texttt{president} \rightarrow \texttt{presAddress}$

  $\{\texttt{title,year}\}^+ =$
  $\{\texttt{title,year,studioName,president,presAddress}\}$
  Since $\{\texttt{title,year}\}^+$ = all attributes then $\texttt{title,year}$ is a key. Only record *minimal* keys.

Follows From

- S_1 and S_2 are sets of functional dependencies.
- S_2 follows from S_1 if every relation instance satisfying S_1 also satisfies S_2.
- Example:
  Given the attribute $\{A, B, C, D, E, F\}$ and the functional dependencies $AB \rightarrow C$, $BC \rightarrow AD$, $D \rightarrow E$ and $CF \rightarrow B$, does $AB \rightarrow D$ and $D \rightarrow A$ follow?

  $D \in \{A, B\}^+ = \{A, B, C, D, E\}$ hence $AB \rightarrow D$ follows from the given dependencies.
  $A \notin \{D\}^+ = \{D, E\}$ hence $D \rightarrow A$ does not follows from the given dependencies.

Projecting a Set of Functional Dependencies

Given a relation R and a set of functional dependencies S that hold in R, what is the minimal set of functional dependencies (FD's) that hold in a second relation R_i computed by the projection $R_i = \pi_L(R)$?

1. Let T be the set of FD's of R_i. Initially, T is empty.
2. For each set of attributes $\overline{X}$, a subset of the attributes of R_i, compute $\overline{X}^+$ with respect to the set S which may involve attributes of R but not R_i. Add to T all nontrivial FD's $\overline{X} \rightarrow \overline{A}$ such that $\overline{A}$ is both in $\overline{X}^+$ and an attribute of R_i.
3. T may not be a minimal set of FD's of R_i. We can construct a minimal set by modifying T as follows:
   - 3.1 If $F \in T$ follows from $F' \in T$, remove F from T.
   - 3.2 Let $\overline{Y} \rightarrow \overline{B} \in T$, with at least two attributes in $\overline{Y}$, and let $\overline{Z}$ be $\overline{Y}$ with one of its attributes removed. If $\overline{Z} \rightarrow \overline{B}$ follows from the FD's in T (including $\overline{Y} \rightarrow \overline{B}$), then replace $\overline{Y} \rightarrow \overline{B}$ by $\overline{Z} \rightarrow \overline{B}$.
   - 3.3 Repeat the above steps in all possible ways until no more changes to T can be made.

Projecting a Set of Functional Dependencies (2)

- Example:
  Let $R(A, B, C, D)$ have FD's $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow D$.
  What is the minimal set of FD's of $R_1(A, C, D)$?
- Start with the closures of the singleton sets, and then move on to the doubleton sets if necessary.
- $\{A\}^+ = \{A, B, C, D\}$. Hence $A \rightarrow C$ and $A \rightarrow D$ hold in R_1.
  - Where did $A \rightarrow C$ and $A \rightarrow D$ come from?
  - $A \rightarrow B$ is true in R, but since B is not an attribute of R_1 $A \rightarrow B$ makes no sense in R_1.
- Since $\{A\}^+$ includes all attributes of R_1, we need not consider any superset of $\{A\}$.
- The only doubleton we need to consider is $\{C, D\}^+ = \{C, D\}$. Hence $C \rightarrow D$ holds in R_1.
- Since $A \rightarrow D$ follows from $A \rightarrow C$ and $C \rightarrow D$, the minimal set of FD's of R_1 is $A \rightarrow C$ and $C \rightarrow D$

1NF

- A set of relations (tables) is in **first normal form** (1NF) if:
  1. Tuples cannot have repeat groups of similar data (*atomicity*).
  2. Each tuple (row of a table) must have a unique identifier (*primary key*).
- The *primary key* can be composed of multiple attributes (columns). This is called a *concatenated primary key*.

2NF

- A set of relations is in **second normal form** (2NF) if there is no *partial dependency* on a *concatenated* key in any relation.
  - *concatenated key*: A key composed of more than one attribute.
  - *partial dependency*: A functional dependency between an subset of a concatenated key and another set of attributes.
- Each attribute in a relation with a concatenated primary key must depend on the entire concatenated key for its existence.

