CS 540
Database Management Systems

Constraints, Schema Normalization

Oregon State University

Constraints

- Restrict the data stored in the database.
  - **C1:** ssn contains only numerical symbols.
  - **C2:** People with same ssn have same name.

Emp

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2nd St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

Oregon State University

Constraints

- Express and generate new rules on data
  - **C3:** If two people share all addresses, they are related.
  - **C4:** If two people are related, they share last names.

  => **C5:**If two people share all addresses, they share last names.

- Common sense or domain-specific rules
- Generation = logical implication

Constraints

- Perform inference on the data
  - **C3:** If two people share all addresses, they are related.

    John & *Mary* share all addresses

    => *John* & *Mary* are related

- Probabilistic rules and inference
  - **C6:** If two people share an address, they are related. (p=0.6)

    John & *Mary* share an address

    => *John* & *Mary* are related (p=0.6)

- We use **constraints** to express rules & restrictions.

OSU Oregon State UNIVERSITY

Challenges of Constraints on Large Data

- Hard to check efficiently
  - **C2:** Tuples with equal ssn values have equal names.
  - Efficient algorithm?

- Hard to do (probabilistic) inference
  - **C3:** If two people share an address, they are related. (0.6)
  - **C6:** If two people have similar names, they are related. (0.4)
  - ....

    John & *Mary* share an address,

    John & *Joe* have similar names,

    …

    => *John* & *Mary* are related ?

Challenges of Constraints on Large Data

- Hard to generate new rules from existing ones
  - Efficient algorithm for first-order logic rules?

- Systems implement subsets of constraints
  - Frequently used
  - Efficient methods of checking & generating new rules

- We focus on **Functional Dependencies (FD)**
  - Other subsets with efficient algorithms

    Alice Book (Part C)
  - FDs as template to learn others.

Challenges of Constraints on Large Data

- We discuss checking and rules generation

- Probabilistic inference is discussed at the end of the course.

Functional dependency (FD)

- Given set of attributes X, Y in relation R, the **functional dependency** X→ Y means that all tuples in R that agree on attributes in X must also agree on Y.

`ssn` → `name`? Yes

`ssn` → `address` ? No

`Emp`

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2nd St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

Keys

- A **key** in R is a set of attributes of R that functionally determines *all attributes* in R and *none of its subsets* is a key.
  - `ssn?  {ssn, address}?  {ssn, name, address}?`
- **Super-key** is a set of attributes that contains a key.

Emp

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2nd St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

Oregon State University

How to find FDs?

- From the domain and experts.
- Logically implied by other FDs.

- *Example*:

  `movies(title, year, actor, cost, revenue, b-buster)`

  FD_1 : `title, year, actor` ➔ `cost`

  FD_2 : `title, year, actor` ➔ `revenue`

  Implied FD_3: `title, year, actor` ➔ `cost, revenue`

Closure of a set of FDs

- All FDs implied by a set of FDs.
- Logical implication is hard for generic rules
  - But, efficient for FDs!
- Generate new FDs using Armstrong's axioms.
- Using the closure of FDs, we can find keys.

Oregon State University

Armstrong's axioms

- **Reflexivity**: $A_1,\ldots, A_n \rightarrow A_1$
  generally, $A_1,\ldots, A_n \rightarrow A_i,\ldots, A_j$; $1 <= i, j <= n$ (*trivial FD*)

- **Augmentation**:
  If $A_1,\ldots, A_n \rightarrow B_1,\ldots, B_m$ then
  $A_1,\ldots, A_n, C_1,\ldots, C_k \rightarrow B_1,\ldots, B_m, C_1,\ldots, C_k$

- **Transitivity**:
  If $A_1,\ldots, A_n \rightarrow B_1,\ldots, B_m$ and $B_1,\ldots, B_m \rightarrow C_1,\ldots,C_k$
  then $A_1,\ldots, A_n \rightarrow C_1,\ldots, C_k$

Computing the closure of a set of FDs (U)

- **U+ = U.**
- Repeat
  - Apply **reflexivity** and **augmentation** to each FD in **U+** and add the resulting FDs to **U+**.
  - Apply **transitivity** to each pairs of FDs in **U+** and add the resulting FDs to **U+**.
- Until **U+** does not change anymore.

