The accounts payable system in third normal form

This is our goal: “third normal form” for the tables.

The first three normal forms

First (1NF)
The value stored at the intersection of each row and column must be a scalar value, and a table must not contain any repeating columns.

Second (2NF)
Every non-key column must depend on the entire primary key.

Third (3NF)
Every non-key column must depend only on the primary key.

Most designers stop at the third normal form

We need to work on what “depends on” really means here. Also the 3NF definition assumes only one key in the table, see R&G, pg. 617 for complete definition that covers cases of multiple keys for a table.

The next four normal forms

Boyce-Codd (BCNF)
A non-key column can’t be dependent on another non-key column.

Fourth (4NF)
A table must not have more than one multivalued dependency, where the primary key has a one-to-many relationship to non-key columns.

Fifth (5NF)
The data structure is split into smaller and smaller tables until all redundancy has been eliminated.

Domain-key (DKNF) or sixth (6NF)
Every constraint on the relationship is dependent only on key constraints and domain constraints, where a domain is the set of allowable values for a column.

Of these, we’ll cover only BCNF. The above definition of BCNF isn’t clear—see R&G, pg 616. Note that BCNF is the same as 3NF for tables with one key.

The benefits of normalization

- More tables, and each table with an index for its primary key. That makes data retrieval more efficient.
- Each table contains information about a single entity. That makes data retrieval and insert, update, and delete operations more efficient.
- Data redundancy is minimized, which simplifies maintenance and reduces storage.
- In cases of huge tables (data warehousing) the benefits of normalization may be outweighed by need to keep related data nearby for fast access.

Note that noSQL databases like MongoDB usually keep related data together in “documents”, i.e., use denormalized models. See MongoDB documentation.
Normalization examples

The invoice data with a column that contains repeating values

<table>
<thead>
<tr>
<th>invoice_id</th>
<th>invoice_number</th>
<th>invoice_total</th>
<th>invoice_due_date</th>
<th>vendor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
<td>12/31/2023</td>
<td>Vendor 1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>200</td>
<td>1/31/2024</td>
<td>Vendor 2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>300</td>
<td>2/28/2025</td>
<td>Vendor 3</td>
</tr>
</tbody>
</table>

Normalization examples

The invoice data with a column that contains repeating values

<table>
<thead>
<tr>
<th>invoice_id</th>
<th>invoice_number</th>
<th>invoice_total</th>
<th>invoice_due_date</th>
<th>vendor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
<td>12/31/2023</td>
<td>Vendor 1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>200</td>
<td>1/31/2024</td>
<td>Vendor 2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>300</td>
<td>2/28/2025</td>
<td>Vendor 3</td>
</tr>
</tbody>
</table>

These examples are not even first normal form

First (1NF)
The value stored at the intersection of each row and column must be a scalar value, and a table must not contain any repeating columns.

The invoice data in first normal form

<table>
<thead>
<tr>
<th>invoice_id</th>
<th>invoice_number</th>
<th>invoice_total</th>
<th>invoice_due_date</th>
<th>vendor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
<td>12/31/2023</td>
<td>Vendor 1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>200</td>
<td>1/31/2024</td>
<td>Vendor 2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>300</td>
<td>2/28/2025</td>
<td>Vendor 3</td>
</tr>
</tbody>
</table>

The invoice data in second normal form

Every non-key column must depend on the entire primary key.

The A/P system in second normal form

<table>
<thead>
<tr>
<th>invoice_id</th>
<th>invoice_number</th>
<th>invoice_total</th>
<th>invoice_due_date</th>
<th>vendor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
<td>12/31/2023</td>
<td>Vendor 1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>200</td>
<td>1/31/2024</td>
<td>Vendor 2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>300</td>
<td>2/28/2025</td>
<td>Vendor 3</td>
</tr>
</tbody>
</table>

“Y depends on X” idea

- We will formalize this idea soon
- If “Y depends on X”, then each X value is associated with precisely one Y value in the table. i.e., if 2 rows agree on X, they agree on Y
  - Here: vendor_name depends on invoice_id
  - 2 rows that agree on invoice_id must agree on vendor_name
- That makes sense: the invoice is talking about one vendor selling things.
- On the other hand, item_description depends on (invoice_id, invoice_sequence)
  - OK: the item being described is different for each invoice_sequence (each line item of the invoice)
Third normal form: Every non-key column must depend only on the primary key, not some other column for example. (Here assuming the table has only one key, “the” primary key)

We clearly want a vendor_id, a sure unique id, so we can say the vendor_address depends on the vendor_id, etc.