3NF & BCNF

- A set of relations is in **third normal form** (3NF) if there is no dependencies on non-key attributes.
- A set of relations is in **Boyce-Codd normal form** (BCNF) if for each non-trivial functional dependency $\overline{A} \rightarrow \overline{B}$ in relation R, $\overline{A}$ is a *superkey* of R.
  - $\overline{A}$ is a *superkey* of R if $\overline{A}$ contains a key.
  - BCNF is a slightly stronger version then 3NF that handles relations with two or more overlapping candidate keys.

Design by Decomposition

- One approach to a "good" relational database design is to move through a series of designs that satisfy 1NF, 2NF, 3NF and then onto BCNF and possibly 4NF.
- We will adopt a different approach where we will start with a "mega-relation" that includes all the attributes and then decompose the "mega-relation" into a set of smaller relations that are in BCNF or 4NF.
  - The decomposition process will be driven by the functional dependencies.
  - If we join the decomposed relations (tables) we should get the original "mega-relation", i.e. a *lossless join*.
  - Resulting relations (tables) will have no anomalies (but still may have some shortcomings).

Lossless Join

- Consider our student relation example
  `Student(sID,sName,address,SSName,SSID,SSCity,score,grade)`
- One possible decomposition could be:
  `S1(sID,sName,address,score,grade)`
  `S2(SSName,SSID,SSCity)`
  - attributes of S1 $\cup$ attributes of S2 $=$ attributes of Student✓
  - $S1 \bowtie S2 = Student$✓
- Another possible decomposition is:
  `S3(sID,sName,address,SSName,SSID)`
  `S4(sName,SSName,SSCity,score,grade)`
  - attributes of S3 $\cup$ attributes of S4 $=$ attributes of Student✓
  - $S3 \bowtie S4 \stackrel{?}{=} Student$
    - Most like not (why?) and information will be lost in the join.

BCNF Decomposition Algorithm

- Given a set of relations and a set of functional dependencies:
  - Repeat until all relations are in BCNF
    - Select any relation $R'(\overline{A}, \overline{B}, \overline{C})$ that violates the BCNF condition ($\forall \overline{A} \rightarrow \overline{B}$, $\overline{A}$ is a superkey).
    - Decompose R' into $R_1(\overline{A}, \overline{B})$ and $R_2(\overline{A}, \overline{C})$.
    - Compute functional dependencies for R_1 and R_2.
    - Compute keys for R_1 and R_2.
- The resulting relations can be joined (without loss) to recover the original "mega-relation".
- Decomposition process removes anomalies.

BCNF Decomposition Example

`Student(sID,sName,address,SSName,SSID,SSCity,score,grade)`
`sID` $\rightarrow$ `sName,address,score` `score` $\rightarrow$ `grade`
`SSID` $\rightarrow$ `SSName,SSCity`

The key is `sID,SSID`.

Not in BCNF, so let us choose to decompose on `SSID` $\rightarrow$ `SSName,SSCity`.

`S1(`<u>`SSID`</u>`,SSName,SSCity)` ✓
`S2(`<u>`sID`</u>`,sName,address,`<u>`SSID`</u>`,score,grade)`

Let us choose to decompose on `score` $\rightarrow$ `grade`.

`S3(`<u>`score`</u>`,grade)` ✓
`S4(`<u>`sID`</u>`,sName,address,`<u>`SSID`</u>`,score)`

Decompose on `sID` $\rightarrow$ `sName,address,score`.

`S5(`<u>`sID`</u>`,sName,address,score)` ✓
`S6(`<u>`sID`</u>`,`<u>`SSID`</u>`)` ✓

Improving the BCNF Decomposition Algorithm

- Given a set of relations and a set of functional dependencies:
  - Repeat until all relations are in BCNF
    - Select any relation $R'(\overline{A}, \overline{B}, \overline{C})$ that violates the BCNF condition ($\forall \overline{A} \rightarrow \overline{B}$, $\overline{A}$ is a superkey).
    - Extend $\overline{A} \rightarrow \overline{B}$ to $\overline{A} \rightarrow \overline{B}\,\overline{A}^{+}$.
    - Decompose R' into $R_1(\overline{A}, \overline{B}\,\overline{A}^{+})$ and $R_2(\overline{A}, \overline{C} \backslash \overline{A}^{+})$.
    - Compute functional dependencies for R_1 and R_2.
    - Compute keys for R_1 and R_2.
- Can produce fewer relations.

