Normalization

Achieving BCNF Schemas

Example 1

R = (A, B, C)
F = {A → B, B → C}
Candidate keys = {A}

B → C

R1 = (B, C)
F1 = {B → C}
Candidate keys = {B}
BCNF = true

R2 = (A, B)
F2 = {A → B}
Candidate keys = {A}
BCNF = true

Dependency preservation ???
Yes

Lossless ???
Yes by construction
Example 2a

R = (A, B, C, D, E)
F = \{A \rightarrow B, BC \rightarrow D\}
Candidate keys = \{ACE\}
BCNF = Violated by \{A \rightarrow B, BC \rightarrow D\}

A \rightarrow B

R1 = (A, B)
F1 = \{A \rightarrow B\}
Candidate keys = \{A\}
BCNF = true

R2 = (A, C, D, E)
F2 = \{AC \rightarrow D\}
Candidate keys = \{ACE\}
BCNF = false (AC \rightarrow D)

From A \rightarrow B and BC \rightarrow D by pseudo-transitivity

Example 2b

R = (A, B, C, D, E)
F = \{A \rightarrow B, BC \rightarrow D\}
Candidate keys = \{ACE\}
BCNF = Violated by \{A \rightarrow B, BC \rightarrow D\} etc...

A \rightarrow B

R1 = (A, B)
F1 = \{A \rightarrow B\}
Candidate keys = \{A\}
BCNF = true

R2 = (A, C, D, E)
F2 = \{AC \rightarrow D\}
Candidate keys = \{ACE\}
BCNF = true

From A \rightarrow B and BC \rightarrow D by pseudo-transitivity

AC \rightarrow D

R3 = (A, C, D)
F3 = \{AC \rightarrow D\}
Candidate keys = \{AC\}
BCNF = true

R4 = (A, C, E)
F4 = \{\} [[ only trivial ]]\nCandidate keys = \{ACE\}
BCNF = true

Dependency preservation ???
We can check:
A \rightarrow B (R1), AC \rightarrow D (R3),
but we lost BC \rightarrow D
So this is not a dependency-preserving decomposition.

You will not be required to do anything like this.
Example 2c

\[ R = (A, B, C, D, E) \]
\[ F = \{ A \rightarrow B, BC \rightarrow D \} \]
Candidate keys = \{ACE\}
BCNF = Violated by \{A \rightarrow B, BC \rightarrow D\}

\[ BC \rightarrow D \]

\[ R1 = (BCD) \]
\[ F1 = \{ BC \rightarrow D \} \]
Candidate keys = \{BC\}
BCNF = true

\[ A \rightarrow B \]

\[ R2 = (A, B, C, E) \]
\[ F2 = \{ A \rightarrow B \} \]
Candidate keys = \{ACE\}
BCNF = false (A \rightarrow B)

R1 = (BCD)
F1 = \{ BC \rightarrow D \}
Candidate keys = \{BC\}
BCNF = true

\[ BC \rightarrow D \]

\[ R3 = (A, B) \]
\[ F3 = \{ A \rightarrow B \} \]
Candidate keys = \{A\}
BCNF = true

\[ E \rightarrow HA \]

\[ R4 = (A, C, E) \]
\[ F4 = \{ A \rightarrow B \} \]
Candidate keys = \{ACE\}
BCNF = true

Dependency preservation ???

Example 3

\[ R = (A, B, C, D, E, H) \]
\[ F = \{ A \rightarrow BC, E \rightarrow HA \} \]
Candidate keys = \{DE\}
BCNF = Violated by \{A \rightarrow BC\} and \{E \rightarrow HA\}

\[ A \rightarrow BC \]

\[ R1 = (A, B, C) \]
\[ F1 = \{ A \rightarrow BC \} \]
Candidate keys = \{A\}
BCNF = true

\[ E \rightarrow HA \]

\[ R3 = (E, H, A) \]
\[ F3 = \{ E \rightarrow HA \} \]
Candidate keys = \{E\}
BCNF = true

\[ R2 = (A, D, E, H) \]
\[ F2 = \{ E \rightarrow HA \} \]
Candidate keys = \{DE\}
BCNF = false (E \rightarrow HA)

\[ R4 = (ED) \]
\[ F4 = \{ E \rightarrow HA \} \]
Candidate keys = \{DE\}
BCNF = true

Dependency preservation ???