Here vendor_address depends on vendor_name (assuming that’s a good id), not on just invoice_id, the table’s PK.

Third normal form: Every non-key column must depend only on the primary key, not some other column for example. (Here assuming the table has only one key, “the” primary key)

Here X → Y means Y depends on X: if 2 rows agree on X, they also agree on Y, or, equivalently, disagree on Y iff disagree on X

It’s always true that key → other cols

Here also R → W: agree on R means agree on W

Why Schema Refinement?

We have learned the advantages of relational tables … but how to decide on the relational schema?

At one extreme, store everything in single table

Huge redundancy

Leads to anomalies!

We need to break the information into several tables

How many tables, and with what structures?

Having too many tables can also cause problems

E.g., performance, difficulty in checking constraints

Sample Relation

Hourly_Emps (ssn, name, lot, rating, wage, hrs_worked)

Denote relation schema by attribute initials: SNLRWH

Constraints (functional dependencies, or FDs for short)

ssn is the key: S → SNLRWH

rating determines wage: R → W

E.g., worker with rating 10 receives 20$/hr

Here X → Y means Y depends on X: if 2 rows agree on X, they also agree on Y, or, equivalently, disagree on Y iff disagree on X

It’s always true that key → other cols

Here also R → W: agree on R means agree on W

Problems due to R → W in Hourly_Emps

Update anomaly: Change value of W only in a tuple, end up with a dependency violation

Insertion anomaly: How to insert employee if we don’t know hourly wage for that rating?

Deletion anomaly: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>5</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>
Removing Anomalies

Wages  
<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>H</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>40</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>30</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Create 2 smaller tables!

Finding another FD lying in a table

From hw4, and createdb.sql:

```
cREATE TABLE flights(
    flno int PRIMARY KEY,
    origin VARCHAR(20) NOT NULL,
    destination VARCHAR(20) NOT NULL,
    depars VARCHAR(20),
    arrives VARCHAR(20),
    price DECIMAL(7,2))
```

What’s distance? It’s the distance between the origin and destination airports, so the FD: origin, destination → distance lies in the table.

We can check the data for FD consistency

```
SQL> SELECT origin, destination, MIN(distance), MAX(distance) FROM flights
2 GROUP BY origin, destination;
```

```
ORIGIN               DESTINATION          MIN(DISTANCE) MAX(DISTANCE)
-------------------- ------------------ ------------------
Los Angeles          Chicago                       1749          1749
Detroit              New York                       470           470
Pittsburgh           New York                       303           303
Los Angeles          Sydney                        7487          7487
Chicago              New York                      802           802
Los Angeles          Honolulu                      2551          2551
...                  ...                           ...             ...
```

The rest check out too: there’s only one distance between two cities in the table.

```
SQL> SELECT origin, destination, COUNT(*) FROM flights
2 GROUP BY origin, destination;
```

```
ORIGIN               DESTINATION            COUNT(*)
-------------------- ------------------ ----------
Los Angeles          Chicago                       1
Detroit              New York                      1
Pittsburgh           New York                      1
Los Angeles          Sydney                        1
Chicago              New York                      1
Los Angeles          Honolulu                      2
Madison              Pittsburgh                    1
Los Angeles          Tokyo                         1
Madison              Detroit                       1
...                  ...                           ...
```

The rest are 1s, so only LA to Honolulu tests the FD, but it passed. LA to Honolulu: 2 rows with the same origin, destination, same distance.

Why do we care?