Multivalued Dependencies

- When two sets of attributes are independent of each other, then we have a *multivalued dependency*.
- Consider the following relation about film stars, their houses and their films.

name	address	city	film	year
Tom Cruise	12 Star Ave.	Hollywood	Mission: Impossible	1996
Tom Cruise	12 Star Ave.	Hollywood	Top Gun	1986
Tom Cruise	5 Oscar St.	New York	Mission: Impossible	1996
Tom Cruise	12 Star Ave.	Hollywood	Knight and Day	2010
Tom Cruise	5 Oscar St.	New York	Top Gun	1986
Tom Cruise	5 Oscar St.	New York	Knight and Day	2010

- There is no reason why where a film star lives and the films they make should be dependent on each other.
- There is no BCNF violations in this relation (since there is no non-trivial functional dependencies).
- But there appears to be a lot of duplication. With just 2 houses and 3 films we have 6 tuples.

Multivalued Dependencies (2)

- Formally a *multivalued dependency*, $\overline{A} \twoheadrightarrow \overline{B}$, in the relation $R(\overline{C})$ is defined:

$$\begin{array}{rlcll} \forall t, u \in R: & t[\overline{A}] & = & u[\overline{A}] & \text{then} \\ \exists v \in R: & v[\overline{A}] & = & t[\overline{A}] & \text{and} \\ & v[\overline{B}] & = & t[\overline{B}] & \text{and} \\ & v[\overline{C} \backslash \overline{A} \backslash \overline{B}] & = & u[\overline{C} \backslash \overline{A} \backslash \overline{B}] & \text{and} \end{array}$$

- Also called *tuple-generating dependencies*.

R

	$\overline{C}$		
	$\overline{A}$	$\overline{B}$	$\overline{C} \backslash \overline{A} \backslash \overline{B}$
t	$\overline{a}$	$\overline{b_1}$	$\overline{r_1}$
u	$\overline{a}$	$\overline{b_2}$	$\overline{r_2}$
v	$\overline{a}$	$\overline{b_1}$	$\overline{r_2}$
w	$\overline{a}$	$\overline{b_2}$	$\overline{r_1}$
	$\vdots$	$\vdots$	$\vdots$

swapping t and u

Multivalued Dependencies (3)

- All instances of a relation must adhere to a multivalued dependency.
- A multivalued dependency is based on knowledge of the world you are modelling.
- Example 1: Application to universities that include your hobbies.
  $sID \twoheadrightarrow uName$, $sID \twoheadrightarrow hobby$

sID	uName	hobby
123	DCU	football
123	TCD	golf
123	DCU	golf
123	TCD	football
⋮	⋮	⋮

Database Design Functional Dependency Normal Forms Decomposition

Multivalued Dependencies (3)

- Example 2: Applications to universities where you selectively reveal your hobbies to different universities.
  No multivalued dependencies

sID	uName	hobby
123	DCU	football
123	TCD	golf
⋮	⋮	⋮

- Example 3: Applications to universities where:
  - You selectively reveal your hobbies.
  - Apply to each university once on a specific day.
  - You can apply for multiple degrees.

`Apply(sID, uName, date, degree, hobby)`
$sID, uName \rightarrow date$
$sID, uName, date \twoheadrightarrow degree$

Rules for Multivalued Dependencies

- A multivalued dependency (MVD), $\overline{A} \twoheadrightarrow \overline{B}$, is a *trivial multivalued dependency* if $\overline{B} \subseteq \overline{A}$ or $\overline{A} \cup \overline{B} =$ all attributes. Otherwise it is a non-trivial MVD.
- A functional dependency is a multivalued dependency. If $\overline{A} \rightarrow \overline{B}$ then $\overline{A} \twoheadrightarrow \overline{B}$.

if an MVD

	$\overline{A}$	$\overline{B}$	the rest
t	$\overline{a}$	$\overline{b_1}$	$\overline{r_1}$
u	$\overline{a}$	$\overline{b_2}$	$\overline{r_2}$
v	$\overline{a}$	$\overline{b_1}$	$\overline{r_2}$
	⋮	⋮	⋮