Useful rules

- They are derived from axioms and we may use them to derive implied FDs faster.
- **Decomposition**

  If $A_1,\ldots, A_n \rightarrow B_1,\ldots, B_m$ then $A_1,\ldots, A_n \rightarrow B_1$, $A_1,\ldots, A_n \rightarrow B_2$, …, and $A_1,\ldots, A_n \rightarrow B_m$.

  Proof: ?
- **Union**

  If $A_1,\ldots, A_n \rightarrow B_1$, …, and $A_1,\ldots, A_n \rightarrow B_m$ then $A_1,\ldots, A_n \rightarrow B_1,\ldots, B_m$.

  Proof: assignment

Computing the closure of a set of FDs

`movies(title, year, actor, cost, revenue, b-buster)`

FD_1 `title, year, actor` ➔ `cost`

FD_2 `title, year, actor` ➔ `revenue`

FD_3 `cost, revenue` ➔ `b-buster`

Apply union on FD_1 and FD_2:

FD_4: `title, year, actor` ➔ `cost, revenue`

Apply transitivity on FD_4 and FD_3:

FD_5: `title, year, actor` ➔ `b-buster`

Apply union on FD_4 and FD_5:

FD_6: `title, year, actor` ➔ `cost,revenue,b-buster`

Oregon State University

Enforcing FDs

- Update '*John*' to '*Richard*' in the first tuple.
  - violates `ssn` ➔ `name`.
  - update anomaly.

Emp

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2nd St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

- Write a program that checks the database after each update for violations of all FDs.
  - Large relations and many FDs => inefficient.

More problems

- Delete John’s addresses => lose his ssn and name.
  - **deletion anomaly**.
- Insert a tuple with new ssn and name but no address.
  - **insertion anomaly**.

`Emp`

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2nd St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

- One may use NULL values to solve these problems.
  - They do not necessarily indicate lack of knowledge.
  - It is hard to write correct SQL queries over the database.

Normalization

- Transform the schema to a schema without update, deletion, and insertion anomalies.

Emp

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2^{nd} St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

➔

Emp-name

ssn	name
111122222	John
454565761	Charles

Emp-addr

ssn	address
111122222	21 Kings St.
111122222	234 2^{nd} St.
454565761	31 Kings St.
454565761	2 Harrison St.

- update anomaly?
- deletion anomaly?
- insertion anomaly?

Normalization

- Given schema $\mathbf{S_1}$, find schema $\mathbf{S_2}$ that does not have update/deletion/insertion anomalies.

`Emp`

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2nd St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

➔

`Emp-name`

ssn	name
111122222	John
454565761	Charles

`Emp-addr`

ssn	address
111122222	21 Kings St.
111122222	234 2nd St.
454565761	31 Kings St.
454565761	2 Harrison St.

$\mathbf{S_1}$ = ({`Emp`}, {`ssn`➔`name`})

$\mathbf{S_2}$ = ({`Emp-name`, `Emp-addr`}, {`ssn`➔`name`})

- Schema $\mathbf{S_2}$ is in a **normal form.**

Many normal forms

	Normal form	Defined by	In
1NF	First normal form	Two versions: E.F. Codd (1970), C.J. Date (2003)	1970 [1] and 2003 [9]
2NF	Second normal form	E.F. Codd	1971 [2]
3NF	Third normal form	Two versions: E.F. Codd (1971), C. Zaniolo (1982)	1971 [2] and 1982 [10]
EKNF	Elementary Key Normal Form	C. Zaniolo	1982 [10]
BCNF	Boyce–Codd normal form	Raymond F. Boyce and E.F. Codd	1974 [11]
4NF	Fourth normal form	Ronald Fagin	1977 [12]
5NF	Fifth normal form	Ronald Fagin	1979 [13]
DKNF	Domain/key normal form	Ronald Fagin	1981 [14]
6NF	Sixth normal form	C.J. Date, Hugh Darwen, and Nikos Lorentzos	2002 [15]

Oregon State UNIVERSITY

Boyce-Codd normal form (BCNF)

- Relation R is in **BCNF**, if and only if:
  - For each non-trivial FD X ➔Y, X is a super-key of R.
- Every attribute depends only on super-keys.