We can check:
A \rightarrow BC (R1), E \rightarrow HA (R3),
Dependency-preserving decomposition
Outline

- Mechanisms and definitions to work with FDs:
  - Closures, candidate keys, canonical covers etc...
  - Armstrong axioms
- Decompositions:
  - Loss-less decompositions, Dependency-preserving decompositions
- BCNF:
  - How to achieve a BCNF schema
  - BCNF may not preserve dependencies
- 3NF: Solves the above problem
- BCNF allows for redundancy
- 4NF: Solves the above problem

BCNF may not preserve dependencies

- $R = \{J, K, L\}$
- $F = \{JK \rightarrow L, L \rightarrow K\}$

- Two candidate keys = $JK$ and $JL$
  - $R$ is not in BCNF
- Any decomposition of $R$ will fail to preserve $JK \rightarrow L$
- This implies that testing for $JK \rightarrow L$ requires a join
BCNF may not preserve dependencies

- Not always possible to find a dependency-preserving decomposition that is in BCNF.
- PTIME to determine if there exists a dependency-preserving decomposition in BCNF
  - in size of F
- NP-Hard to find one if it exists
- Better results exist if $F$ satisfies certain properties

Prime attributes

- Definition: Prime attributes
  - An attribute that is contained in a candidate key for $R$
- Example 1:
  - $R = \{A, B, C, D, E, H\}$, $F = \{A \to BC, E \to HA\}$,
  - Candidate keys = \{ED\}
  - Prime attributes: $D, E$
- Example 2:
  - $R = \{J, K, L\}$, $F = \{JK \to L, L \to K\}$,
  - Candidate keys = \{JL, JK\}
  - Prime attributes: $J, K, L$

- Observation/Intuition:
  1. A key has no redundancy (is not repeated in a relation)
  2. A prime attribute has limited redundancy
**3NF to the rescue**

*R is in 3NF (3rd Normal Form) if:*

- Given a relation schema *R* and a set of functional dependencies *F*:
  - every FD, \( \alpha \rightarrow \beta \), is either:
    - Trivial, or
    - \( \alpha \) is a superkey of *R*, or
    - All attributes in \((\beta - \alpha)\) are prime

**Why is 3NF good?**

- Lossless
- Preserves dependencies.
- Limited redundancy

---

**3NF and Redundancy**

- **Why does redundancy arise?**
  - Given a FD, \( \alpha \rightarrow \beta \), if \( \alpha \) is repeated \((\beta - \alpha)\) has to be repeated
    - If rule 1 is satisfied, \((\beta - \alpha)\) is empty, so not a problem.
    - If rule 2 is satisfied, \( \alpha \) can’t be repeated, so this doesn’t happen either
      - If not, rule 3 says \((\beta - \alpha)\) must contain only prime attributes
    - This limits the redundancy somewhat.

- 3NF relaxes BCNF by allowing some (hopefully limited) redundancy

- Why good?
  - *There always exists a dependency-preserving lossless decomposition in 3NF.*
Decomposing into 3NF

Note: We usually simplify by using $F$ instead of $F_c$ for 3NF in this class. This gives a correct, but perhaps not as minimal, decomposition.

let $F_c$ be a canonical cover for $F$
$i := 0$
for each functional dependency $\alpha \rightarrow \beta$ in $F_c$
    $i := i + 1$
    $R_i := \alpha \beta$
if none of the schemas $R_j$, $j = 1, 2, \ldots, i$ contains a candidate key for $R$
    $i := i + 1$
    $R_i := \text{any candidate key for } R$
/* Optionally, remove redundant relations */
repeat
    if any schema $R_j$ is contained in another schema $R_k$
        /* Delete $R_j$ */
        $R_j := R_k$
        $i := i - 1$
until no more $R_j$s can be deleted
return $(R_1, R_2, \ldots, R_i)$

Figure 8.12  Dependency-preserving, lossless decomposition into 3NF.