$\overline{A} \rightarrow \overline{B}$

	$\overline{A}$	$\overline{B}$	the rest	
t	$\overline{a}$	$\overline{b_1}$	$\overline{r_1}$	
u	$\overline{a}$	$\overline{b_1}$	$\overline{r_2}$	
v	$\overline{a}$	$\overline{b_1}$	$\overline{r_2}$	$v = u$
	⋮	⋮	⋮	

- Intersection Rule: If $\overline{A} \twoheadrightarrow \overline{B}$ and $\overline{A} \twoheadrightarrow \overline{C}$ then $\overline{A} \twoheadrightarrow \overline{B} \cap \overline{C}$.
- Transitivity Rule: If $\overline{A} \twoheadrightarrow \overline{B}$ and $\overline{B} \twoheadrightarrow \overline{C}$ then $\overline{A} \twoheadrightarrow \overline{C} - \overline{B}$.

Fourth Normal Form Decomposition Algorithm

- A set of relations is in 4NF if for each non-trivial $\overline{A} \twoheadrightarrow \overline{B}$, $\overline{A}$ is a superkey.
- Given a set of relations and a set of functional dependencies FDs and multivalued dependencies MVDs:
  - Repeat until all relations are in 4NF
    - Select any relation $R'(\overline{A}, \overline{B}, \overline{C})$ that violates the 4NF condition.
    - Decompose R' into $R_1(\overline{A}, \overline{B})$ and $R_2(\overline{A}, \overline{C})$.
    - Compute the FDs and MVDs for R_1 and R_2.
    - Compute keys for R_1 and R_2.

4NF Decomposition Example 1

Apply(sID, uName, hobby)
sID $\twoheadrightarrow$ uName

The only key is sID, uName, hobby and hence a 4NF violation.
Decompose using sID $\twoheadrightarrow$ uName.

A1(<u>sID</u>,<u>uName</u>)
A2(<u>sID</u>,<u>hobby</u>)

There are no FDs and MVDs so we are now in 4NF.

4NF Decomposition Example 2

Apply(sID, uName, date, degree, hobby)
sID,uName $\rightarrow$ date
sID,uName,date $\twoheadrightarrow$ degree

Again the key is the complete set of attributes and there is a 4NF violation.
Decompose using sID,uName,date $\twoheadrightarrow$ degree.

A1(<u>sID</u>, <u>uName</u>, <u>date</u>, <u>degree</u>)
A2(<u>sID</u>, <u>uName</u>, <u>date</u>, <u>hobby</u>)

Both A1 and A2 are 4NF violations. Decompose A1 using sID,uName $\rightarrow$ date.

A3(<u>sID</u>, <u>uName</u>, date) ✓
A4(<u>sID</u>, <u>uName</u>, <u>degree</u>) ✓

Decompose A2 using sID,uName $\rightarrow$ date.

A5(<u>sID</u>, <u>uName</u>, <u>hobby</u>) ✓

Disadvantages of BCNF and 4NF

- In general, decomposing a set of relations into BCNF or 4NF yields a good design with no redundant information and no anomalies.
- However, there can be disadvantages resulting from the relations being "too decomposed".
  - Important combinations of information may be split between relations and the relations need to be joined before recovering this information.
  - Dependency information may not be stored in some relation and would require a join to recover it.
  - Common queries on the data may require information that has been decomposed across several relations.

Over-Decomposing Example 1

Consider `Apply(sID, uName, date, degree)`
where a student can apply for one degree in each university and universities have non-overlapping application dates.

From the above description we can infer
`sID,uName`→`date,degree` and `date`→`uName`

Keys: `sID,uName`

This is not in BCNF and decomposing on `date`→`uName` yields:

`A1(date,uName)`
`A2(sID,date,degree)`

This design has separated the student ID from the name of the university they were applying to. Might be better to keep the original relation which was in 3NF.

Over-Decomposing Example 2

Consider `Results(sID,sName,exam1,exam2,exam3)`
where exams can be taken multiple times.

This gives dependencies `sID`$\rightarrow$`sName`, `sID,sName` $\twoheadrightarrow$ `exam1`, `sID,sName` $\twoheadrightarrow$ `exam2`

Decomposing into 4NF yields:
`R3(sID,sName)`
`R4(sID,exam1)`
`R5(sID,exam2)`
`R6(sID,exam3)`

If most queries need to return the student's name and a value calculated from the best scores in `exam1`, `exam2` and `exam3` then there will be a repetition of the same joins in each query.