- **Example:** `ssn`➔`name`
  - ssn is not a super-key.
  - *Employee* is not in BCNF.

`Emp`

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2nd St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

Oregon State University

Boyce-Codd normal form (BCNF)

- **Example:** `ssn`➔`name`
  - ssn is not a super-key.
  - *Employee* is not in BCNF.

Emp

ssn	name	address
111122222	John	21 Kings St.
111122222	John	234 2nd St.
454565761	Charles	31 Kings St.
454565761	Charles	2 Harrison St.

- Decompose *Emp*.
  - super-key of *Emp-name*?
  - super-key of *Emp-addr?*
  - Schema is in BCNF.

Emp-name

ssn	name
111122222	John
454565761	Charles

Emp-addr

ssn	address
111122222	21 Kings St.
111122222	234 2nd St.
454565761	31 Kings St.
454565761	2 Harrison St.

BCNF decomposition of R

- Find the closure of FDs in R.
- Find the keys for R.
- Pick an FD $A \rightarrow B$ that violates the BCNF condition in R.
  - select the largest possible B.
- Decompose relation R to relational R1 and R2.

R

Other attributes	A	B

R1

Other attributes	A

R2

A	B

- Repeat until there is no BCNF violation left.

BCNF decomposition example

`Emp2(ssn, name, street, city, state, zip)`

FD$_1$ `ssn` → `name`

FD$_2$ `zip` → `state`

Key: `{ssn, street, city, zip}`

FD$_1$ violates BCNF.

`Emp2(ssn, name)`

`Emp-addr(ssn, street, city, state, zip)`

FD$_2$ violates BCNF.

`Emp2(ssn, name)`

`Emp-addr(ssn, street, city, zip)`

`Location(zip, state)`

Oregon State University

The danger of normalization

- Losing information.
- We like to preserve the information stored in R.
  - Recover the tuples in R from R's BCNF decomposition: lossless decomposition
  - Recover all FDs in R from its BCNF decomposition: dependency preserving
- Check these conditions after normalizing a relation.

Lossless decomposition

- If (R1, R2) is a decomposition of R, then

  Join (R1, R2) = R for all instances of R, R1, R2.
- **Example:** R(A, B, C) ➔ R1(A, B), R2(A, C)

A	B	C
a1	b1	c1
a1	b2	c2

➔

A	B
a1	b1
a1	b2

A	C
a1	c1
a1	c2

A	C
a1	c1
a1	c2

A	B
a1	b1
a1	b2

➔

A	B	C
a1	b1	c1
a1	b1	c2
a1	b2	c1
a1	b2	c2

we get bogus tuples, not a lossless decomposition.

Oregon State University

BCNF decomposition is lossless

- **Example:** {R(A, B, C), A→B} {R1 (A, B), R2 (A, C), A→B}

A	B	C
a1	b1	c1
a1	b1	c2

→

A	B
a1	b1

A	C
a1	c1
a1	c2

A	C
a1	c1
a1	c2

A	B
a1	b1

→

A	B	C
a1	b1	c1
a1	b1	c2

- The join does *not* produce bogus tuples.
- Generally, if the join attribute is a key, we are OK.
- It is true for all BCNF decompositions.

Oregon State University

BCNF is **not** dependency preserving

`Emp(ssn, name, address), ssn→name, name,address→ssn`

BCNF decomposition:

`Emp-name(ssn, name) ssn→name`

`Emp-addr(ssn, address)` No FD! **not** dependency preserving

- What will happen if we lose dependency?

`Emp-name`

ssn	name
111122222	John
444455555	John

`Emp-addr`

ssn	address
111122222	21 Kings St.
111122222	21 Kings St.
444455555	21 Kings St.
444455555	21 Kings St.

→

`Emp`

ssn	name	address
111122222	John	21 Kings St.
111122222	John	21 Kings St.
444455555	John	21 Kings St.
444455555	John	21 Kings St.

Looks OK to the system

Does **not** satisfy `name,address→ssn`

Oregon State University

BCNF is **not** dependency preserving

- What will happen if we lose a dependency?

`Emp-name`

ssn	name
111122222	John
444455555	John

`Emp-addr`

ssn	address
111122222	21 Kings St.
111122222	21 Kings St.
444455555	21 Kings St.
444455555	21 Kings St.