Possible anomalies in this situation:

- **Delete anomaly:**
  Delete all flights from Boston to Ithaca
  End up losing distance information on this link

- **Insert anomaly:**
  Add a flight from Boston to Ithaca
  Need to check if the distance is consistent with other rows

- **Update anomaly:**
  Correct the distance: need to check for all the cases, or break the dependency.

  As a database professional, need to keep an eye on table designs and possibly point out potential problems, esp. early, before the group has invested a lot of development work in their design.

Normalization in practice

We can create another table to hold this FD information:

```
cREATE TABLE links(
    origin VARCHAR(20),
    destination VARCHAR(20),
    distance INT,
    PRIMARY KEY (origin, destination)
);
cREATE TABLE flights(
    flno INT PRIMARY KEY,
    origin VARCHAR(20) NOT NULL,
    destination VARCHAR(20) NOT NULL,
    depar VARCHAR(20),
    arrives VARCHAR(20),
    price DECIMAL(7,2),
    FOREIGN KEY (origin, destination) REFERENCES links
);
```
Dealing with Redundancy

- **Redundancy** is at the root of redundant storage, insert/delete/update anomalies.
- Integrity constraints, in particular **functional dependencies**, can be used to help identify redundancy.
- Main refinement technique: **decomposition** (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
  - Decomposition may sometimes affect performance. Why?
  - What problems (if any) does decomposition cause?
    - Incorrect data
    - Loss of dependencies

Functional Dependencies (FDs)

- A functional dependency **X → Y** holds over relation R if for every instance r of R:
  - \( t_1, t_2 \in r, \pi_X(t_1) = \pi_X(t_2) \implies \pi_Y(t_1) = \pi_Y(t_2) \)
- Given two tuples in r, if the X values agree, Y values must also agree.
- In R \( \rightarrow \ W \) example, all emps with rating 8 have wage 10, etc.
- FD is a statement about all allowable relations.
- Identified based on semantics of application (business logic).
- Given an instance r of R, we can check if it violates some FD f, but we cannot tell if f holds over R!

FDs and Keys

- FDs are a **generalization** of keys.
  - A key uniquely identifies all attribute values in a tuple.
  - That is a particular case of FD …
  - … but not all FDs must determine ALL attributes.
- K is a **key** for R means that K \( \rightarrow \) R
- However, K \( \rightarrow \) R does not require K to be **minimal**.
- K can be a **superkey** as well.

Reasoning About FDs

- Given FD set F, we can usually infer additional FDs:
  - \( \overline{\overline{F}} = \text{closure of } F \) is the set of all FDs that are implied by F.
- **Armstrong's Axioms** (X, Y, Z are sets of attributes):
  - Reflexivity: If \( Y \subseteq X \), then \( X \rightarrow Y \)
  - Augmentation: If \( X \rightarrow Y \), then \( XZ \rightarrow YZ \) for any Z.
  - Transitivity: If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \).
- These are sound and **complete** inference rules for FDs!

Reasoning About FDs (cont'd)

- Additional rules
  - Not necessary, but helpful.
  - Can be proved from Armstrong’s axioms.
- Union and decomposition (splitting):
  - \( X \rightarrow Y \) and \( X \rightarrow Z \) imply \( X \rightarrow YZ \)
  - \( X \rightarrow YZ \) implies \( X \rightarrow Y \) and \( X \rightarrow Z \).

An Example of FD Inference

- Pg. 613: a contract with id C is an agreement that a supplier S will supply Q items of part P to project J associated with department D: the value of this contract is V.
- Contracts(cid, sid, jid, did, pid, qty, value), and:
  - Contract id, supplier, project, department, part
  - C is the key: \( C \rightarrow CSJDPOV \)
  - Project purchases each part using single contract: \( JP \rightarrow C \)
  - A certain project J and part P purchased determines a particular contract C.
  - Dept purchases at most one part from a supplier: \( SD \rightarrow P \)
  - A certain department D and supplier determine a certain part P if any...
An Example of FD Inference

Contracts(cid,sid,jid,did,pid,qty,value), and:
- Contract id, supplier, project, department, part
- C is the key: C → CSJDPQV
- Project purchases each part using single contract: JP → C
- Dept purchases at most one part from a supplier: SD → P

JP → C, C → CSJDPQV imply JP → CSJDPQV
SD → P implies SDJ → JP
SDJ → JP, JP → CSJDPQV imply SDJ → CSJDPQV
So SDJ is a superkey for the table.