3NF Example

- $(R) = (A,B,C,D,E,F,G,H)$
- Function Dependencies
  - $F = \{A \rightarrow CGH, AD \rightarrow C, DE \rightarrow F, G \rightarrow H\}$
    - $R_1 = \{ACGH\}$, $R_2 = \{ADC\}$, $R_3 = \{DEF\}$, $R_4 = \{GH\}$
    - $R_1 = \{ACGH\}$, $R_2 = \{ADC\}$, $R_3 = \{DEF\}$, $R_4 = \{GH\}$, $R_5 = \{ABDE\}$
    - $R_1 = \{ACGH\}$, $R_2 = \{ADC\}$, $R_3 = \{DEF\}$, $R_4 = \{ABDE\}$
  - Somewhat better if start from canonical cover $F_c =$
    - $\{A \rightarrow CGH, AD \rightarrow C, DE \rightarrow F, G \rightarrow H\}$
    - $\{A \rightarrow CGH, AD \rightarrow C, DE \rightarrow F, G \rightarrow H\}$
      - $H$ is extra in $A \rightarrow CGH$
      - $D$ extra in $AD \rightarrow C$, merge $A \rightarrow C$ into $A \rightarrow CG$
    - $\{A \rightarrow CG, DE \rightarrow F, G \rightarrow H\}$
      - $R_1 = \{ACG\}$, $R_2 = \{DEF\}$, $R_3 = \{GH\}$
      - $R_1 = \{ACG\}$, $R_2 = \{DEF\}$, $R_3 = \{GH\}$, $R_4 = \{ABDE\}$
  - Lossless: Each (except $R_4$) has a single FD that is a key
  - Preserves dependencies: each carried through a single subrelation
Mechanisms and definitions to work with FDs
- Closures, candidate keys, canonical covers etc...
- Armstrong axioms

Decompositions
- Loss-less decompositions, Dependency-preserving decompositions

BCNF
- How to achieve a BCNF schema
- BCNF may not preserve dependencies
- 3NF: Solves the above problem
- BCNF allows for redundancy
- 4NF: Solves the above problem

**BCNF and redundancy**

<table>
<thead>
<tr>
<th>MovieTitle</th>
<th>MovieYear</th>
<th>StarName</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
<tr>
<td>Indiana Jones</td>
<td>198x</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Indiana Jones</td>
<td>198x</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
<tr>
<td>Witness</td>
<td>19xx</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Witness</td>
<td>19xx</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Lot of redundancy

FDs ? No non-trivial FDs.

So the schema is trivially in BCNF (and 3NF)

What went wrong ?
Multi-valued Dependencies

- The redundancy is because of multi-valued dependencies
- Denoted:
  - starname → address
  - starname → movietitle, movieyear
- Does not happen if schema is constructed from an E/R diagram
- Functional dependencies are a special case of multi-valued dependencies

<table>
<thead>
<tr>
<th>MovieTitle</th>
<th>MovieYear</th>
<th>StarName</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
<tr>
<td>Indiana Jones</td>
<td>198x</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Indiana Jones</td>
<td>198x</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
<tr>
<td>Witness</td>
<td>19xx</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Witness</td>
<td>19xx</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
</tbody>
</table>

Multi-valued Dependencies

- FDs rule out certain tuples
- MVDs require tuples of certain form

- Read up if interested.

- Not on test.
Comparing the normal forms

<table>
<thead>
<tr>
<th></th>
<th>3NF</th>
<th>BCNF</th>
<th>4NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminates redundancy because of FD's</td>
<td>Mostly</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Eliminates redundancy because of MVD's</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Preserves FDs</td>
<td>Yes.</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
<tr>
<td>Preserves MVDs</td>
<td>Maybe</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
</tbody>
</table>

- 4NF is typically desired and achieved.
- A good E/R diagram won’t generate non-4NF relations at all
- Choice between 3NF and BCNF is up to the designer

Database design process

- So three ways to come up with a schema:
  1. Using E/R diagram
     - If good, then little normalization is needed
     - Tends to generate 4NF designs
  2. A universal relation $R$ that contains all attributes.
     - Called universal relation approach
     - Note that MVDs will be needed in this case
  3. An *ad hoc* schema that is then normalized
     - MVDs may be needed in this case
Recap

- What about 1\textsuperscript{st} and 2\textsuperscript{nd} normal forms?
- 1NF:
  - Essentially says that no set-valued attributes allowed
  - Formally, a domain is called \textit{atomic} if the elements of the domain are considered indivisible
  - A schema is in 1NF if the domains of all attributes are atomic
  - We assumed 1NF throughout the discussion
    - Non 1NF is just not a good idea
- 2NF:
  - Mainly historic interest
  - See Exercise 7.19 in the book if interested