➔

`Emp`

ssn	name	address
111122222	John	21 Kings St.
111122222	John	21 Kings St.
444455555	John	21 Kings St.
444455555	John	21 Kings St.

Looks OK to the system

Does **not** satisfy `name,address`➔`ssn`

Given FD A ➔B, If normalization puts A and B in different relations, it is not dependency preserving.

Dependency preserving normal form: 3NF

- Relation R is in 3rd normal form if for each non-trivial FD X➔Y in R
  - X is a super-key or
  - Y is a part of a key.

- Every non-key attribute must depend on a key, the whole key, and nothing but the key!

Dependency preserving normal form: 3NF

- Relation R is in 3rd normal form if for each non-trivial FD X➔Y in R
  - X is a super-key or
  - Y is a part of a key.
- **Example:**

```
Emp(ssn, name, address)
ssn➔name, address,name➔ssn
```

3NF but not BCNF.

- 3NF is a lossless decomposition and preserves dependencies.

Minimal basis (minimal cover)

- The FD sets **U1** and **U2** are *equivalent* if and only if

  U1+ = U2+.

- **U2** is a *minimal basis* for **U1** iff:
  - **U2** and **U1** are equivalent.
  - The FDs in **U2** have only one attribute in their right-hand side.
  - If we remove an FD from **U2** or an attribute from an FD in **U2**, **U2** will not be equivalent to **U1**.

- **Intuition:** a smaller set of FDs with fewer attributes that implies **U+.**

Oregon State University

Finding minimal basis of U

1. **Standard form**: replace each FD $A_1,\ldots,A_n \rightarrow B_1, \ldots, B_m$ by $A_1,\ldots,A_n \rightarrow B_1, \ldots, A_1,\ldots,A_n \rightarrow B_m$

2. **Minimize left hand side**: for each attribute A_j and each FD $A_1,\ldots, A_j,\ldots, A_n \rightarrow B_i$, check if you can remove A_j from the FD while preserving **U**+.

1. **Delete redundant FD:** check each remaining FD to see if you can remove it while preserving **U+.**

- The minimal basis for **U** is not necessarily unique.

Finding minimal basis of **U**

- **Example:**

  U = {B ➔ A, D ➔ A, AB ➔ D}

1. It is already in standard form.
2. We can remove A from {AB ➔ D} to get

   U = {B ➔ A, D ➔ A, B ➔ D}

3. We can remove B ➔ A as it is implied by transitivity from others.

3NF synthesizing algorithm

Input: relation R and set of FDs **U.**

Output: Normalized schema S

1. Find a minimal basis **M** for **U.**
2. For each FD A➔B in **M**, if AB is not covered by any relation in S, add Ri = (A,B) to S.
3. If none of the relations in S contain a super-key for R, add a relation to S whose attributes form a key for R.

3NF synthesizing algorithm

1. Find a minimal basis **M** for **U.**
2. For each FD A→B in **M**, if AB is not covered by any relation in S, add Ri = (A,B) to S.
3. If none of the relations in S contain a super-key for R, add a relation to S whose attributes form a key for R.

- *Why step 3?*

Consider relation R(A,B,C) with **U**={A→B, C→B}.

Minimal basis of **U** is **U** : 3NF = $R_1(\underline{A},B)$ $R_2(\underline{C},B)$

lossless join property? 3NF = $R_1(\underline{A},B)$ $R_2(\underline{C},B)$ $R_3(\underline{A},\underline{C})$

3NF versus BCNF

- BCNF eliminates more redundancies than 3NF.
- Normalization must be dependency preserving, unless there is a strong reason.
- Try BCNF, but if it is not dependency preserving use 3NF.

De-normalization

- Normalization improves data quality, but it has some drawbacks.
  - **Performance**
    - queries on normalized schemas need more joins.
  - **Readability:** normalized schemas are hard to understand.
    - more relations.
    - related attributes in different relations
    - many connections between relations.

De-normalization

- DB designers find the trade-off based on workload.
  - No write in Online Analytical Processing (OLAP)
    - ➔ argues against normalization.
  - Write in Online Transaction Processing (OLTP)
    - ➔ argues for normalization.