Another Example of FD Inference

Relation Hourly(ssn, rating, wage)
Given S→R, R→W
- Then S→W by transitivity
- S→R, S→W implies S→RW by union
- SS→RWS by augmentation
- S→RWS by definition of SS as union of attributes
- Thus S is a superkey, and indivisible, so a key
- And R→W is a FD lying in the table outside the key
- This non-key FD drives a decomposition, as we will see.

Attribute Closure

Attribute closure of X (denoted X+) wrt FD set F:
- Set of all attributes A such that X → A is in F+
- Set of all attributes that can be determined starting from attributes in X and using FDs in F

Algorithm:
- X+ = X
- Repeat
- Y=X+ (remember starting X+ value for this pass)
- Search all FDs in F with LHS completely included in X+
- Add RHS of each such FD to X+
- Until Y=X+ (same as at start of this pass)

Relation Hourly(ssn, rating, wage)
Given (1) S→R, (2) R→W
S+ = S, look for FDs with LHS in S, find S→ R
S+ = SR, look for FDs with LHS in SR, find R→W
S+ = SRW = Relation, so S is a superkey, and a key
R+ = R, look for FDs with LHS in R, find R→W
R+ = RW, done
W+ = W, done

Attribute Closure : second example

X+ = X
Repeat
Y=X+ (remember starting X+ value)
Search all FDs in F with LHS completely included in X+
Add RHS of each such FD to X+
Until Y=X+ (same as at start of this pass)

Relation Hourly(ssn, rating, wage)
Given (1) S→R, (2) R→W
S+ = S, look for FDs with LHS in S, find S→ R
S+ = SR, look for FDs with LHS in SR, find R→W
S+ = SRW = Relation, so S is a superkey, and a key
R+ = R, look for FDs with LHS in R, find R→W
R+ = RW, done
W+ = W, done

Verifying if a given FD is in FD-set closure

- Computing the closure of a set of FDs (F+) can be expensive
- Size of closure is exponential in number of attributes!
- But if we just want to check if a given FD X→Y is in the closure of a set of FDs F:
- Can be done efficiently without need to know F+
- Compute X+ wrt F
- Check if Y is in X+

Example: is JDP→V in F+?
Compute JDP→ JDPJ by JDP→ C
JDPJ→ R by C→ R, so contains V
Answer: Yes
Verifying if a given FD is in FD-set closure (another example)

- But if we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs $F$:
  - Can be done efficiently without need to know $F^+$
  - Compute $X^+ \text{ wrt } F$
  - Check if $Y$ is in $X^+$

Example: Relation Hourly(ssn, rating, wage)
Given $F$ (1) $S \rightarrow R$, (2) $R \rightarrow W$
Is $S \rightarrow W$ in $F^+$?
Compute $S^+ = SR$ by (1), $= SRW$ by (2)
We see $W$ in $S^+$
Answer: Yes

Verifying if attribute set is a key

- Key verification can also be done with attribute closure
- To verify if $X$ is a key, two conditions needed:
  - $X^+ = R$
  - $X$ is minimal

How to test minimality
- Removing an attribute from $X$ results in $X'$ such that $X'^+ \not< R$

Normal Forms: next time

- If a relation is in a certain normal form (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized.

Simple cases:
- Consider a relation $R$ with attributes $AB$
  - No FDs hold: There is no (provable) redundancy, so is BCNF and 3NF
  - Given $A \rightarrow B$:
    - Several tuples could have the same A value
    - If so, they'll all have the same B value!
    - But a relation is a set of rows, so this can't happen
    - So there are no duplicates in A, and A must be a key
    - Sure $A \rightarrow B$, so $AA \rightarrow AB$, $AA = A$, so $A \rightarrow AB$, key definition
    - This will turn out to be BCNF and 3NF