Recap

- We would like our relation schemas to:
  - \textit{Not allow potential redundancy} because of FDs or MVDs
  - Be \textit{dependency-preserving}:
    - Make it easy to check for dependencies
    - Since they are a form of integrity constraints
- Functional Dependencies/Multi-valued Dependencies
  - Domain knowledge about the data properties
- Normal forms
  - Defines the rules that schemas must follow
  - 4NF is preferred, but 3NF is sometimes used instead
Recap

- **Denormalization**
  - After doing the normalization, we may have too many tables
  - We may *denormalize* for performance reasons
    - Too many tables → too many joins during queries
  - A better option is to use *views* instead
    - If a specific set of tables is joined often, create a view on the join

- **More advanced normal forms**
  - project-join normal form (PJNF or 5NF)
  - domain-key normal form
  - Rarely used in practice

Exam

- Definitions / short answer
- SQL queries <=> RA
- Reduce E/R diagram to relation schema
- Functional dependences
  - Armstrong's axioms, auxiliary axioms
- BCNF, 3NF
- 75 minutes, closed everything, in class.
Mappings between objects and DBs

- Object-relational mapping (ORM) systems allow
  - Specification of mapping between programming language objects and database tuples
  - Automatic creation of database tuples upon creation of objects
  - Automatic update/delete of database tuples when objects are update/deleted
  - Interface to retrieve objects satisfying specified conditions
    - Tuples in database are queried, and object created from the tuples

- Details in Section 9.6.2
  - Hibernate ORM for Java
  - Django ORM for Python

- Assignment 4 will be using the “peewee” ORM.
Databases

- Data Models
  - Conceptual representation of the data
- Data Retrieval
  - How to ask questions of the database
  - How to answer those questions
- Data Storage
  - How/where to store data, how to access it
- Data Integrity
  - Manage crashes, concurrency
  - Manage semantic inconsistencies

Query Processing/Storage

- Given a input user query, decide how to "execute" it
- Specify sequence of pages to be brought in memory
- Operate upon the tuples to produce results
- Brining pages from disk to memory
- Managing the limited memory
- Storage hierarchy
- How are relations mapped to files?
- How are tuples mapped to disk blocks?
Outline

- Storage hierarchy
- Disks
- RAID
- File Organization
- Etc....

Storage Hierarchy

- Tradeoffs between speed and cost of access
- Volatile vs nonvolatile
  - Volatile: Loses contents when power switched off
- Sequential vs random access
  - Sequential: read the data contiguously
    - select * from employee
  - Random: read the data from anywhere at any time
    - select * from employee where name like ‘__a__b’
- Why care?
  - Need to know how data is stored in order to optimize, to understand what’s going on
How important is this today?

- Trade-offs shifted drastically over last 10-15 years
  - Especially with fast network, SSDs, and large memories
  - However, the volume of data is also growing quite rapidly
- Some observations:
  - Can be cheaper to get to another computer’s memory than local disk
  - Cache is more and more important
  - Data often fits in memory of a single machine, or cluster of machines
  - “Disk” considerations less important
    - Disks still where most of the data lives today
    - Similar reasoning/algorithms will continue to be required

Storage Hierarchy

![Storage Hierarchy Diagram](image-url)
Storage Hierarchy: Cache

- Cache
  - Super fast; volatile; Typically on chip
  - L1 vs L2 vs L3 caches ???
    - L1 about 64KB or so; L2 about 1MB; L3 8MB (on chip) to 256MB (off chip)
    - Huge L3 caches available now-a-days
  - Becoming more and more important to care about this
    - Cache misses are expensive
  - Similar tradeoffs as were seen between main memory and disks
  - Cache-coherency ??
Storage Hierarchy: Cache

K8 core in the AMD Athlon 64 CPU

- Main memory
  - 10s or 100s of ns; volatile
  - Pretty cheap and dropping: 1GByte << $100
  - Main memory databases feasible now-a-days
- Flash memory (EEPROM)
  - Limited number of write/erase cycles
  - Non-volatile, slower than main memory (especially writes)
  - Examples?
- Question
  - How does what we discuss next change if we use flash memory only?
  - Key issue: Random access as cheap as sequential access